Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Tomaž Borštnik

Analiza in primerjava javanskih tehnologij za spletni sloj

diplomsko delo

univerzitetni študijski program prve stopnje
računalništvo in informatika

Mentor: prof. dr. Matjaž Branko Jurič

Ljubljana 2012

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Univerza *v Ljubljani*
Fakulteta *za računalništvo in informatiko*

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00046/2012
Datum: 13.04.2012

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **TOMAŽ BORŠTNIK**

Naslov: **ANALIZA IN PRIMERJAVA JAVANSKIH TEHNOLOGIJ ZA SPLETNI SLOJ**

ANALYSIS AND COMPARISON OF JAVA WEB TIER TECHNOLOGIES

Vrsta naloge: Diplomsko delo univerzitetnega študija prve stopnje

Tematika naloge:

Proučite in analizirajte tehnologije Java Server Faces, Apache Struts, Google Web Toolkit in jQuery. Tehnologije primerjajte po zastavljenih kriterijih. Izdelajte praktično aplikacijo v eni od izbranih tehnologij.

Mentor:

prof. dr. Matjaž B. Jurič

Dekan:

prof. dr. Nikolaj Zimic

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Tomaž Borštnik, z vpisno številko **63090025**, sem avtor diplomskega dela z naslovom:

Analiza in primerjava javanskih tehnologij za spletni sloj

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom prof. dr. Matjaža Branka 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 v zbirki "Dela FRI".

V Ljubljani, dne 25. septembra 2012 Podpis avtorja:

Profesorju dr. Matjažu Branku Juriču se iskreno zahvaljujem za mentorstvo, pomoč in strokovne nasvete.

Za potrpežljivost in podporo pri študiju se zahvaljujem tudi svojim staršem.

Kazalo

Povzetek

Uporabniški vmesniki so pomemben del aplikacij, kajti z njihovo pomočjo omogočimo uporabo programske opreme, jo olajšamo ter pospešimo oziroma naredimo uporabniško interakcijo bolj učinkovito. Poleg tega lahko povečamo zadovoljstvo uporabnikov in s privlačnim uporabniškim vmesnikom privabimo tudi nove. Uporabniški vmesniki pravzaprav podajajo uporabniško izkušnjo. Zato je poznavanje tehnologij, ki nam omogočajo učinkovit razvoj dobrih uporabniških vmesnikov, ključnega pomena. Cilj diplomske naloge je analizirati in primerjati nekatere tehnologije za spletni sloj ter razvoj preproste spletne aplikacije. V teoretičnem delu je bilo treba preučiti tehnologije JavaServer Faces, Apache Struts, GWT in jQuery. Po opravljenem teoretičnem delu se je bilo treba osredotočiti na eno izmed omenjenih tehnologij in izdelati preprosto spletno aplikacijo. Zato je bila razvita spletna igra Potapljanje ladjic, ki temelji na tehnologiji GWT. S tem smo želeli ugotoviti prednosti in slabosti posameznih tehnologij oziroma narediti primerjavo med njimi.

Ključne besede

Java EE, JSF, Apache Struts, GWT, jQuery, uporabniški vmesnik

Abstract

User interfaces are an important part of applications, with their help user interaction is possible, eased, sped up and made more effective. In addition, user satisfaction can be increased and with an attractive user interface new users can be attracted. Therefore, knowing the technologies and features which enable us a fast and effective development of good user interfaces is crucial. The goal of this assignment is to analyze and compare some of the technologies used in web application development and to develop a simple web application. In the theoretical part of the assignment JavaServer Faces, Apache Struts, GWT and jQuery had to be examined. After the theoretical work it was necessary to focus on one of these technologies and build a simple web application. Therefore, a web game based on GWT technology named 'Potapljanje ladjic' was built. This was to compare and determine the advantages and disadvantages of each technology.

Key words:

Java EE, JSF, Apache Struts, GWT, jQuery, user interface

Poglavje 1

Uvod

Izkušnja uporabnika spletne aplikacije je močno odvisna od uporabniškega vmesnika, katerega aplikacija ponuja. Uporabniški vmesniki so prav tako pomembni, gledano s perspektive razvijalcev. Razvijalci si želijo hiter in preprost razvoj le-teh. Obstajajo različni načini povezave med komponentami uporabniškega vmesnika, ki se izvaja na odjemalčevem brskalniku in strežniške kode, ki obdeluje prejete podatke oziroma zahteve. Tehnologije se razlikujejo tudi po načinih prehajanja med stranmi. Mnoge tehnologije ponujajo tudi množico dodatnih komponent za lažjo komunikacijo uporabnika s spletno aplikacijo.

Podjetja ponujajo razvijalcem množico tehnologij za razvoj spletnih aplikacij na različnih platformah. V diplomski nalogi je predstavljena platforma Java EE. Za razvoj javanskih spletnih aplikacij prav tako obstaja veliko ogrodij in knjižnic. Razširjena javanskima ogrodja so Spring MVC, Grails, GWT, Struts in JSF. V diplomski nalogi smo se osredotočili na ogrodja GWT, JSF in Struts ter knjižnico JavaScript poznano po imenu jQuery.

Motivacija diplomske naloge je učinkovita in hitra gradnja dobrih uporabniških vmesnikov. S tem spletne aplikacije privabljajo večje število uporabnikov in omogočajo boljšo uporabniško izkušnjo. Tako smo v diplomski nalogi preučili zgoraj naštete tehnologije. Tehnologija JSF je strežniško ogrodje za gradnjo uporabniških vmesnikov. Struts je spletno ogrodje za

razvoj javanskih spletnih aplikacij. Njegova dobra lastnost je dobra kompatibilnost s tehnologijama, kot sta REST in AJAX. Naslednja tehnologija, imenovana GWT, je ogrodje, namenjeno razvoju in optimizaciji programske kode, ki se izvaja na odjemalcu. Ravno tako kot Struts zagotavlja odlično podporo tehnologiji AJAX. Zadnja tehnologija je knjižnica JavaScript imenovana jQuery. Razvijalcem omogoča kompatibilnost na vseh brskalnikih, preprostejše pisanje kode JavaScript ter ponuja tudi razne dodatke, ki naredijo uporabniški vmesnik privlačnejši.

V diplomski nalogi smo najprej analizirali omenjene tehnologije. Analizirali smo njihovo delovanje in način pisanja spletnih aplikacij oziroma gradnjo uporabniških vmesnikov z izbrano tehnologijo. Nato smo naredili primerjavo med tehnologijami. Najprej smo primerjali tehnologiji JSF in Struts ter GWT in jQuery. Sledila je še primerjava po izbranih kriterijih med javanskimi tehnologijami. V omenjeni primerjavi nismo upoštevali tehnologije jQuery, saj ni javanska tehnologija. Za tem sledi poglavje o praktični aplikaciji, ki opisuje razvoj spletne igre s pomočjo tehnologije GWT. Zadnje poglavje v diplomski nalogi opisuje sklepne ugotovitve.

Poglavje 2

JavaServer Faces

2.1 Osnovne informacije

Tehnologija JavaServer Faces (JSF) je strežniško ogrodje komponent za grandnjo uporabniških vmesnikov. Narejeno je v Javi in je namenjeno razvoju spletnih aplikacij. Pomembne komponente tehnologije JSF so vmesnik za predstavitev uporabniških komponent in upravljanje z njenim stanjem, upravljanje dogodkov, strežniška validacija in pretvorba med različnimi tipi podatkov. Prav tako podpira internacionalizacijo. Dobro definiran programski model in knjižnica oznak olajšujeta gradnjo in vzdrževanje spletnih aplikacij [2].

2.2 Analiza tehnologije

Uporabniški vmesnik, kreiran z uporabo tehnologije JSF, teče na strežniku. Tehnologija JavaServer Pages (JSP) omogoča kreiranje dinamično generirane spletne strani, ki bazira na HTML ali XML. Stran *mojaForma.jsp* v spletnem vsebniku je pravzaprav stran JSP, ki vključuje oznake JSF.

Ena največjih prednosti tehnologije JSF je, da omogoča popolno ločitev obnašanja in predstavitve spletne aplikacije. Tehnologija JSP omogoča samo delno ločitev, saj ne moremo zahteve HTTP povezati s komponento in jo

upravljati z uporabniškimi elementi kot objekti s stanjem na strežniku. To nam omogoča JSF. Ločitev logike od predstavitve prav tako omogoča vsakemu članu razvojne skupine, da se osredotoča na svoj del razvojnega procesa in zagotavlja preprost programski model za povezovanje delov v celoto. Naslednji pomemben cilj tehnologije JSF je spodbuditev uporabniških komponent in spletnih konceptov, brez omejevanja na določeno skriptno tehnologijo ali označevalni jezik. Najpomembnejše je, da tehnologija JSF zagotavlja bogato arhitekturo za upravljanje stanja komponent, procesiranje podatkov komponent, validacijo uporabniškega vhoda in rokovanje z dogodki [3].

2.2.1 Razvoj aplikacije

V večini primerov je aplikacija JSF kot vsaka druga javanska spletna aplikacija. Tipična aplikacija JSF vključuje več delov. Ti so: množica strani JSP (čeprav to ni nujno) in množica zrn ozadja, ki so javanska zrna ter definirajo lastnosti in funkcije uporabniške komponente na strani. Vključuje tudi aplikacijsko konfiguracijsko datoteko z viri, ki definira navigacijska pravila ter konfiguracijo zrn in drugih objektov, kot so lastne komponente. Sledi namestitveni deskriptor in opcijsko množica komponent razvijalca. Razvoj aplikacije običajno zahteva naslednje naloge:

- instanca preslikovanja FacesServlet,
- kreiranje strani z uporabo uporabniških komponent in oznak jedra,
- definiranje navigacije strani v aplikacijsko konfiguracijski datoteki z viri,
- razvoj zrn ozadja,
- dodajanje deklaracij, namenjenh upravljanim zrnom, v aplikacijsko konfiguracijski datoteki z viri.

2.2.2 Kreiranje strani

Ta naloga vsebuje določanje uporabniških komponent na strani, povezovanje teh komponent z zrni in dodajanje oznak, ki registrirajo validatorje, pretvornike ali poslušalce na komponente. Če želimo uporabljati komponente JSF na strani JSP, moramo tem stranem dati dostop do dveh standardnih knjižnic oznak. Ti dve knjižnici sta *HTML component tag library* in *core tag library*. Vključimo jih z direktivo *taglib* [3].

```
1 <%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
2 <%@ taglib uri="http:.//java.sun.com/jsf/core" prefix="f" %>
```

Vse strani JSF so predstavljene z drevesno strukturo. Koren tega drevesa predstavlja oznaka *view*. Vse ostale oznake morajo biti znotraj nje. Definirana je v *core tag library*. Oznaka *form* predstavlja komponento za predstavitev obrazca, ki omogoča, da uporabnik vnese podatke in jih pošlje na strežnik, običajno s klikom na gumb. Vse oznake, ki predstavljajo uporabniške komponente in spadajo med *editable*, morajo biti gnezdene znotraj oznake *form*. Lahko ji določimo *id*, ki predstavlja povezavo s komponento na strežniku. Dodajanje label nam predstavlja oznaka *outputText*. S pomočjo atributa *value* pridobimo vrednost, največkrat z uporabo EL (*expression language*). Vnosno polje predstavlja komponenta *inputText*. Atribut *id* nam predstavlja objekt na strežniku s tem ključem. To je prav tako obvezno, če želimo s pomočjo oznake *message* prikazati sporočilo o napaki. Naslednji atribut je *label*, ki določa ime komponente pri prikazu sporočila o napaki.

```
1 <%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
2 <%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
3 <f:view>
4   <h:form id="obrazec">
5     <h:inputText id="uporaniskaSt" label="Uporabniska stevilka"
6     value="#{UporabniskaStZrno.uporaniskaSt}">
7     <f:validateLongRange
8         minimum="#{UporabniskaStZrno.minimum}"
9         maximum="#{UporabniskaStZrno.maximum}" />
10    </h:inputText>
```

```
11    </h:form>
12 </f:view>
```

V zgornjem primeru je na vnosno polje dodan validator *validateLongRange*. Ta preverja, ali so vneseni podatki znotraj omejitev. Obseg določimo z atributoma *minimum* in *maximum*. Poglejmo si še gumb za oddajo forme. To storimo z oznako *<h:commandButton id="submit"action="success"value="Oddaj"/>*. Atribut *action* določa cilj, ki pomaga navigacijskemu mehanizmu ugotoviti, katero stran mora prikazati naslednjo.

2.2.3 Definiranje navigacijskih pravil

Definiranje določa prikazno stran ob uporabnikovrm pritisku na gumb ali povezavo. Navigacija za aplikacijo je definirana v konfiguracijski datoteki aplikacije z uporabo *powerful rule-based* sistema. Tu je primer:

```
1 <navigation-rule>
2     <from-view-id>/greeting.jsp</from-view-id>
3     <navigation-case>
4         <from-outcome>success</from-outcome>
5         <to-view-id>/response.jsp</to-view-id>
6     </navigation-case>
7 </navigation-rule>
8 <navigation-rule>
9     <from-view-id>/response.jsp</from-view-id>
10    <navigation-case>
11        <from-outcome>success</from-outcome>
12        <to-view-id>/greeting.jsp</to-view-id>
13    </navigation-case>
14 </navigation-rule>
```

Pravilo določa, da se ob pritisku gumba na strani *greeting.jsp* prikaže stran *response.jsp*. Pri tem velja pogoj, da je logični izhod *success*. Ta izhod je definiran z atributom v uporabniški komponenti, ki odda obrazec.

2.2.4 Razvoj zrn

Javanska zrna so razredi, napisani v programskem jeziku Java ter se držijo predpisane konvencije. Kriteriji za zadostitev konvenciji:

- ima definiran privzeti konstruktor ali konstruktor z anotacijo *@Inject*,
- ima ustrezne *get* in *set* metode za dostop do privatnih atributov,
- vsi atributi so privatni,
- razred implementira vmesnik *java.io.Serializable*.

Poznamo dva tipa javanskih zrn, in sicer: sejna zrna in sporočilno vodena zrna. Sporočilna zrna so na strežniku v vsebniku EJB. Tipična aplikacija ima nekaj zrn za vsako stran aplikacije. Zrno navadno definira tudi lastnosti in metode, ki so povezane z uporabniškimi komponentami, uporabljenimi na strani [1, 4].

```
1 Integer uporabniskaSt = null;
2 ...
3 public void setUporabniskaSt(Integer uporabniska_st) {
4     uporabniskaSt = uporabniska_st;
5  }
6 public Integer getUporabniskaSt() {
7     return uporabniskaSt;
8 }
9 public String getResponse() {
10     if(uporabniskaSt != null &&
11          uporabniskaSt.compareTo(randomInt) == 0) {
12             return "Cestitamo! Zadel si pravo stevilo.";
13     } else {
14         return "Stevilo "+ uporabniskaSt+ " je napacno.";
15     }
16 }
```

2.2.5 Dodajanje deklaracij upravljalskim zrnom

Po razvoju zrn, uporabljenih v aplikaciji, moramo le-te konfigurirati v konfiguracijski datoteki aplikacije. To storimo zato, da implementacija JSF avtomatično kreira nove instance zrna, ko jih potrebuje.

```
1 <managed-bean>
2     <managed-bean-name>UporabniskaStZrno</managed-bean-name>
3     <managed-bean-class>
4         uganiStevilo.UporabniskaStZrno
5     </managed-bean-class>
6     <managed-bean-scope>session</managed-bean-scope>
7     <managed-property>
8         <property-name>minimum</property-name>
9         <property-class>long</property-class>
10        <value>0</value>
11    </managed-property>
12    <managed-property>
13        <property-name>maximum</property-name>
14        <property-class>long</property-class>
15        <value>10</value>
16    </managed-property>
17 </managed-bean>
```

Zgornja deklaracija pomeni, da se vrednost lastnosti *minimum* zrna *UporabniskaStZrno* inicializira na 0 in vrednost lastnosti *maximum* na vrednost 10 ter sta dodani v doseg seje. Protokol *HTTP* je brez stanja. Zato strežnik ne more identificirati zahteve, ali je ta prišla od istega ali drugega odjemalca. Da si strežnik zapomni odjemalca in njegove aktivnosti, imamo na voljo sejo. Seja nam omogoča povezovanje informacij z individualnimi obiskovalci [3].

2.2.6 Instanca FacesServlet

Vse aplikacije JSF morajo vključevati instanco *FacesServlet* v svojem namestitvenem deskriptorju. Instanca *FacesServlet* sprejema prihajajoče zahteve ter jih posreduje življenjskemu ciklu in inicializira vire. Primer namestitvenega deskriptorja:

```
1 <servlet>
2     <display-name>FacesServlet</display-name>
3     <servlet-name>FacesServlet</servlet-name>
4     <servlet-class>javax.faces.webapp.FacesServlet
5     </servlet-class>
6     <load-on-startup>1</load-on-startup>
7 </servlet>
8 <servlet-mapping>
9     <servlet-name>FacesServlet</servlet-name>
10    <url-pattern>/guess/*</url-pattern>
11 </servlet-mapping>
```

Preslikovanje *FacesServleta* uporablja predpono, ki nam prikazuje, kako identificirati stran JSP, ki vsebuje komponente JSF. Zaradi tega mora URL do prve strani aplikacije vsebovati preslikovanje. Primer take povezave *<a href="guess/greeting.jsp">* [3].

Poglavje 3

Apache Struts

3.1 Osnovne informacije

Apache Struts je spletno ogrodje, ki je brezplačno in namenjeno razvoju javanskih spletnih aplikacij. Spletne aplikacije, ki bazirajo na JSP, včasih pomešajo podatkovno kodo z oblikovanjem in poslovno logiko. To se je v praksi izkazalo kot nerodno, ker razvijalci in oblikovalci posegajo v iste datoteke in s tem potrebujejo tudi razvijalska oziroma oblikovalska znanja. Struts je ogrodje, namenjeno razvoju spletnih aplikacij po modelu *Model-View-Controler* (MVC). Več o modelu MVC je v poglavju o primerjavi. Ogrodje zagotavlja tri ključne komponenete. Te so: rokovalec zahteve, ki zagotavlja razvijalcu povezavo s standardnim URI, rokovalec odgovora, ki prenaša nadzor do drugih virov, ki končajo odziv, in knjižnica oznak, ki pomaga razvijalcem razviti interaktivno, na obrazcih bazirajočo aplikacijo s strežniškimi stranmi. Arhitektura Struts omogoča uporabo s konvencionalnimi aplikacijami *Representational State Transfer* (REST) in s tehnologijama, kot sta *Simple Object Access Protocol* (SOAP) in *Asynchronous JavaScript and XML* (AJAX) [5].

3.2 Analiza tehnologije

Za osnovno uporabo Struts 2 potrebujemo zip datoteko *Essental Dependencies Only*. Za delovanje Struts 2 zahteva *Servlet API 2.4*, *JSP 2.0* in *Java 5* ali novejšo verzijo.

Ogrodje uporablja akcije za obdelovanje obrazcev HTML in ostalih zahtev. Glede na preslikavo, naloženo iz *struts.xml*, akcijski razred vrača rezultat, kot je *SUCCESS*, *ERROR* ali *INPUT*. Dobljen rezultat izbere ustrezno stran, akcijo ali spletni vir. Ker strežniške strani najpogosteje vključujejo dinamične podatke, nam ogrodje zagotavlja množico oznak, ki jih lahko uporabljamo skupaj z oznakami HTML.

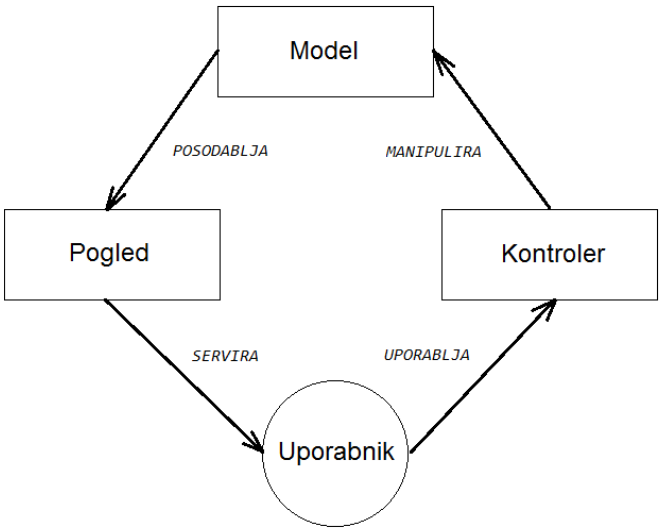
3.2.1 Razvoj aplikacije

Preprosto spletno aplikacijo s pozdravnim sporočilom kreiramo na naslednji način. Treba je ustvariti sledeče strani:

- razred za hranjenje pozdravnega sporočila (model),
- strežniško stran, ki predstavlja sporočilo (pogled),
- akcijski razred za kontrolo interakcije med uporabnikom, modelom in pogledom (kontroler),
- preslikovanje za povezavo akcijski razred in pogled.

S postavitvijo teh komponent sledimo vzorcu MVC, ki smo ga omenili na začetku poglavja. Vzorec MVC poskrbi za preprosto upravljanje aplikacije, ko ta postane kompleksna.

Model MVC (Slika 3.1) je bil formuliran leta 1970 v podjetju *Xerox PARC* s strani *Trygvea Reenskauga*. Namen modela je ločitev predstavitve informacij od uporabniške interakcije. Razdeljen je na tri osnovne dele, model, pogled in kontroler. Model predstavlja podatke aplikacije in poslovna pravila. Kontroler sprejme vhodne podatke in jih pretvori v ukaze za model ali pogled. Pogled je lahko kakršna koli predstavitev podatkov [5, 8].



Slika 3.1: Model MVC

Spodaj je prikazan razred, ki nam predstavlja model. V razredu so javne *set* in *get* metode, ki omogočajo dostop do privatnega atributa *message*. Ogrodje Struts 2 zahteva, da objekti, ki jih uporabljamo, sledijo konvenciji javanskih zrn. Konvencija je opisana v prejšnjem poglavju.

```
1  package org.apache.struts.helloworld.model;
2
3  public class MessageStore {
4
5     private String message;
6
7     public MessageStore() {
8        setMessage("Pozdravljen");
9     }
10
11    public String getMessage() {
12       return message;
13    }
14
15    public void setMessage(String message) {
```

```
16        this.message = message;
17    }
18 }
```

Sledi implementacija akcijskega razreda, ki nam predstavlja kontroler. Razred se odzove na uporabniško akcijo. Izvede se ena ali več metod akcijskega razreda, katera vrne rezultat tipa *String*.

```
1  package org.apache.struts.helloworld.action;
2
3  import org.apache.struts.helloworld.model.MessageStore;
4  import com.opensymphony.xwork2.ActionSupport;
5
6  public class HelloWorldAction extends ActionSupport {
7
8     private static final long serialVersionUID = 1L;
9     private MessageStore messageStore;
10
11    public String execute() throws Exception {
12       messageStore = new MessageStore();
13       return SUCCESS;
14    }
15
16    public MessageStore getMessageStore() {
17       return messageStore;
18    }
19
20    public void setMessageStore(MessageStore messageStore) {
21       this.messageStore = messageStore;
22    }
23 }
```

Za odgovor na uporabniško akcijo ogrodje Struts 2 ustvari objekt razreda *HelloWorldAction* in pokliče metodo *execute()*. V zgornjem primeru ustvari objekt razreda *MessageStore* in vrne konstanto *SUCCESS*.

Sledi tretji korak kreiranja pogleda. Potrebujemo strežniško stran, ki nam predstavlja sporočilo, ki je shranjeno v razredu *MessageStore* [5].

```
1 <%@ page language="java" contentType="text/html; charset=ISO
      -8859-1"
2     pageEncoding="ISO-8859-1"%>
3 <%@ taglib prefix="s" uri="/struts-tags" %>
4 <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "
      http://www.w3.org/TR/html4/loose.dtd">
5 <html>
6 <head>
7 <meta http-equiv="Content-Type" content="text/html; charset=ISO
      -8859-1">
8 <title>Hello World!</title>
9 </head>
10 <body>
11     <h2><s:property value="messageStore.message" /></h2>
12 </body>
13 </html>
```

Direktiva *taglib* pove vsebniku servletov, da stran uporablja oznake ogrodja Struts 2. Označujejo se s predpono s. Oznaka *s:property* prikazuje vrnjeno vrednost metode *getMessageStore()* kontrolnega razreda *HelloWorldAction*. Ta metoda vrača objekt *MessageStore*. Z dodajanjem pripone *.message* povemo ogrodju Struts 2, da pokliče metodo *getMessage()* nad objektom *MessageStore*.

Potrebna je še konfiguracija v datoteki *struts.xml*. Potrebujemo preslikavo, da povežemo URL s kontrolnim razredom *HelloWorldAction* in pogledom *HelloWorld.jsp*. To pove ogrodju Struts 2, kateri razred se bo odzval na uporabniško zahtevo, katera metoda tega razreda se bo izvedla in kateri pogled se bo vrnil glede na rezultat metode [5].

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE struts PUBLIC
3     "-//Apache Software Foundation//DTD Struts Configuration
          2.0//EN"
4     "http://struts.apache.org/dtds/struts-2.0.dtd">
5
6 <struts>
7   <constant name="struts.devMode" value="true" />
```

```
8   <package name="basicstruts2" extends="struts-default">
9
10  <action name="index">
11    <result>/index.jsp</result>
12  </action>
13
14  <action name="hello" class="org.apache.struts.helloworld.
        action.HelloWorldAction" method="execute">
15    <result name="success">/HelloWorld.jsp</result>
16  </action>
17 </package>
18 </struts>
```

3.2.2 Konfiguracija XML

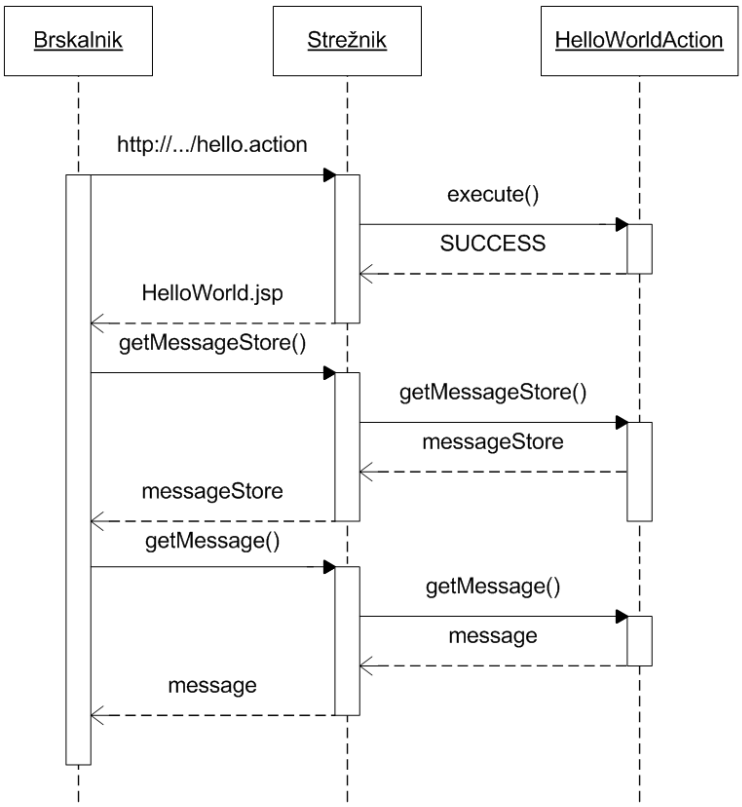
Za določanje odnosa med naslovi URL, Javanskimi razredi in prikaznimi stranmi Struts 2 lahko uporablja konfiguracijsko datoteko XML ali anotacije (lahko tudi oboje). Pri konfliktnih situacijah se uporabi vsebina iz datoteke XML. Spodnja konfiguracija pove ogrodju, če se URL konča z *index.action*, naj strežnik odjemalca preusmeri na *index.jsp*. Ko kliknemo na povezavo ali oddamo obrazec v spletni aplikaciji, se vhod ne pošlje na drugo strežniško stran. Pošlje se do javanskega razreda. Razredi se imenujejo *Actions*. Ko se sproži akcija, rezultat izbere vire za generiranje odgovora. Vir je največkrat strežniška stran, lahko pa je tudi datoteka pdf, razpredelnica excel ali javanskimi applet [5].

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE struts PUBLIC
3     "-//Apache Software Foundation//DTD Struts Configuration
          2.0//EN"
4     "http://struts.apache.org/dtds/struts-2.0.dtd">
5 <struts>
6   <constant name="struts.devMode" value="true" />
7   <package name="basicstruts2" extends="struts-default">
8
9       <action name="index">
```

```
10            <result>/index.jsp</result>
11        </action>
12
13    </package>
14 </struts>
```

3.2.3 Delovanje

Delovanje je opisano in prikazano z diagramom zaporedja na sliki 3.2. Brskalnik pošlje spletnemu strežniku zahtevo *http://.../hello.action*. Vsebnik od spletnega srežnika prejme zahtevo za vir *hello.action*. Glede na nastavitve v *web.xml* vsebnik poišče vse zahteve, ki se nanašajo na *org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter*, vključno z zahtevami $*.action$. *StrutsPrepareAndExecuteFilter* predstavlja vstopno točko v ogrodje. Ogrodje nato išče akcijsko preslikovanje z imenom *hello*. Ogrodje najde preslikovanje na razred *HelloWorldAction*. Ustvari se instanca in pokliče se metoda *execute()*. Metoda ustvari objekt *MessageStore* in vrne *SUCCESS*. Ogrodje preveri preslikovanje. S tem ugotovi, kaj se vrne, če je vrnjen niz *SUCCESS*. Nato ogrodje pove vsebniku, naj generira odgovor z virom *HelloWorld.jsp*. Ko je stran *HelloWorld.jsp* obdelana, oznaka *<s:property value="messageStore.message"/>* pokliče metodo *getMessageStore()* akcije *HelloWorld*. Nato pokliče metodo *getMessage()* objekta *MessageStore*, ki ga vrne *getMessageStore()*. Do odjemalca se pošlje dokument HTML [5].



Slika 3.2: Diagram zaporedja

Poglavje 4

Google Web Toolkit

4.1 Osnovne informacije

Google Web Toolkit (GWT) je razvojno ogrodje za gradnjo in optimizacijo kompleksnih aplikacij, ki jih izvaja brskalnik. GWT SDK zagotavlja množico Java APIjev in gradnikov. To omogoča pisanje aplikacij AJAX v Javi, ki se kasneje prevedejo v optimizirano kodo JavaScript, ki deluje na vseh brskalnikih, vključno z mobilnimi brskalniki za Andorid in iOS.

4.2 Analiza tehnologije

GWT ne določa, kakšna bo pozicija strani HTML. Aplikacija lahko zavzame celotno okno brskalnika, največkrat pa je vključena v že obstoječo stran. GWT poskrbi za kompatibilnost med različnimi brskalniki. Pri programiranju obsežnejših aplikacij je še vedno potrebno testiranje na brskalnikih. Kljub temu je to velika prednost za ogrodje GWT. Ogrodje zagotavlja množico gradnikov. Gradniki imajo privzet stil, katerega po želji spreminjamo s CSS [6].

4.2.1 Gradniki

Kot smo omenili, GWT zagotavlja množico gradnikov. Podobni so gradnikom HTML. Za instanco gradnika *Button* se skriva oznaka HTML <*button*>.

Od tod sledi, da gumbe in druge komponente oblikujeta brskalnik in operacijski sistem. Prednost takega pristopa je, da so komponente hitre, dostopne in najbolj poznane uporabnikom. Lahko so tudi oblikovane s CSS.

Za vnosno polje GWT zagotavlja več gradnikov:

- TextBox - ena vrstica vnosnega polja,
- PassWordTextBox - vnosno polje, ki vizualno maskira vhod,
- TextArea - večvrstično vnosno polje,
- SuggestBox - vnosno polje, ki prikazuje vnaprej definirano množico vnosov.

Naslednji gradnik je label. Label se ne preslika v oznako HTML *<label>*, ampak v oznako * *, ki vsebuje besedilo. To besedilo ni interpretirano kot HTML. Pri tem moramo upoštevati, da je * * bločni in ne vrstični element HTML.

Uporabnik lahko kreira svoje gradnike. Zgoraj predstavljeni gradniki so le osnovni. GWT ponuja tudi bolj napredne. Gradnik *FlexTable* kreira celice na zahtevo. S tem omogoča preprosto dodajanje, brisanje in spreminjanje celic tabele [6].

4.2.2 Panele

Gradnike dodajamo na panele in s tem določimo lego uporabniških elementov. Panele so lahko gnezdene ena v drugo. GWT ponuja več panel. Najpogosteje uporabljeni sta horizontalna in vertikalna panela. Najpomembnejši za razumevanje je tako imenovana korenska panela. Ta ni viden v uporabniškem vmesniku. Predstavlja vsebnik za dinamične elemente naše aplikacije. Vedno je na vrhu katerekoli uporabniške hierarhije ogrodja GWT. Obstajata dva načina uporabe korenske panele. Lahko ovija celotno telo strani ali samo specifične elemente, vključene v telo. Panela deluje kot ovoj okrog elementa odjemalčeve strani HTML. Privzeto ovija oznako HTML *<body>*.

```
RootPanel.get()       // Privzeto. Ovija HTML element <body>.
RootPanel.get("panela") // Ovija vse HTML elmente z id panela.
```

Stran lahko vsebuje več korenskih panel. Vsak je lahko samostojno implementiran, ovit v svojo korensko panelo, neodvisno od drugih panel [6].

4.2.3 Implementacija panel in gradnikov

Če želimo, da se aplikacija prikaže takoj, jo implementiramo v metodi *onModuleLoad()*.

```
package com.google.gwt.sample.primer.client;

import com.google.gwt.core.client.EntryPoint;
import com.google.gwt.user.client.ui.Button;
import com.google.gwt.user.client.ui.FlexTable;
import com.google.gwt.user.client.ui.HorizontalPanel;
import com.google.gwt.user.client.ui.Label;
import com.google.gwt.user.client.ui.TextBox;
import com.google.gwt.user.client.ui.VerticalPanel;
import com.google.gwt.user.client.ui.RootPanel;

public class Primer implements EntryPoint {

  private VerticalPanel panela1 = new VerticalPanel();
  private FlexTable tabela = new FlexTable();
  private HorizontalPanel panela2 = new HorizontalPanel();
  private TextBox vnosnoPolje = new TextBox();
  private Button gumb = new Button("Dodaj");
  private Label labela = new Label();

  /**
   * Vstopna metoda.
   */
  public void onModuleLoad() {
   tabela.setText(0, 0, "Izdelek");
    tabela.setText(0, 1, "Cena");
```

```
27
28      panela2.add(vnosnoPolje);
29       panela2.add(gumb);
30
31       panela1.add(tabela);
32       panela1.add(panela2);
33       panela1.add(labela);
34
35      RootPanel.get().add(panela1);
36    }
37 }
```

Koda generira aplikacijo GWT. Vmesnik *EntryPoint* omogoča razredu *Primer*, da se vede kot vstopna točka v modul. Vstopna metoda *onModuleLoad()* se pokliče avtomatsko, ko se nalaga razred, ki implementira vmesnik *EntryPoint*. Aplikacija vsebuje tabelo s stolpcema "Izdelek"in "Cena". Tabela je dodana na vrh strani. Pod njo je "panela2", ki vsebuje vnosno polje in gumb. Na dnu se nahaja še labela z imenom labela [6].

4.2.4 Upravljanje z dogodki

Kot večina ogrodij, tudi ogrodje GWT bazira na dogodkih. To pomeni, da se koda odzove, ko zazna nek dogodek. Najpogosteje je ta dogodek sproži uporabnik, kateri uporablja miško in tipkovnico za komunikacijo z uporabniškim vmesnikom.

Dogodki v GWT uporabljajo vmesniški model rokovalca dogodka, ki je podoben večini ostalim ogrodjem. Za primeren gradnik podamo ustrezen vmesnik rokovalca dogodka, s čimer opišemo dogodek. Vmesnik definira eno ali več metod, katere lahko gradnik pokliče in oznani dogodek.

Klik miške realiziramo tako, da podamo objekt, ki implementira vmesnik *ClickHandler*. Vmesnik vsebuje metodo *onClick()* [6].

```
1     gumb.addClickHandler(new ClickHandler() {
2       public void onClick(ClickEvent event) {
3         // koda ki se izvede ob kliku na gumb
```

```
4       }
5     });
```

Zgornja koda mora biti v metodi *onModuleLoad()*. Pristop je podoben tudi pri dogodkih tipkovnice.

```
1     vnosnoPolje.addKeyPressHandler(new KeyPressHandler() {
2       public void onKeyPress(KeyPressEvent event) {
3         if (event.getCharCode() == KeyCodes.KEY_ENTER) {
4           // koda ki se izvede ob pritisku tipke enter
5         }
6       }
7     });
```

Na odjemalčevi strani se bo aplikacija odzivala brez pošiljanja kakršnih koli zahtev na strežnik ali ponovnega nalaganja strani.

4.2.5 Validacija

Validacijo uporabljamo za preverjanje pravilnosti vnosa. Po pretvorbi uporabniškega vhoda v standardno obliko uporabimo regularni izraz za preverbo. Regularni izraz ima enak pomen v Javi in JavaScriptu [6].

```
1     final String vnos = vnosnoPolje.getText().toUpperCase().trim
          ();
2     if (!vnos.matches("^[0-9A-Z\\.]{1,10}$")) {
3       Window.alert("'" + vnos + "' ni veljaven vnos.");
4       vnosnoPolje.selectAll();
5       return;
6     }
```

Koda vsebino vnosnega polja pretvori in shrani v niz vnos. Nato preveri, če niz vsebuje števila od 0 do 9, velike črke od A do Z in piko. Preveri tudi, če ima niz dolžino od 1 do 10 znakov. V primeru izpolnjenega pogoja je vnos veljaven. V nasprotnem primeru se uporabniku prikaže opozorilno okno in vrednost v vnosnem polju se označi [6].

4.2.6 Samodejno spreminjanje podatkov

GWT omogoča preprosto posodabljanje aplikacijskega vsebnika. Z uporabo GWT razreda, imenovanega *Timer*, bomo spreminjali vrednosti v tabeli.

```
1     Timer casovnik = new Timer() {
2       @Override
3       public void run() {
4         posodobiTabelo();
5       }
6     };
7     casovnik.scheduleRepeating(INTERVAL_OSVEZEVANJA);
8
9    ...
10
11    private void posodobiTabelo() {
12      tabela.setText(1,1,Random.nextDouble() * MAX_CENA)
13    }
```

Poglavje 5

jQuery

5.1 Osnovne informacije

JQuery je hitra in jedrnata knjižnica JavaScript, ki poenostavlja pregled dokumenta HTML, rokovanje z dogodki, animacijo, omogoča interakcijo AJAX in poskrbi za kompatibilnost kode JavaScript med različnimi brskalniki. Omenjene prednosti omogočajo hiter razvoj spletnih strani. Knjižnjica je oblikovana tako, da spremeni način pisanja kode JavaScript. Vsak od uporabnikov bi se strinjal z njihovim sloganom, ki se glasi *write less, do more*. Vse, kar počnemo z uporabo jQueryja je, da beremo ali manipuliramo z modelom objektnega elementa (DOM). DOM je aplikacijski programski vmesnik za veljvne HTML in dobro oblikovane XML dokumente. Definira logično strukturo dokumenta, in način dostopa in manipulacije z dokumentom. Logična struktura dokumenta je predstavljena v obliki drevesa. Z DOM lahko razvijalci gradijo dokumente, se sprehajajo po njegovi strukturi, jo spreminjajo, brišejo in dodajajo elemente ter njihovo vsebino. [7].

5.2 Analiza tehnologije

Za pisanje kode potrebujemo urejevalnik besedil in brskalnik. Našo kodo vključimo v dokument HTML, treba je dodati oznako *script*. Atribut *src*

vsebuje pot do datoteke s kodo jQuery. Za primer vzemimo naslednjo preprosto spletno stran. Če je datoteka s kodo jQuery v enakem imeniku kot datoteka HTML, potem v atributu *src* navedemo samo ime datoteke [7].

```
<html>
  <head>
    <meta charset="utf-8">
    <title>Primer</title>
  </head>
  <body>
    <a href="http://jquery.com/">jQuery</a>
    <script src="jquery.js"></script>
    <script>

    </script>
  </body>
</html>
```

Knjižnica jQuery ima preprost stavek. Ta preverja dokument in čaka dokler ni pripravljen na izvajanje. Dogodek se imenuje *ready*. Znotraj dogodka *ready* lahko dodajamo rokovalce. Spodnji primer kode prikazuje dodajanje rokovalnika ob kliku na povezavo. Ob proženju dogodka se v opozorilnem oknu izpiše vsebina "Hvala za obisk!" [7].

```
$(document).ready(function(){
    $("a").click(function(event){
      alert("Hvala za obisk!");
    });
 });
```

Za klik in večino ostalih dogodkov lahko onemogočimo privzeto obnašanje. To naredimo s klicem *event.preventDefault()* v obravnavi dogodka. [7]

5.2.1 Dodajanje in brisanje razreda HTML

Za potrebe prikaza moramo dodati nekatere oblikovalne informacije v glavi našega dokumenta.

```
<style>
    a.test { font-weight: bold; }
</style>
```

Ko imamo definiran razred, lahko z uporabo jQueryja izvedemo klic za dodajanje razreda *$("a").addClass("test");*. Za odstranjevanje razreda jQuery ponuja *$("a").removeClass("test");*. Ob dodajanju razredov je potrebna pozornost, kajti HTML dovoljuje dodajanje več razredov enemu elementu [7].

5.2.2 Posebni učinki

JQuery zagotavlja nekaj priročnih učinkov, ki naredijo spletno stran izstopajočo. Ti učinki so *show(), hide(), toggle(), fadeIn(), fadeOut()* itd. Našteti učinki so le osnovni, saj ponuja še mnoge druge. Naslednja koda prikazuje implementacijo učinka, ki ob kliku skrije povezavo [7].

```
$("a").click(function(event){
    event.preventDefault();
    $(this).hide("slow");
});
```

5.2.3 Povratni klici in funkcije

Povratni klic je funkcija, ki je podana kot argument drugi funkciji in se izvrši, ko je starševska funkcija izvedena. Posebna lastnost funkcij s povratnim klicem je, da se izvedejo za staršem in pred izvajanjem povratnega klica.

Povratni klic brez argumentov podamo na naslednji način *$.get('mojastran.html', primerCallBack);*. Drugi argument je ime funkcije, ki ni niz. Ime funkcije v argumentu nima oklepajev. Funkcije v Javascriptu so tako imenovani prvorazredni člani. To pomeni, da so lahko podane kot spremenljive reference in so izvedene kasneje.

Če želimo klicati funkcijo povratnega klica z argumenti, bi najprej pomisli na naslednji način *$.get('mojastran.html', primerCallBack('arg1', 'arg2'));*.

To ne bi delovalo, ker se najprej kliče *primerCallBack('arg1', 'arg2')*, nato se vrnjena vrednost poda kot drugi argument *$.get()*. Težava v zgornjem primeru je, da je funkcija ovrednotena, preden je podana kot funkcija. Javascript z razširitvijo jQuery pričakuje kazalec na funkcijo. To težavo rešimo s kreiranjem anonimne funkcije, ki je registrirana kot funkcija povratnega klica. Anonimne funkcije opravijo samo eno stvar. Pokličejo *primerCallBack* z ustrezno vrednostjo parametrov [7].

```
1 $.get('mojastran.html', function(){
2   primerCallBack('arg1', 'arg2');
3 });
```

5.2.4 Selektorji in dogodki

JQuery zagotavlja dva pristopa za izbiro elementov. Prvi pristop zahteva kombinacijo selektorjev CSS in XPath, kateri so podani kot nizi v konstruktorju jQuery *($("div > ul a"))*. Drugi pristop uporablja nekaj metod objekta jQuery. Dobra lastnost obeh pristopov je, da jih lahko kombiniramo. Najbolj osnovna operacija je izbira seznama. Če ima seznam id z vrednostjo "seznam", ga v klasičnem JavaScriptu izberemo z uporabo *document.getElementById("seznam")*. V jQueryju to naredimo na sledeč način:

```
1 $(document).ready(function() {
2     $("#seznam").addClass("ozadje");
3  });
```

V zgornji kodi nismo le izbrali elementa, ampak smo mu dodali tudi razred "ozadje". Za izbiro otrok tega seznama uporabimo selektor *$("#seznam > li")*. Izbiro zadnjega elementa tega seznama dosežemo s pomočjo *$("#seznam > li:last")*. To je le drobec selektorjev, katere ponuja jQuery. Za primerjavo in analizo nam to zadošča. Več o selektorjih lahko najdemo na uradni spletni strani.

V jQueryju obstaja ekvivalent dogodkov, definiranih v JavaScriptu, kot sta onclick in onsubmit. Dodani so tudi nekateri drugi dogodki, kot sta ready ali hover.

S selektorji in dogodki lahko realiziramo marsikaj. Za iskanje nadaljnjih potomcev že izbranih elementov lahko uporabimo *find()*. Metoda *each()* omogoča iteriranje čez vsak element in omogoča nadaljnje obdelovanje. Razvijalci jQueryja se zavzemajo za čim krajšo kodo. To lahko sklepamo že iz njihovega slogana, ki smo ga omenili na začetku poglavja. S tem naj bi omogočili lažje branje in razumevanje kode. Okrajšavo za *$(document).ready(callback)* prikazuje naslednja koda:

```
1 $(function() {
2    // koda, ki bo izvedena, ko bo DOM pripravljen
3  });
```

5.2.5 AJAX in jQuery

AJAX je skupina medsebojno povezanih tehnik spletnega razvoja. Uporablja se za kreiranje asinhronih aplikacij. Z AJAXom lahko spletne aplikacije pošijajo in prejemajo podatke od strežnika asinhrono (v ozadju), brez osveževanja spletne strani. Za dinamično spreminjanje vsebine jQuery ponuja nekaj metod. Med njimi sta najbolj pogosto uporabljeni *$.post()* in *$.get()*, ki sta okrajšavi *$.ajax()*.

```
1 $.post("test.php", { ime: "Janez", cas: "2" },
2    function(data) {
3      alert("Prejeti podatki: " + podatki);
4    });
```

S klicem iz zgornjega primera koda pošljemo zahtevo POST do strežnika, natančneje datoteki test.php. Nato strežnik na podlagi podanih podatkov tvori odgovor, ki se pošlje do odjemalca. V zgornjem primeru se podatki prikažejo uporabniku preko opozorilnega okna [7].

5.3 Razvoj tehnologije

Prva stabilna verzija jQueryja je bila na voljo 26. avgusta 2006. Konec leta 2007 so predstavili knjižnico uporabniških vmesnikov za jQuery, imenovano jQueryUI. Leta 2010 so začeli z razvojem mobilnega ogrodja za pametne telefone in tablične računalnike. Trenutno je na voljo verzija 1.1.1 tega ogrodja [7].

Poglavje 6

Primerjava tehnologij

Primerjavo omenjenih tehnologij lahko razdelimo v dve skupini. Prvo skupino sestavljata tehnologiji JSF in Struts. Drugo pa ogrodje GWT in knjižnica JavaScript, imenovana jQuery. Tehnologiji iz prve skupine sta si blizu po načinu gradnje spletnih aplikaciji. Obe sledita arhitekturnemu modelu MVC. GWT in jQuery si nista tako blizu, toda vseeno imata skupne lastnosti. Tehnologiji sta namenjeni predvsem gradnji dinamičnih aplikacij. Hkrati se koda obeh tehnologiji lahko izvaja na strani odjemalca.

6.1 JSF in Apache Struts

Kot smo omenili na začetku tega poglavja, obe omenjeni tehnologiji sledita modelu MVC.

Prednosti JSF:

- hiter in preprost razvoj,
- veliko knjižnic s komponentami.

Prednosti Struts 2:

- preprosta arhitektura - preprosto za razširitev,
- preprosta spreminjanje knjižnice oznak,

- navigacija temelji na kontrolerjih ali na straneh.

Temeljna razlika med tehnologijama je v osnovni paradigmi obeh. JSF je komponentno ogrodje, medtem ko je Struts 2 akcijsko ogrodje. V komponentnem ogrodju so elementi na strani prvotno razviti kot komponente. Te imajo dogodke in koda je napisana za delo s temi dogodki. Akcijska ogrodja nam dajejo možnost, da povežemo URL naslove z aktivnostmi in kodo v ozadju. Ogrodje je močno povezano s ciklom zahteve HTTP in njenim formatom. Za spletne strani, ki se osredotočajo na zagotavljanje vsebine za uporabnika, je akcijsko ogrodje boljše. Komponentna ogrodja so boljša pri aplikacijah z veliko obrazci in kontrolami. Tu je potek bolj kontroliran. Ogrodje prikrije zahteve HTTP in uporabi višji nivo abstrakcije [2, 5, 8].

6.2 GWT in jQuery

Največja prednost ogrodja GWT je, da aplikacije pišemo v Javi. Ob prevajanju se javanska koda prevede v optimizirano JavaScript kodo. Prav tako je preprost za učenje in hiter razvoj z uporabo strandardnih javanskih orodij. Slabosti, ki jih navajajo, je počasno prevajanje in težavno testiranje. Očitajo tudi, da je bolj podoben knjižnici oznak JSP kot pa spletnemu ogrodju. JQuery je za razliko od GWT knjižnica in ne ogrodje. Poskuša poenostaviti kodo JavaScript. Njegova največja prednost pred drugimi knjižnicami JavaScript je preprostost za uporabo. Poleg preproste sintakse je koda jQuery tudi kratka. Drugače povedano, neko stvar naredimo z veliko manj stavki kot v drugih knjižnicah JavaScript. Ponuja nam tudi množico funkcij in primerjav v primerjavi z ostalimi knjižnicami. Uporabnik ima na voljo tudi veliko dodatkov, ki močno skrajšajo čas razvoja. Nudi tudi odlično dokumentacijo. Omogoča tudi preprosto uporabo AJAXa. Slabost jQueryja je, da nam kljub vsem dodatkom in prednostim v nekaterih primerih ne preostane drugega, kot pisanje kode JavaScript. Druga težava je, da za svoje delovanje potrebuje datoteko JavaScript [6, 7].

6.3 Primerjava z naborom kriterijev

Kriteriji za primerjavo tehnologij se skozi čas močno spreminjajo. Trenutno so najpomembnejši naslednji kriteriji: produktivnost razvijalca, vtis razvijalca, krivulja učenja, zdravje projekta, razpoložljivost razvijalca, gibanje delovnih mest, predloge(templates), komponente, AJAX, dodatki, skalabilnost, testiranje, i18n in l10n, validacija, večjezikovna podpora, kakovost dokumentacije, izdane knjige, podpora REST, mobilna podpora, stopnja tveganja. Vsakega od naštetih kriterijev predstavimo s tremi vrednostmi: 0, 0,5 in 1. V tej lestvici nam 1 predstavlja dobro, 0 slabo. Vrednosti predstavimo s tabelo 6.1 [9].

Produktivnost razvijalca ocenjuje, kako hitro razvijalec razvije aplikacijo. Na to točko vpliva tudi testiranje, saj je pri nekaterih točkah potrebno tudi prevajanje kode, kar pa vpliva tudi na produktivnost razvijalca. Vtis razvijalca je pomemben, saj so navadno razvijalci ponosni na ogrodje in o tem govorijo tudi zunaj podjetja. Vsekakor se pridobi z razvojem praktične aplikacije in ne z branjem teoretičnih zmogljivosti. Krivulja učenja se pokaže po več kot tednu dni učenja. Takrat ogrodje spoznamo in se že soočamo z njegovimi težavami. Zdravje projekta je pomembno, ker se ne želimo učiti ogrodji, ki niso zanesljiva. Razpoložljivost razvijalca je najbolj pomembna za ljudi, ki zaposlujejo. V primeru, da je tehnologija nova, ne moremo zaposliti razvijalca z desetimi leti delovnih izkušenj. Gibanje delovnih mest je pomembno, za razvijalce brez redne zaposlitve. Govorimo o povpraševanju na trgu za posamezno tehnologijo. Predloge omogočajo lažje delo razvijalca. Če ogrodje omogoča komponente, je dobro, ker jih preprosto samo uporabimo. Pri kriteriju AJAX govorimo o sami podpori tej tehnologiji in ali je vključena v njo. Če je AJAX vključen v tehnologijo, potem je posodabljanje izdaje lažje. Dodatki so zaželeni, saj razvijalcem omogočajo preprosto vključitev že rešenih problemov, drugih aplikacij in podobno. Skalabilnost pove, kako se aplikacija odziva pri velikem številu zahtev na strežnik, velikemu številu uporabnikov in veliko dostopov do podatkov. Kriterij testiranje poskuša zaobjeti, kako preprosto je testiranje aplikacije z izbranim ogrodjem. Internacionalizacija

in lokalizacija se označujeta z i18n in l10n. I18n je okrajšava za internacionalizacijo. L in n pomenita prvo in zadno črko angleške besede *internationalization*. Številka med črkama pa predstavlja število črk med l in n. Uporabo so skovali pri podjetju DEC med leti 1970 in 1980. Enako je z oznako l10n ,le da izhaja iz angleške besede *localization*. Validacija je pomembna kjer je oblika podatkov striktno določena in pomembna. Kakovostna dokumentacija je pomembna za vzdrževanje, razumevanje kode, pregled, iskanje napak in pri spoznavajo s tehnologijo. Izdane knjige pomenijo, kako je tehnologija razširjena. Podpora REST je pomembna tako s strani strežnika kot odjemalca. Kriterij mobilna podpora ocenjuje možnost dostopa in upravljanja z mobilnimi napravami. Stopnja tveganja je bolj pomembna za podjetja kot za razvijalce [9].

Iz tabele 6.1 je razvidno, da je tehnologija GWT najbolje ocenjena. Slabost se pokaže pri večjezikovni podpori, množici komponent, predlog in testiranju. Za GWT sta preostali tehnologiji s tabele. Pozorni moramo biti na razliko, saj sta JSF 2.0 in Struts 2 veliko bolj enakovredni, kajti odstopata samo za 1 točko, medtem ko je GWT boljši od JSF 2.0 za 3,5 točke. JSF 2.0 izgubi prednost v primerjavi s Struts 2 pri vtisu razvijalcev in podpori REST. Vtis je najverjetneje slabši zaradi verzije 1.0. Takrat je ob prihodu drugih tehnologij veliko podjetij opustilo JSF. Trenutno so v ospredju javanske tehnologije, kot so Spring MVC, Grails, GWT, Wicket in Struts 2. To so tehnologije, ki imajo oceno nad 14,5 točke.

Ko se podjetja sprašujejo, katero tehnologijo izbrati, morajo najprej sestaviti seznam prioritet, ki so pomembne za podjetje oziroma aplikacijo. Sledi izbira treh od štirih ogrodji. Z vsakim ogrodjem podjetje en teden razvija enako aplikacijo. Nato sledi ugotavljanje, katero mesto zavzame ogrodje glede na izbrane zahteve [9].

Obstaja še veliko podobnih tehnologij in vsaka ima neko množico uporabnikov. Mnenja, katera tehnologija je boljša, so si nasprotujoča. Če na vse tehnologije pogledamo objektivno, pridemo do ugotovitve, da najboljše tehnologije ni. Vsak razvijalec oziroma skupina ima svoj okus. Od tod

kriterij/tehnologija	JSF 2.0	Struts 2	GWT
produktivnost razvijalca	0,5	0,5	1
vtis razvijalca	0	0,5	1
krivulja učenja	0,5	1	1
zdravje projekta	1	0,5	1
razpoložljivost razvijalca	1	0,5	1
gibanje delovnih mest	1	1	1
predloge	0,5	1	0,5
komponente	1	0	0,5
AJAX	0,5	0,5	1
dodatki	1	0,5	1
skalabilnost	0,5	1	1
testiranje	0,5	1	0,5
i18n in l10n	0,5	1	1
validacija	0,5	1	1
večjezikovna podpora	1	0,5	0
kakovost dokumentacije	0,5	0,5	1
izdane knjige	1	1	1
podpora REST	0	0,5	0,5
mobilna podpora	1	1	1
stopnja tveganja	1	1	1
skupaj	13,5	14,5	17

Tabela 6.1:

lahko sklepamo, da imamo na voljo veliko dobrih tehnologij, moramo se samo odločiti, katera nam najbolj ustreza. Stvar je podobna pri izbiri avtomobila. Pri neomejenih sredstvih imamo na voljo ogromno avtomobilov. Toda mnenja, kateri je najboljši, so si deljena. Iz tega sledi, da je tudi prihodnost razvoja svetla, saj je na trgu veliko konkurence. Za razvijalce je dobro, da poznajo več kot eno spletno ogrodje. Večina strokovnjakov meni, da ne smemo samo zaupati člankom z opisi tehnologije. Najbolje je, da tehnologijo preizkusimo sami. Glede na pozitivne rezultate primerjave smo izdelali spletno aplikacijo s pomočjo ogrodja GWT. Pregled aplikacije sledi v naslednjem poglavju.

Poglavje 7

Praktična aplikacija

7.1 Razvojna orodja in tehnologija

Aplikacijo smo razvili s pomočjo razvojnega okolja Eclipse in dodatka Google za Eclipse (GPE), ki omogoča podporo za GWT in aplikacijsko jedro spletnega projekta. Dodatek vključuje tudi preprosto verzijo GWT Designerja. GPE podpira Eclipse 4.2 (Juno). Z GPE lahko preprosto kreiramo *Google Cloud Endpoint* in dostopamo do njih z operacijskima sistemoma Android in iOS ter spletnimi aplikacijami. Dodatek omogoča preprosto kreiranje jedra spletne aplikacije. Zagotavlja:

- čarovnika za izdelavo začetnega jedra aplikacije,
- *Cloud SQL*,
- podporo JPA za *Cloud SQL*,
- vključevanje najnovejših Googlovih programsko aplikacijskih vmesnikov,
- uporabo jedra v ozadju Android projekta.

Razvijalcem pomaga učinkovito kreirati bogato uporabniško izkušnjo, generirati kakovostno kodo AJAX in optimizirati zmogljivosti s *Speed Tracker-*

jem. Celotna aplikacija je bila razvita izključno v programskem jeziku Java z uporabo dodatka GPE [6].

7.2 Specifikacija zahtev

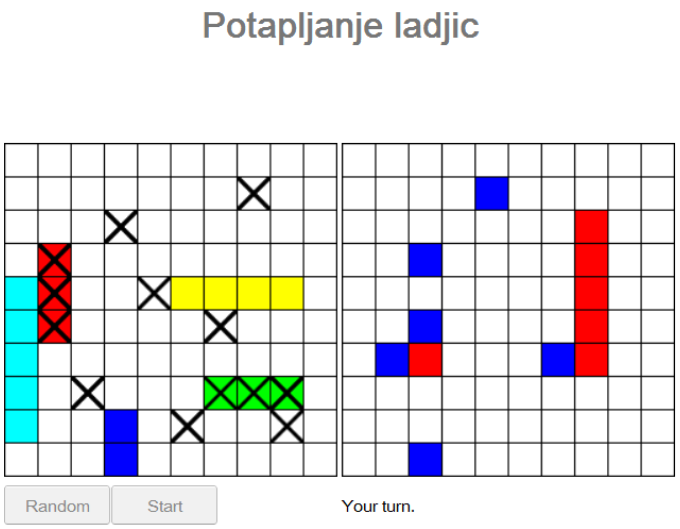
S pridobljenim znajem smo želeli razviti spletno aplikacijo, ki se večinoma izvaja na odjemalcu. Natančneje smo želeli razviti spletno igro z imenom Potapljanje ladjic. Najprimernejše se nam je zdelo ogrodje GWT, ker nudi preprosto pisanje kode na odjemalcu, brez potrebe po znanju skriptnega jezika JavaScript. Nudi nam tudi preprosto izvajanje AJAX zahtev s pomočjo *Remote Procedure Call* (RPC). RPC je notranji komunikacijski proces, ki omogoča programu, da pokliče proceduro v drugem naslovnem prostoru brez potrebe po kodiranju malenkosti pri oddaljeni interakciji.

Igra naj bi omogočala več igralcem (uporabnikom) igranje igre med seboj. Določanje nasprotnikov naj bi bilo naključno. Razporeditev ladij je lahko prav tako naključno. Igro naj bi začel igralec, ki je moral čakati na nasprotnika. Igralec ima možnost pregleda nad svojimi napadi in napadi nasprotnika. Aplikacija naj tudi nekako sporoča stanje igre o tem, kdo je na potezi in ali je igre konec.

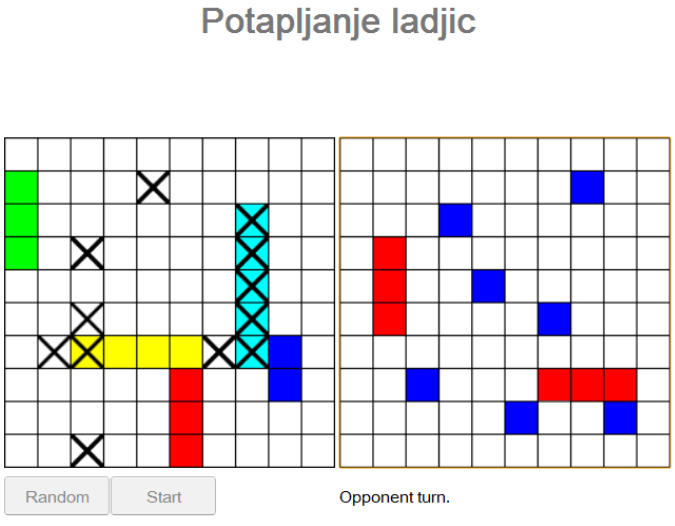
7.3 Implementacija

S spletno igro (Slika 7.1, 7.2) smo želeli prikazati praktično uporabo ogrodja GWT in izpostaviti njegove prednosti, ter hkrati ugotoviti njegove slabosti v praktični uporabi. Glavna prednost, ki smo jo omenili že pri primerjavi tehnologij, je pisanje javanske kode, ki se izvaja v brskalniku in preprosta realizacija AJAXa. Za risalno ploščo smo uporabili razred GWT z imenom Canvas, ki predstavlja oznako HTML5 <*canvas*>.

Najprej smo ustvarili programsko kodo, ki se izvaja na odjemalcu. Ta del vsebuje razporeditev ladij in pripravljenost uporabnika na igro. Razporeditev ladij smo realizirali naključno z dodanimi omejitvami, ki preprečijo



Slika 7.1: Prikaz igre prvega igralca



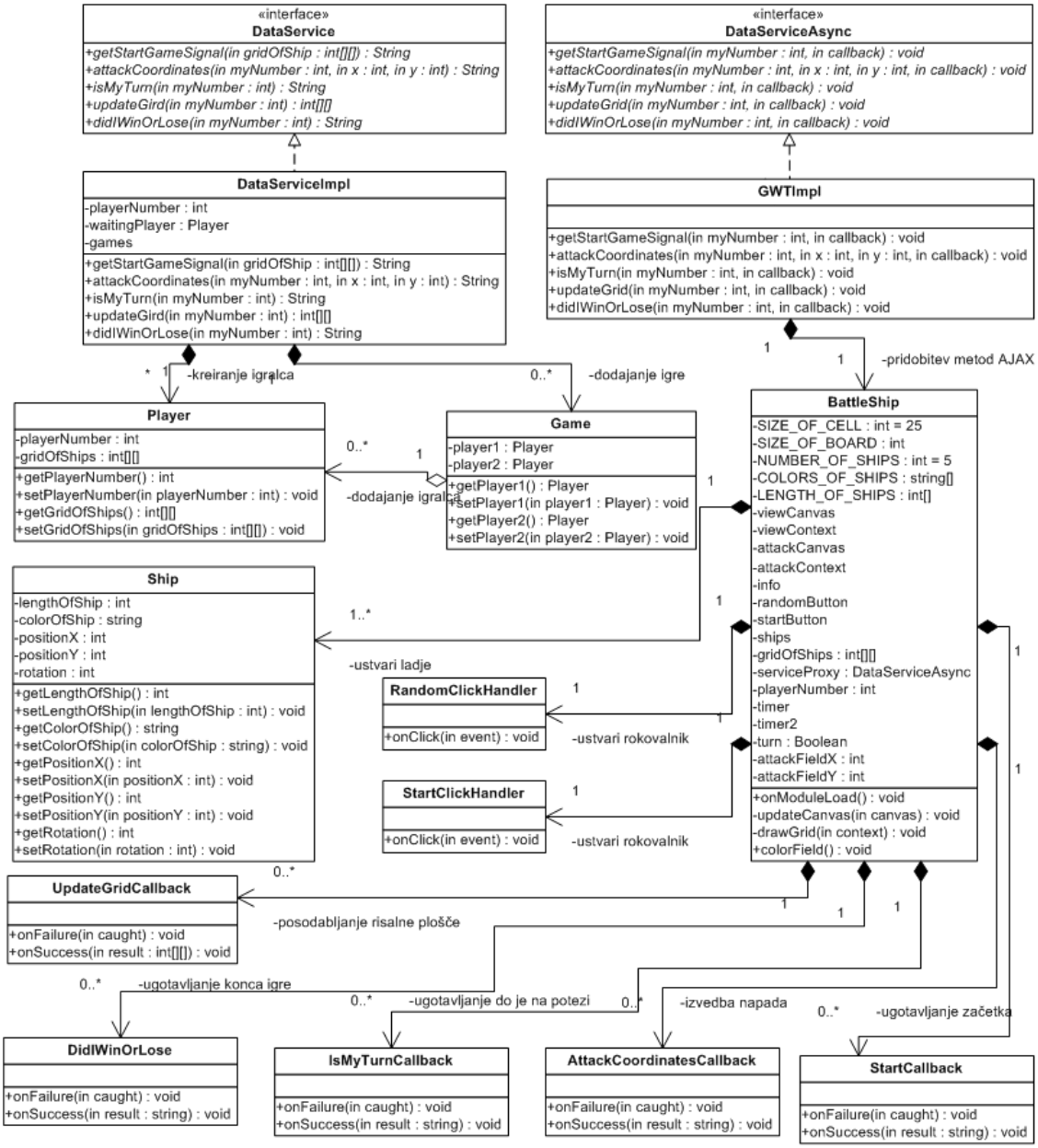
Slika 7.2: Prikaz igre drugega igralca

prekrivanje ladij in izpad ladje iz igralnega polja. Igralno polje je velikosti 10x10. Uporabnik ima na igralni plošči razporejenih pet ladij. Ladje so različnih velikosti in barv. Najmanjša zavzema dve polji, nato ji sledita dve ladij s tremi polji ter po ena s štirimi in petimi polji. Plošča z razporeditvijo ladij je na uporabnikovi levi strani. Desna stran je namenjena napadu na nasprotnikove ladje. Napadena polja se tudi ustrezno obarvajo. Na levi strani se prikazujejo tudi napadi nasprotnika. Na strežniku se hranijo igre in stanje igralcev pri posamezni igri. Odjemalec od strežnika dobiva odgovore o tem, kdo je na potezi, kam je napadel nasprotnik in ali je igre konec. Podrobnejša zgradba aplikacije je z razrednim diagramom prikazana na sliki 7.3. Pomembnejše komponente prikazuje komponentni diagram na sliki 7.4. Odsek delovanja je prikazano z diagramom zaporedja na sliki 7.5.

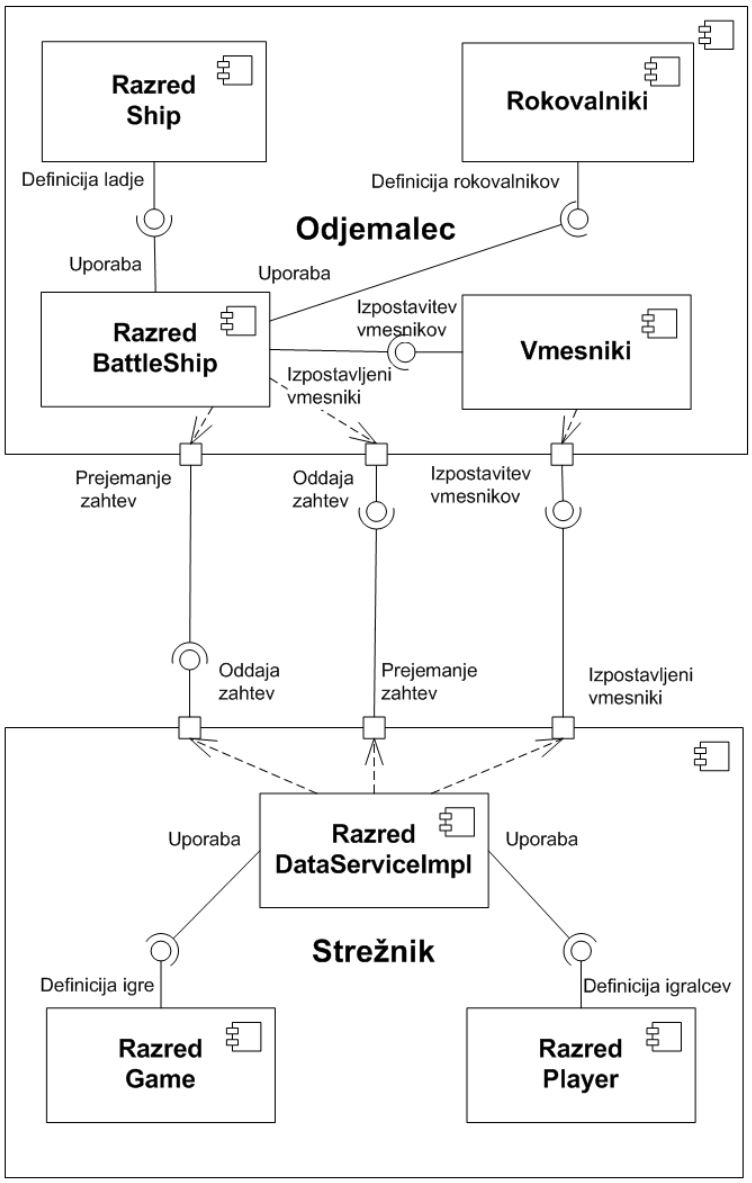
7.4 Delovanje

Ob zagonu aplikacije se v brskalniku naloži naslov igre, dva elementa za risanje ter dva gumba. Risanje poteka s pomočjo razredov *Canvas* in *Contex2d*. Z objektom razreda *Canvas* pridobimo objekt razreda *Contex2d*, in sicer z metodo *getContex2d()*. Vrnjen objekt nam omogoča risanje po površini, ki je določena z metodami objekta razreda *Canvas*. Z metodama *setHeight()* in *setWidth()* nastavimo širno in višino risalne plošče. Na risalni plošči se ob zagonu izriše mreža 10x10. Rišemo z metodami objekta razreda *Contex2d*. Nekaj osnovnih metod:

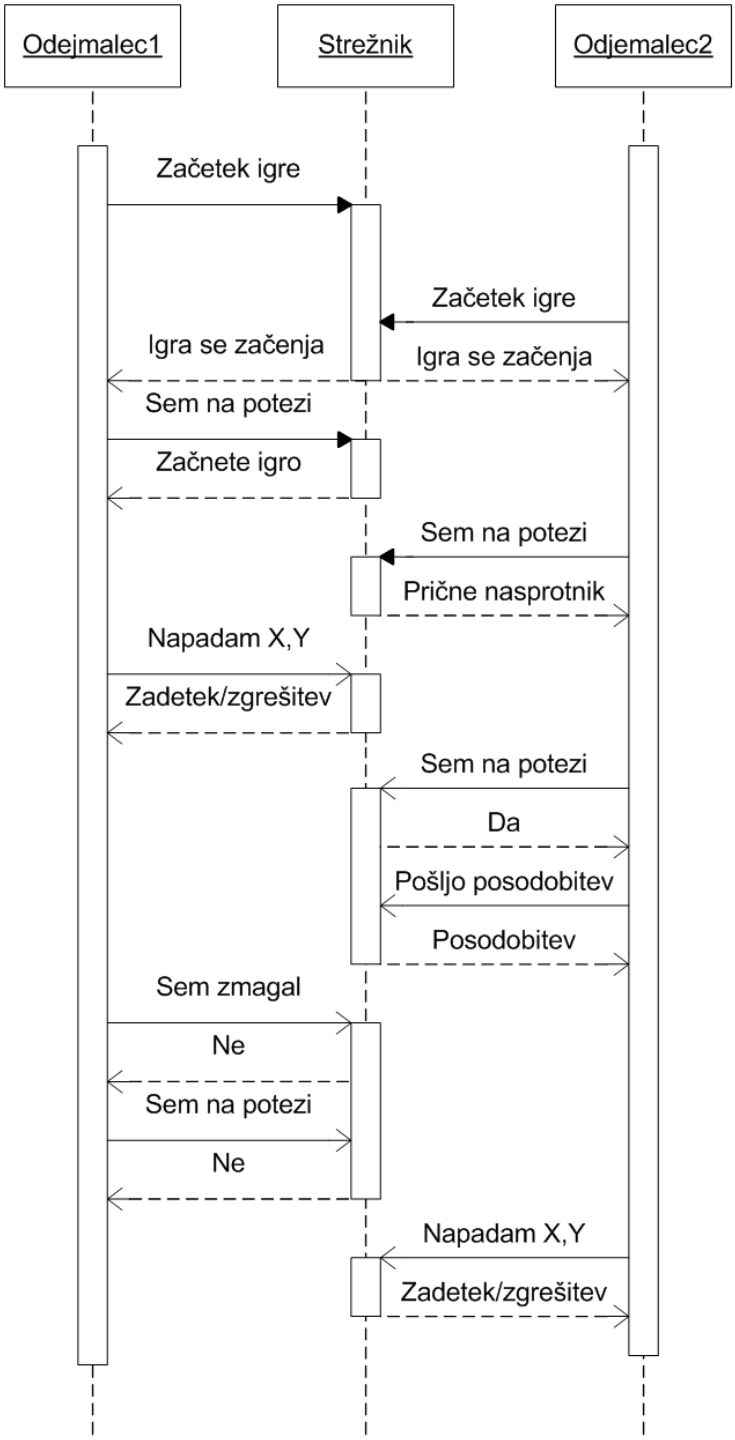
- *setFillStyle(String fillStyleColor)* - nastavitev barve,
- *fillRect(double x, double y, double w, double h)* - izris polnega pravokotinika,
- *setLineWidth(double lineWidth)* - nastavitev debeline črte,
- *moveTo(double x, double y)* - nastavitev začetne pozicije za izris črte,
- *lineTo(double x, double y)* - izris črte iz začetne pozicije do točke (x,y).



Slika 7.3: Razredni diagram



Slika 7.4: Komponentni diagram



Slika 7.5: Diagram zaporedja

Pojavita se gumb za naključno razporeditev ladij in gumb za začetek igre. Uporabnik ima omogočen samo gumb za naključno razporeditev. Za onemogočanje klika na gumb smo uporabili metodo *setEnabled(false)* nad objektom razreda *Button*. Pri naključni razporeditvi v zanki ustvarjamo ladje s pomočjo razreda *Ship* in jim naključno določamo koordninati X in Y ter rotacijo. Razred *Ship* vsebuje pet privatnih atributov. Dolžino ladje predstavlja atribut *lengthOfShip* tipa *int*. Barvo ladje imamo predstavljeno z atributom *colorOfShip*, ki je tipa *String*. Ostali trije atributi so *positionX*, *positionY* in *rotation* tipa *int*. Koordinati x in y določata, kje je ladja. Predstavljata najbolj levo točko ladje, če je ladja v rotaciji vodoravno oziroma najvišjo, če je ladja v rotaciji navpično. Pozicijo oziroma rotacijo ladje tudi določamo naključno. Za naključno določanje smo uporabili razred *Random* iz paketa *com.google.gwt.user.client*. Nato preverimo, ali so izbrane koordinate in rotacija veljavne. To storimo tako, da preverimo prekrivanje z ostalimi ladjami in izpad ladje iz mreže. Preverjanje prekrivanja smo opravili tako, da če se ladja preslika v tabelo z razporeditvijo ladij na mesto, ki je različno od nič, imamo prekrivanje. Izpad ladje preverimo tako, da začetnima koordinatama prištejemo dolžino ladje in z upoštevanje rotacije ugotovimo, ali presega velikosti tabele z razporeditvijo ladij. Ko so vse ladje razporejene, jih izrišemo s pomočjo zgoraj omenjenih metod. Rišemo na podlagi razporeditve v tabeli z razporeditvijo ladij. Nato se uporabniku omogoči gumb za začetek igre in hkrati se mu gumb za naključno razporeditev onemogoči. Ob pritisku na gumb za začetek igre se izvede AJAX zahteva. Zahteva se izvede s klicem metode *getStartGameSignal* nad objektom razreda, ki implementira vmesnik *DataService*. Strežnik nato uporabniku poišče nasprotnika. Najprej pogleda, ali kakšen uporabnik že čaka. Čakajoč igralec je na strežniku povezan s statičnim atributom razreda *Player*. Razred *Player* vsebuje dva privatna atributa, in sicer *playerNumber*, ki predstavlja številko igralca in *gridOfShip*, ki predstavlja tabelo z razporeditvijo igralčevih ladij. Če kakšen igralec čaka, ju strežnik združi v igro in določi, kdo igro začne. Igro začne čakajoči igralec. V primeru, da ni čakajočega uporabnika, strežnik čaka na novega uporab-

nika. Sledi izvajanje treh AJAX klicev na vsako sekundo. To smo izvedli s pomočjo razreda *Timer*, natančenje s klicem metode *scheduleRepating(1000)* nad objektom tega razreda. Število 1000 predstavlj število milisekund. Odjemalec strežnik sprašuje, kdo je na vrsti, kam je napadel nasprotnik in ali je igre konec. Strežnik na vprašanji odgovarja s pomočjo dveh dvo dimenzionalnih tabel z razporeditvijo ladij obeh igralcev. Prazna mesta v tabeli so označena s številom 0. Ladje so predstavljene s številkami od 1 od 5. Napad nasprotnika pa je predstavljen z številom -1. Zahtevo za napad pošljemo z metodo *attackCoordinates(int myNumber, int x, int y)*. *MyNumber* je število, ki enolično določa igralca. S tem se igralec identificira strežniku. Preostala dva atributa predstavljata napadeni koordinati. Strežnik nato poišče igro, v kateri igralec sodeluje. To naredi tako, da pregleda seznam z igrami. Seznam z igrami je predstavljen z objektom tipa *LinkedList*. Igre so predstavljene z objekti razreda *Game*. Razred *Game* vsebuje dva privatna atribura razreda *Player* ter *get* in *set* metode. Ko najde igro, spremeni nasprotnikovo tabelo z razporeditvijo ladij tako, da na napadeno mesto zapiše število -1. Odjemalec za ugotavljanje, kdo je na potezi, uporablja metodo *isMyTurn(int myNumber)*. Da strežnik ugotovi, kdo je na potezi, poišče igro enako kot pri napadu ter prešteje število negativnih števil v obeh igralčevih tabelah z razporeditvijo ladij. Na potezi je tisti uporabnik, ki ima manj negativnih števil. V primeru enakosti je na vrsti igralec, ki je začel igro. Za ugotavljanje konca igre odjemalec uporabi metodo *didIWinOrLose(int myNumber)*. Kdaj je igre konec, strežnik ugotavlja tako, da poišče igro in pogleda, če v kateri od tabel z razporeditvijo ladij ni več številke večje od nič. Izris napada nasprotnika je na levi risalni plošči uporabniškega vmesnika. V napadeno polje se izrišeta dve diagonalni črti. Ko je uporabnik na potezi, je omogočen rokovalnik *ClickHandler* nad objektom razreda *Canvas*, ki zazna pritisk na desni risalni površini uporabniškega vmesnika. Če je uporabnik zadel nasprotnikovo ladjo, se polje obarva rdeče, v nasprotnem primeru modro. Zadetek se ugotavlja ob napadu. Če je vrednost na napadenih koordinatah različna od nič, imamo zadetek, v nasprotnem primeru zgrešitev. Ko eden od igralcev

potopi vse nasprotnikove ladje, je igre konec. Tretji AJAX klic, ki se izvajaa je metoda *updateGrid(int myNumber)*. Metoda poskrbi za posodabljanje leve risalne površine. S pomočjo parametra v metodi metoda poišče igro ter vrne igralčevo tabelo z razporeditvijo ladij. Nato programska koda na odjemalčevi strani izriše posodobitve z že omenjenimi metodami za risanje. Za ponoven začetek igre je treba osvežiti spletno stran. Igralcema se pod desno risalno površino vseskozi prikazujejo sporočila z informacijami o igri. Te informacije so: sporočilo o čankanju na nasprotnikov napad, kdo je na potezi in kdo je zmagovalec igre. Informacije izpisujemo s pomočjo objekta razreda *Label* in metode *setText(String text)*.

Poglavje 8

Sklepne ugotovitve

Razvijalci imajo različne okuse. Zaradi tega obstaja veliko odličnih tehnologij za razvoj spletnih aplikacij. Iz tega sklepamo, da najboljše tehnologije ni. So le bolj in manj primerne za določeno vrsto spletnih aplikacij. Ogrodje GWT je primerno za t.i. bogate odjemalce (ang. rich client). To pomeni, da se aplikacija osredotoča na odjemalca. Uporabniški vmesnik je naložen pri odjemalcu takoj, ko se aplikacija zažene. Take aplikacije omogočajo manj kot bogate spletne aplikacije (ang. rich internet application). Primer bogate spletne aplikacije je Pokerstars, ki vsebuje moderen dizajn, množico efektov in multimedijske dodatke. V to kategorijo spadajo tehnologije Java FX in Adobe Flex. Tehnologija JSF spada med ogrodja za razvoj klasičnih spletnih aplikacij (ang. classical web application). Enako velja za ogrodje Struts. Primer klasičnih spletnih aplikacij je Amazon. Aplikacija je osredotočena na strežniški del, za uporabniški vmesnik pa uporablja več strani, zaznamke in navigacijo. Glede na današnje razmere imajo take aplikacije tudi AJAX. JQuery je knjižnica JavaScript z največjim številom uporabnikov. Je preprosta za učenje in uporabo. Zanjo obstaja veliko dodatkov. Z jQueryjem manipuliramo z modelom objektnega dokumenta (DOM).

Pri implementaciji praktične aplikacije smo se dodobra spoznali z ogrodjem GWT. Sama aplikacija poudarja prednosti te tehnologije. Izvaja se večinoma na odjemalčevi strani, ki pošilja AJAX zahteve do strežnika. Prvi

AJAX klic se izvede ob pritsku na gumb za začetek igre. Uporabnik čaka na nasprotnika. Drugi, tretji in četrti AJAX klic se izvajajo vsako sekundo, kjer odjemalec preverja, ali je na vrsti, ali je zmagal oziroma izgubil ter izvaja posodabljanje mreže z napadi nasprotnika. Zadnji AJAX klic se izvaja ob napadu na določeno polje. V ogrodju GWT je implementacija takih aplikacij proti ostalim javanskim ogrodjem zelo učinkovita. Težave se pojavijo pri obrazcih. V takih primerih je boljša uporaba drugih tehnologij. Zato se tehnolgija GWT večinoma kombinira z ostalimi javanskimi tehnologijami za spletni sloj.

Druga pomembna ugotovite, ki smo jo ugotovili pri razvoju praktične aplikacije, je, da GWT omogoča preprosto pisanje aplikacij, ki so centrirane na odjemalca. Programsko kodo pišemo v Javi. To omogoča pisanje tako strežniške kot odjemalčeve programske kode v samo enem jeziku. Tako razvijalcu ni potrebno pisanje aplikacije v dveh programskih jezikih. Kasneje se programska koda prevede v optimizirano kodo JavaScipt. Pri prevajanju se programska koda na odjemalcu tudi optimizira. Z izkušnjami, pridobljenimi pri izdelavi diplomske naloge, smo ugotovili, da je koda dobro optimizirana, saj spletna aplikacija, ki smo jo razvili, deluje hitro in učinkovito.

Med izdelavo diplomske naloge se je izkazalo, da je za spletne razvijalce ključno poznavanje več tehnologij ter prednosti in slabosti posamezne tehnologije. S tem lahko izberejo pravo tehnologijo za izdelavo želene spletne aplikacije. Tako je razvoj hitrejši in učinkovitejši. Kriteriji za ocenjevanje ogrodij se hitro spreminjajo, saj poskušajo zajeti trenutne trende razvoja spletnih aplikacij.

Literatura

[1] M. B. Jurič, Postopki razvoja programske opreme: gradivo predmeta, Fakulteta za računalništvo in informatiko 2011/2012

[2] (2012)Oracle Dostopno na:
www.oracle.com

[3] (2012)Oracle documentation Dostopno na:
docs.oracle.com

[4] 2012 Java Beans Dostopno na:
http://en.wikipedia.org/wiki/JavaBeans

[5] (2012)The Apache Software Fundation Dostopno na:
http://struts.apache.org/

[6] (2012)Google Web Toolkit Dostopno na:
https://developers.google.com/web-toolkit/

[7] (2012)jQuery Fundation Dostopno na:
jQuery.com

[8] (2012)Model-view-controller (MVC) Dostopno na:
http://en.wikipedia.org/wiki/Model-view-controller

[9] (2012)Raible designs Dostopno na:
raibledesigns.com/rd/category/Java