Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Saša Makorič

Analiza uporabe GWT za razvoj spletnih aplikacij

diplomsko delo

univerzitetni študijski program prve stopnje
računalništvo in informatika

Mentor: prof. dr. Matjaž Branko Jurič

Ljubljana 2013

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

[1] V dogovorju z mentorjem lahko kandidat diplomsko delo s pripadajočo izvorno kodo izda tudi pod katero izmed alternativnih licenc, ki ponuja določen del pravic vsem: npr. Creative Commons, GNU GPL. V tem primeru na to mesto vstavite opis licence, na primer tekst [**?**]

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: 00086/2013
Datum: 15.04.2013

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

Kandidat: **SAŠA MAKORIČ**

Naslov: **ANALIZA UPORABE GWT ZA RAZVOJ SPLETNIH APLIKACIJ**
GWT ANALYSIS FOR WEB APPLICATION DEVELOPMENT

Vrsta naloge: Diplomsko delo univerzitetnega študija prve stopnje

Tematika naloge:

Analizirajte koncepte razvoja spletnih aplikacij. Podrobno proučite ogrodje GWT in opišite arhitekturne koncepte ter model delovanja ogrodja. Preglejte ključne gradnike za razvoj uporabniškega vmesnika in načine komunikacije s strežnikom. Naštejte prednosti in slabosti ogrodja in ga primerjajte z JSF, Structs in jQuery. Na praktičnem primeru aplikacije prikažite najboljše prakse pri razvoju.

Mentor:

prof. dr. Matjaž B. Jurič

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Dekan:

prof. dr. Nikolaj Zimic

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Saša Makorič, z vpisno številko **63080128**, sem avtor diplomskega dela z naslovom:

Analiza uporabe GWT za razvoj spletnih aplikacij

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 11. januarja 2011 Podpis avtorja:

Kazalo

Povzetek

V diplomski nalogi je predstavljena analiza uporabe ogrodja GWT za razvoj spletnih aplikacij. GWT ponuja funkcionalnosti za razvoj celovitih spletnih aplikacij, tako uporabniškega vmesnika kot aplikacijske logike izvajanja na strežniku.
V celoti so predstavljeni arhitekturni koncepti ter model delovanja ogrodja. Velik poudarek leži na prevajalniku, ki uporablja koncept odložene navezave, s katerim lahko brez dodatnega posega v samo programsko kodo zagotovimo koherentno delovanje na različnih brskalnikih.
Predstavljena je uporaba ogrodja za grajenje grafičnega vmesnika in interakcijo s strežnikom. Celotni grafični vmesnik se preko prevajalnika iz izvorne kode Java za čim bolj gladko uporabniško izkušnjo prevede v JavaScript. GWT nam nudi obilico uporabnih grafičnih gradnikov, ki jih lahko poljubno spreminjamo ali sami generiramo ter jih vključimo v aplikacijo. Vsi gradniki lahko enostavno komunicirajo s strežnikom preko klicev RPC.
Predstavljene so tudi slabosti in prednosti uporabe ogrodja GWT ter primerjava z različnimi ogrodji. GWT je primerjan s podobnimi spletnimi ogrodji Java kot sta Struts in JSF in ogrodjem JavaScript jQuery.
V praktičnem primeru si lahko ogledamo razvoj GWT v realnem svetu. Videti je mogoče, kako aplikacijo GWT enostavno povežemo s tehnologijami Java, kot sta JPA in EJB, prikaz najboljših praks v razvoju ter dodajanje zunanjih knjižnic, kot je lib-gwt-svg, za generacijo vektorske grafike.

Ključne besede: GWT, analiza, spletna aplikacija

Abstract

This undergraduate thesis presents the analysis of the GWT framework usage for the development of web applications. GWT offers functionality for the development of complete web applications, as the user interface as the implementation of the application logic on the server.
The thesis shows the architectural concepts and the model of framework operation. There is a great emphasis on the compiler, which uses the concept of deffered binding, providing a coherent render on various browsers without interfering with the programming code.
The thesis also describes the use of the framework for building a graphical interface and the interaction with the server. The complete graphic interface translates itself through the modulator from the Java source code to JavaScript in order to obtain an easy user experience. GWT offers us plenty of graphic widgets that we could change at will or we could generate them and include into the application. All the widgets communicate easily with the server through RPC calls.
Disadvantages and advantages of GWT framework are discussed, and the comparison of various frameworks is performed. GWT is compared to similar Java web frameworks, such as JSF and JavaScript framework (jQuery)..
In practical example we demonstrate the GWT development in in a real world scenario. We show how to connect the GWT application to Java's technologies, such as JPA and EJB. Finally, we describe the best practices in the development of external libraries (e.g. gtw-lib-svg) for the generation of vector graphics.

Keywords: GWT, analysis, web application

Seznam uporabljenih kratic

- AJAX (Asynchronous JavaScript and XML) – skupina tehnologij za ustvarjanje dinamičnih spletnih strani.
- CSS (Cascade Style Sheet) – kaskadna stilska predloga; predstavi stil in način prikaza spletnih strani.
- DOM (Document Object Model) – je standard, ki je neodvisen od jezika in platforme ter opisuje objektno zgradbo spletne strani.
- EJB (Enterprise JavaBeans) – strežniški model za enkapsuliranje poslovne logike.
- GWT (Google Web Toolkit) – ogrodje Java za razvoj spletnih aplkacij.
- HTML, HTML5 (HyperText Markup Language) – jezik za označevanje besedila spletnih strani.
- HTTP (Hypertext Transfer Protocol) – protokol na aplikacijski plasti, namenjen prenosu informacij.
- JPA (Java Persistence API) – orodje Java za upravljanje z entitetami.
- JPQL (Java Persistence Query Language) – povpraševalni jezik za iskanje v entitetah JPA.
- JSNI (JavaScript Native Interface) – vmesnik za integracijo kode JavaScript kode s kodo GWT.
- JSON (JavaScript Object Notation) – format za prenos podatkov med spletnimi aplikacijami in spletnimi strežniki.
- JVM (Java Virtual Machine) – navidezni stroj Java, ki izvaja nižje nivojsko operacijsko kodo Java.
- RPC (Remote Procedure Call) – oddaljeni klic čez internetno omrežje.

- XML (Extensible Markup Language) – razširljiv označevalni jezik; format podatkov za izmenjavo strukturiranih dokumentov na spletu.

Poglavje 1

Uvod

Selitev namiznih aplikacij na splet je že vrsto let več kot očitna. Razlog, zakaj se to dogaja, je dokaj preprost, aplikacija na spletu je enostavnejša za uporabnika. Ni več potrebe po zamudnem nameščanju same aplikacije, niti nameščanju posodobitev, potrebna je le povezava na internet ter brskalnik. Na to dejstvo se morajo prilagoditi tudi razvijalci aplikacij.
Trenutno obstaja že množica odprtokodnih ali plačljivih ogrodij za razvijalce spletnih aplikacij, ki jih ponujajo različna podjetja. Zahteve so učinkovita in preprosta koda – "piši manj, naredi več".
V diplomski nalogi si bomo ogledali ogrodje, razvito na platformi Java EE. Znan problem razvoja na tej platformi so kompleksna ogrodja, na katerih je, preden začnemo s pravim delom, najprej potrebno narediti obilico konfiguracije, k čemur naj bi pripomoglo ogrodje GWT, ki obljublja veliko mero avtomatizacije za razbremenitev programerskega dela.

Poglavje 2

Google Web Toolkit

Google Web Toolkit je odprtokodno orodje, v celoti razvito s programskim jezikom Java za razvijanje dinamičnih spletnih aplikacij. Cilj GWT je ustvarjati dinamične spletne aplikacije AJAX v čim bolj hitrem, enostavnem in optimiziranem načinu. Z uporabo GWT lahko razvijamo in razhroščujemo aplikacije AJAX, ki so s prevajalnikom GWT prevedene v brskalniku, skladnem z JavaScript in HTML.
GWT ponuja za razvoj spletnih aplikacij to, kar sama Java ponuja že v osnovi, in to je koncept "write once, run anywhere" [6].

2.1 Kako deluje GWT?

Z uporabo GWT lahko razvijamo celotno spletno rešitev.
Odjemalčeva stran je v celoti razvita z uporabo programskega jezika Java v specifičnem okolju GWT. Sam GWT nudi tako obilico gradnikov, enostavnih za uporabo, kot metode za lažje razvijanje dinamičnih spletnih komponent in servisnih klicev za pridobivanje podatkov. Celotna koda za odjemalca je nato preko prevajalnika GWT prevedena v optimizirano kodo JavaScript, ki dinamično generira dokument HTML.
Z uporabo sistema GWT RPC lahko enostavno komuniciramo z strežnikom Java. Podatke se prenaša z obkjekti Java, ki jih dinamično serializira preva-

jalnik GWT. GWT lahko povežemo tudi s kakšnim drugim spletnim strežnikom, ki ne podpira platforme Java , in sicer z uporabo zahtevkov HTTP ter razreda GWT Java za branje objektov JSON.

2.2 Osnove kodiranja v GWT

2.2.1 Modul GWT

Ta modul je enkapsulacija funkcionalosti, saj definira, kje je specifična logika za določeno funckionalnost. V njem definiramo lokacijo razredov o uporabniškem vmesniku ter lokacijo razredov, ki se izvajajo na strežniku. V njem je prav tako potrebno definirati tudi uporabo dodatnih knjižnic.
Te informacije uporabi prevajalnik, da lahko pravilno razporedi vso prevedeno kodo in pravilno upošteva odloženo navezavo. Skupno s paketi Java imajo poimenovanje ločeno s pikami (na primer sm.gwt.floorplaner.client). Vsak modul je specificiran v datoteki gwt.xml, ki vsebuje vse razrede Java, potrebne za delovanje aplikacije.

Pri GWT razrede, ki so potrebni za delovanje aplikacije, fizično ločujemo v tri osnovne module.

- Client: Razredi, ki predstavljajo vidni del naše aplikacije. Celotna koda, ki je napisana v sledečih razredih, je ob prevajanju spremenjena v JavaScript. Zaradi omejitev, ki jih JavaScript zahteva, ne moremo uporabiti vseh knjižnic Java.

- Server: Vsi razredi, ki jih bomo potrebovali za delo na strežniku. V večini primerov so to servisni klici RPC za shranjevanje in branje podatkov. Kode se prevede v operacijsko kodo Java, da jo lahko spletni strežnik prevaja. Lahko uporabljamo vse knjižnice Java, ki so na voljo.

- Shared: V večini primerov so razredi, ki definirajo objekte za izmenjavo med odjemalčevo ter strežniško stranjo. Koda se prevede tako v operacijsko kodo kot tudi v JavaScript.

2.2.2 JSNI: Uporaba programskega jezika JavaScript

Pri razvoju aplikacije GWT se lahko srečamo s težavo, kjer moramo v našo aplikacijo vključiti kodo JavaScript, ker imamo zaradi razširjanja prejšnje aplikacije že spisano, ali pa nam GWT enostavno ne nudi zadostne funkcionalnosti. V takem primeru lahko uporabljamo vmesnik JSNI (JavaScript Native Interface).
Uporaba je zelo enostavna in praktično ekvivalentna uporabi metode Java. Potrebno je definirati posebno metodo native, ki je prikazana v izvorni kodi 2.1.

```
public static native void alert(String msg) /*-{
$wnd.alert(msg);
}-*/;
```

Izvorna koda 2.1: Primer metode JSNI.

Slednjo metodo nato lahko poljubno kličemo čez celoten razred. Je zelo fleksibilen sistem, ki ga lahko poljubno združujemo s čisto kodo Java (kličemo kodo Java iz metode JSNI ali obratno).

2.3 Razlike med programskim jezikom Java ter GWT

Čeprav je GWT razvit z programskim jezikom Java, velik del aplikacije v GWT sestoji iz jezika JavaScript, kar nam v nekaterih primerih omejuje uporabo vseh možnih knjižnic, ki so prisotne v okolju Java.
Torej GWT uporablja podrazred celotnega okolja Java JRE. Razlike se pojavijo med prehodom iz razvijalnega načina (aplikacija zagnana v celoti v operacijski kodi Java) v produkcijski način ("front-end"v JavaScript). Torej je ob uporabi knjižnic, ki jih ni mogoče direktno prevesti v okolje JavaScript, potrebno biti pazljiv :

- Vsi primitivni tipi (*boolean, byte, char, short, int, long, float, double*), tipi *Object*, *String*, tabele, uporabniško deifinirani razredi so podpirani z manjšmi razlikami:

  - Aritmetični tipi, kot so *int, short, char*, so v JavaScript prevedeni v 64-mestno število v plavajoči vejici. Kar pomeni, da so v Javi aritmetični tipi spremenljivk, kot je *int*, povoženi izven dovoljenih mej. Operacije na tipu float pa so izvršene kot tip double, kar povzroči "napako"prevelike natančnosti.
  - Long: zaradi pomanjkanja 64-bitnega celega števila v JavaScript se tipi long prevedejo v pare 32-bitnih celih števil.
  - Plavajoča vejica: Nimamo podpore za enojno ali dvojno natančnost plavajoče vejice z uporabo ključa *strictfp*. Uporaba ključa *strictfp* je odsvetovana.

- Izjeme: *try, catch, finally* in uporabniško definirane izjeme so celotno podprte, čeprav funkcija *Throwable.getStackTrace()* ni izvedena, je torej ignorirana. Izjeme tipov *NullPointerException*, *StackOverflowError* in *OutOfMemoryError* so mapirane kot *JavaScriptException* in jih mogoče mapirati nazaj v tip izjeme Java.

- Večnitnost in Sinhronizacija: izvajanje JavaScript se odvija v enonitnem okolju , torej bo uporaba funkcij kot *Object.wait()* ali *Object.notify()* oziroma uporaba ključne besede synchronized enostavno spregledana.

- Dinamično nalaganje razredov: koda je prevedena v eno celoto in je v času izvajanja ni možno spreminjati, zato dinamično nalaganje razredov ni mogoče.

- Regularni izrazi: regularni izrazi so med Java in JavaScript v majhnem obsegu različno interpretirani. Uporaba regularnih izrazov, kot na primer v metodi *split()*, je priporočena v pravilih, definiranih za JavaScript.

- Serializacija: v GWT Java serializacija ni podprta, zato sistem GWT RPC avtomatično opravi serializacijo potrebnih objektov sam.

Zaradi kompleksnosti nekaterih knjižnic, ki jih nudi Java JRE, in so potrebne za razvoj spletnih aplikacij, je skupaj z knjižnico GWT posredovanih nekaj dodatnih knjižnic in metod, ki uporabljajo podmnožico iz okolja Java JRE:

- *com.google.gwt.i18n.client.DateTimeFormat* : Podpira podmnožico iz *java.util.DateTimeFormat*.
- *com.google.gwt.i18n.client.NumberFormat* : Podpira podmnožico. iz *java.util.NumberFormat*.
- *com.google.gwt.user.client.rpc.IsSerializable* : markirni razred je za prenašanje objektov v servisnih klicih uporabljen podobno kot *java.io.Serializable*.
- *com.google.gwt.user.client.Timer* : Poenostavljena verzija *java.util.Timer* razreda za potrebe enonitnega okolja, ki je prisoten v brskalnikovem okolju.

2.4 GWT prevajalnik

Prevajalnik GWT je bistvo tehnologije GWT. Sposobnost specifičnega prevajanja za različne brskalnike ter prevajanje kode Java v kodo JavaScript je možno le zaradi posebne arhitekture prevajalnika. Kot podobnost lahko primerjamo s prevajalnikom Javac, ki kodo Java spremeni v operacijsko kodo, da jo operacijski sistem lažje interpretira. Prevajalnik GWT pa kodo prevede v JavaScript. V obeh primerih tako dosežemo iskano prenosljivost ne glede na arhitekturo izvajalnega sistema.

2.4.1 Prevajanje v JavaScript

Zaradi striktnega ločevanja kode med odjemalčevo in strežniško stranjo lahko dosežemo, da imamo kodo prevedeno specifično za določen brskalnik. Koda

na odjemalčevi strani, ki je prevedena v JavaScript, je tako specifična za določen brskalnik, kar nam olajša delo za optimizacijo, ki nam jo sam GWT nudi. Slednje dosežemo s postavitvijo pravil v sami kodi. Prevajalnik ob prevajanju upošteva vsa pravila, ter lahko tako izhodno kodo pravilno optimizira. Tako sam prevajalnik razrede, metode in atribute, ki niso nujni, odstrani ali pa doda. To je ena glavnih prednosti prevajalnika GWT pred drugimi knjižnicami JavaScript, ki jih je potrebno za pravilno delovanje odjemalcu posredovati v celoti. Tako lahko odjemalcu posredujemo le del, ki je za njega relevanten.

2.4.2 Odložena navezava

Vsak brskalnik ima svoje načine, kako interpretirati kodi JavaScript in HTML. Prav zaradi teh razlik, ki jih je potrebno upoštevati ter specifično reševati, je nastal projekt GWT. Jedro razvoja GWT je njegov prevajalnik, ki izvorno kodo Java za specifične potrebe prevede na več načinov.
Pri klasičnem načinu programiranja v Javi bi uporabili njeno sposobnost dinamičnega nalaganja razredov med izvajanjem, za kar se uporablja Java Reflection. Za specifičen brskalnik,bi specifično kodo razporedili v podrazrede, ki bi se ob izvajanju naložili in izvedli glede na potrebe izvajajočega brskalnika. Slednje pa pri aplikacijah GWT ni mogoče. Zaradi arhitekture JavaScript namreč dinamično nalaganje razredov ni mogoče.
Namesto dinamičnega imamo odloženo nalaganje. Lahko si ga predstavljamo kot "dinamično nalaganje, ki se izvede med prevajanjem namesto izvajanjem" [17].

2.4.2.1 Prednosti

Prednost odložene navezave je, da nam sam prevajalnik proizvede kodo JavaScript, ki je že optimizirana glede na brskalnik. Dodatne prednosti posameznega prevajanja pa so:

- Program porabi manj časa za razvijanje z avtomatičnim generiranjem

kode pri implementaciji in kreiranjem razredov proksi (uporabljeno pri modulu GWT RPC).

- Drastično zmanjša obseg kode JavaScript, ki jo odjemalec potrebuje za izvajanje aplikacije.
- Ker so različne implementacije že ob prevajalnem času vezane na posamezne potrebe, pri izvajalnem času nimamo več časovne kazni kot pri dinamičnem nalaganju.

2.4.2.2 Prevajanje glede na Brskalnik

Velik del API-ja, ki vsebuje grafične gradnike, ima že implicitno vgrajeno funkcionalnost, da sam opravi delo prevoda kode za specifičen brskalnik, ne da bi sam uporabnik to opazil.
Prevajalnik razume vse razširjene brskalnike (Internet Explorer, Firefox, Chrome, Safari idr.) in kodo prevede za specifične potrebe posameznega brskalnika. V privzeti nastavitvi gre prevajalnik skozi vse cikle prevajalnega postopka za vse brskalnike, kar zelo podaljša prevajalni čas aplikacije. Za razvijalne potrebe v praksi omejimo prevajalnik na enega ali dva brskalnika. Postopek je skrajno enostaven, opravi se z vpisom ene vrstice v modulu GWT: *<set-property name="user.agent"value="gecko1_8"/>*. Pri slednjem smo prevajalniku GWT ukazali, naj kodo prevede le za brskalnike Firefox.

2.4.2.3 Prevajanje glede na lokalizacijo

Uporabnost ločenega prevajanja pokaže svojo moč pri lokalizaciji spletne aplikacije. Tako lahko dosežemo, da se aplikacija prevede za vsak posamezen jezik posebej, in ko na primer odjemalec, ki uporablja Firefox v slovenskem jeziku, dostopa do aplikacije GWT, mu avtomatično ponudimo aplikacijo, ki je prevedena za tip brskalnika Firefox, z I18n lokalizacijo, nastavljeno na slovenski jezik.
Slednje dosežemo z uporabo *<extend-property name="locale"values=šl"/>* v modulu GWT. V tem primeru se prevajalni čas še dodatno poslabša. Namreč

vsak jezik je potrebno množiti s tipom brskalnika. Na primer, če hočemo aplikacijo, ki bo delovala na petih brskalnikih v petih jezikih, imamo na voljo kar 25 prevajalnih ciklov.

2.4.3 Struktura prevedene kode

Generirana koda se nahaja v direktorijah po strukturi, ki je definirana po standardu WAR (ang. Web Application Archive), kar omogoča lažjo postavitev aplikacije na aplikacijski strežnik, kot sta Tomcat ali Jboss.
V direktoriju se nato razvrstijo datoteke, potrebne za zagon aplikacije. Pomembne datoteke so:

- *<ime modula>.cache.html*
- *<ime modula>.nocache.js*
- *<alfanumerično>.gwt.rpc*

2.4.3.1 Datoteke tipa .cache.html

Datoteke tega tipa vsebujejo aplikacijsko logiko. To je koda, prevedena v JavaScript, v ovojnici HTML. Ovojnica HTML je uporabljena za lažjo kompresijo. Določeni brskalniki namreč nepravilno upravljajo s čistimi datotekami JavaScript.
Datoteke se poimenujejo glede na izračun vsote MD5 njihove vsebine. To nam zagotavlja unikatno ime glede na vsebino ter izboljša shranjevanje v predpomnilniku brskalnika (shranjevanje v predpomnilnik poteka le ob vsebinski spremembi datoteke).

2.4.3.2 Datoteke tipa .nocache.js

V njih se nahaja koda, ki se specifično generira ob prevajanju, in je prevedena kot odložena navezava. Vsebuje metode, specifične za posamezni brskalnik, jezik aplikacije itd.
Datoteke tega tipa se ne smejo shranjevati v brskalnikov predpomnilnik, iz

tudi tega sledi poimenovanje *nocache*. Datoteke se morajo ob vsakem dostopu do aplikacije ponovno naložiti, ker jih prevajalnik GWT vsakič regenerira z istim imenom. Da se aplikacijski podatki ne bi shranjevali v predpomnilnik, dosežemo z datoteko gwt.js, ki vsaki datoteki .nocache doda HTTP metodo GET s časovnim parametrom, ki ga brskalnik razume kot dinamični zahtevek HTTP in ga zato ponovno naloži.

2.4.3.3 Datoteke tip .gwt.rpc

V tej datoteki je zapisana serializacijska politika, torej kateri uporabljeni tipi objektov so serializirani za uporabo preko mreže. Tako morajo biti tudi objekti, definirani s strani uporabnika, ki implementirajo java.io.Serializable, uporabljen pri klicih RPC, definirani v datotekah .gwt.rpc.

2.5 Potek zagona GWT aplikacije

Za boljše razumevanje vseh datotek, ki jih GWT generira, si oglejmo, kako poteka zagon aplikacije GWT.

1. Brskalnik naloži in procesira glavno stran HTML.

2. Ko brskalnik doseže oznako HTML *<script src=«ime_modula>.nocache.js»* naloži in procesira celotno kodo JavaScript v datoteki.

3. V datoteki .nocache.js se nahaja koda JavaScript z ukazi za parcialno prevajanje, ki nato naloži potrebno datoteko .cache.html s specifičnimi konfiguracijami za brskalnik in lokalizacijo aplikacije.

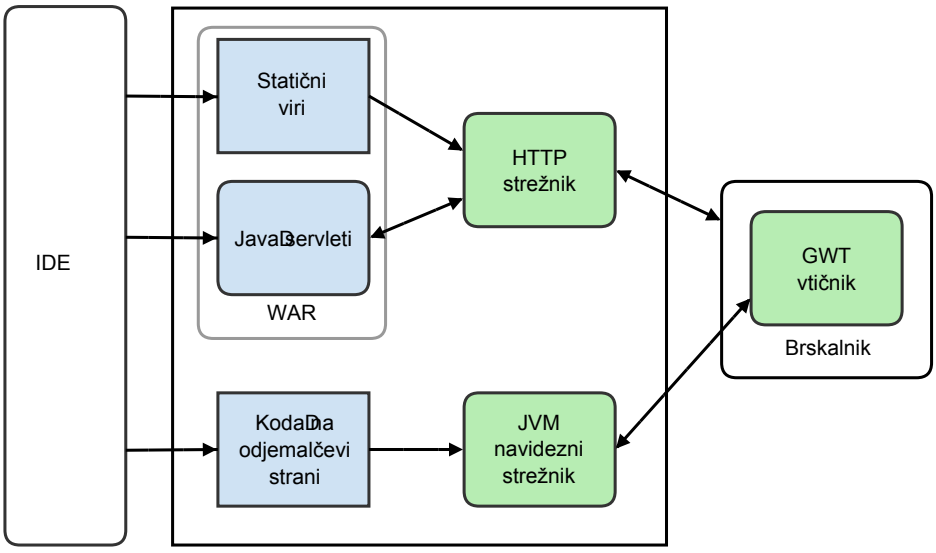
4. Nato koda v datoteki .nocache.js postavi na gostujoči strani oznako *<iframe>*, v katero se naloži vsebina iz .cache.html, ki vsebuje dejansko programsko logiko in aplikacijo.

2.6 Razvijanje in umestitev aplikacij GWT

Potek razvijanja aplikacij GWT poteka v dveh fazah, ki sta ločeni zaradi praktičnih razlogov: razvijalni način (ang. development mode) ter produkcijski način (ang. production mode).

2.6.1 Razvijalni način

Razvijanje aplikacije večino časa poteka v tem načinu. Tako se koda ne prevaja v JavaScript, ampak nam navidezni stroj Java (ang. Java Virtual Machine – JVM) izvaja celotno kodo, torej tako odjemalčevo kodo, ki bi se drugače prevedla v JavaScript, kot tudi strežniško. S tem načinom, ki nam omogoča tradicionalno metodo programiranja "kodiranje-testiranje-razhroščevanje"(ang. code-test-debug), je razvijanje karseda hitro, saj prevajanje celotne aplikacije ob vsaki spremembi ni več potrebno. Na diagramu (Slika 2.1) si lahko ogledamo potek ene seje v ravzijalnem načinu.



Slika 2.1: Diagram poteka ene seje v razvijalnem načinu.

Za izvajanje aplikacij v razvijalnem načinu je v enem od podprtih brskalnikov potrebna vsaj verzija GWT 2.0. Strežniška koda pa se izvaja na internem strežniku Jetty, po potrebi pa se lahko direktno izvaja na željenem spletnem strežniku, ki bo v bodoče gostil produkcijsko verzijo. V tem primeru se lahko izognemo v začetku določenim težavam, ki nam lahko prinašajo različne implementacije spletnih strežnikov.
Razhroščevanje poteka enako kot pri drugih podobnih tehnologijah. S poljubnim delovnim okoljem (npr. Eclipse IDE) nastavimo prekinitveno točko, ki prekine izvajanje ne glede na to, ali razhroščujemo odjemalčevo ali strežniško stran (v razvijalnem načinu se namreč vsa koda izvaja v JVM).

2.6.2 Produkcijski način in umestitev

Po končanem razvijanju in testiranju v razvijalnem načinu lahko celotno aplikacijo umestimo v spletni strežnik (potrebujemo Java EE skladen spletni strežnik). V tem primeru se celotna koda prenese na spletni stržnik, ki je zadolžen za izvajanje aplikacije.
Produkcijski način je primeren za testiranje hitrosti in odzivnosti aplikacije, saj ni več odvisen od JVM, ki je simuliral delovanje dela kode JavaScript v operacijski kodi Java. Nazadnje je aplikacija specifično prevedena za vse brskalnike, ki jih želimo podpreti.

Poglavje 3

Grafični gradniki v GWT

Gradnja uporabniškega vmesnika je zelo podobna drugim ogrodjem Java, kot sta Swing in SWT, le da se pri GWT, za razliko od prej omenjenih, kjer se vmesniki izrišejo z uporabo grafike po pikslih, tu elementi HTML kreirajo dinamčno. Vsi elementi so dinamično kreirani s kodo JavaScript, prevedeno iz kode Java.
Pri uporabi grafičnih elementov posebno ravnanje glede kompatibilnosti z različnimi brskalniki ni potrebno. To izvira tudi iz dejstva, da so vsi gradniki generirani v element HTML, ki je za določen brskalnik najbolj naraven. Na primer, gradnik *Button* se generira v element HTML *<button>* za razliko od nekaterih grafičnih ogrodij, ki sintetično generirajo gradnike, kjer bi se gradnik kot *Button* zgeneriral v element * * z vključenim elementom *<label>*, ki bi predstavljal gumb. Kar ločuje GWT od drugih orodij, je uporaba Java objektov za vizualne elemente. Ne glede na to, ali je element panel, grafični gradnik ali katerikoli drugi element, je predstavljen kot javanski objekt in se tako tudi obnaša. Vsak element nosi metode *getter* in *setter* za definiranje njihovih lastnosti. Elemente lahko uporabljamo v seznamih in tabelah ter jih lahko poljubno kloniramo, kar se lahko dobro izkorišča pri prenosljivosti po meri narejenega elementa za uporabo v različnih aplikacijah.
Vsi vizualni objekti so pridobljeni iz objekta UIObject. Slednji objekt ovije element DOM in ne sprejema dogodkov. Vse metode objekta so dosegljive

podrazredom. Tako imamo za vse vizualne elemente standardne metode, ki vključujejo:

- delo s CSS stili (*addStyleName()*, *setStyleName()*, *getStyleName()*),
- nastavitve velikosti elementa (*setHeight()*, *setWidth()*),
- delo z elementom DOM, ki je osnova objekta (*getElement()*),
- razne metode za nastavljanje elementa DOM (*setTitle()*, *isVisible()*, *getOffsetHeight()*, *getOffsetWidth()*).

3.1 Paneli

Osnovni gradniki vmesnika so ti. paneli. Uporablja se jih za postavitev grafičnih gradnikov v aplikaciji, delujejo kot ovojnica, na katero prilepimo poljubno število gradnikov in jih lahko poljubno postavljamo. Vsi izvirajo iz razreda *Panel*, ki je abstrakten razred, ki izvira iz razreda *widget*, je torej grafični gradnik, ki lahko vsebuje druge gradnike. Razdeljujemo jih v dve kategoriji: osnovni paneli ter postavitveni paneli.

3.1.1 Osnovni paneli

So enostavnejši paneli, ki nudijo osnovne funkcije in postavitve za vsebovane gradnike. Vsi paneli te oblike so oviti z elementom * * , ki je v hierarhiji gradnika najvišji. Glede na izbran panel nato v najvišjem * * elementu na ravni hierarhije DOM GWT prilepi naslednji element, ki nam nudi različne funckionalnosti. Tako na primer FormPanel ovije HTML *<form>* element in nam nudi funkcionalnost GET in POST ali pa FlexTable, ki nam predstavlja *<table>* element.
Posebno pozornost je potrebno nameniti elementu RootPanel, ki je panel na najvišji ravni hierarhije aplikacije DOM, na katerega so v končni fazi prilepljeni vsi drugi grafični gradniki. Tak panel je samo eden in tako prestavlja ti. singleton panel, do katerega lahko dostopamo preko metode RootPanel.get(),

ki je vidna čez celotno aplikacijo. Deluje kot ovojnica za element <body> uporabljenega dokumenta HTML.

3.1.2 Postavitveni paneli

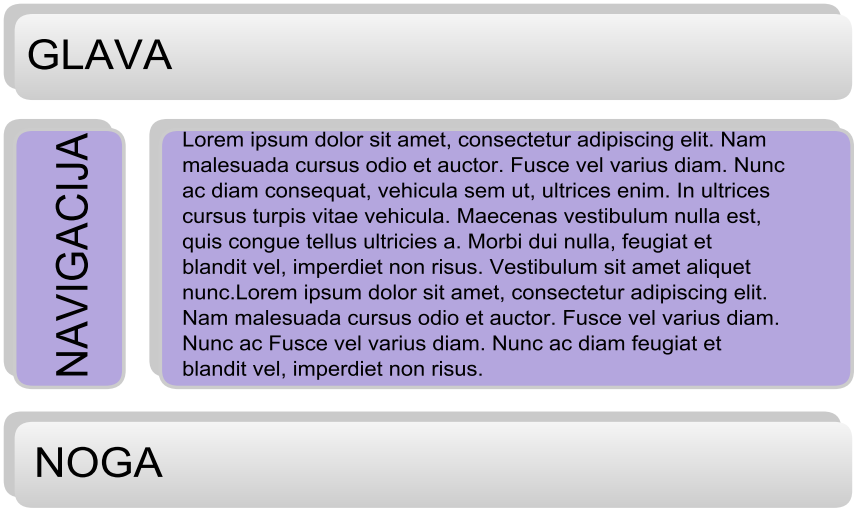
So bili predstavljeni z GWT-različico 2.0. Pred njmi je bila gradnja vmesnika zelo nepredvidljiva in vmesnik ni pravilno deloval na vseh brskalnikih. Novi paneli pa so zgrajeni tako, da čim bolje izkoristijo prav brskalnikovo sposobnost postavitve elmentov HTML. Vsi paneli izhajajo iz enega razreda (Layout razred), ki z razliko od ostalih elementov GWT deluje direktno na hierarhijo DOM. Razred Layout uporabljajo samo gradniki višjega nivoja hierarhije DOM in je namenjen temu, da deluje kot ovojnica za druge elemente (tipično * * element). Vsebovani gradniki se postavijo avtomatsko in se ob spremembi njihovih karatkeristik, kot sta velikost ali robovi¸obnašajo v predvidljivem načinu – njihova postavitev v prostoru se ne spremeni.
Pri postavitvenih panelih imamo elemente, ki nam že v osnovi nudijo najbolj uporabljene postavitve pri gradnji spletnih uporabniških vmesnikov: glava, noga, navigacija, zavihki. Dober primer za tako postavitev je DockLayoutPanel, na katerem lahko z zelo malo truda definiramo izgled celotne aplikacije. Uporaba je prikazana v izvorni kodi 3.1, končni izgled pa na sliki 3.1.

```
DockLayoutPanel p = new DockLayoutPanel(Unit.EM);
p.addNorth(new HTML("GLAVA"), 2);
p.addSouth(new HTML("NOGA"), 2);
p.addWest(new HTML("NAVIGACIJA"), 10);
p.add(new HTML(vsebina));
```

Izvorna koda 3.1: Generacija gradnika DockLayoutPanel.

3.2 Grafični gradniki

Vsi grafični gradniki so vsebovani v panelih, ki prevzemajo nalogo njihovih postavitev. Grafični gradniki in paneli delujejo na vseh brskalnikih enako in



Slika 3.1: Izgled po prevedeni kodi gradnik DockLayoutPanel.

zaradi tega ni več potrebe po pisanju specializirane kode za vsak brskalnik posebej.
Primeri so gumbi, drevesa, vnosna polja, kar nam za gradnjo vmesnikov ponuja večina orodij. Prednost GWT pa je v tem, da lahko grafične gradnike naredimo tudi sami in jih nato tudipoljubno uporabljamo.

3.2.1 Gradnja grafičnih gradnikov z uporabo Composite

Potrebna je razširitev razreda *Composite*. To je specializiran grafični gradnik, ki lahko vsebuje druge komponente (tipično panel) ampak deluje tako, kot da je v celoti grafični gradnik. Na zelo enostaven način lahko tako kombiniramo različne panele in grafične gradnike ter nato vse skupaj uporabljamo kot en sam grafični gradnik. Veliko grafičnih gradnikov, ki so že vsebovani v GWT, je sestavljenih z uporabo *Composite* razreda, na primer *TabPanel*, ki je setavljen iz elementov *TabBar* in *DeckPanel*).
Prednost uporabe Composite je v tem, da lahko namesto gradnje kompleksnih gradnikov z raširjanjem že obstoječih sami nastavimo, kakšne metode

se lahko uporabljajo, kar je bolje, kot da se metode podeduje iz nadrazreda.

3.2.2 Gradnja grafičnih gradnikov na ravni izvorne kode

Možno je tudi, čeprav bolj kompleksno, sestaviti grafični gradnik na ravni kode Java, kot so sestavljeni osnovni grafični gradniki. Nov gradnik tako razširja osnovni razred *Widget*, ki je podrazred razreda *UIObject*. Ker imata osnovna razreda *UIObject* ter *Widget* definirane samo splošne metode JavaScript za prikaz elementa, je za delo s postavitvijo ter stili potrebno napisati specifične metode za interakcijo s takimi elementi. Pri tem je potrebno biti pazljivi glede kompatibilnosti z različnimi brskalniki. Tako je takemu gradniku potrebno dodati še ukaze za prevajalnik GWT, da lahko pravilno izolira brskalnikovo specifično kodo.

3.3 Dogodki in upravljalci dogodkov

Upravljalčev vmesnik definira eno ali več metod, ki jih grafični gradnik pokliče ob dogodku. Če imamo razred, ki uporablja dogodke določenega tipa, implementira vmesnik upravljalca, ki je zadolžen za izbrani dogodek. Na primer, razred gumba objavi dogodke tipa *ClickEvent*, ki jih upravlja upravljalec *ClickHandler*.
Ob uporabi upravljalca za vsakega grafičnega gradnika posebej lahko pride do prevelike obremenitve spomina, zato lahko uporabimo samo enega upravljalca, ki je deljen med več grafičnih gradnikov. Ti sami sebe deklarirajo kot izvor dogodka, da lahko en sam upravljalec razlikuje med pošiljatelji dogodkov. Ta način uporablja manj spomina, potrebuje pa več kode.

3.4 Uporaba CSS

Uporaba stilnih predlog CSS je idealna za uporabo v spletnih aplikacijah. GWT spodbuja razvijalce v uporabo CSS z integriranimi funkcijami za delo s stili. Vsak gradnik ima definirane metode setter za dodajanje stilskega

imena gradniku, ki je direktno povezano na ime v predlogi CSS. Tako lahko ločimo stilske lastnosti gradnikov od aplikcijske logike, kar pomaga pri hitrejši generaciji aplikacije, hitrosti aplikacije, lažjemu spreminjanju stilov ter manjši porabi pomnilnika.

3.4.1 Stili grafičnih gradnikov

Vsi grafični gradniki GWT že imajo svoj CSS za grafično oblikovanje. To nam olajša delo, saj lahko spremenimo ta privzeti stil in s tem dobimo koherentni prikaz vseh grafičnih gradnikov, ki spadajo v isto družino. Privzeto vsaki grafični gradnik dobi svoje ime razreda za uporabo CSS, ki je sestavljeno iz *gwt-<classname>*. Na primer, Label grafični gradnik se imenuje gwt-Label. Vsem grafični gradnikom in panelom lahko definiramo svoj stil CSS, ki ga nastavimo z porabo metode setStyleName() ali pa dodamo stil z uporabo metode addStyleName(). Dodajanje datotek CSS Datoteko CSS lahko vežemo na več načinov:

- z uporabo *<link>* elementa na glavni strani,
- z *<stylesheet>* elementom v datoteki XML module,
- z uporabo *CssResource* metoda vsebovane v *CleintBundle*,
- z uporabo vrstičnega elemeneta *<ui:style>* v predlogi *UiBinder*.

Prva dva načina nista več v veljavi.

3.4.2 Privzete teme GWT

Sam GWT nudi že tri privzete teme za uporabo: *standard, chrome* in *dark*. Definiramo jih v datoteki module z odkomentiranjem določene vrstice.

3.5 UiBinder

Ker je gradnja spletne strani najbolj naravna z uporabo HTML in CSS, ima GWT ogrodje *UiBinder*, ki z uporabo markirnega jezika XML gradi spletne

strani z grafičnimi gradniki kot dodatneoznake HTML (tako po meri generirani grafični gradniki kot tudi grafični gradniki GWT).
Z uporabo sistema *UiBinder* olajšamo delo tudi brskalniku. Generiran gradnik je brskalniku podan kot dolg niz HTML, kar brskalniku olajša generacijo strukture DOM. Potrebno pa je razumeti, da *UiBinder* ne more biti uporabljen za dinamično spletno aplikacijo, ampak le za postavitev dinamičnih elementov.
V primeru kode lahko vidimo, kako so bil postavljeni elementi, ki so bili zgrajeni kot lastni grafični gradniki. Taki elementi imajo definiran svoj imenski prostor in so lahko na istem *UiBinder* elementu uporabljeni kadarkoli.
Vsi elementi *UiBinder* so vezani na javanski razred istega imena, ki element inicializira preko metode *initWidget()*. Z vsemi poljubnimi elementi, ki so v XML definirani z uporabo značke *ui:field*, lahko nato v javanskem razredu poljubno manipuliramo kot z javanskimi spremenljivkami.

3.6 Uporaba slik

Tipična spletna stran je sestavljena iz velikega števila malih slik. GWT vse slike združi v eno, ki jo nato odjemalec naloži iz strežnika, kjer se obnaša kot objekt Java. S tem načinom upravljanja s slikami odpravimo tri velike težave:

- odpade potreba po povpraševanju strežnika z zahtevo HTTP za vsako sliko posebej,
- tudi ko imamo na odjemalčevem brskalniku shranjene slike v predpomnilniku, je še vedno potrebno povpraševanje (zahteva 304) za stanje slike (spremenjena/nespremenjena),
- HTTP 1.1 zahteva omejitev odhodnih povezav, kar bi nato blokiralo klice RPC.

GWT rešuje slednje težave z uporabo razreda *ImageResource* (Izvorna koda 3.2), ki vse slike združi v eno. Ob zagonu tako imamo samo eno sliko, do ka-

tere je potrebna ena sama povezava. Slike definiramo v poljubnem razredu, ki razširja razred *ClientBundle*.

```
interface Resources extends ClientBundle {
@Source("logo.png")
ImageResource logo();
}
```

Izvorna koda 3.2: Definiranje slikovnih virov.

Tam, kjer nato želimo uporabiti sliko, inicializiramo naš razred z metodo *GWT.create()* in sliko lahko uporabljamo kot objekt Java (Izvorna koda 3.3).

```
Resources resources = GWT.create(Resources.class);
Image img = new Image(resources.logo());
```

Izvorna koda 3.3: Uporaba slikovnih virov.

Poleg tega ima objekt *Image* ob nalaganju aplikacije že predefinirano velikost slike, tudi če slika še ni bila naložena, to preprečuje nerodno obnašanje spletnih aplikacij ob nalaganju slike ("poskakovanje slik"), saj so lahko s predefinirano velikostjo brskalnika v pričakovanju naložen prazen prostor (ang. placeholder).

3.7 Internacionalizacija aplikacij GWT

GWT nam nudi več možnosti internacionalizacije naših aplikacij, v naslednjem sklopu si bomo ogledali dve najbolj uporabljeni.

3.7.1 Internacionalizacija z uporabo statičnih nizov

Namesto nizov uporabljamo oznake, ki so v paru z določenim nizom, ki določa tekst v izbranem jeziku. Je najpogosteje uporabljena, saj je z vidika hitrosti izvajanja aplikacije najbolj učinkovita. Prevodi so vnaprej definirani v datotekah .properties, ki se ob prevajanju glede na določitev v modulu GWT

pravilno prevedejo in se ob zahtevi ponudijo uporabniku. Slednje datoteke so standardne datoteke Java properties s to razliko, da so vse UTF-8 kodirane, kar pomeni, da lahko brez problemov uporabljamo znake UNICODE kot š,č,ž, itn.
Na izbiro imamo dva vmesnika za statično prevajanje *Constants* in *Messages*. Največja razlika med njima je da lahko ta, da vmesnik *Messages* sprejema tudi spremenljivke, ki jih lahko podamo v sporočilo.

3.7.2 Internacionalizacija z uporabo dinamičnih nizov

Uporablja se, če imamo že obstoječo aplikacijo na aplikacijskem strežniku, ki ne podpira lokalizacije GWT. Z uporabo razreda *Dictionary* lahko na gostiteljski strani preverimo, kateri prevodi so že nameščeni. Slednji razred vsebuje metode za pregled parov ključ/vrednost, ki so definirani kot spremenljivke JavaScript.

3.8 Upravljanje z zgodovino brskalnika

Mehanizem GWT je zelo podoben drugim mehanizmom AJAX za upravljanje z brskalnikovo zgodovino, na primer RSH (ang. Really Simple History), ki upravlja s fragmenti URL, ki shranijo interno stanje aplikacije in ne povzročijo osvežitve celotnega spletnega mesta med prehodom od enega do drugega. Uporablja žetone, ki predstavljajo določeno stanje aplikacije, ki ga je mogoče doseči preko navigacije skozi brskalnikovo zgodovino. Žeton je alfanumerični niz, ki je dodan na koncu naslova aplikacije URL, definiran je z znakom # pred njegovim imenom.
Primer žetona: *http://www.aplikacija.si/aplikacijezzetonom#Token1*

Ko brskalnik dostopa do žetona Token1, aplikacija vrne interno stanje na del, ki je definiran z omenjenim žetonom. Žetone se definira programsko, s pomočjo razreda *History*, ki predstavlja brskalnikov sklad z zgodovino. Postavimo jih v aplikacijo z uporabo metode *History.newItem("ime_žetona")*. Mesto v aplikaciji, kjer imamo element, ki potrebuje zgodovino, vežemo z

upravljalcem dogodkov *History.addValueChangeHandler()*, ki se odziva na spremembe v zgodovinskem skladu.

3.8.1 Upravljanje z aplikacijami, ki shranjujejo stanje

Posebej je potrebno omeniti aplikacije, ki shranjujejo velike količine podatkov. Interno stanje je tako odvisno od količine informacij, ki je prisotna ob tistem stanju. Dober primer je anketna aplikacija, ki sestoji iz izpolnjevanja obrazcev, kjer je novo vprašanje odvisno od prejšnjega odgovora. V takih primerih je potrebno shraniti določeno količino informacij, ki aplikacijo še lahko postavi v prejšnje interno stanje. Rešitev je shranjevanje informacije ali nekakšnega kazalca do nje v žetonih. V en žeton tako lahko vpišemo več daljših nizev, ki nosijo informacijo, podobno kot pri uporabi zahtevkov GET. Za lažjo razlago si oglejmo potek med prehodi stanj v tabeli 3.1.

Žeton	Akcija
Page=start;data=null	Prehod na začetek ankete. Nimamo podatkov, če so, jih izbrišemo.
Page=form1;data=info	Prehod na prvi del ankete. Ohranimo le podatke o demografski informaciji uporabnika.
Page=form2;data=detail	Prehod na drugi del ankete. Ohranimo podatke o informaciji ter o dodatnih izpolnjenih poljih.
Page=end;data=all	Prehod na konec ankete. Izpišemo vse podatke ter pazimo, da jih ne pošljemo naprej še enkrat.

Tabela 3.1: Primer prehodov med stanji aplikacije.

Aplikacija torej take primere rešuje z uporabo programske logike. V zgornjem primeru tako navigacijo postavimo glede na *Page=<zeton>* in poiščemo

podatke, ki se odzivajo na ključ, podan v parametru *Data=<dataname>*, ki je lahko shranjeni v seji, lokalnem brskalnikovem pomnilniku ali pa v podatkovni bazi.

Poglavje 4

Komunikacija s strežnikom

Kot vse spletne aplikacije, je tudi tako, ki je narejena s tehnologijo GWT, potrebno povezovati s strežnikom. Za komunikacijo z njim imamo na voljo dva možna načina. Za delo s klici servletov Java lahko uporabimo ogrodje GWT-RPC, kjer sam GWT poskrbi za nižje nivojske postopke, kot je serializacija objektov Java. Lahko pa tudi uporabimo odjemalčeve razrede HTTP za gradnjo lastnih zahtevkov HTTP.

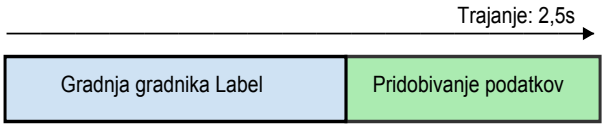
4.1 Koda na strežniku

Pri GWT imamo mehanizem RPC na osnovi servletov Java. Ta sistem prevede kodo v JavaScript (za delo na odjemalčevi strani) ter v operacijsko kodo Java za pravilno serializacijo objektov Java za prenašanje skozi omrežja. Za razliko od odjemalčeve se koda na strežniku ne prevede v JavaScript, zato lahko uporablja vse knjižnice Java (standardne in nestandardne).

4.2 Klici RPC

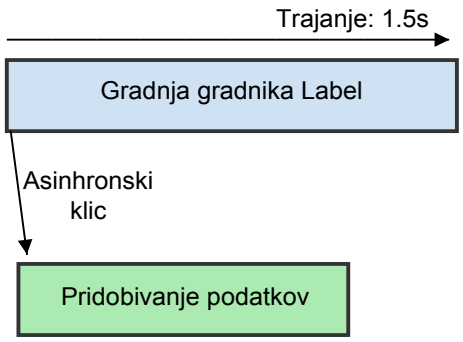
RPC je mehanizem za interakcijo med odjemalčevim ter strežniškim delom aplikacije, ki je včasih poimenovan tudi strežniški klic. Klici RPC so v osnovi klici AJAX. S tem več ni potreben klic na strežnik za celotno stran ob vsaki

spremembi, ampak le za del strani, ki ga potrebujemo. GWT RPC-sistem zelo poenostavi pošiljane objektov Java skozi omrežje.
Vsi klici RPC so asinhronskega načina. GWT RPC-sistem nam ne nudi možnosti klicev sinhronskega načina. Razlog je v načelu GWT izvajanja ne glede na odjemalčev brskalnik. Uporaba sinhronskih klicev namreč ni mogoča zaradi narave jezika JavaScript, ki ga večina brskalnikov izvaja kot eno samo nit. Uporaba sinhronskega bi tako zahtevala dodatno delo za programerja, ki bi moral spreminjati klice glede na uprabljeni brskalnik.
Prednost asinhronskega klica je očitna. Zaradi pomanjkanja možnosti večino izvajanja lahko z asinhronskimi klici dosežemo paralelno izvajanje, če hočemo na primer ustvariti en gradnik, ki bi vsebovali dolg niz, ki je shranjen v neki tabeli. S sinhronskim načinom bi najprej zgradili gradnik v HTML ter nato poslali klic na strežnik, ki vrne niz. Če bi gradnja gradnika, na primer *Label*, trajala npr. 1,5 sekunde, klic na strežnik pa še dodatno sekundo, bi za celotno opravilo potrebovalo 2,5 sekunde. To si lahko ogledamo na sliki 4.1.



Slika 4.1: Primer poteka sinhronskega klica.

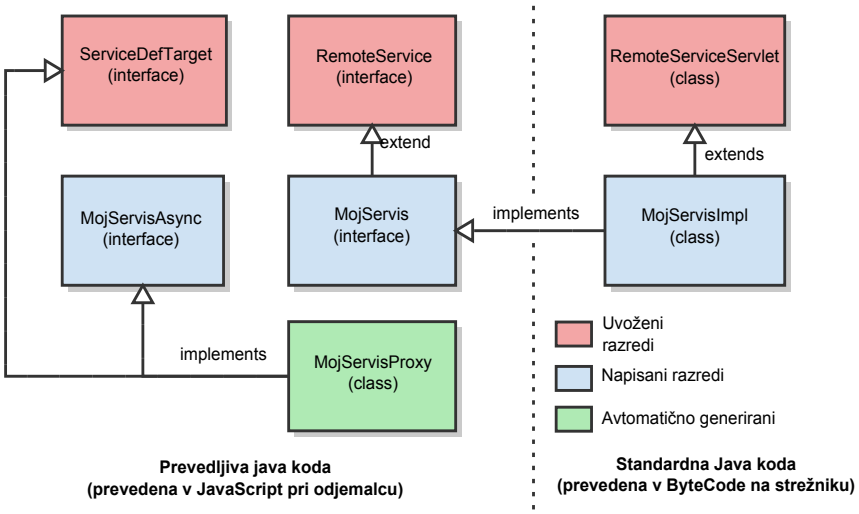
Najprej bi potekala gradnja gradnika, nato bi se začelo pridobivanje podatkov. Torej potrebovali bi 2,5 sekunde za celotno opravilo.
Isti primer ob uporabi asinhronskega klica (Slika 4.2) bi trajal 1,5 sekunde, kar je celoten čas potreben za gradnjo gradnika. Podatki se ob takem načinu pridobijo med časom same gradnje in ne vplivajo na celoten čas, če seveda se pridobivanje podatkov ne izvršuje dalj časa od same gradnje gradnika.



Slika 4.2: Primer poteka asinhronskega klica.

4.2.1 Uporaba klicev RPC

Definicija enega servisnega klica zahteva implementacijo več razredov v lastnoročno spisane razrede, poleg teh GWT ustvarijo še dodatne proxy razrede za pomoč uporabniku. Vse vmesnike in razrede lahko vidimo na spodnjem diagramu (Slika 4.3).



Slika 4.3: Diagram klica RPC.

4.3 Uporaba HTTP zahtevkov

Dela aplikacije GWT za odjemalčevo stran ni potrebno nujno povezati s strežnikom Java. Tako lahko uporabimo strežnike, ki nimajo podpore za platformo Java. Tako komunikacija s strežnikom preko servletov Java ni več mogoča. GWT v takem primeru nudi razrede HTTP, ki so nam v pomoč pri ustvarjanju zahtev HTTP. Gradnja takih zahtev HTTP je podobna kot pri drugih ogrodjih. Potrebno pa je paziti na dve zahtevi, ki jih GWT RPC-sistem odpravi sam.

Prva izhaja iz enonitnosti arhitekture JavaScript. Če med klicem, ki se izvršuje, pride do nepričakovane napake, se aplikacija "obesi"in ne more nadaljevati. Za odpravljanje take težave je potrebno dodati metodo, ki se odziva na tako napako, in tako dovoli nadaljevanje aplikacije.

Druga izhaja iz pravil SOP, ki ne dovoljuje pošiljanja zahtev strežniku, ki ni izvorni strežnik aplikacije. Če nimamo komunikacije preko Java Servlet, nimamo niti možnosti prenašanja objektov. Torej podatki se ne morejo prenašati kot objekti, ampak GWT nudi podporo za nize JSON. Dobljen niz prevedemo v objekt *JavaScriptObject* z uporabo metode JSNI (Izvorna koda 4.1).

```
public static native JavaScriptObject parseJson(String jsonStr)
    /*-{
return eval(jsonStr);
}-*/;
```

Izvorna koda 4.1: Metoda *JavaScriptObject* za razčlenitev objektov JSON.

Dobljeni objekt nato uporabljamo podobno kot odgovor na klasičeno zahtevo HTTP.

4.4 Serializacija objektov

Trenutna verzija GWT (2.5.0) podpira *java.io.Serializable* implementacijo serializacije objektov za prenašanje skozi mrežo. Pri starejših verzijah do verzije 1.4.0, so se objekti serializirali preko serializacije GWT z vmesnikom *isSerializable*. Razlike med java.io in vmesnikom GWT ni precejšnja, cilj obeh je namreč enak. Razlog, zakaj *java.io.Serializable* ni bil uporabljen, je ta, da sistem serializacije GWT namreč ni zmožen opravljati enakega dela kot standardni v vmesniku Java. Serializacijski mehanizem GWT je enostavnejši in ni zmožen serializirati objektov, kot je na primer ID serializacijska verzija.

Z GWT, verzije od 1.4 naprej, je za serializacijo možna tudi uporaba vmesnika java.io.Serializable, ampak pod enim pogojem. Vsi objekti, ki se bodo prenašali skozi klice RPC, morajo biti vsebovani v datoteki .gwt.rpc, ki je seznam vseh dovoljenih objektov za serializacijo. Datoteka mora biti naložena na aplikacijski strežnik mora biri skupaj z aplikacijo dosegljiva preko servleta *RemoteServiceServlet* preko klica *SerlvetContext.getResource()*.

Zaradi take implementacije ni mogoče uporabiti vseh objektov, ki so prisotni v okolju Java JRE. Uporaba objektov, kot je *Throwable*, ni mogoča. Odjemalčeva stran namreč nima zmogljivosti serializirati/deserializirati. Enostavnejše povedano, odjemalčeva stran je omejena na uporabo objektov, ki so lahko uporabljeni v okolju JavaScript, torej objekti tipa *String, Integer, Float*, ki imajo možnost serializiranja ter uporabe za pošiljanje preko klica RPC.

Poglavje 5

Prednosti in slabosti

GWT je arhitekturno zelo močna in predvsem uporabna knjižnica za izdelavo spletnih in mobilnih aplikacij. V naslednjem razdelku bom opisal pet prednosti ter pet slabosti, ki se po dolgoročno gledano ob razvijanju aplikacij po mojem mnenju pokažejo največkrat.

5.1 Prednosti

Najprej si oglejmo prednosti, ki nam jih ponuja GWT:

1. Prenešen koncept objektnega programiranja na odjemalčev del aplikacije
   Še posebej se slednja prednost pokaže, če je programer domač v okoljih Java AWT ali Swing. Krivulja učenja je zelo majhna. Tudi če programer sam nima izkušenj z razvojem takšnih aplikacij, mu bo znanje Jave pri delu s strežniško kodo aplikacij prišlo v veliko pomoč.

2. Front-end koda v Javascript
   JavaScript od zmeraj asociramo s hitrostjo in prenosljivostjo in prav tako GWT. S tem ko se Java koda prevede v JavaScript, nam to predstavlja kar dve prednosti. Prvič, dobimo vse kar je v JavaScriptu dobrega (najbolj hitrost in prenosljivost), in drugič, v JavaScriptu sploh

ne programiramo – razhroščevanje, ki nam nudi isti jezik kot Java, v JavaScriptu ni možno.

3. Razvijan iz strani Googla
   Projekt, ki je v razvoju giganta, kot je Google, ne bo kar tako padel v vodo. Od leta 2006, ko je bila izdana prva verzija, razvoj poteka neprekinjeno (zadnja verzija izdana 1. marca 2013). Poleg tega je to še ena najbolj intuitivnih dokumentacij, ki je ves čas ažurirana s primeri. Poleg razvijalcev je tu še velika skupnost, ki je v stalni pripravljenosti na straneh kot StackOverflow.

4. Obilica dodatnih odprtnokodnih knjižnic
   Ogromno dodatnih knjižnic, kot že omenjeni lib-svg-gwt in GwtQuery, ki nam še bolj olajšata razvoj ter doprineseta k večji funkcionalnosti novih aplikacij, ali Smart GWT, ki nam olajša delo z vizualnim delom aplikacij.

5. Testiranje
   Do umestitve aplikacije ni nikoli dokončno prevedena, kar pomeni, da je v celotnem razvijalnem delu aplikacija zagnana v JVM – razhroščevanje.

5.2 Slabosti

Oglejmo si še slabosti. Kot zanimivost bomo videli, da nekaj slabosti izhaja prav iz konceptov, ki nam doprinesejo prednosti ogrodja GWT:

1. Prevajanje kode
   Čeprav je res, da načeloma kode ni potrebno prevajati ob vsaki spremembi, se v praksi izkaže, da pa je potrebno velikokrat za pravilno testiranje storiti prav to. Čas, ki ga potrebuje prevajalnik, da kodo iz Jave spremeni v JavaScript, je največja težava razvoja GWT.

2. Oblikovanje aplikacij

   Ena najslabših nalog pri aplikaciji GWT je oblikovanje. Ogrodje že samo postavi veliko pravil CSS, ki jih težko obidemo. Pri tem je potrebno standardni proces oblikovanja prilagoditi tehnologiji. Večja težava je v tem, da je linija med programerjem ter oblikovalcem težko ločljiva, saj ju GWT združi v enega.

3. "Front-end koda"v Javascript

   Poleg vsega dobrega JavaScript prinese tudi veliko slabega. Ena večjih težav je ranljivost jezika, ki je delno rešljiva s raznimi dolgotrajnimi prijemi s kodo GWT. Tudi velika prednost, ki je prenosljivost med brskalniki, rada pokaže slabo stran: ni še 100

4. Krivulja učenja

   Brez izkušenj z Javo bo učenje te tehnologije zelo velik zalogaj. Zaradi relativne novosti, za katero razen dokumentacije še vedno ne obstaja veliko virov, iz katerih se je možno učiti.

5. Vzdrževanje kode

   Zaradi združevanja vizualnega dela z logiko se ob velikem obsegu kode ter klicev med strežniško in odjemalčevo stranjo lahko zelo enostavno izgubimo. Na začetku vsakega projekta je potrebno dodobra definirati pravilno strukturo, ki se jo je treba držati do konca.

Poglavje 6

Primerjava z drugimi tehnologijami

6.1 Primerjava s podobnimi Java spletnimi ogrodji

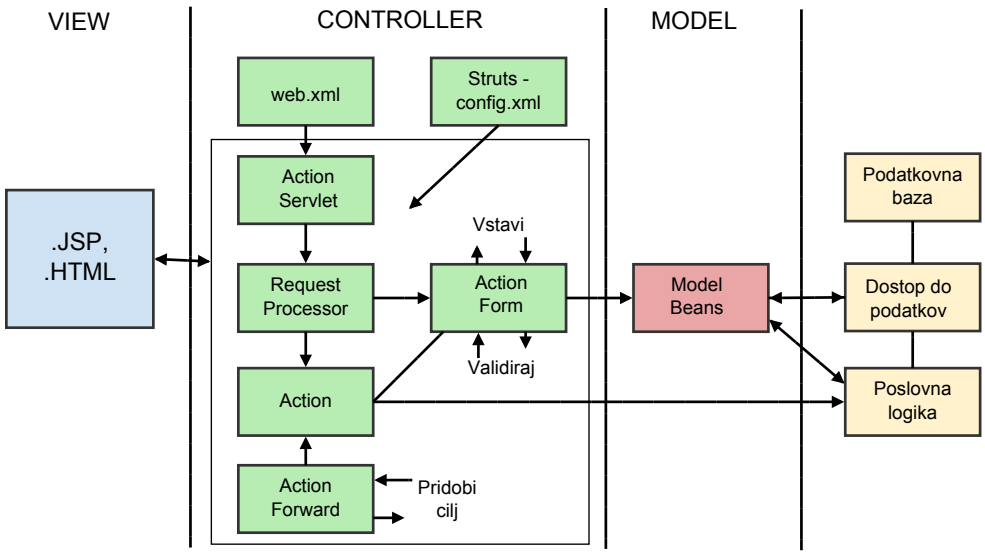
6.1.1 Primerjava s Struts

Struts je eno najstarejših odprtokodnih spletnih ogrodji za razvoj v Javi, ki ga je razvilo podjetje Apache Software Fondation. Samo ogrodje deluje na osnovi zahtevkov (ang. Request Based Framework). Vsaka aplikacija je razvita v čistem modelu MVC. Zahteva striktno ločevanje med strežniško in odjemalčevo programsko kodo.

6.1.1.1 Komunikacija s strežnikov

V osnovi Struts nima podpore za asinhronsko komunikacijo (z verzijo 2.2 ima delno podporo z Jquery vtičnikom), kar pomeni da vsa komunikacija poteka preko zahtevkov. Odjemalčeva zahteva poteka po modelu MVC. Najprej se zahteva imenovana Action pošlje upravljalcu dogodkov , ki izvede potrebno logiko, da na modelu pridobi potrebne podatke, ki se nato preko upravljalca dogodkov zapišejo nazaj na odjemalčev del apliakacije. Potek zahteve si

lahko ogledamo na spodnjem diagramu (Slika 6.1)



Slika 6.1: Struts življenski cikel HTTP zahteve v MVC modelu.

Takoj opazimo, da imajo s komunikacijo GWT skupno le to, da uporabljajo servlete Java.

6.1.1.2 Enostavnost uporabe

V primerjavi z GWT Struts ni kompleknso ogrodje, saj deluje na čistem modelu MVC v dokaj enostavni arhitekturi. Najbolj pomembno znanje je potek procesa zahteva/odgovor, ki pa je kompleksna operacija že za najbolj enostavne zahteve. Recimo, za samo eno operacijo za pridobitev podatkvo izpodatkovne baze je potrebno implementirati celotno logiko: akcije, preslikavo URL v XML datotekah, podatkovna zrna s konfiguracijo v datotekah XML in prikaz. Vsi razredi, konfiguracijske datoteke in prikaz pa morajo vsebovati točna imena resursov, sam prevajalnik pa nima možnosti avtomatičnega pregleda skladnosti med datotekami.

6.1.1.3 Razvijanje, podpora in skalabilnost

Za razvijanje v Struts je še zmeraj potrebno imeti dovolj znanja v HTML in JS . Struts je akcijsko in ne komponento ogrodje kot GWT ali JSF, vse komponente pa je potrebno ustvariti z uporabo značk HTML. Zaradi tega prototipno programiranje skorajda ni mogoče, saj je pred samim pravim delom potrebne veliko konfiguracije.
Podpora pri GWT in Struts ne bi smela povzročiti večjih težav, obe kodi sta razumljivi in jasno je definirano, kje naj se določena komponenta nahaja. Glede skalabilnosti pa pri Struts nimamo druge možnosti, kot da razširimo infrastukturni sistem, kjer je aplikacija zagnana, saj je celotna aplikacija na strežniškem delu aplikacije.

6.1.1.4 Grafični vmesnik

Struts nam ne nudi orodji za delo z grafičnim vmesnikom. Vmesnik je razvit s klasičnimi tehnologijami kot HTML, z dodatkom JavaScript ali Jquery. To nam nudi najbolj fleksiblno okolje glede vizualnosti, ampak podaljša čas razvoja.

6.1.1.5 Dokumentacija in skupnosti

Uradna dokumentacije ogrodja Struts je zelo nepraktična in nevzdrževana v primerjavi z dokumentacijo GWT. Po drugi strani pa je že samo zaradi starosti ogrodja veliko neuradne dokumentacije, nastale s strani zelo močne skupnosti [5] [16].

6.1.2 Primerjava s JSF

JavaServer Faces je javanska specifikacija, torej standardizirana knjižnica, prisotna v Java EE za razvoj spletnih aplikacij. Od JSF verzije 2 za razvijanje grafičnih gradnikov uporablja tehnologijo Facelets za privzeti sistem predlog. Podpira tudi druge sisteme, kot je JSP, ki je bil v verzijah 1.x standarden.

6.1.2.1 Komunikacija s strežnikom

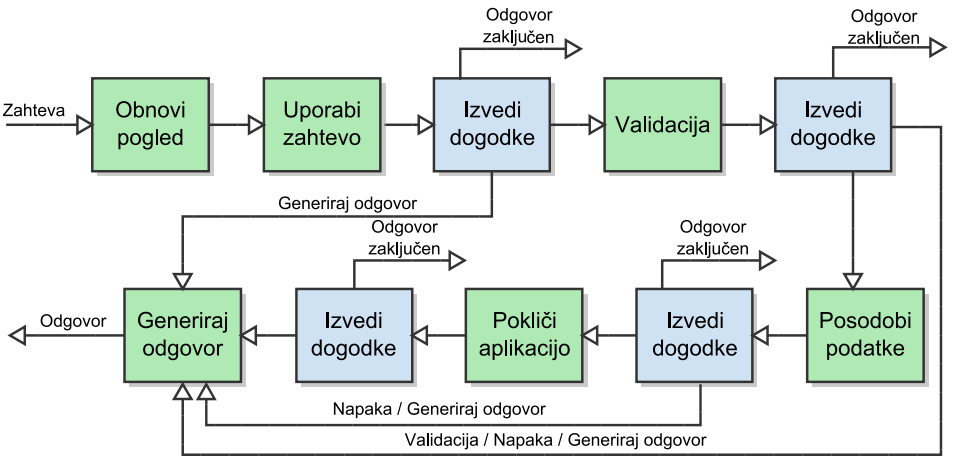
Vsako zahtevo procesira FacesServlet, ki sestoji iz več faz. Odjemalčeva zahteva najprej naloži potrebno prikazno predlogo, kjer generira drevo komponenente, nato procesira vse dogodke, ki so se pojavili, ter posodobi morebitne podatke v zrnih Java, ter nato zgenerira odgovor (tipično v HTML) ter ga pošlje odjemalcu. Novo stanje komponente je shranjeno ob koncu vsake zahteve in je ažurirano ob naslednji zahtevi.
JSF nudi tudi asinhronsko komunikacijo s pomočjo AJAX. Specifikacija JSF 2.0 nam ponuja že vgrajene komponente z podporo AJAX, kjer poteka cikel JSF samo za del gradnika, ki je specificiran v zahtevo Strežniški odgovor v takem primeru spremeni komponente na ravni hierarhije DOM.
V primerjavi z JSF pri GWT nimamo sinhronske komunikacije s strežnikom, ker je tudi ne potrebujemo (aplikacija v celoti na eni strani HTML). Asinhronska komunikacija pa je podobna, ker poteka s klici AJAX preko omrežja.

6.1.2.2 Enostavnost uporabe

JSF je zelo kompleksno ogrodje, kar je posledica standardiziranja v Java EE. Dobra stran tega je, da s Java EE imamo že vse kar potrebujemo, sam JSF ima dostop do celotnega JRE. Krivulja učenja je malo bolj strma kot pri GWT, zato je potrebno zelo dobro razumeti življenski cikel JSF (Slika 6.2).

6.1.2.3 Razvijanje, podpora in skalabilnost

JSF nam nudi tehnologijo predlog Facelets, ki nam že v osnovi nudi veliko uporabnih gradnikov. Z razliko od gradnikov GWT so tu definirani kot značke JSF. Gradniki so vezani na zrna Java za pridobivanje podatkov, ki hranijo aplikacijsko stanje gradnikov. Nimamo možnosti take stopnje manipulacije z njimi kot pri GWT, kjer so gradniki prikazani kot objekti, ki jih lahko poljubno prenašamo skozi celotno aplikacijo. Zmožnost generiranja hitrih prototipov aplikacije je v JSF na slabšem kot pri GWT. Za prototi-



Slika 6.2: Življenski cikel JSF generiranja komponent

pno aplikacijo je namreč potrebna skoraj enaka konfiguracija kot za celotno aplikacijo.
Skalabilnost je pri JSF boljša kot pri ogrodijh Struts, saj nam ponuja uporabo asinhornskih klicev, še zmeraj pa je velik del aplikacije na strežniški strani, za kar je ob veliki povečavi prometa potrebna nadgradnja infrastukture.

6.1.2.4 Grafični vmesnik

Pri JSF imamo neko srednjo pot med GWT in Struts, kjer pri GWT značk HTML skorajda ne uporabljamo, pri Struts nimamo izbire, pri JSF pa uporabljamo veliko grafičnih gradnikov, definiranih v predlogah Facelets. Obstaja obilica zunanjih knjižnic, ki jih lahko uporabljamo z razširitev osnovnih grafičnih gradnikov, kot sta RichFaces ali PrimeFaces, podobno kot pri GWT. Tako JSF kot tudiGWT uporablja CSS za spreminjanje stilov gradnikov, kjer same knjižnice že vsebujejo obilico uporabnih tem.

6.1.2.5 Dokumentacija in skupnosti

Zaradi prisotnosti JSF v okolju Java EE je dokumentacije ogromno. Na trenutke pa je zelo neberljiva in pomankljiva s praktičnimi primeri. To pa popravi skupnost, ki sledi tehnologiji JSF, ki je izmed vseh treh najbolj razvita [5] [15].

6.2 Primerjava s JQuery

Obe tehnologiji omogočata lažje razvijanje in podporo spletnih aplikacij, ki so zgrajene v jeziku Javascript. Končen rezultat razvoja v obeh jezikih je torej isti – spletna aplikacija JavaScript, zato je primerjava obeh orodji za razvoj uporabniška vmesnika smiselna. Jquery ni namenjen razvoju celotnega okolja aplikacije, ampak samo za del aplikacije, ki je namenjen odjemalcu.

6.2.0.6 Komunikacija s strežnikom

JQuery nam ponuja metode AJAX za komunikacijo s strežnikom, kar je sila podobno GWT RPC-sistemu za komunikacijo s strežnikom, torej imamo asinhronsko možnost komunikacije.

6.2.0.7 Enostavnost uporabe

Če primerjamo jQuery s različnimi Java ogrodji glede njegove kompleksnosti, kar vključuje tudi GWT, skorajda ni primerjave. JQuery je najmanj kompleksno orodje od primerjanih. Na željeni dokument HTML preko *<script>* oznake jQuery le dodamo izvorno kodo in že lahko uporabljamo vse funckije ogrodja brez dodatnih konfiguracij. JQuery tudi nima potrebe po prevajanju.

6.2.0.8 Razvijanje, podpora in skalabilnost

Razvoj in programiranje sta med jeziki zelo različena. Jquery je zgrajen nad jezikom JavaScript in nam torej ponuja vse njegove zmogljivosti in pomankljivosti. Sam Jquery tudi ni mišljen za razvoj večjih kompleksnih spletnih aplikacij, kjer je celotna vsebina postavljena na samo en dokument HTML za razliko od GWT, ki je bil razvit s tem ciljem. Pravo moč Jquery pokaže pri tradicionalnih spletnih straneh, kjer imamo več HTML strani z različno vsebino. Dodajanje elementov jQuery je namreč zelo hitro ter učinkovito. Pri takih projektih je GWT skorajda nesmiselna izbira, ki bi pripeljala do uporabe obilice kode za najmanjše stvari. Razvijanje v Jquery je lahko zelo mukotrpno, nimamo pravega objektno orientiranega programiranja, zato je pomanjkanje dobrega razhroščevalnika ob razvoju večjih projektov izrazito. Tudi pomanjkanje obstoja resnično dobrega razvojnega okolja vpliva razvoj in še posebej na testiranje jQuery kode. Pri GWT pa imamo vtičnike za razvojno okolje Eclipse, ki nam ponuja celotno podporo pri pisanju in razhroščevanju odjemalčeva kode, kar dosežemo s prevajanjem v razvojno verzijo aplikacije, kjer je odjemalčeva koda zagnana v JVM.

6.2.0.9 Grafični vmesnik

Cilj uporabe GWT je čim manjša uporaba kode HTML v razvojnem ciklu. Imamo namreč gradnike za tudi najenostavnejše elemente HTML, kot je na primer * * element. Pri jQuery je integracija med kodama HTML in jQuery skorajda neizogibna, čeprav bi lahko tudi enostavnejše elemente dinamično generirali preko kode, toda to bi bilo nesmiselno.
Jquery nam že v osnovi ponuja veliko več gradnikov kot osnovni GWT, ki so boljše oblikovani in dodelani. Dobra za gradnike GWT pa je možnost enostavne razširitve in dodajanje novih gradnikov. Poleg tega pa obstaja tudi obilica knjižnic, kot sta GWT-EXT ali SmartGWT, ki nam ponujajo že pripravljene grafične gradnike za takojšnjo rabo.
Obe tehnologiji uporabljata stilne predloge CSS, ki jih enostavno dodamo k projektu. Pri Jquery jih lahko brez večjih težav spreminjamo in že obstoje-

čim dodajamo nove. Pri GWT imamo že definirane privzete predloge CSS, ki vsakemu gradniku dodajo stil, ki ga je za lastne spremembe potrebno povoziti.
Velika pomanjklivost pri jQuery v primerjavi z GWT je pomanjkanje optimizacije kode, saj mora odjemalec naložiti celotno izvorno kodo, šele nato pa lahko generira komponente HTML iz logike JavaScript.

6.2.0.10 Dokumentacija in skupnosti

Obe tehnologiji imata zelo dobro dokumentacijo. Razvijata se zelo hitro, s tem pa se razvija tudi dokumetacija, ki je polna primerov in zelo enostavna za razumevanje. Skupnosti so zelo razširjene in tudi uradno podprte s strani razvojnih skupin obeh tehnologij.

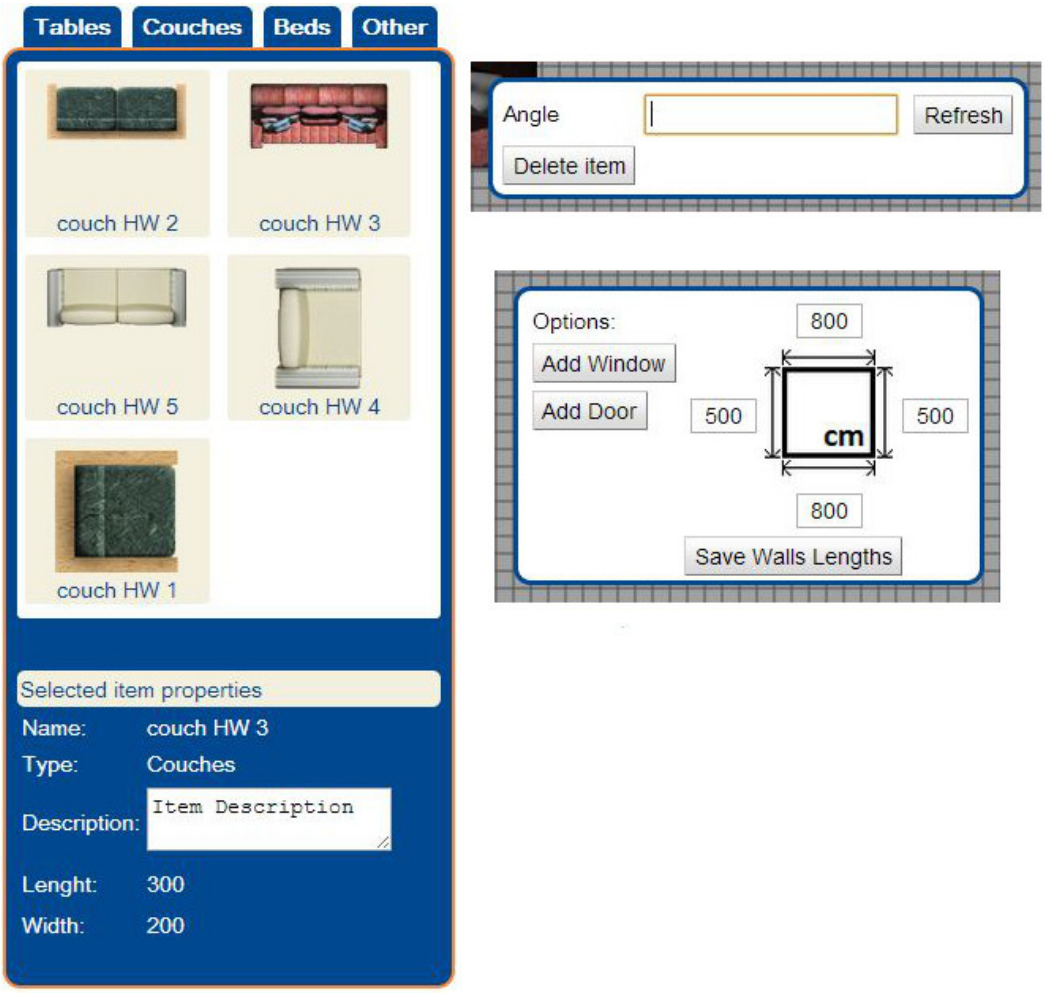
Poglavje 7

Praktični primer aplikacije GWT

7.1 FloorPlaner

Za prikaz praktičnega primera je bila razvita aplikacija za za lažje opremljanje prostorov. Slednja aplikacija je bila izbrana zato, ker lahko pokaže vse moči ogrodja GWT, tako gradnjo vmesnika kot aplikacijsko logiko na strežniku. Aplikacija omogoča prijavljenemu uporabniku iskanje pohištva ter njegovo postavitev v navideznem dvodimeonzionalnem prostoru po principu "primi in spusti".
Aplikacija je v celoti razvita z uporabo ogrodja GWT, z dodatkom tehnologij JPA in EJB Java.

Sama aplikacija sestoji iz dveh delov: izbirni meni (Slika 7.1), ter risalne plošča (Slika 7.2). V izbirnem meniju lahko izberemo željeni predmet, ter pregledamo njegove lastnosti. Izbrani predmet nato lahko poljubno premikamo ter rotiramo po risalni plošči. Ob desnem kliku se nam na risalni plošči prikažejo konfiguracijski meniji za spreminjanje lastnosti izbranega predmeta ter risalne plošče.



Slika 7.1: Izbirni meni ter konfiguracijski pojavni okni.

Slika 7.2: Risalna plošča.

7.2 Uporabljene tehnologije

7.2.1 Programski jezik Java

Java je objektno programski jezik, iz katerega je zgrajena knjižnica GWT. Največja prednost jezika je prenosljivost, saj lahko deluje na vseh virtualnih napravah Java, ki nato ne glede na računalniško arhitekturo izvajajo razrede Java. Trenutna verzija je 7, ki je bila tudi uporabljena za namene izdelave aplikacije [7].

7.2.2 Spletni strežnik JBoss EAP

Odprtokodni spletni strežnik, ki streže spletne strani odjemalcem in je bil uporabljen za postavitev praktičnega primera. Aplikacijski spletni strežnik Jboss nudi podporo za podporo Java EE (Enterprise Edition), kar nam nudi možnost uporabe JPA (Java Persistance API) in EJB (Enterprise JavaBeans) za lažje delo z objekti za branje in shranjevanje v bazo [9].

7.2.3 Razvojno okolje Eclipse

Je odprtno kodno razvojno okolje, napisano večinoma v Javi, zaradi tega tudi najbolj prijazno za razvoj v javanskem okolju. Eclipse ima na voljo tudi široko paleto dodatkov (ti. plug-in). Za potrebe knjižnice GWT je bil uporabljen Google plugin, ki nam omogoča izdelavo in prevajanje aplikacij GWT [10].

7.2.4 Podatkovna baza Postgres

Odprtno kodni objektno-relacijski sistem za upravljanje s podatkovnimi bazami. Deluje na vseh razširjenih operacijskih sistemih, kateri implementira večino SQL:2008 standardov. Zadovoljuje predvsem potrebe po enostavnosti uporabe, standarnem naboru spremenljivk ter hitrostjo izvajanja zahtev po podatkovni bazi [8].

7.2.5 SquirrelSQL

Preprosto odprtnokodno orodje, ki omogča administracijo vseh vrst podatkovnih baz. Deluje preko gonilnikov JDBC,s katerim se povežemo na željeno bazo. V tem primeru je bil uporabljen postgreSQL JDBC-gonilnik 3. Generacija verzije 9.2. [11].

7.2.6 SVG

SVG je označevalni jezik za definiranje dvodimenzionalne grafične aplikacije in slike, ter povezane skripte za manipulacijo grafičnih objektov.
SVG dovoljuje tri tipe grafičnih objektov: vektorsko grafiko, rastersko grafiko in ttekst. Grafični objekti kot rasterske slike PNG in JPEG so lahko grupiranje, preoblikovanje in kompozicija v prej oblikovanje objekte [13].

7.2.6.1 Vektorska grafika

Je uporaba geometričnih primitivnih oblik, kot so pike, črte, vijuge, oblike in poligoni, , za prikaz slik in grafike na računalniku na osnovi matematičnih enačb. Vektorska grafika bazira na vektorjih, ki se izrisujejo po določenih kontrolnih točkah. Vsaka točka ima pozicijo na oseh delovnega območja x in y. Vsaka točka nosi tudi neko informacijo kot sta: svoja lokacija na delovnem območje ter smer vektorja, ki nato definira pot (ang. track). Vsaki poti lahko definiramo barvo, obliko, širino, in polnilo. Vektorska grafika prav zaradi svoje strukture nima takih problemov kot npr. rasterska [12].

7.3 Razvijanje grafičnega vmesnika

Grafični vmesnik je vceloti razvit z uporabo tehnologije GWT z dodatkom dveh knjižnic:

- GwtQuery, ki doda funkcionalnosti jQuery za direktno uporabo v Java razredih.
  Knjižnico vežemo v projekt GWT z vpisom vrstice
  <inherits name='com.google.gwt.query.Query'/> v datoteko gwtmodul.

  Slednje vključi v projekt GWT razred *$*, kateri upravlja s jQuery funkcijami.Primer, ki doda drsni efekt gradniku z razredom CSS *login* (Izvorna koda 7.1) [14].

```
$(".login").show();Isti primer ob uporabi asinhronskega
    klica, pa je ponazorjen s sliko 4.2.
$(".login").animate("top:'100px'", 350);
```

Izvorna koda 7.1: Primer uporabe knjižnjice GwtQuery.

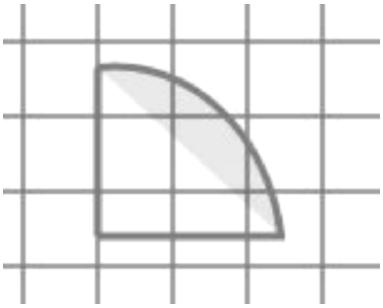
- Lib-gwt-svg: ki doda funckionalosti risanja vektorske grafike.V knjižnjici razpolagamo z metodami za definicijo oznake SVG, ki jih nato lahko brskalnik interpretira. Deluje v vseh ključnih brskalnikih. Knjižnico vežemo na projekt analogno kot knjižnico GwtQuery [13].

  Knjižnjica je bila uporabljena v aplikaciji za generacijo vizualnih elementov. Spodnji primer (Izvorna koda 7.2) prikazuje generacijo vrat v ptičji perspektivi (Slika 7.3).

```
public OMElement createDoor(Door d){
        OMElement layer=creteLayer(svg, 0, 0);
        OMElement line=doc.createSVGLineElement(0, 45, 50,
            45);
        OMElement line2=doc.createSVGLineElement(0, 0, 0,
            45);
        layer.setAttribute("style", "stroke:␣grey;␣stroke-
            width:2;fill-opacity:0.1;");
        OMSVGPathElement curve = doc.createSVGPathElement();
        curve.setAttribute("d","M␣0␣0␣a45␣53␣1␣0␣1␣49␣45");

        layer.appendChild(line);
        layer.appendChild(line2);
        layer.appendChild(curve);
        layer.setAttribute("x",""+ d.getX());
        layer.setAttribute("y",""+ d.getY());
        return layer;
}
```

Izvorna koda 7.2: Generacija vektorske risbe z uporabo knjižnice Lib-gwt-svg.



Slika 7.3: Rezultat izrisa viden v brskalniku

7.3.1 Gradnja novega grafičnega gradnika - LabeledImage

LabeledImage omogoča združitev slike ter oznako v enem gradniku. Torej ni več potrebno vključiti več gradnikov GWT v enega, ampak se direktno zgradi samo eden.
Gradnik je bil narejen z razširitvijo razreda Widget (Izvorna koda 7.3), ki nam nudi možnost manipulacije z hierarhijo DOM.

Potek kreacije poteka v več fazah:

1. kreiranje HTML elemetna preko metode *Document.get().createElement("oznaka HTML elementa")*,
2. opcijsko dodajanje atributov z metodo *element.setAttribute("oznaka", "vrednost")*,
3. postavitev elementa na hierarhijo DOM, uporaba metode *setElement(Element)*.

.

```
public class LabeledImage extends Widget implements
    HasClickHandlers{
        public LabeledImage(String url, String labelText, int
            imageHeight, int imageWidth) {
                //Generiranje potrebnih elmentov
                Element element = Document.get().createElement("
                    div");
                Element image = Document.get().createElement("
                    img");
                Element label = Document.get().createElement("
                    label");
                //Dodajanje atributov ter postavitev na glavni
                    element
                image.setAttribute("width", String.valueOf(
                    imageWidth));
                element.appendChild(image);
                //Postavitev elementa na DOM strukturo
                setElement(element);
        }

        @Override
        public HandlerRegistration addClickHandler(ClickHandler
            handler) {
                return this.addDomHandler(handler, ClickEvent.
                    getType());
        }
}
```

Izvorna koda 7.3: Gradnja novega grafičnega gradnika LabeledImage.

Nov gradnik sprejme 4 parametre:

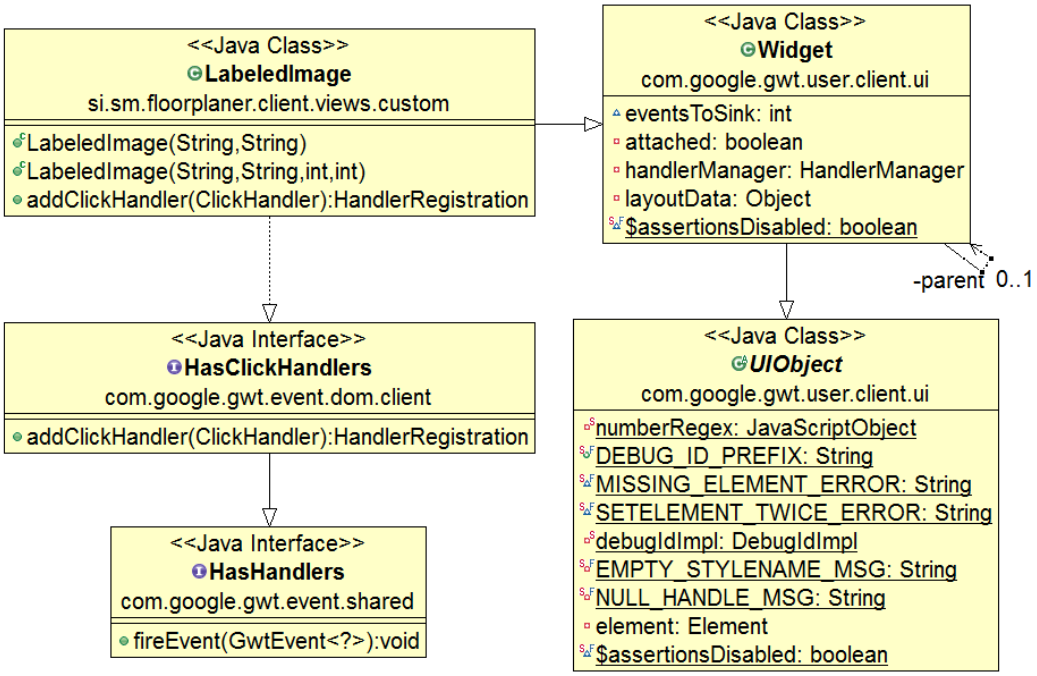
- *url*: niz, ki predstavlja lokacijo slike,
- *labelText*: niz, ki definira tekst pod sliko,
- *imageHeight*: višina slike,
- *imageWidth*: širina slike.

Za potrebe naše aplikacije potrebujemo možnost upravljanja z dogodki, za te potrebe implementiramo razred *HasClickHandler*, ki nam nudi registracijo dogodkov. Na aplikaciji nato vežemo dogodek pritiska na levi gumb miške z uporabo metode *addClickHandlers()* (Izvorna koda 7.4).

```
LabeledImage l = new LabeledImage("http://www.slika.com/slika.
    jpg", "Primer slike", 60, 60);

l.addClickHandler(new ClickHandler() {
        @Override
        public void onClick(ClickEvent event) {
                doStuff();
        }
});
```

Izvorna koda 7.4: Uporaba grafičnega gradnika LabeledImage.



Slika 7.4: UML razredni diagram gradnika LabeledImage.

7.3.2 Uporaba razreda EntryPoint

Za vsako aplikacijo GWT je potrebno definirati en razred, ki implementira *EntryPoint* (Izvorna koda 7.5). Ta razred predstavlja razred *Main* pri uporabi čistega okolja Java. Aplikacija zagon se prične v njem.

```
public class FloorPlaner implements EntryPoint{
        @Override
        public void onModuleLoad() {
                MainApp main= new MainApp();
                RootPanel.get().add(main);
        }
}
```

Izvorna koda 7.5: Primer implementacije EntryPoint razreda.

Najprej inicializiramo najvišji razred v naši hierarhiji, ki vsebuje celotni aplikacijski grafični vmesnik. Z uporabo metode *RootPanel.get()* vežemo celotno aplikacijo na gostujočo stran.

7.4 Aplikacijska logika

7.4.1 Uporaba Singleton metode

Metoda singleton (Izvorna koda 7.6) nam omogoča, da lahko brez potrebe redundance instanc razreda dostopamo do ključnih podatkov skozi celotno aplikacijo. V našem primeru je bila uporabljena na razredu, ki vsebuje trenutno stanje v aplikaciji, kar je ključno za izris.

```
public class SestavManager implements HasValueChangeHandlers<
    Sestav>{
        static SestavManager instance= new SestavManager();
        private Sestav sestav;
        public static SestavManager getInstance() {
                return instance;
        }
}
```

Izvorna koda 7.6: Definicija singleton metode.

Razred implementira vmesnik *HasValueChangeHandlers*, ki nam nudi registracijo za upravljanje dogodkov. Tako ob klicu metode *setSestav(Sestav)* že avtomatsko obvesti vse poslušalce, vezane na instanco razreda SestavManager z uporabo metode *addValueChangeHandler*. S takim načinom lahko implementiramo tudi zgodovino stanj aplikacije, posamezna stanja namreč lahko shranjujemo v sezname, ter jih po potrebo obnovimo – na primer, enostavna funkcija razveljavitve dogodka.

7.4.2 Primer klica RPC

Potrebno je ustvariti tri vmesnike ali razrede (Slika 7.5), ki bodo razširljali funkcionalnost, ki jo nudeno GWT RPC-infrastukture:

- Sinhronski vmesnik (Izvorna koda 7.7), ki razširja RemoteService vmesnik (GWT-RPC), kjer so definirane vse metode, ki jih bomo uporabili.

```
@RemoteServiceRelativePath("loginService")
public interface LoginService extends RemoteService {

        List<LoginProperty> getLogin();
}
```

Izvorna koda 7.7: Sinhronski vmesnik.

- Nato potrebujemo na strežniški strani implementacijo našega sinhronskega vmesnika (Izvorna koda 7.8), ki mora vsebovati vse definirane metode v sinhronskem vmesniku. Slednji še razširja *RemoteServiceServlet* vmesnik, ki nam ponuja možnosti servleta Java.

```
@WebServlet("/floorplaner/loginService")
public class LoginServiceImpl
                extends RemoteServiceServlet implements
                    LoginService {
public List<LoginProperty> getLogin() {
        List<LoginProperty> lpList=new ArrayList<
            LoginProperty>();
        for(Person p:loginManager.getAllPersons()){
                        .   .   .
        }
        return lpList;
}
}
```

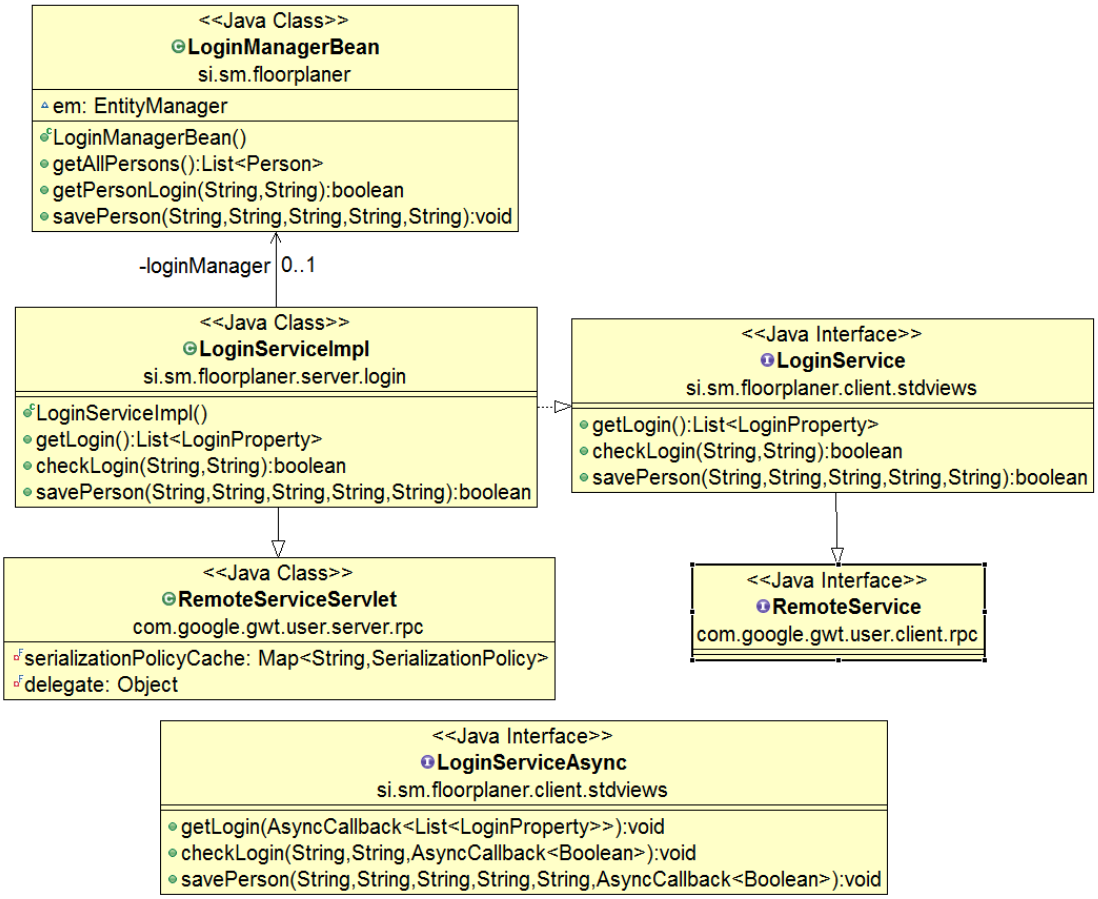
Izvorna koda 7.8: Implementacija sinhronskega vmesnika.

- Za uporabo slednjega servisnega klica potrebujemo še asinhronski del na odjemalčevi strani (Izvorna koda 7.9).

```
public interface LoginServiceAsync {
        void getLogin(AsyncCallback<List<LoginProperty>>
            handler);
}
```

Izvorna koda 7.9: Asinhronski vmesnik.

Ko imamo celoten klic uspešno definiran, ga lahko preprosto kličemo na odjemalčevi strani aplikacije, tako da najprej z metodo *GWT.create(LoginService.class)* dobimo objekt, ki je naš asinhornski vmesnik, ki vsebuje vse metode, definirane v njem.



Slika 7.5: UML razredni diagram klica RPC za prijavo v aplikacijo.

Poglavje 8

Sklep in ugotovitve

Namen diplomske naloge je bil preučiti ogrodje GWT z vidika uporabnosti za razvijanje spletnih aplikacij ter njegove prednosti in slabosti in posledično odgovoriti na vprašanje, zakaj uporabljati GWT.
Za nekoga, ki je mu je okolje Java domače, to sploh ni vprašanje, ampak dejstvo. Krivulja učenja je skorajda ničla. Koncept objektnega programiranja je uporabljen v vseh vidikih ogrodja GWT, na značke HTML pa lahko skoraj pozabimo. Zagon spletnih aplikacij je hiter ter opimiziran za prikaz s strani glavnega aduta, prevajalnika GWT. Strežniška ter odjemalčeva programska koda je pisana v jezikih Java, kar pomeni, da razvijalcu ni potrebno pisati v različnih programskih jezikih. Torej lahko sklepamo, da je za razvijalca v programskem jeziku Java, ki hoče razviti spletno aplikacijo, GWT najbolj naravna izbira.
Slabšo plat prikaže GWT pri razvoju samega izgleda aplikacije. V njem kot v večini ogrodji na osnovi komponent nimamo več fleksibilnosti, ponujene v čistem okolju HTML. Za določene komponente bo spreminjati izgled prava mora ali pa bo potrebno graditi nove komponente, kar se prikaže na razvojnem času. Dodatne težave nastanejo ob potrebi obrazcev na spletni aplikaciji, še bolj pa se zaplete, če je stanje podatkov odvisno od prejšnjega stanja, pri čemer je za pravilen prikaz potrebne veliko dodatne logike. Seveda je GWT še zmeraj v razvoju in smo komaj pri različici 2.5. Že sam problem

komponent, ki sestojijo iz vnosnih obrazcev, je bil rešen z uvedbo sistema *UiBinder*, ki pa je uporabo kode HTML v aplikaciji prinesel nazaj.
Med izdelavo aplikacije se je pokazalo, kako enostavna je integracija GWT z dodatnimi tehnologijami in sožitje z njmi. V aplikaciji sem tako na odjemalčevem delu uporabljal JQuery ter vtičnik SVG, na strežniškem delu pa EJB in JPA. Za razvijanje take vrste aplikacij se je GWT izkazal kot učinkovito in fleksibilno orodje, manj priporočen pa je za gradnjo tradicionalnih spletnih strani.

Literatura

[1] Uradna dokumentacija GWT projekta. Dostopno na:
http://www.gwtproject.org/doc/latest/DevGuide.html

[2] Primerjava spletnih JVM ogrodji. Dostopno na:
https://spreadsheets.google.com/pub?key=0AtkkDCT2WDMXdC1HOEtnUHpCejJMbUhGeGJWUmh

[3] S, Neethling, "Understanding the GWT compiler", dostopno na:
http://css.dzone.com/news/understanding-gwt-compiler

[4] G. Marwaha, "GWT – Pros and Cons", dostopno na:
http://www.javacodegeeks.com/2012/01/gwt-pros-and-cons.html

[5] RebelLabs, The Curious Coder's Java Web Framework Comparison,
dostopno na:
http://zeroturnaround.com/rebellabs/the-curious-coders-java-web-frameworks-compa

[6] "Write once, run anywhere", dostopno na:
http://en.wikipedia.org/wiki/Write_once,_run_anywhere

[7] Wikipedia: Java programski jezik, dostopno na:
http://en.wikipedia.org/wiki/Java_%28programming_language%29

[8] Postgres podatkovna baza, dostopno na:
http://www.postgresql.org/about/

[9] JBoss aplikacijski strežnik, dostopno na:
http://en.wikipedia.org/wiki/JBoss_Enterprise_Application_Platform

[10] Razvojno okolje Eclipse, dostopno na:
http://en.wikipedia.org/wiki/Eclipse_%28software%29

[11] Orodje SquirrelSQL, dostopno na:
http://squirrel-sql.sourceforge.net/#overview

[12] Wikipedia: Vektorska grafika , dostopno na:
http://en.wikipedia.org/wiki/Scalable_Vector_Graphics

[13] Vectomatic svg knjižnjica za GWT , dostopno na:
www.vectomatic.org/libs/lib-gwt-svg

[14] GwtQuery: JQuery klon za uporabo v GWT, dostopno na:
http://code.google.com/p/gwtquery/

[15] Dokumentacija Java EE 5: življenski cikel JSF, dostopno na:
http://docs.oracle.com/javaee/5/tutorial/doc/bnaqq.html

[16] Apache Struts, dostopno na:
http://struts.apache.org/

[17] GWT Dokumentacija: Deffered binding, dostopno na:
http://www.gwtproject.org/doc/latest/FAQ_Client.html#Deferred_Binding