VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

KOMPONENTA JSF PRO VYKRESLOVÁNÍ GRAFŮ S PODPOROU AJAX

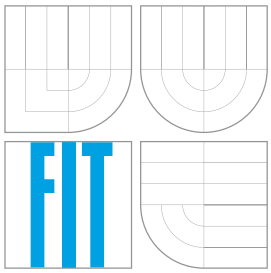
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

LUKÁŠ MACKO

BRNO 2013

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

KOMPONENTA JSF PRO VYKRESLOVÁNÍ GRAFŮ S PODPOROU AJAX

AJAX-ENABLED JSF COMPONENT TO WORK WITH CHART-PLOTTING LIBRARIES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — LUKÁŠ MACKO
AUTHOR

VEDOUCÍ PRÁCE — Ing. RADEK BURGET, Ph.D.
SUPERVISOR

BRNO 2013

Abstrakt

Cílem této práce bylo vytvořit komponentu pro framework Java Server Faces vykreslující grafy. Komponenta představuje abstrakci nad javascriptovou knihovnou. Umožňuje vykreslovat čarové, sloupcové a koláčové grafy. Rovněž podporuje obsluhu událostí u klienta i na straně serveru. Práce popisuje návrh a implementaci s použitím Component Development Kit.

Abstract

The aim of this thesis is to create a JavaServer Faces componet for chart plotting. The component is an abstraction of javascript library. It allows to plot line, bar and pie charts. It supports client-side and server-side event handling. The thesis describes design and implementation using the Component Development Kit.

Klíčová slova

komponenta pro vykreslování grafů, Component Development Kit, JSF, JavaServer Faces

Keywords

chart plotting component, Component Development Kit, JSF, JavaServer Faces

Citace

Lukáš Macko: Komponenta JSF pro vykreslování grafů s podporou AJAX, bakalářská práce, Brno, FIT VUT v Brně, 2013

Komponenta JSF pro vykreslování grafů s podporou AJAX

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Radka Burgeta, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Lukáš Macko
14. května 2013

Poděkování

Chcel by som poďakovať Ing. Radkovi Burgetovi, Ph.D. za vedenie mojej práce a technickému konzultantovi Bc. Lukášovi Fryčovi za cenné rady, pomoc a usmernenia pri jej vypracovaní. V neposlednom rade ďakujem rodičom a ostatným, ktorí ma podporovali počas štúdia.

Obsah

Kapitola 1

Úvod

V súčasnosti sa pre tvorbu rozsiahlych aplikácii na internete, ktoré vyžadujú podporu transakčného spracovania, rýchlosť a bezpečnosť, využíva trojvrstvový model. Komplexné riešenie pre aplikácie tohto druhu poskytuje platforma Java EE. Táto platforma zahŕňa niekoľko technológií, medzi inými aj technológiu JavaServer Faces(JSF) pre tvorbu užívateľského rozhrania.

Cieľom tejto práce bolo vytvorenie komponentu pre JSF vykresľujúceho grafy s ohľadom na rozšíriteľnosť, konfigurovateľnosť a podporou AJAX. V súvislosti so svojim cieľom, táto práca oboznamuje čitateľa s možnosťami tvorby grafov na webe, technológiou JSF a možnosťami tvorby vlastných komponentov. Na práci som spolupracoval s firmou RedHat a vyvíjaný komponent bol určený pre knižnicu RichFaces, ktorá zatiaľ komponent tohto druhu neobsahuje. Proces vývoja komponentu a súvisiace teoretické poznatky sú obsahom tejto technickej správy. Text je členený na kapitoly.

Kapitola 2 predstavuje použité technológie. Začína uvedením platformy Java EE a vysvetlením akú úlohu v rámci nej zastáva JSF. Následne predstavuje framework JSF, jeho štruktúru a charakteristické črty. Taktiež je tu uvedená technológia Maven, použitá pre správu projektu, a Arquillian, zabezpečujúci automatizované testovanie.

Ďalšia kapitola 3 sa zameriava na možnosti tvorby komponentov v JSF. Začína tvorbou kompozitných komponentov, implementovaných v jazyku XML. Pokračuje tvorbou vlastných komponentov v jazyku Java. V poslednej časti ukazuje ako je možné proces vývoja týchto komponentov uľahčiť pomocou sady nástrojov Component Development Kit.

Kapitola 4 predstavuje rôzne spôsoby vykresľovania grafov na webe. Dôraz je kladený na porovnanie javascriptových riešení, z ktorých bude zvolené riešenie použité vo výslednom komponente. Hodnotiaciami kritériami boli možnosti poskytované knižnicou, korektné zobrazenie v rôznych webových prehliadačoch, stav vývoja a podpora komunity.

Návrhu komponentu a odôvodneniu výberu javascriptového riešenia sa venuje kapitola 5. Vymedzuje vlastnosti zvolené pre implementáciu. Oboznamuje čitateľa o špecifikách rôznych druhov grafov, o návrhu dátového modelu a udalostiach, ktorých obsluhu bude komponent umožňovať.

Na kapitolu zaoberajúcu sa návrhom nadväzuje kapitola 6 popisujúca zaujímavé časti implementácie komponentu. Taktiež predstavuje aplikáciu, ktorá demonštruje použitie komponentu.

Kapitola 7 sa zaoberá postupmi použitými pri testovaní vytvoreného komponentu. V poslednej kapitole 8 sú zhrnuté výsledky dosiahnuté v tejto práci. Taktiež je tu uvedené plánované pokračovanie projektu a možné rozšírenia do budúcnosti.

Kapitola 2

Použité technológie

Táto kapitola sa venuje technológiám použitým v práci. Začína predstavením platformy pre tvorbu podnikových aplikácii v jazyku java – Java EE 6. Predstavuje trojvrstvový model pre tvorbu aplikácii v tomto rámci, úlohu kontajnera a možnosti konfigurácie aplikácií. Následne sa venuje konkrétne frameworku JavaServer Faces. Ukazuje namapovanie tejto technológie na návrhový vzor Model-View-Controller(MVC). Ďalej sú stručne predstavené pomocné technológie. Konkrétne sa jedná o technológiu Maven, ktorá bola využitá na správu projektu a Arquillian pre tvorbu automatizovaných testov. Táto kapitola vychádza zo zdrojov [15] [12] [14].

2.1 JAVA EE

Java EE je špecifikácia, ktorá zastrešuje špecifikácie viacerých technológii a definuje platformu pre vývoj distribuovaných podnikových aplikácii. Jedná sa zvyčajne o softvér, pre rôzne podniky, ich zamestnancov, dodávateľov alebo zákazníkov. Pre aplikácie tohto druhu je charakteristická ich rozsiahlosť, komplexnosť, dôraz na spoľahlivosť, škálovateľnosť a bezpečnosť. Z dôvodu prehľadnejšieho návrhu, implementácie a jednoduchšej údržby sú tieto aplikácie rozdelené do viacerých vrstiev. Jednotlivé vrstvy sú označované ako klientská, webová, biznis vrstva a dátová vrstva. Aj napriek tomu, že aplikácia môže obsahovať štyri vrstvy, Java EE aplikácie sú zvyčajne považované za trojvrstvové ako uvádza [15]. Dôvodom je, že sú distribuované medzi tri miesta: klientské zariadenia, Java EE server a databázový server.

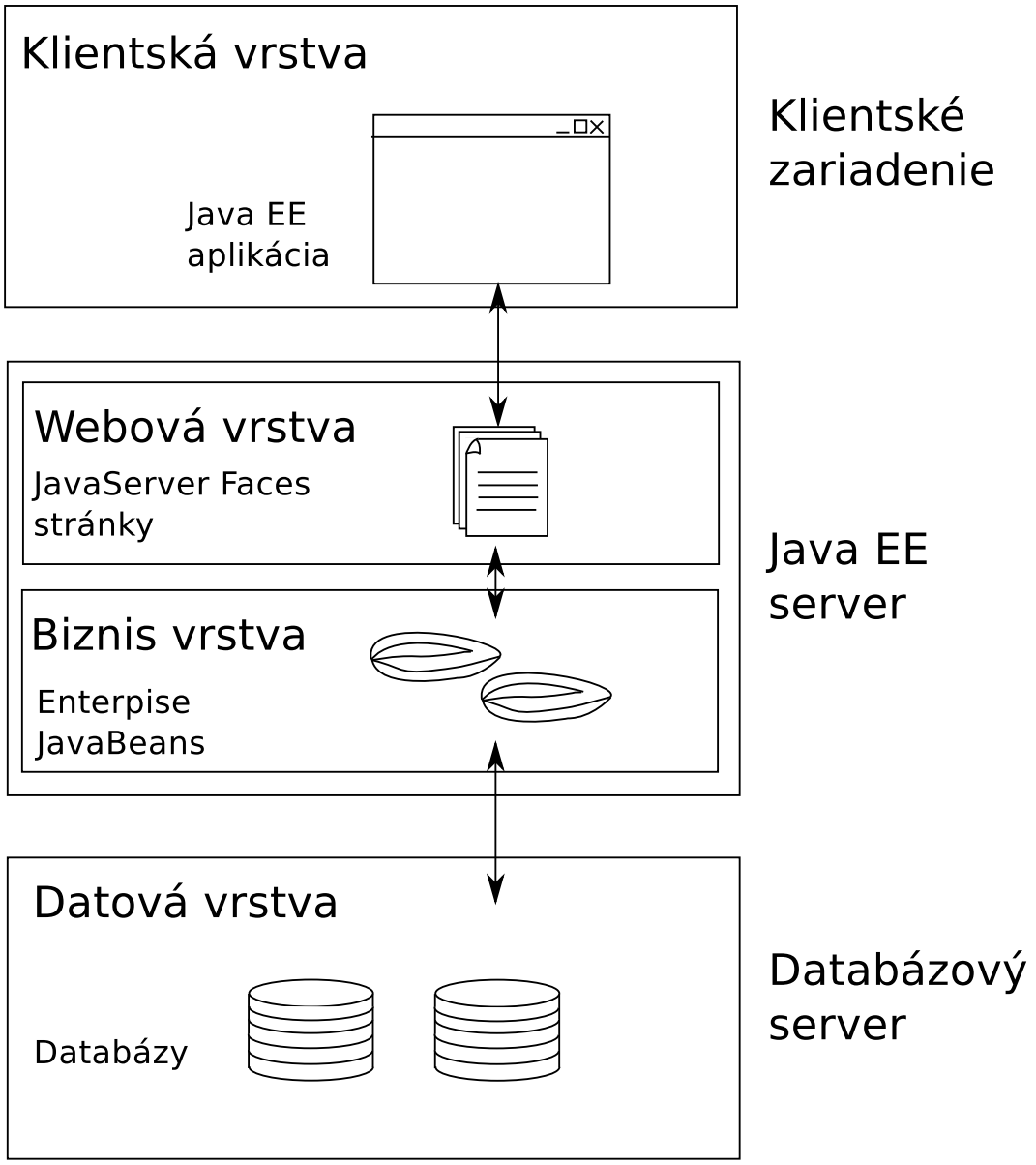
Trojvrstvový model Java EE aplikácií

Klientskú časť aplikácie predstavuje webový prehliadač, ktorý zasiela požiadavky webovej vrstve aplikácie. Úlohou tejto vrstvy je zobrazenie dát pre klienta. Taktiež zabezpečuje spracovanie užívateľskej interakcie a vstupných dát od klienta. Umožňuje navigáciu v rámci aplikácie a stará sa o udržiavanie stavu dát medzi jednotlivými HTTP požiadavkami v rámci sedenia (angl. *session*). Pre implementáciu webovej vrstvy sa používa v rámci platformy Java EE technológia JavaServer Faces, ktorá je podrobnejšie popísaná v nasledujúcej časti 2.2.

Biznis vrstva predstavuje miesto, kde je sústredená logika aplikácie. Implementuje funkčnosť špecifického odvetvia, pre ktoré je aplikácia určená. Prijíma dáta z webovej vrstvy a zabezpečuje ich spracovanie a uloženie s pomocou dátovej vrstvy. Pre jej implementáciu

sa využíva technológia Enterprise JavaBeans, ktorá naviac pridáva podporu transakčného spracovania a bezpečnosti.

Dátová vrstva obsahuje databázové servery a prostriedky pre prácu s nimi. S touto vrstvou, komunikuje biznis vrstva za účelom získavania a ukladania dát.



Obrázok 2.1: Vrstvy Java EE aplikácie [15]

Úloha kontajnera a konfigurácia aplikácií

Pri rozsiahlych aplikáciach, ktoré vyžadujú spracovanie transakcií, správu stavu a viacvláknovosť môže dochádzať k tomu, že kód samotnej aplikácie sa stáva komplikovaný až nezrozumiteľný. Java EE predchádza tomuto problému rozdelením aplikácie do modulov, označovaných aj ako komponenty aplikácie, a použitím kontajnerov. Kontajner tvorí prostredie pre

beh určitého druhu komponentov. Predstavuje rozhranie medzi nízkoúrovňovými detailami a modulmi tvoriacimi aplikáciu. Nízkoúrovňové detaily sú zapúzdrené do služieb a prostredníctvom kontajnera poskytované komponentom. Java EE obsahuje štyri štandardné kontajnery, z ktorých každý zabezpečuje beh určitého druhu komponentov. Konkrétne sa jedná o Application Client Container, Applet Container, Web Container a EJB Container. JSF aplikácie bežia vo Web Containeri.

Pred spustením aplikácie sú jej jednotlivé komponenty preložené, zabalené do archívov a nasadené do kontajnera. Web aplikácie sú zabalené do súboru s príponou .war (Web archív). S nasadením do kontajnera súvisí aj konfigurácia rôznych vlastností komponentov aj celkovej aplikácie. Konfigurácia môže byť špecifikovaná pomocou XML súborov alebo využitím anotácii.

2.2 Architektúra Java Server Faces

JavaServer Faces(JSF) je framework určený pre tvorbu užívateľských rozhraní webových aplikácií vytvorených v jazyku Java. Využíva model riadený udalosťami. Užívateľské rozhranie stavia z jednotlivých komponentov a pripomína tak tvorbu tradičného grafického užívateľského rozhrania desktopových aplikácií. Framework obsahuje sadu základných komponentov a taktiež podporuje tvorbu vlastných komponentov. Procesom tvorby vlastných komponentov sa zaoberá nasledujúca kapitola 3.

Komponenty si udržujú svoj stav medzi jednotlivými požiadavkami aj napriek tomu, že JSF pracuje nad bezstavovým protokolom HTTP. Samotné požiadavky sú obsluhované presne definovaným postupom označovaným ako životný cyklus spracovania požiadavky, ktorý je podrobnejšie popísaný v časti 2.2.2. Framework taktiež vyžaduje striktné oddelenie aplikačnej logiky od prezentačnej vrstvy – uplatňuje návrhový vzor MVC ako ukazuje obrázok 2.2.

2.2.1 Uplatnenie návrhového vzoru MVC v JavaServer Faces

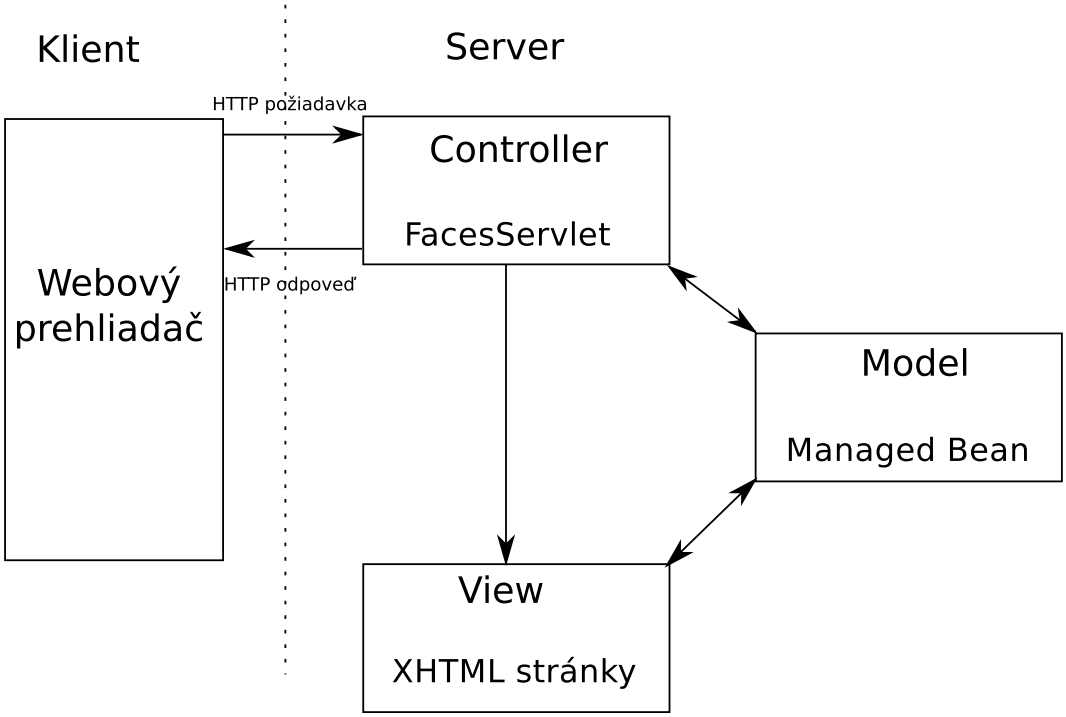
Controller v JSF zabezpečuje spracovanie jednotlivých požiadaviek smerovaných na určitú URL adresu, zvyčajne `/faces/*` špecifikovanú v konfiguračnom súbore `web.xml`. Controller spracuje požiadavku, presne definovaným postupom označovaným ako životný cyklus spracovania požiadavky, a presmeruje klienta na určenú stránku. Samotný controller je implementovaný triedou `FacesServlet`, ktorá je súčasťou frameworku.

Model tvoria triedy, ktoré vystavujú svoje premenné a metódy – umožňujú ich naviazanie na atribúty komponentov v prezentačnej vrstve. Tieto triedy musia mať špecifikované meno, ktoré bude použité pre ich adresáciu a rozsah ich životnosti v rámci aplikácie. Viac informácií o týchto triedach sa nachádza v časti 2.2.4.

Prezentačná vrstva je zložená z jednotlivých stránok označovaných ako facelets. Tie určujú rozloženie jednotlivých komponentov užívateľského rozhrania a taktiež definujú namapovanie dát z modelu na konkrétne komponenty.

2.2.2 Životný cyklus spracovania požiadavky

Spracovanie HTTP požiadaviek je neoddeliteľnou súčasťou webových aplikácii. Postupne ako sa web vyvíjal od jednoduchého statického zobrazenia až k súčastnému dynamickému prostrediu, kedy jednotlivé požiadavky obsahujú množstvo parametrov, ich spracovanie



Obrázok 2.2: Návrhový vzor MVC v JSF [14]

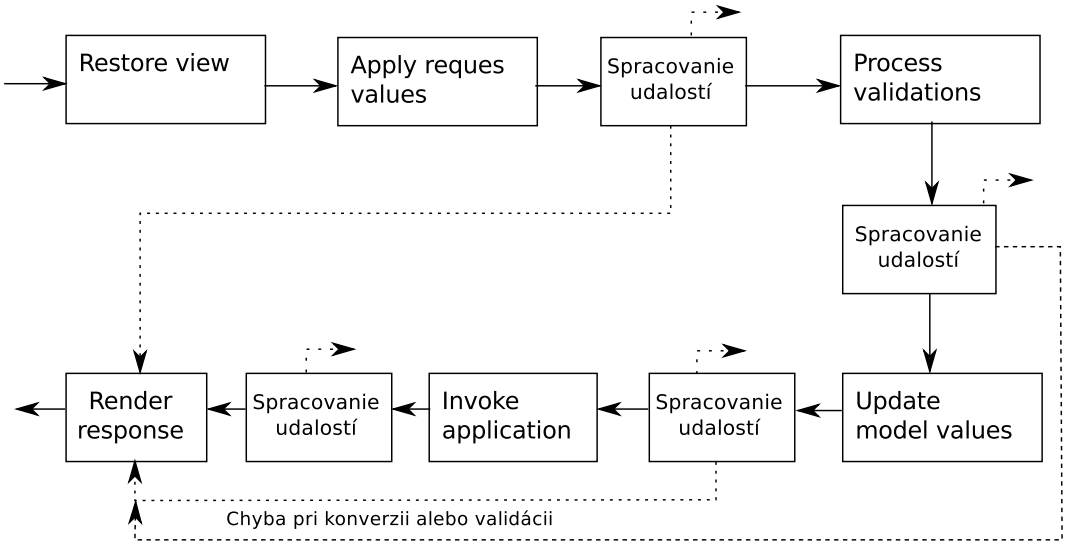
sa stalo komplexnejším. Nezávisle na programovacom jazyku či frameworku, spracovanie prichádzajúcej požiadavky zahŕňa:

- získanie stavu aplikácie z predchádzajúcich požiadaviek,
- prevedenie parametrov na dátové typy zodpovedajúce serveru spojené s validáciou,
- aktualizácia dát na serveri,
- použitie získaných parametrov v aplikácii napríklad v komunikácii s DB,
- odoslanie odpovede klientovi.

Životný cyklus spracovania požiadavky v JSF vykonáva tieto štandardné činnosti potrebné pre obsluhu požiadavky automaticky a v presne definovanom poradí. Zabezpečuje synchronizáciu medzi jednotlivými komponentami užívateľského rozhrania klienta a ich obrazom na serveri medzi jednotlivými požiadavkami. Sled fáz živtoného cyklu spracovania požiadavky je ukázaný na obrázku 2.3.

Restore view (Obnovenie stromu komponentov)

JSF si udržuje na serveri strom užívateľských komponentov, ktoré zodpovedajú grafickému užívateľskému rozhraniu tak ako je zobrazené používateľovi. V tejto fáze životného cyklu spracovania požiadavky sa buď obnoví strom komponentov uložený z predchádzajúcich požiadaviek alebo dôjde k vytvoreniu nového. Objekt stromu komponentov je následne uložený v objekte triedy `FacesContext` pre ďalšie fázy cyklu.



Obrázok 2.3: Životný cyklus spracovania požiadavky

Apply request values (Spracovanie hodnôt)

Po obnovení stromu komponentov sú spracovávané parametre prichádzajúcej požiadavky. Parametre sú vo forme názov-hodnota. JSF zabezpečí aktualizáciu komponentov volaním metódy `processDecodes()` na koreni stromu komponentov UIViewRoot. Táto metóda následne rekurzívne volá metódy `decode()` na jednotlivých komponentoch pre spracovanie parametrov. V komponentoch sú získané hodnoty parametrov z požiadavky použité na aktualizáciu stavu komponentu.

Process validations (Validácia hodnôt)

Po získaní hodnôt nasleduje ich konverzia a validácia. Hodnoty získané z požiadavky sú transformované z textovej reprezentácie v HTTP požiadavke na požadované dátové typy jazyka Java. Cieľom tejto fázy je zabezpečiť, aby aplikačná logika pracovala s očakávanými hodnotami a taktiež oddeliť ošetrenie nevalidných hodnôt od samotného algoritmu. Validáciu zabezpečuje metóda `processValidators()`, ktorá je vyvolaná na všetkých komponentoch kaskádovitým spôsobom ako v predchádzajúcej fáze.

Update model values (Akutalizácia Modelu)

Po úspešnej konverzii a validácii hodnôt nasleduje aktualizácia modelu. V tejto fáze sú nové hodnoty komponentov prepísané do modelu – Managed Bean. Jedná sa o automatickú aktualizáciu premenných, ktoré boli naviazané na atribúty komponentu. Túto úlohu zabezpečuje metóda `processUpdates()`.

Invoke application (Spustenie aplikácie)

Vo fáze označenej ako spustenie aplikácie dochádza k použitiu hodnôt v modeli, ktoré boli získané z požiadavky. Deje sa tak v obsluhe udalostí, ktoré boli vytvorené počas predchádzajúcich fáz. V obsluhe udalostí dochádza k volaniu biznis logiky. Táto fáza taktiež rozhoduje

o prípadnej navigácii na ďalšiu stránku. K vykonaniu tejto fázy dôjde len v prípade, že predchádzajúce fázy prebehli bez chýb.

Render response (Tvorba odpovede)

V poslednej fáze dochádza k vytvoreniu odpovede na požiadavku klienta. Komponenty sú vykreslené do podoby, v akej budú zobrazené klientovi. Túto činnosť zabezpečuje metóda `encode()`. Na záver je výsledný strom komponentov uložený, aby bol dostupný pre prípadné nasledujúce požiadavky.

2.2.3 Facelets

Facelet je termín, ktorý sa používa k označeniu deklaratívneho jazyka pre tvorbu prezentačnej vrstvy JSF v rámci konceptu MVC. Táto technológia nahradila staršiu prezentačnú technológiu JavaServer Pages používanú v predchádzajúcich verziách frameworku. Na rozdiel od nej neumožňuje vkladanie Java kódu do jednotlivých stránok. Cielene tak oddeľuje prezentačnú vrstvu od aplikačnej logiky. Pre tvorbu jednotlivých stránok využíva syntax značkovacieho jazyka XHTML.

Stránka predstavuje strom komponentov, nad ktorým sú vykonávané operácie popísané v predchádzajúcej časti 2.2.2. Komponenty sú zoskupené do knižníc. Framework obsahuje základnú knižnicu komponentov a taktiež umožňuje tvorbu vlastných komponentov. Pred použitím komponentu je nutné definovať na stránke menný priestor knižnice, z ktorej pochádza.

Ukážka 2.1: Deklarácia knižnice vo facelete

```
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:h="http://java.sun.com/jsf/html">
```

Po definovaní menného priestoru je možné používať komponenty z danej knižnice. Definícia v príklade 2.1 umožňuje použitie štandardných HTML komponentov. Previazanie komponentov s hodnotami z modelu je uvedené v nasledujúcej časti 2.2.4.

Facelety umožňujú použitie šablón, pre minimalizáciu duplicitného kódu. Špeciálnym druhom šablón sú kompozitné komponenty, ktoré vytvoria novú komponentu zoskupením už existujúcich. Tvorba tohto druhu komponentov je podrobnejšie popísaná v časti 3.1.

2.2.4 Managed Beans

ManagedBeans sú objekty tried, ktoré zoskupujú dáta zobrazené v prezentačnej vrstve. Umožňujú naviazanie svojich premenných a metód na atribúty komponentov užívateľského rozhrania. Tvoria vrstvu model v koncepte MVC.

ManagedBean je charakterizovaná dvoma vlastnosťami – menom a rozsahom platnosti. Meno môže byť špecifikované explicitne alebo automaticky odvodené z názvu triedy. Druhou vlastnosťou je rozsah platnosti Managed Bean. Je možné použiť nasledujúce rozsahy: v rámci aplikácie, sedenia, stránky alebo požiadavky. Rozsah ManagedBean je dôležitý preto, že programátor manuálne nevytvára jej novú inštanciu. ManagedBean je spravovaná (angl. *managed*), teda aj vytvorená automaticky. Pokiaľ sa v stránke nachádza výraz požadujúci hodnotu z modelu, JSF zabezpečí určenie ManagedBean, na ktorú odkazuje. Tá je následne vyhľadaná v špecifikovanom rozsahu. V prípade, že sa v danom rozsahu nenájde, je vytvorená nová inštancia.

Pre prepojenie premenných a metód Manged Bean s prezentačnou vrstvou sa používa Expression Language(EL). Tento jazyk umožňuje obojsmerné naviazanie hodnôt. V rámci JSF je zabezpečené získanie hodnoty z ManagedBean vo fáze vykreslenia a jej aktualizáciu vo fáze aktualizácie modelu. Ukážka 2.2 ukazuje použitie EL pre naviazania premennej z ManagedBean.

Ukážka 2.2: Použitie EL

```
<h:outputText value="#{user.name}"/>
```

Existujú dva spôsoby, ktorými je možné definovať triedy ako Managed Bean. Prvým je vytvorenie záznamu v konfiguračnom súbore `faces-config.xml`. Ako uvádza [12] preferovanejším spôsobom je použitie anotácie, ktoré je ukázané na príklade 2.3.

Ukážka 2.3: Deklarácia Managed Bean

```
import javax.faces.bean.ManagedBean
import javax.faces.bean.SessionScoped
@ManagedBean
@SessionScoped
public class UserBean { ...
```

2.3 Maven

Maven je nástroj, ktorý na základe projektového objektového modelu (POM) špecifikovaného v súbore `pom.xml` umožňuje spravovať projekt a činnosti s ním súvisiace. Zabezpečuje vytvorenie nového projektu, správu závislostí, zostavenie výsledku, testovanie, nasadenie projektu na server a dokumentáciu. Každý projekt identifikujú tri parametre: [6]

- groupID – identifikátor organizácie alebo skupiny projektov
- artifactID – identifikátor projektu
- version – verzia projektu

Nástroj je možné ovládať z príkazového riadku ako aj z integrovaných vývojových prostredí. Bol použitý na správu projektu vytvoreného v tejto práci.

2.4 Arquillian

Arquillian je testovacia platforma pre tvorbu automatizovaných integračných, akceptačných a funkčných testov aplikácií napísaných v jazyku Java. Arquillian spája existujúce testovacie frameworky a pridáva automatizáciu nasadenia testov, správu kontajnera a ďalšie rozšírenia. Arquillian stavia na troch základných princípoch: [1]

- testy by mali byť prenositeľné medzi podporovanými aplikačnými serverami
- testy je možné spúšťať pomocou integrovaného vývojového prostredia alebo príkazového riadku
- platforma rozširuje a spája existujúce testovacie frameworky

Kapitola 3

Tvorba komponentov v JSF

Framework JSF obsahuje sadu základných komponentov. Pokiaľ pre aplikáciu nepostačujú základné komponenty, je možné použiť komponenty tretej strany spôsobom spomenutým v časti o prezentačnej vrstve 2.2.3 alebo vytvoriť si vlastné komponenty. JSF poskytuje možnosť tvorby dvoch druhov komponentov. Prvým sú kompozitné komponenty – komponenty vytvorené skladaním existujúcich komponentov. V tomto prípade sú komponenty vytvárané pomocou Facelet šablón, teda v jazyku XML. Druhým druhom sú vlastné komponenty, ktoré sú implementované v jazyku Java. V poslednej časti tejto kapitoly je v stručnosti predstavený nástroj Component Development Kit (CDK). Tento nástroj je vyvíjaný pod projektom RichFaces[9] a pomocou neho je možné uľahčiť proces tvorby komponentov implementovaných v jazyku Java. Táto kapitola vychádza z poznatkov uvedených v knihe [12].

3.1 Kompozitné komponenty

Od verzie frameworku JSF2.0 je možné vytvárať kompozitné komponenty (angl. *Composite components*). Jedná sa o špeciálny druh šablóny, ktorá zoskupuje niekoľko už existujúcich komponentov. Komponent je definovaný vo forme XHTML stránky, ukážka štruktúry je uvedená nižšie. Hlavnou výhodou tohto druhu vlastných komponentov je, že pre ich vytvorenie nie je nutné písať kód v jazyku Java a vytvárať XML konfiguračné súbory. Niektoré integrované vývojové prostredia poskytujú možnosť vyextrahovania kompozitných komponentov z existujúcich stránok.[1]

V ukážke 3.1 kódu je príklad vytvorenia kompozitného komponentu. Pre jeho definíciu sú využité dva elementy `<cc:interface>` a `<cc:implementation>`.

cc:interface	špecifikuje rozhranie – priestor pre deklaráciu atribútov, ktoré môžu byť špecifikované pri použití komponentu
cc:implementation	definuje šablónu komponentu, jeho vykreslenie. Keď uvažujeme o cc:interface ako o abstrakcii komponentu, táto časť je jeho implementáciou

Ukážka 3.1: Kompozitný komponent

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD␣XHTML␣1.0␣Transitional//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

[1] https://netbeans.org/kb/docs/web/jsf20-support.html

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:cc="http://java.sun.com/jsf/composite"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core">

   <cc:interface>
      <cc:attribute name="likeCount"/>
      <cc:attribute name="incLikes"
          method-signature="void␣action()"/>
   </cc:interface>

   <cc:implementation>
      <h:form class="rate-form">
         <h:outputText value="#{cc.attrs.likeCount}" /><br/>
            <h:commandButton value="Like" >
               <f:ajax event="click"
              listener="#{cc.attrs.incLikes}" render="@form"/>
            </h:commandButton>
      </h:form>
   </cc:implementation>
</html>
```

Kompozitné komponenty priniesli jednoduchosť do tvorby nových komponentov vo frameworku JSF. Prispievajú k prehľadnejšiemu kódu stránky a pre väčšinu prípadov sú postačujúcou alternatívou k vlastným komponentom vytvoreným v jazyku Java. V prípade potreby komplexnejších požiadaviek ich ale nedokážu plne nahradiť.

3.2 Vlastné komponenty

V prípade, že požiadavky na vlastnú komponentu presahujú možnosti kompozitných komponentov je potrebné vytvoriť vlastný komponent v jazyku Java. Implementovať potomka triedy `UIComponentBase`. Úlohou tejto triedy – komponentu bude špecifikovať atribúty pre jeho prispôsobenie, zabezpečiť vytvorenie časti užívateľského rozhrania a spracovanie interakcie s ním.

3.2.1 Atribúty komponentu

Pre deklaráciu každého atribútu komponentu je nutné implementovať metódy pre získanie a uloženie hodnoty. Metódy budú využívať rozhranie `StateHelper` pre udržiavanie stavu atribútov medzi jednotlivými požiadavkami.

3.2.2 Vykreslenie komponentu

Komponent tvorí časť užívateľského rozhrania. Aby ho bolo možné zobraziť klientovi je nutné naimplementovať jeho vykreslenie. Túto činnosť zabezpečuje metóda `encodeAll()`, prípadne metódy `encodeBegin()`, `encodeChildren()` a `encodeEnd()`. Ich výstupom je kód v ľubovoľnom značkovacom jazyku, zvyčajne HTML. Pre jeho tvorbu sa používa objekt triedy `ResponseWriter`, odovzdaný v parameteri `FacesContext`. Obsahuje metódy

pre výpis značiek a ich atribútov. Čiastočne tak uľahčuje generovanie značiek predstavujúcich grafickú reprezentáciu komponentu. Celý proces vykreslenia prebieha v poslednej fáze životného cyklu spracovania požiadavky.

3.2.3 Spracovanie parametrov z HTTP požiadavky

V prípade užívateľskej aktivity je na server zaslaná HTTP požiadavka. Jej spracovanie a následné aktualizovanie stavu komponentu je možné implementovať v metóde `decode()`. Pre získanie parametrov z požiadavky slúži metóda `getRequestParameterMap()`.

3.2.4 Súbor popisujúci značky komponentu

Komponent je doplnený konfiguračným súborom `component-taglib.xml`. Ten definuje menný priestor pre komponent a spája triedu, ktorá obsahuje kód komponentu, so značkou, pod ktorou bude komponent dostupný pre použitie vo facelete. Môže tiež definovať aj atribúty danej značky a ich dátove typy.

3.2.5 Použitie renderera

Úlohu vykreslenia komponentu a spracovania parametrov z prichádzajúcej požiadavky je možné delegovať na triedu označovanú ako `renderer`. Komponent ich môže obsahovať viacero, špecializovaných napríklad podľa typu klientského zariadenia alebo výstupného značkovacieho jazyka.

3.3 Component Development Kit

V predchádzajúcej časti boli predstavené činnosti, ktoré zahŕňa tvorba vlastných komponentov. Tento proces obsahujúci generovanie HTML kódu v jazyku Java a tvorbu konfiguračných xml súborov je možné uľahčiť použitím nástroja Component Development Kit(CDK), ktorý je súčasťou projektu RichFaces. Nasledujúca časť oboznamuje čitateľa o jeho prínose pre vývoj komponentov. Konkrétne uvádza zjednodušenie deklarácie atribútov komponentu, jeho vykreslenia a tvorby súboru popisujúceho značky. Táto časť vychádza z informácii uvedených v príručke [16].

3.3.1 Trieda komponentu

Oproti predchádzajúcemu postupu uvedenému v časti 3.2.1, nie je nutné triedu implementovať. Pre určenie typu komponentu, skupiny, do ktorej patrí, a jeho značky je možné využiť anotáciu triedy `@JsfComponent`. Deklaráciu atribútu je tiež možné nahradiť vytvorením abstraktnej metódy a pridaním anotácie. Príklad definície triedy s použitím CDK je uvedený v ukážke 3.2.

Ukážka 3.2: Definícia triedy s použitím CDK

```
@JsfComponent(
        type = "sk.lukasmacko.richfaces.chart.component.Chart",
        family = "sk.lukasmacko.Chart",
        renderer = @JsfRenderer(
                        type = "sk.lukasmacko.chartRenderer"),
        tag = @Tag(name = "chart"))
```

```
abstract public class AbstractChart extends UIComponentBase{
    @Attribute
    public abstract String getTitle();
}
```

3.3.2 Vykreslenie komponentu

Pri použití CDK nie je nutné manuálne programovať generovanie HTML reprezentácie komponentu v Jave. Vykreslenie komponentu je možné definovať priamo v jazyku XML ako pri kompozitných komponentoch. CDK obsahuje prekladač, ktorý zabezpečí prevod takejto deklarácie vykreslenia komponentu na zodpovedajúci kód v jazyku Java.

3.3.3 XML konfiguračné súbory

Predchádzajúci postup vyžadoval manuálne vytvorenie súboru, ktorý bude popisovať značky a ich atribúty. Použitie CDK tento proces skracuje na vytvorenie súboru, v ktorom je pomocou anotácie definovaný menný priestor komponentu. Samotné konfiguračné súbory `component.taglibrary.xml` a `faces-config.xml` sú generované automaticky na základe anotácii v triede komponentu. Vygenerovaný súbor `component.taglibrary.xml` naviac obsahuje komentáre ku komponentu a jeho atribútom, ktoré je možné použiť k tvorbe dokumentácie.

Kapitola 4

Vykresľovanie grafov na webe

Táto kapitola poskytuje prehľad rôznych riešení pre vykresľovanie grafov používaných na webe. V úvode popisuje dva prístupy k tvorbe grafov, vykresľovanie na serveri a na klientovi. Taktiež uvádza formy, v akých je výsledné grafy možné zobraziť. Posledná časť sa venuje predstaveniu existujúcich riešení pre vykresľovanie grafov. Dôraz je kladený na javascriptové knižnice, ktoré sú porovnané za účelom výberu jednej z nich pre použitie v implementácií vlastného JSF komponentu.

4.1 Graf

Pre účely tejto práce rozumieme grafom vizuálnu reprezentáciu závislosti dvoch alebo viacerých premenných zobrazených v 2D priestore. Závislosť môže byť vyjadrená lomenou čiarou alebo plochou. Na základe toho existuje niekoľko typov grafov. Grafy zobrazené na webe možeme rozdeliť podľa miesta ich vytvorenia a formy, v akej sú zobrazené užívateľovi.

4.2 Formy zobrazenia grafov

Existuje niekoľko foriem, v akých môžu byť grafy zobrazené na webe. Môžu byť zobrazené ako statický obrázok pomocou značky `<img>`. Taktiež je možné ich zobraziť ako kolekciu útvarov vo formáte SVG, čo je značkovací jazyk vychádzajúci z XML, umožňujúci tvorbu vektorovej grafiky. Ďalším spôsobom je vykreslenie grafu na prvok `<canvas>` pomocou javascriptu. Existuje ešte niekoľko iných foriem ako napríklad zobrazenie grafu pomocou technológie Flash alebo Java Applet, tie ale vyžadujú doinštalovanie externého doplnku do prehliadača.

4.3 Delenie podľa miesta vytvorenia grafu

Samotný proces vytvorenia grafu môže prebiehať na serveri alebo až na klientovi. Pri vykreslení na serveri je výstupom zvyčajne statický obrázok. Pre vykreslenie grafu na klientovi sa zvyčajne využíva javascript a výstupom môže byť svg alebo graf na elemente `<canvas>`. Pokiaľ je graf vykresľovaný pomocou technológie Adobe Flash alebo Java Applet samotné vykreslenie prebieha takisto na klientovi.

4.4 Existujúce riešenia

Nasledujúca časť uvádza niekoľko existujúcich riešení pre vykresľovanie grafov. Podrobnejšie sa zameriava na javascriptové riešenia, z ktorých bude zvolené jedno pre implementáciu komponenty JSF.

- OpenFaces[2] – knižnica JSF komponentov, ktorá umožňuje vykreslenie čiarových, stĺpcových a koláčových grafov. Grafy sú vykresľované pomocou knižnice vytvorenej v jazyku Java.
- PrimeFaces[8] – knižnica JSF komponentov, ktorá grafy vykresľuje pomocou javascriptovej knižnice.
- JpGraph[5] – knižnica napísaná v jazyku PHP vykresľujúca grafy vo forme obrázkov
- Open Flash Chart[7] – knižnica vykresľuje grafy pomocou technológie Flash.

Javascriptové knižnice pre vykresľovanie grafov

Pre účely nasledujúceho predstavenia existujúcich javascriptových knižníc vykresľujúcich grafy budem slovným spojením *základné druhy grafov* označovať skupinu čiarových stĺpcových a koláčových grafov. Prípadné ďalšie druhy grafov budú pri konkrétnom riešení uvedené explicitne.

Bluff

Knižnicu Bluff[13] je možné použiť pre vykresľovanie všetkých základných druhov grafov. Dáta je možné načítať z HTML tabuľky alebo zadať v javascripte pri vytváraní grafu. Grafy sú vykresľované na HTML prvok canvas. Jedinou podporovanou interakciou s grafom je zobrazenie informácie o bode, nad ktorým sa nachádza kurzor myši. Projekt nie je vedený vo verejnom repozitári, jediná dostupná verzia 0.3.6.2 je z januára roku 2011.

D3

D3[11] je komplexné riešenie poskytujúce rôzne možnosti vizualizácie dát. Pre vykreslenie výsledku používa formát SVG. Okrem základných grafov poskytuje možnosti vykreslenia stromových štruktúr a rôznych iných diagramov. Nevýhodou tejto knižnice je ale nízka úroveň abstrakcie pri tvorbe grafov pre vykresľovanie. Na serveri github je vedený verejný repozitár tohto projektu. Knižnica je dostupná pod licenciou BSD.

CanvasXpress

CanvasXpress[18] vykresľuje grafy na element canvas. Podporuje tvorbu všetkých základných druhov grafov. Okrem toho je možné pomocou tohto riešenia vykresľovať aj vennove diagramy a 3D grafy. Knižnica obsahuje podporu pre obsluhu udalostí. Na stránke projektu sa neuvádza odkaz na verejný repozitár. Projekt je distribuovaný pod licenciou LPGL3.

Elycharts

Knižnica Elycharts[3] podporuje len základné druhy grafov. Výsledné grafy sú vo formáte SVG. Knižnica nemá podporu pre obsluhu udalostí a taktiež nemá uvedený repozitár so správou verzií. Dokumentácia projektu nie je úplná. Projekt je možné použiť pod licenciou MIT.

Flotr2

Knižnica Flotr2[4] vykresľuje všetky základné druhy grafov. Naviac umožňuje vykreslenie bublinového grafu, kde sú jednotlivé body grafu určené súradnicami x, y a polomerom kružnice. Taktiež umožňuje vykreslenie grafu zobrazujúceho dianie na burze označovaného anglický termínom candlestick. Grafy sú vykresľované na element canvas. Knižnica nemá podporu obsluhy udalostí. Projekt je vedený vo verejnom repozitári na serveri github a je možné ho použiť pod licenciou MIT.

gRaphael

gRaphael[10] podporuje vykreslenie základných druhov grafov. Grafy sú vytvárané v SVG formáte. Táto knižnica umožňuje obsluhovať kliknutia na graf a pohyb myšou nad bodmi grafu. Zdrojové kódy sú verzované na serveri github.com. Projekt je distribuovaný pod licenciou MIT.

jqPlot

jqPlot[17] umožňuje vykresľovať všetky základné druhy grafov a tiež bublinové a grafy označované ako candlestick. Podporuje užívateľskú interakciu s grafom a obsluhu udalostí. Taktiež poskytuje užívateľovi možnosť upravovať hodnoty zobrazené v grafe a automatické vykresľovanie lineárnej aproximácie. Knižnica vykresľuje grafy na element canvas a výsledok je korektne zobrazený vo všetkých moderných webových prehliadačoch. Návrh knižnice bol vytvorený s ohľadom na rozšíriteľnosť. jqPlot sa skladá zo skupiny modulov, z ktorých každý zabezpečuje určitú čiastkovú funkčnosť. Jedinou externou závislosťou tejto knižnice je jQuery. Celý projekt je stále vyvíjaný a udržiavaný. Zdrojové kódy je možné použiť pod licenciou GPL alebo MIT.

Kapitola 5

Návrh komponentu pre vykresľovanie grafov

Táto kapitola sa venuje návrhu komponentu. Začína výberom javascriptového riešenia, ktoré bude zapúzdrené v navrhovanom komponente. Následne sú predstavené možnosti poskytované zvolenou knižnicou a vymedzené tie z nich, ktoré bude výsledný komponent podporovať. Jednotlivé časti podrobnejšie oboznamujú čitateľa s podporovanými druhmi grafov, návrhom dátového modelu, možnostiach zadávania dát a obsluhe udalostí. V závere kapitoly sú predstavené konfigurovateľné atribúty komponentu.

5.1 Výber javascriptového riešenia pre komponent

Výsledný kompnent bude abstrakciou nad javascriptovým riešením. Umožní programátorom vyvíjajúcim s frameworkom JSF tvoriť grafy jednoduchšie a bez nutnosti písania javascriptového kódu. Prvým krokom pri návrhu komponenty bol výber javascriptovej knižnice, ktorá bude zabezpečovať vykresľovanie grafov. Rozhodujúce kritéria pre výber som po dohode s mojím konzultantom, zvolil nasledovne:

- druhy grafov podporované knižnicou,
- korektné zobrazenie vo všetkých moderných webových prehliadačoch,
- podpora užívateľskej interakcie a možnosť obsluhy vzniknutých udalostí,
- stav vývoja projektu a podpora komunity,
- knižnica musí byť voľne dostupná, aby ju bolo možné použiť ako súčasť open-source projektu.

Na základe týchto parametrov som vypracoval porovnanie, ktorého výsledky boli zhrnuté v predchádzajúcej kapitole 4.

Pre použitie v komponente som zvolil knižnicu jqPlot, nakoľko z uvedených knižníc najlepšie spĺňa vyššie uvedené kritéria. Zdrojové kódy tejto knižnice sú vo verejne dostupnom repozitári, je aktuálne stále vyvíjaná a udržiavaná. Taktiež má diskusné fórum, kde je možné diskutovať prípadné problémy zistené v tejto knižnici. Výsledne grafy sú korektne zobrazované v moderných prehliadačoch. Rozhodujúcim parametrom v prospech tohto riešenia bola podpora udalostí, ktoré komponent umožňuje obsluhovať, ako aj modulárny návrh knižnice.

5.2 Vymedzenie funkčnosti komponentu

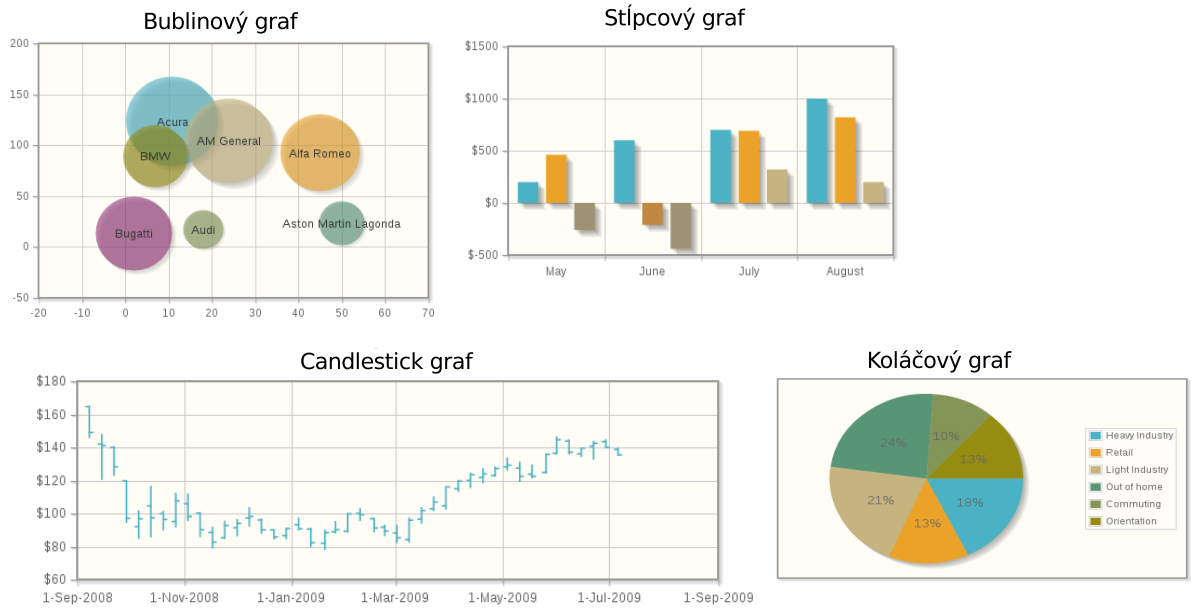
V predchádzajúcej časti som spomenul, že knižnica jqplot je rozdelená do modulov. Každý z nich zabezpečuje istú časť vykreslenia grafov. Pridanie nového druhu grafu, prípadne zmena vykresľovania niektorej časti, je len zmenou nastavení grafu, pričom štruktúra vytvorenia grafu ostáva rovnaká pre všetky druhy grafov. Kód incializujúci graf je zobrazený v ukážke 5.1.

Ukážka 5.1: Vytvorenie grafu s jqPlot

```
$.jqplot(element,data,options)
```

`element`	identifikátor elementu `<div>`, do ktorého bude vykreslený graf
`data`	kolekcia vstupných hodnôt
`options`	štruktúra špecifikujúca vlastnosti grafu

Možnosti (angl. *options*) dovoľujú špecifikovať vlastnosti týkajúce sa vzhľadu grafu, rozsahy a popis osí, možnosti zväčšovania grafu, automatické vykreslenie lineárnej aproximácie a iné. Niektoré druhy grafov, ktoré je možné pomocou knižnice vykresliť, sú ukázané na obrázku 5.1.



Obrázok 5.1: Ukážky grafov

Z rôznych druhov grafov a možností knižnice som po dohode s konzultantom zvolil nasledovné vlastnosti, ktoré bude podporovať vytvorený komponent:

- Komponent bude podporovať čiarový, stĺpcový a koláčový graf. Návrh komponentu ale bude umožňovať pridanie ďalších druhov.
- Komponent bude umožňovať obsluhu užívateľskej interakcie. Podrobnosti o obsluhe udalostí sú uvedené v časti 5.6.

- Dáta pre komponent bude možné zadávať dvoma spôsobmi – vytvorením modelu alebo deklaratívnym spôsobom vo Facelete. Viac informácií o zadávaní dát obsahuje časť 5.5.
- Možnosti konfigurácie komponentu sú uvedené v časti 5.7.

5.3 Podporované druhy grafov

Komponent podporuje čiarový, stĺpcový a koláčový graf. Aj napriek rovnakej štruktúre, ktorá je použitá pri inicializácii grafov, existujú určité rozdiely medzi jednotlivými druhmi. Tieto rozdiely sa týkajú formátu zadávaných dát a obmedzení v oblasti podporovaných dátových typov pre jednotlivé druhy grafov. Nasledujúca časť oboznamuje čitateľa s konkrétnymi špecifikami.

Čiarový graf

Čiarový graf je tvorený kolekciou bodov v 2D priestore spojených úsečkami. Dátový typ nezávislej premennej grafu, určujúcej súradnicu x, môže byť číselný alebo dátumový. Je možné zobraziť viacero závislých premenných v jednom grafe. Dáta sú zadávané vo formáte JSON, ako kolekcia dvojíc. Ukážka 5.2 zobrazuje formát dát vyjadrujúci závislosť na jednej premennej.

Ukážka 5.2: Formát dát v JSON pre čiarový graf

```
[[x1,y1],[x2,y2],...]
```

Stĺpcový graf

Stĺpcový graf zobrazuje závislosť na premennej číselného dátového typu alebo textového reťazca. V prípade číselného dátového typu je formát rovnaký ako pri čiarovom grafe 5.2. V prípade reťazca majú dáta grafu formát ako je uvedené v ukážke 5.3. Hodnoty pre os x sú uvádzané vo vlastnostiach grafu. V prípade zobrazovania závislosti viacerých premenných, bude potrebné ošetriť prípadné nezadanie niektorej z hodnôt pre os y a nahradiť ju nulou, aby nedošlo chybnému zobrazeniu.

Ukážka 5.3: Formát dát v JSON pre stĺpcový graf

```
[y1,y2,...]
```

Koláčový graf

Koláčový graf reprezentuje hodnotu ako podiel plochy kruhovej výseče z celkovej plochy kruhu. Závislá premenná vyjadrujúca plochu musí byť nezáporná. Nezávislá premenná, na osi x, má dátový typ textového reťazca. Formát je rovnaký ako pri čiarovom grafe 5.2. Pri tomto druhu je taktiež potrebné riešiť problém spomínaný pri stĺpcovom grafe – nezadanie hodnoty nezávislej premennej.

5.4 Návrh dátového modelu

Dátový model pre komponenty bude zoskupovať hodnoty vyjadrujúce závislosť jednej premennej v grafe. Bude poskytovať možnosť exportovania hodnôt do formátu JSON, ktorý bude použitý na incializáciu grafu v javascripte. Je potrebné, aby zohľadňoval špecifiká jednotlivých druhov grafov uvedené v predchádzajúcej časti. Pre uloženie bodov grafu som sa rozhodol v dátovom modeli použiť abstraktný dátový typ mapa. Nakoľko grafy môžu zobrazovať závislosť na rôznych dátových typoch (číslo, dátum, textový reťazec), trieda dátového modelu bude generická.

Problém spomínaný v predchádzajúcej časti, kedy je nutné zabezpečiť, aby pri zobrazení viacerých závislých premenných v jednom grafe boli hodnoty korektne zobrazené a nedošlo k ich prehádzaniu, som sa rozhodol riešiť nasledovným spôsobom: Prvá skupina hodnôt určí hodnoty osi x, pre ktoré budú na grafe zobrazené hodnoty závislej premennej. Z ďalších skupín hodnôt budú vybraté len tie hodnoty osi x, ktoré sa nachádzali v prvej. Chýbajúce hodnoty budú nahradené nulou a prebytočné budú ignorované.

5.5 Zadávanie dát

Rozhodol som sa v komponente umožniť zadávať dáta pre graf dvoma spôsobmi:

- tradičný spôsob – v Managed Bean je vytvorená inštancia dátového modelu. Následne sú doňho pridané dáta, ktoré bude graf zobrazovať. Objekt modelu je odovzdaný ako atribút komponentu.
- deklaratívny spôsob – nebude potrebné vytvárať inštanciu dátového modelu. Komponentu bude odovzdaná kolekcia objektov. Následne bude špecifikované, ktoré vlastnosti z objektov budú zobrazené na grafe.

5.6 Spracovanie udalostí

Komponent bude podporovať užívateľskú interakciu a obsluhu takto vzniknutých udalostí. Udalosti bude možné využiť pre zobrazenie detailnejších informácii o hodnotách v grafe, navigáciu alebo úpravu zobrazených hodnôt. Konkrétne bude komponent poskytovať nasledujúce udalosti:

- dataclick – kliknutie na bod v grafe,
- highlight – kurzor myši sa nachádza nad bodom v grafe,
- unhighlight – kurzor myši opustil bod v grafe,
- dragstart – začiatok pohybu bodom grafu,
- pointmove – pohyb bodom grafu,
- dragstop – ukončenie pohybu bodom grafu,

Pre každú z uvedených udalostí bude možné zaregistrovať obslužnú funkciu v javascripte. Udalosť dataclick a dragstop bude naviac možné obsluhovať aj na strane servera.

5.7 Štruktúra značiek a ich atribúty

Po vymedzení možností poskytovaných komponentom som sa zameral na návrh štruktúry značiek, ktoré budú využité pre vytvorenie komponentu. Uvažoval som nad alternatívou, v ktorej by každý druh využíval pre inicializáciu vlastnú značku. Vzhľadom na návrh knižnice, v ktorej sú rôzne grafy incializované jednou funkciou, som sa rozhodol využiť pre všetky druhy grafov jednu značku. Keďže ale komponent poskytuje väčšie množstvo atribútov, ktoré je možné konfigurovať rozdelil som ich špecifikáciu do niekoľkých značiek z dôvodu prehľadnosti.

Chart

title Nadpis uvedený nad grafom

styleClass CSS trieda pre HTML prvok `<div>` obaľujúci graf

ondataclick javascriptová funkcia obsluhujúca udalosť dataclick

onhighlight javascriptová funkcia obsluhujúca udalosť highlight

onunhighlight javascriptová funkcia obsluhujúca udalosť unhighlight

ondragstop javascriptová funkcia obsluhujúca udalosť dragstop

ondragstart javascriptová funkcia obsluhujúca udalosť dragstart

onpointmove javascriptová funkcia obsluhujúca udalosť pointmove

dataClickListener obsluha udalosti dataclick na serveri

dragStopListener obsluha udalosti dragstop na serveri

Series

label popis skupiny bodov vyjadrujúci závislé hodnoty grafu

type atribút špecifikujúci druh grafu

value model obsahujúci dáta pre graf

color farba skupiny bodov

marker symbol použitý na vykreslenie bodu grafu

markerVisible určuje či bude zobrazený symbol v zadaných bodoch grafu, alebo bude graf zobrazený len ako lomená čiara

dragable určuje či bude možné pohybovať bodmi grafu

dragableConstraint obmedzuje pohyb bodmi grafu len pre určitý smer

trendlineVisible automatické vykresľovanie lineárnej aproximácie

Point

x hodnota nezávislej premennej

y hodnota závislej premennej

Xaxis a Yaxis

format formát textu popisujúceho os

label text uvedený pri osi

min minimálna hodnota zobrazená na osi

max maximálna hodnota zobrazená na osi

pad určuje o koľko bude väčší rozsah osí vzhľadom na minimálnu a maximálnu hodnotu v grafe

tickrotation uhol natočenia popisiek osí

Legend

position poloha legendy vzhľadom ku grafu

placement určuje či sa legenda bude nachádzať v ploche grafu alebo vedľa nej

visible rozhoduje či bude legenda vykreslená

Cursor

zoomEn umožňuje zväčšovať graf

constraintZoom umožňuje obmedziť možnosti zväčšovania grafu len na niektorú z osí

cursorStyle vzhľad kurzoru v grafe

tooltipVisible rozhoduje či bude zobrazený popis bodu grafu v prípade, keď sa nad ním nachádza kurzor myši

Kapitola 6

Implementácia navrhnutého komponentu

Táto kapitola sa venuje implementačnej časti projektu. Začína uvedením postupu implementácie. Popisuje rozdelenie projektu do jednotlivých tried a balíčkov. Podrobnejšie sa zaoberá dátovým modelom a popisom spôsobov zadávania dát. Taktiež predstavuje implementáciu obsluhy udalostí. V poslednej časti stručne predstavuje aplikáciu implementovanú pre ukážku použitia komponentu.

6.1 Zvolený postup

Po fáze návrhu 5, vymedzení funkčnosti a jej rozdelenia do jednotlivých značiek, som pristúpil k samotnej implementácii komponentu. Ako bolo uvedené v kapitole 3, technológia JSF umožňuje vytvárať dva druhy komponentov. Pre implementáciu komponentu vykresľujúceho grafy som zvolil prístup uvedený v časti 3.2. Dôvodom k tomu boli v návrhu uvedené použitie hierarchickej štruktúry značiek a spôsoby zadávania dát, ktoré by nebolo možné implementovať kompozitným komponentom. Naviac som použil Component Development Kit(CDK), pre jeho výhody uvedené v časti 3.3. Taktiež som odelil vykreslenie komponentu a spracovanie požiadavky do triedy vlastného renderera.

Implementácia prebiehala spôsobom zhora nadol. Postupnosť jednotlivých implementových častí je uvedená v nasledujúcom zozname.

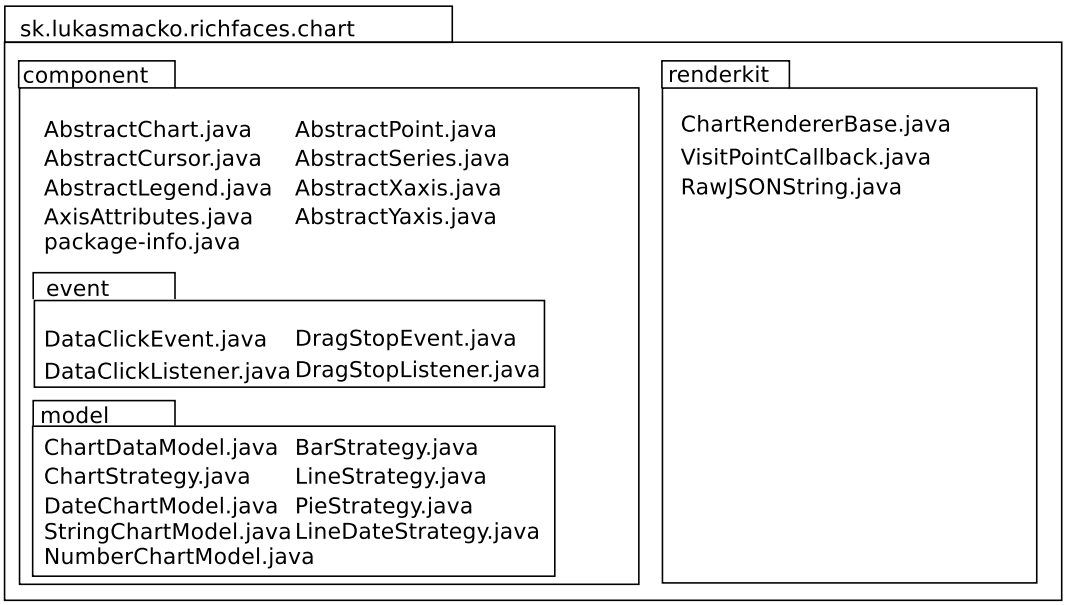
- spracovanie atribútov, graf zobrazoval len staticky zadané dáta
- pridanie možnosti zadávania dát pomocou vytvorenia modelu
- možnosť zadávania dát pomocou iterácie vo facelete
- podpora obsluhy udalostí na klientovi a na serveri
- tvorba ukážkovej aplikácie
- automatizované testy

6.2 Organizácia projektu

Projekt komponentu je rozdelený do balíčkov ukázaných na obrázku 6.1. Abstraktné triedy v balíku `component` definujú atribúty jednotlivých značiek. Triedy v balíku `event`, obsa-

hujú podporu pre obsluhu udalostí na strane servera. Balík `model` obsahuje implementáciu dátového modelu, ktorý je popísaný v časti 6.4. Balík `renderkit` obsahuje triedu zabezpečujúcu vykreslenie komponentu a spracovanie prichádzajúcej požiadavky ako aj s tým súvisiace pomocné triedy.

Súčasťou projektu je aj javascriptová knižnica jqPlot a súbory obsahujúce testy popísané v nasledujúcej kapitole, ktoré nie sú na obrázku zobrazené.



Obrázok 6.1: Štruktúra projektu komponentu

6.3 Vytvorenie komponentu

Komponent je možné použiť na stránke po definícii jeho menného priestoru, konkrétne `http://lukasmacko.sk/chart`. Pre konfiguráciu komponentu je možné značky popísané v návrhu. Príklad vytvorenia grafu s použitím všetkých značiek je zachytený v ukážke 6.1. V niektorých prípadoch vykreslenia grafu ale nemusia byť použité všetky uvedené značky, pokiaľ je postačujúce východzie správanie. Jedinými povinnými značkami sú `chart` a aspoň jedna značka `series`.

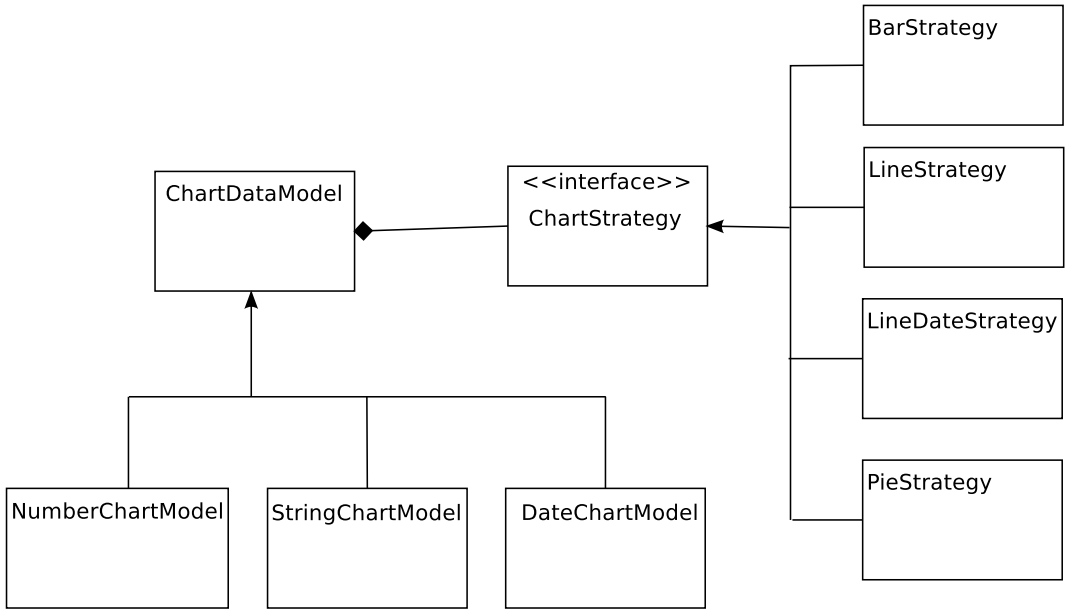
Ukážka 6.1: Použitie vytvorenej komponenty

```
<lm:chart title="Ceny␣ropy␣za␣rok␣2012">
   <lm:series type="line" value="#{bean.prices}"/>
   <lm:xaxis format="YY-M" />
   <lm:yaxis format="$%d" />
   <lm:legend position="right"/>
   <lm:cursor zoomEn="true"/>
</lm:chart>
```

Viac príkladov použitia komponentu spolu s obrázkami sa nachádza v prílohe B.

6.4 Dátový model

Úlohou dátového modelu je zoskupenie hodnôt vyjadrujúcich závislosť jednej premennej, ktorá bude zobrazená na grafe. Vychádza zo špecifikácie požiadaviek uvedených v časti návrhu 5.4. Diagram tried dátového modelu je ukázaný na obrázku 6.2. Jeho jadro tvorí generická trieda `ChartDataModel`. Umožňuje zadávanie dát, kontrolu použitého dátového typu vzhľadom na daný druh grafu, a obsahuje metódu zabezpečujúcu transformáciu dát do formátu JSON vhodného tvaru.



Obrázok 6.2: Triedy dátového modelu

Vzhľadom na požiadavku kontroly kombinácie dátového typu nezávislej premennej a druhu grafu, boli vytvorené triedy troch konkrétnych potomkov. Konkrétne sa jedná o triedy `NumberChartModel`, `StringChartModel` a `DateChartModel`. Každá z nich predstavuje implementáciu pre jeden dátový typ. Tento postup bol použitý, aby bolo možné za behu aplikácie zistiť dátový typ nezávislej premennej grafu. Zistený typ je následne použitý na kontrolu vhodnosti pre zvolený druh grafu vykonanú metódou `isAllowedForChartType()`. Po úspešnej kontrole je možné pristúpiť k transformácii dát do formátu JSON, ktorý bude použitý pre vykreslenie grafu v jazyku javascript.

Úloha transformácie dát do formátu JSON je závislá na type grafu. Rovnako aj pridávanie hodnôt si vyžaduje pri určitých druhoch grafov kontrolu alebo uloženie dodatočnej informácie. Tá je potrebná, aby bolo možné korektne zobraziť dáta pri vykreslení viacerých závislých premenných. Pri implementácii týchto činností som použil návrhový vzor stratégia. Metódu zabezpečujúcu výstup do JSON a metódu pridávajúcu novú hodnotu do dát grafu som vyčlenil do rozhrania `ChartStrategy`. Následne som vytovril implementácie tohto rozhrania zohľadňujúce požiadavky jednotlivých druhov grafov.

Tento prístup naviac umožňuje v budúcnosti pridanie ďalších druhov grafov.

6.5 Zadávanie dát

Dáta, ktoré bude grafový komponent zobrazovať sú zadávané pomocou značky `series`. Tá predstavuje skupinu hodnôt reprezentujúcich jednu závislú premennú. Komponent umožňuje zobrazovať aj závislosť viacerých premenných, pridaním ďalšej značky.

Ako je spomenuté v návrhu 5.5, dáta je možné zadávať dvoma spôsobmi. Prvým z nich je vytvorenie inštancie dátového modelu v ManagedBean a jeho odovzdanie pomocou atribútu `value`. Tento spôsob bol použitý aj v ukážke 6.1.

Druhou možnosťou, ktorá odstraňuje potrebu vytvárať inštanciu dátového modelu, je použitie iteračného komponentu `<a4j:repeat>`, ktorý je súčasťou knižnice RichFaces. Tento komponent na základe iterácie nad vstupným zoznamom objektov vytvorí jednotlivé body grafu. Je potrebné špecifikovať, ktoré premenné objektov budú zobrazené na osi x a y. Tento postup je znázornený na ukážke 6.2.

Ukážka 6.2: Zadávanie dát grafu iteráciou

```
<lm:chart>
  <lm:series type="line">
    <a4j:repeat value="#{bean.list}" var="item">
       <lm:point x="#{item.date}" y="#{item.price}">
    </a4j:repeat>
  </lm:series>
</lm:chart>
```

Pri použití tohto postupu je dátový model vytvorený automaticky. Zabezpečuje to `renderer` komponentu pri vykreslení. Spôsoby zadávania dát je možné vzájomne kombinovať v jednom grafe pri použití rovnakého dátového typu.

6.6 Obsluha udalostí

Komponent umožňuje obsluhu udalostí vzniknutých v dôsledku užívateľskej interakcie. Javascriptové obslužné funkcie pre konkrétne udalosti je možné špecifikovať v atribútoch značky `chart`. Obslužným funkciám je odovzdaný objekt nesúci informácie o vzniknutej udalosti. Obsahuje nasledujúce informácie o bode grafu, ktorého sa udalosť dotýka:

- seriesIndex – index skupiny bodov vyjadrujúcich závislosť jednej premennej
- pointIndex – index konkrétneho bodu v rámci skupiny
- x – hodnota nezávislej premennej
- y – hodnota závislej premennej

Pre udalosti `dataclick` a `dragstop`, ako uvádza návrh 5.6, je naviac možné špecifikovať obsluhu aj na serveri. Obsluhou je metóda akceptujúca parameter triedy `DataClickEvent` alebo `DragStopEvent`, taktiež sa definuje v atribúte značky `chart`. Pri vzniknutí udalosti na klientovi je odoslaná AJAX požiadavka na server a komponent zabezpečí vyvolanie takto špecifikovanej obsluhy.

6.7 Ukážkova aplikácia

Ako súčasť mojej bakalárskej práce som vytvoril aj aplikáciu, ktorá demonštruje použitie vytvoreného komponentu. Jej ukážka je zobrazená na obrázku 6.3. Aplikácia zobrazuje predpokladané zisky a výdaje oddelení firmy počas niekoľkých mesiacov, ktoré je možné upravovať pohybom myši. Na základe príjmov a ziskov je automaticky vypočítaný celkový stav oddelení v daných mesiacoch a taktiež ich podiel na celkovom zisku a výdajoch firmy. Aplikácia demonštruje nasledujúce vlastnosti komponentu:

- podporované druhy grafov,
- možnosti zadávani dát,
- obsluhu udalosti na serveri.

Obrázok 6.3: Ukážka demo aplikácie komponentu

Kapitola 7

Testovanie

Po implementácii bol komponent testovaný na serveri Glassfish 3.1.2 a JBoss AS 7.1.1 spôsobom popísaným v tejto kapitole. Okrem toho som vytvoril dva automatizované testy pomocou technológie Arquillian. Automatizované testy sú zamerané na obsluhu udalostí ako na klientovi tak aj na serveri.

7.1 Manuálne testovanie

Počas vývoja komponentu som priebežne testoval implementovanú funkčnosť. Využíval som k tomu aplikáciu, ktorá obsahovala grafy s rôznou konfiguráciou. Testoval som korektnosť zmeny konfigurácie vlastností grafov, zadávania dát a obsluhy udalostí. Všetky chyby zistené počas testovania boli odstránené.

7.2 Automatizované testy

Automatizované testy boli vytvorené pomocou technológie Arquillian a jej rozšírenie Graphene integrujúce Selenium WebDriver. Táto technológia spolu s uvedenými rozšíreniami umožňuje automatizovane nahrať na server testovanú aplikáciu, spustiť prehliadač, simulovať užívateľskú interakciu a overiť odozvu aplikácie.

Pomocou nej som vytvoril testy, ktoré overujú korektnosť obsluhy udalosti kliknutia na bod v grafe pre všetky podporované druhy grafov. Testy pre každý druh grafu a obsluhu na klientovi a na serveri majú nasledujúcu štruktúru:

- vyhľadanie elementu canvas testovaného grafu na stránke,
- kliknutie na špecifikovanú pozíciu canvas. Obsluha udalosti vypíše na stránku text o bode grafu, na ktorý bolo kliknuté,
- porovnanie textu z očakávaným výsledkom.

Kapitola 8

Záver

Cieľom tejto bakalárskej práce bolo vytvoriť komponent vykresľujúci grafy pre framework JSF. Dôraz bol kladený na podporu obsluhy užívateľskej interakcie a rozšíriteľnosť komponentu.

Vo výsledku bol implementovaný komponent s využitím nástroja Component Development Kit, zapúzdrujúci javascriptovú knižnicu jqPlot, ktorá zabezpečuje samotné vykreslenie. Komponent umožňuje programátorom, ktorí ho budú používať, tvorbu čiarových stĺpcových a koláčových grafov bez písania kódu v jazyku javascript. Návrh komponentu počíta s možnosťou pridania podpory pre ďalšie druhy grafov. Umožňuje taktiež obsluhu udalostí, vzniknutých po interakcii užívateľa, na klientovi a aj na strane servera. Za najvýraznejšie črty komponentu považujem rovnakú hierarchickú štruktúru značiek pre tvorbu rôznych druhov grafov a deklaratívny spôsob zadávania dát. Tieto charakteristiky odlišujú moje riešenie od súčasných dostupných komponentov.

Uvedené vlastnosti komponentu sú demonštrované pomocou ukážkovej aplikácie, ktorá bola tiež vytvorená ako súčasť tejto bakalárskej práce. Naviac, projekt ešte obsahuje ďalšiu aplikáciu, ktorá umožňuje zmenu konfigurácie komponentu priamo na stránke.

Po úprave drobných formálnych požiadaviek bude komponent zaradený medzi nové komponenty knižnice RichFaces. Na vývoji projektu by som chcel ďalej pokračovať počas štipendijného programu Google Summer of Code, do ktorého som sa prihlásil. Plánovaným vylepšením projektu je rozšírenie sady automatizovaných testov a pridanie podpory nových druhov grafov.

Literatura

[1] Arquillian · Write Real Tests. [online] [cit. 2013-05-10].
URL http://arquillian.org/invasion/

[2] Chart — OpenFaces documentation. [online] [cit. 2013-05-10].
URL http://openfaces.org/documentation/developersGuide/chart.html

[3] Elycharts. [online] [cit. 2013-05-10].
URL http://elycharts.com/

[4] flotr2. [online] [cit. 2013-05-10].
URL http://www.humblesoftware.com/flotr2/

[5] JpGraph - Most powerful PHP-driven charts. [online] [cit. 2013-05-10].
URL http://jpgraph.net/

[6] Maven - What is Maven? [online] [cit. 2013-05-10].
URL http://maven.apache.org/what-is-maven.html

[7] Open Flash Chart. [online] [cit. 2013-05-10].
URL http://teethgrinder.co.uk/open-flash-chart/

[8] PrimeFaces. [online] [cit. 2013-05-10].
URL http://primefaces.org/

[9] RichFaces - JBoss Community. [online] [cit. 2013-05-10].
URL http://www.jboss.org/richfaces

[10] Baranovskiy, D.: gRaphaël—Charting JavaScript Library. [online] [cit. 2013-05-10].
URL http://g.raphaeljs.com/

[11] Bostock, M.: D3.js - Data-Driven Documents. [online] [cit. 2013-05-10].
URL http://d3js.org/

[12] Burns, E.; Schalk, C.: *JavaServer Faces 2.0, The Complete Reference*. JavaServer Faces 2.0: The Complete Reference, McGraw-Hill Education, 2009, ISBN 9780071625104.

[13] Coglan, J.: Bluff: Beautiful graphs in JavaScript. [online] [cit. 2013-05-10].
URL http://bluff.jcoglan.com/

[14] Goncalves, A.: *Beginning Java^{TM} EE 6 Platform with GlassFish^{TM} 3: From Novice to Professional*. Expert's voice in Java technology, Apress, 2009, ISBN 9781430219545.

[15] Jendrock, Eric and Evans, Ian and Gollapudi, Devika and Haase, Kim and Srivathsa, Chinmayee: *The Java EE 6 Tutorial: Basic Concepts*. Upper Saddle River, NJ, USA: Prentice Hall Press, 2010, ISBN 0137081855, 9780137081851.

[16] Katz, M.; Shaikovsky, I.: *Practical RichFaces*. Expert's voice in Java technology, Apress, 2011, ISBN 9781430234494.

[17] Leonello, C.: jqPlot Charts and Graphs for jQuery. [online] [cit. 2013-05-10]. URL http://www.jqplot.com/index.php

[18] Neuhaus, I.: CanvasXpress. [online] [cit. 2013-05-10]. URL http://canvasxpress.org/

Příloha A

Obsah CD

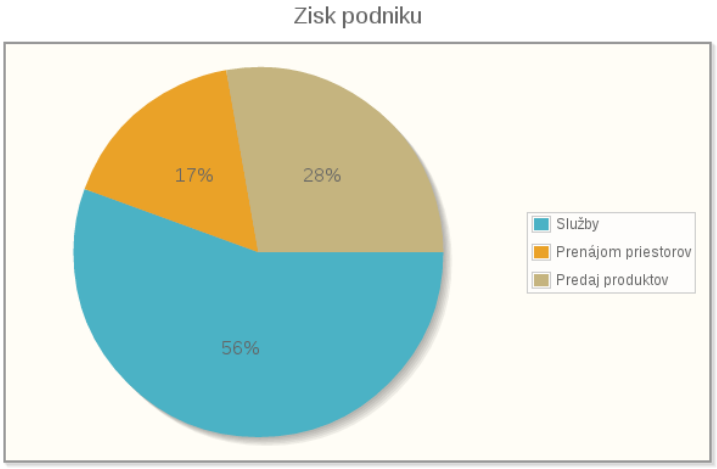
`doc/`	adresár obsahujúci text technickej správy a obrázky
`src/`	zdrojové kódy vytvorené počas bakalárskej práce
	`ChartComponent` komponent pre vykresľovanie grafov
	`Customization` aplikácia pre konfiguráciu komponentu
	`Demo` ukážková aplikácia popísaná v kapitole 6.7
`target/`	výsledky prekladu projektov z adresára `src`
`report.pdf`	technická správa
`README`	súbor popisuje obsah CD a tiež obsahuje informácie potrebné pre preklad zdrojových súborov

Aplikácia `Demo` je dostupná online na adrese
https://chartdemo-lukasmacko.rhcloud.com/Demo-1.0-SNAPSHOT/
Aplikácia `Customization` je dostupná online na adrese
https://chartdemo-lukasmacko.rhcloud.com/Customization-1.0-SNAPSHOT/

Příloha B

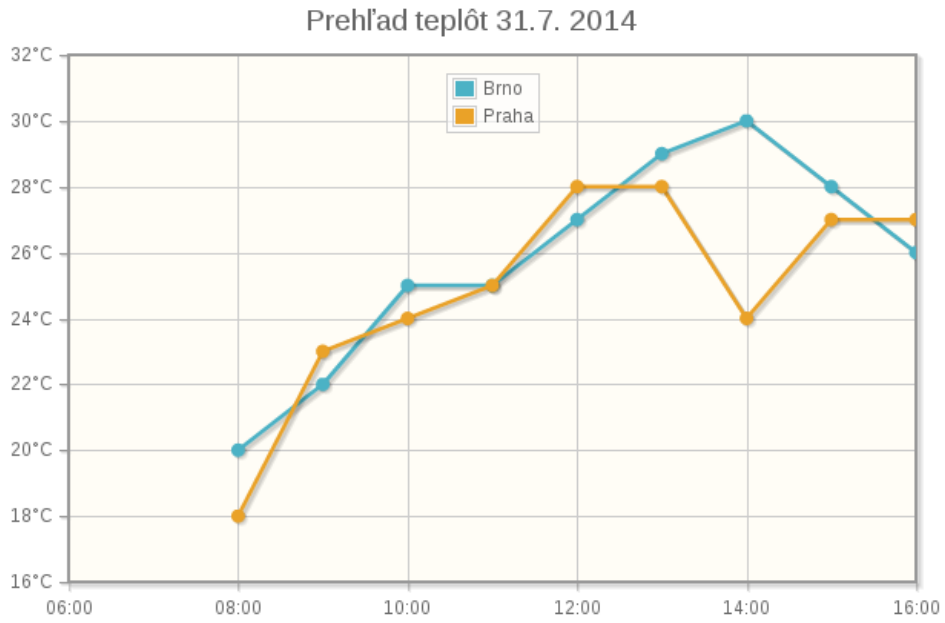
Príklady použitia komponentu

Táto kapitola ukazuje niekoľko príkladov grafov vytvorených pomocou komponentu. Obrázky sú doplnené úsekmi kódu, ktorý bol použitý na ich vykreslenie.



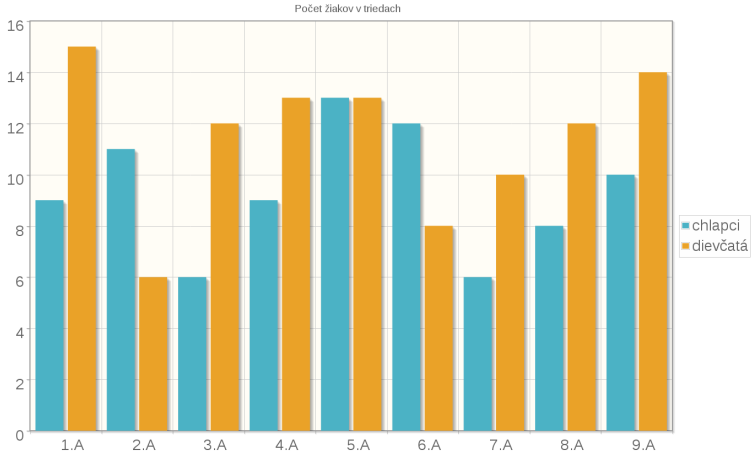
Obrázok B.1: Koláčový graf

```
<lm:chart id="pieChart" title="Zisk␣podniku">
   <lm:series value ="#{myBean.pieChartModel}" type="pie" />
   <lm:legend/>
</lm:chart>
```



Obrázok B.2: Čiarový graf

```
<lm:chart id="temperature" title="Prehlad teplôt 31.7. 2014" >
   <lm:series value="#{data.brno}" type="line" label="Brno"/>
   <lm:series value="#{data.praha}" type="line" label="Praha"/>
   <lm:xaxis format="%H:%M"  />
   <lm:yaxis format="%d°C" />
   <lm:legend position="top"/>
</lm:chart>
```



Obrázok B.3: Stĺpcový graf

```
<lm:chart title="Počet žiakov v triedach" >
  <lm:series type="bar" label="chlapci">
    <a4j:repeat value="#{myBean.chlapci}" var="ch">
       <lm:point x="#{ch.trieda}" y="#{ch.pocet}"/>
    </a4j:repeat>
  </lm:series>

  <lm:series type="bar" label="dievčatá">
    <a4j:repeat value="#{myBean.dievcata}" var="d">
       <lm:point x="#{d.trieda}" y="#{d.pocet}"/>
    </a4j:repeat>
  </lm:series>
  <lm:legend placement="outside"/>
</lm:chart>
```