VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Absolvování individuální odborné praxe

Individual Professional Practice in the Company

2014

Zbyněk Fišer

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Zbyněk Fišer**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Absolvování individuální odborné praxe
Individual Professional Practice in the Company

Zásady pro vypracování:

1. Student vykoná individuální praxi ve firmě: GIRITON Systems s.r.o.
2. Struktura závěrečné zprávy:
a) Popis odborného zaměření firmy, u které student vykonal odbornou praxi a popis pracovního zařazení studenta
b) Seznam úkolů zadaných studentovi v průběhu odborné praxe s vyjádřením jejich časové náročnosti
c) Zvolený postup řešení zadaných úkolů
d) Teoretické a praktické znalosti a dovednosti získané v průběhu studia uplatněné studentem v průběhu odborné praxe
e) Znalosti či dovednosti scházející studentovi v průběhu odborné praxe
f) Dosažené výsledky v průběhu odborné praxe a její celkové zhodnocení

Seznam doporučené odborné literatury:

Podle pokynů konzultanta, který vedl odbornou praxi studenta.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Mgr. Miloš Kudělka, Ph.D.**

Konzultant bakalářské práce: Jan Gřeš, MSc.

Datum zadání: 01.09.2013
Datum odevzdání: 07.05.2014

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 *Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.*

V Ostravě 7. května 2014 .....

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 7. května 2014 .....

Rád bych poděkoval svémů mentorovi Janu Gřešovi, za možnost vykonání praxe ve firmě Giriton Systems s.r.o. a cenné rady a ochotu v průběhu bakalářské praxe. Dále bych chtěl poděkovat kolegům a spolužákům na praxi, za skvělý kolektiv a spolupráci.
Nakonec bych chtěl poděkovat svému vedoucímu bakalářky Miloši Kudělkovi, za pomoc a cenné rady při psaní bakalářské práce.

Abstrakt

Obsah bakalářské práce se zabývá popisem vykonané individuální odborné praxe ve firmě GIRITON Systems s.r.o. První část popisuje odborné zaměření firmy a firmu samotnou. Dále jsou popsány technologie, použité při plnění zadaných úkolů. Třetí část se zabývá informačním systémem GIRITON. Jsou zde vysvětleny základní funkce systému a jeho účel a použití v reálném prostředí. Jádrem této bakalářské práce je postup řešení zadaných úkolů v průběhu praxe. Každý úkol obsahuje zadání, postup řešení a závěr. V dalších dvou sekcích jsou popsány využité a chybějicí znalosti. Nakoneci bakalářské práce uvádím závěrečné zhodnocení o probíhané praxi

Klíčová slova: GIRITON Systems s.r.o, Java, Vaadin, framework, odborná praxe, design, komponenta, uživatelské rozhraní, implementace

Abstract

The content of the thesis is about individual professional practice in company GIRITON Systems s.r.o.. The first part describes the professional focus of the company and the company itself. Next part describes technologies used to accomplish assigned tasks. The third part deals with the information system GIRITON. There are explained basic functions of the system and its purpose and use in the real enviroment. The core of this thesis is about the procedure of solving given tasks during practice. Each task includes a description, process solution and conclusion. In the next two parts I describe used knowledge during the practice, and knowledge gaps. At the end of thesis, I do the final evaluation of the ended practice.

Keywords: GIRITON Systems s.r.o, Java, Vaadin, framework, professional practice, design, component, user interface, implementation

Seznam použitých zkratek a symbolů

RIA	–	Ritch internet application
GWT	–	Google web toolkit
DOM	–	Document object model
HTML	–	Hypertext markup language
XML	–	Extensible markup language
Sass	–	Syntactically Awesome Style Sheets
CSS	–	Cascade Style Sheets
RFID	–	Radio Frequency Identification
PDA	–	Personal data access
SVN	–	Subversion

Obsah

1 Úvod

Vykonání bakalářské práce stylem odborné praxe jsem si vybral z mnoha důvodů. Hlavním důvodem byla možnost získání praxe v oboru. Dalším důvodem byl výběr témat bakalářských prací. Ty které mě zaujali již byli zabrané. Na možnosti odborné praxe mě navíc zaujala šance budoucí kariéry. Dále jsem chtěl zjistit jak probíhá vývoj informačních systémů v týmu a naučit se spolupracovat s ostatními v týmu. Tyto zkušenosti se nedají získat jinak než v praxi.

2 Obecné informace

2.1 O firmě

Firma GIRITON Systems s.r.o. je mladá firma, která vnikla na český trh roku 2011 pod vedením zakladatele Jana Gřeše. Zabývá vývojem moderního podnikového software na míru konkrétnímu zákazníkovi. Původně byla firma zaměřená pouze na vývoj informačního systému pro řízení výroby, avšak časem se portfolio rozšířilo.

2.2 Popis odborného zaměření firmy

Firma GIRITON Systems tvoří variabilní cloudové systémy zpracovávající všechna zákazníkem požadovaná data. Od kontroly docházky, sběr dat v terénu a ve firmě, kontroly skladů až po celkové řízení výroby. Zákazník má možnost díky moderním technologiím přistupovat ke svým datům odkudkoli na internetu.
Velkým potenciálem této firmy je využití levných tabletů formou plně funkčních PDA stanic komunikujících se serverem. Tablety mají připojenou čtečku čárových kódu.
Koncový pracovník jednoduše načte čárový kód prováděné operace a systém data v reálném čase zpracuje. Díky tomu má zákazník okamžitý přehled o aktuálním stavu objednávek, skladů nebo docházce zaměstnanců ve své firmě. Systémy navíc nabízí generování ročně definovatelných reportů, grafů a výpisů v určitém časovém období, přesně podle požadavků zákazníka.
Systémy jsou vyvíjeny v moderním Java Vaadin frameworku pro tvorbu webových aplikací.

2.3 Pracovní zařazení studenta

Při vykonávání praxe jsem si vyzkoušel práci v různých částech vývoje softwaru, jakož jsou např.: návrh designu, vývoj apliakce, testování systému atp.. Avšak moje hlavní pozice ve firmě byla Vaadin software developer.

3 Použité technologie

- Vaadin framework
- SeleniumHQ
- Jasper Reports
- Syntactically awesome style sheets

3.1 Vaadin Framework

Primární technologií, kterou jsem používal, je *Vaadin* Java framework pro vývoj webových aplikací. Je navržen pro tvorbu bohatých internetových aplikací (RIA) běžící v internetovém prohlížeči. Servrová architektura je navržena s důrazem na bezpečnost webové aplikace. Spolu s modelem komponent umožňuje jednoduchý a rychlý vývoj.
Celá webová apliakce je psaná v programovacím jazyce Java. Ten se následně překládá do HTML a javascriptu. Pro překlad se používá nástroj *Google web toolkit*.
Oproti ostatním webovým frameworkům je Vaadin díky své architektuře navržen ve větší míře pro vývoj RIA aplikací. Programovací model se podobá vývoji desktopové aplikace s logikou eventů a jejich posluchačů. Ke komunikaci se serverem se využívá AJAX. Vaadin framework se stylem programování podobá Java knihovně *Swing*. Díky své komplexnosti není nejlepším kadidátem při tvorbě jednoduchých webových stránek.

3.2 SeleniumHQ

SeleniumHQ je technologie automatizující práci v internetových prohlížečích. Využívá se především k dynamickému testování webových aplikací. Jednoduše řečeno, simuluje akce skutečného uživatele. Umožňuje nadefinování přesných pohybů ukazatele myši v prostředí webového prolížeče.
Orientace ukazatele se provádí na základě vyhledávání v DOM struktuře dokumentu. K vyhledávání se využívá dotazovací jazyk *XPath* pro vyhledávání v XML dokumentech.
SeleniumHQ není knihovna pouze pro jazyk Java. Podporuje i ostatní programovací jazyky jako jsou například C# a Python.

3.3 iReport Designer

IReport Designer je grafický editor usnadňující tvorbu Jasper výpisů. V jádru obsahuje knihovnu *JasperReports Library*, která se stará o generování výpisů do různých typů for-

mátu. Umožňuje generování výpisu do HTML, PDF, docx, OpenOffice a mnoho dalších. Formát výpisu se definuje pomocí XML elementů, které se ukládájí do souborů s příponou *.jrxml*. Report se poté prožene přes jasper report knihovnu, která se postará o naplnění reportu daty z databáze. Následně se report vytiskne do požadovaného formátu.

3.4 Sass - Syntactically awesome style sheets

Zkráceně Sass je CSS 3 preprocessor. Vzhled webových aplikací se definuje pomocí CSS maker. Jak aplikace roste, roste i CSS kód a stává se nepřehledným. Preprocessory řeší tento problém. Doplňují čisté CSS o nové funkce a možnosti. Sass přidává do čistého CSS dědičnost, proměnné, jednoduché početní operace a mnoho dalších funkcí, usnadňující vývoj a správu Kaskádových stylů webové stránky. Před posláním požadované stránky uživateli, se provede kompilace a vytvoří se jeden velký CSS soubor. Ten se pošle spolu s požadovanou stránkou uživateli.
Nevýhoda je nustnost překladu Sass souborů na čisté CSS a proto i delší doba odpovědi uživateli. V produkční verzi aplikace se překlad provede pouze při spuštění aplikace, avšak při vývoji se překlad provádí při každém dotazu.

4 Popis vyvíjené aplikace

V průběhu praxe jsem spolupracoval na cloudovém informačním systému GIRITON. GIRITON je nástroj pro přehlednost procesu výroby a správu zakázek. Je určen především pro firmy, zabývající se výrobním a zpracovatelským průmyslem. Díky cloudovému řešení je systém dostupný z prostředí internetu.
GIRITON systém je velmi komplexní aplikace skládající se ze spousty menších i větších modulů.

Vlastnosti systému GIRITON: [8]

- Řízení a monitorování výroby
- Použití čárových kódů a RFID čipů
- Monitorování skladu a objednávek
- Podpora exportu podkladů pro mzdy
- Sledování docházky

4.1 Řízení a monitorování výroby

Modul pro řízení a monitorování výroby je hlavní součást systému GIRITON. K řízení výroby se využívá tabletů a chytrých telefonů, které jsou bezdrátově připojeny do sítě GIRITON. V tabletech je nainstalována android aplikace pro tablety, přes kterou se zaměstnanci přihlásí ke svému účtu ve firmě.
Po vykonání pracovních úkolů, zadá pracovník jejich čísla do tabletu. Data se okamžitě odešlou na server GIRITON. Zde jsou data zpracována a uložena do databáze. Pokud pracovník dostává úkolovou mzdu, obratem se mu na displeji zobrazí vydělaná částka peněz. Při odesílání dat se do systému uloží také kdo provedl jaký úkol. Tyto údaje využívá modul pro výpočet úkolové mzdy zaměstnance. V případě poruchy systému se tablet přepne do offline režimu. Data se ukládají do interní paměti tabletu a při prvním připojení do sítě se data okamžitě odešlou na server.

4.2 Čárové kódy a RFID čipy

Aplikace Řízení výroby umožňuje vkládání čísel výrobních úkolů za pomocí čtečky čárových kódů. Výhodou použití této čtečky je omezení faktoru lidské chyby při zadávání čísel výrobních procesů do systému. Usnadňuje evidenci materiálu, jejich výdeje a příjmy do skladu. Dále umožnujě export a evidenci zákaznických objednávek.
RFID čipy se využívají pro monitorování docházky zaměstnanců ve firmě.

4.3 Monitorování skladu a objednávek

GIRITON systém nabízí snadnou správu a přehled o aktuálním stavu skladů a objednávek. Umožňuje nastavení upozornění v případě nízkého stavu materiálu ve skladu. Podporuje nastavení z jakých materiálů a polotovarů se produkt skládá.
Systém dále umožňuje přesné monitorování a správu objednávek. V případě nově vystavené objednávky systém vyhledá, zda již nejsou některé produkty na skladě, a podle zjištěných informací automaticky vytvoří nový výrobní úkol.
Aplikace *Řízení firem* používaná v tabletech umožňuje příjem a výdej materiálu do skladu a obsahuje agendu pro expedici objednávek.

4.4 Podpora exportu podkladů pro mzdy

GIRITON systém podporuje plnou správu zaměstnanců ve firmě. Poskytuje detailní přehled o provedených úkolech zaměstance. Na základě těchto informací, poskytuje podporu pro export podkladů pro výpočet úkolové mzdy.
V případě použití docházkového modulu, podporuje i výpočet časové mzdy zaměstnance.

4.5 Sledování docházky

Volitelným modulem GIRITON systému je správa a sledování docházky. Tento modul využívá speciální docházkovou android aplikaci, která pomocí RFID karet zaznamenává docházku zaměstnanců. S pomocí tohoto modulu, je možný výpočet časové mzdy zaměstnance. GIRITON systém nabízí i pouze docházkový systém bez modulů pro řízení výroby.

5 Úkoly zadané v průběhu praxe

Před samotným začátkem praxe jsem dostal za úkol naučit se základy práce s Vaadin frameworkem pro tvorbu webových aplikací. Poté jsem byl vržen rovnou do praxe.
V průběhu jsem dostal spoustu menších i větších úkolů, týkajících se vývoje aplikace a jejích modulů. Na některých jsem spolupracoval s více kolegy a některé jsem řešil samostatně. Rozhodl jsem se popsat jen ty nejpodstatnější a nejzajímavější.

5.1 Seznam vykonaných úkolů

5.1.1 Úkoly které budu dále popisovat:

- PDA simulátor
- Využití *CustomLayout* komponenty
- Vývoj a úprava uživatelského rozhraní
  - Formulář přihlášení
  - Editor kódu skladových akcí
  - Do PDA simulátoru přidat kartu objednávky
  - Přeprogramovat *VEditorTabMaterial* s využitím *CustomLayout* komponenty
- Vytvoření defaultního stylu tlačítek a jejich barevné variace
- Optimalizace přepínání agend
- Integrační testy
  - Report editor
  - Přidání a odebrání materiálu
- Jasper reporty
- Modul desktopových upozornění

5.1.2 Ostatní úkoly:

- Vytvořit *CalendarWidgetProvider* zobrazující počet naplánovaných úkolů
- Upravt progres bar dialog systémových akcí - úprava vzhledu

- Otestovat *Dcharts* Vaadin add-on
- Implementovat metodu pro zjištění DPI obrazovky zařízení.
- Dopsat anotace pro třídění a vyhledávání entit v databázi
- Vytvořit integrační test přidání nového pracovního příkazu
- Upravit výšku a styl řádků tabulek Multitable 2 a 3
- Vyhledat ceny licencí Jasper Reportů
- V agendě výplaty zabarvit víkendové řádky tabulek šedě
- Hledání vhodného čárového kódu, který je větší než *EAN13*
- Přepsat komponentu pro výběr měsíce s využitím komboboxů
- Analyzovat problém zacyklení aplikace při odstranení výrobního příkazu
- Upravit styl mobilní verze GIRITON systému
- Otestovat posílání výrobních příkazů z android aplikace na server
- Otestovat funkci *Animator* Vaadin add-onu pro použití v mobilní verzi aplikace
- Přidat funkci automatického označení nově přidané akce ve skladu
- Opravit zalamování řádků v chatu
- Úpravit stylu příchozích a odchozích zpráv v chatu
- Opravit vykreslování vzorového čísla čárového kódu při jeho výběru v report editoru
- Vyhledat a opravit chybu proč se nenahrávají nastavené hodnoty čárových kódů v report editoru
- Upravit integrační test, tak aby nevyužíval funkci čekající na odpověď prohlížeče
- Pokračovat s vypisováním funkcí různých docházkových systémů na trhu

5.2 Řešení vykonaných úkolů

5.2.1 PDA simulátor

Zadání

Hlavní aplikace přijímá data z různých tabletů a PDA. Pomocí android aplikace se posílají informace o stavu výroby na server, který je následně zpracuje. Mým úkolem bylo spolupracovat na vytvoření Vaadin verze této mobilní apliakce. PDA simulátor by měl mít stejné funkce a vzhled jako originální Android verze. Funkční simulátor se pak vloží do systému.

Postup řešení

Nejprve jsem pracoval na verzi oddělené od hlavní aplikace a až po implementaci většiny uživatelského rozhraní se PDA simulátor importoval. Využívá se v agendě pro správu výrobních příkazů.

Jedna z agend PDA simulátoru, kterou jsem vytvářel, zobrazuje výpis sum odvedené práce v zadaném datu. Skládá se z výběru data a textového pole pro výpis. Výběr data je v Android aplikaci tvořen třemi textovými poli pro den, měsíc a rok. Textová pole mají ovládací tlačítka plus a mínus a potvrzovací tlačítko OK.

Jelikož tato komponenta defaultně ve Vaadin Frameworku není, napsal jsem vlastní implementaci. Nejprve jsem vytvořil třídu *NumberPicker*, obsahující univerzální textové pole s plus a mínus tlačítky. Třídě *NumberPicker* se dá nastavit horní a spodní hranice rozsahu. Tuto menší komponentu jsem použil celkem třikrát v celkové implementaci komponenty pro výběr datumu.

Komponentu pro výběr datumu jsem nazval *DatePicker*. Třída *DatePicker* má na starost změnu číselného rozrahu komponent třídy *NumberPicker*. V závislosti na měsíci se mění maximální počet dnů.

Datum je spravován pomocí třídy *LocalDate* od *JodaTime* 2.2 API pro programovací jazyk Java.

Další částí PDA simulátoru, na které jsem spolupracoval, bylo Menu. Menu se skládá z odkazů na různé agendy a tlačítka zpět. Po společné týmové domluvě jsme zvolili modální implementaci menu. Díky tomu nemusíme řešit, v jaké agendě jsme byli před vstupem do menu. Po výběru položky v menu se změní aktuálně viditelná agenda. Ta se nahraje do předpřipraveného kontejneru jako kořenová komponenta a zobrazí se uživateli.

Položky v menu jsou uloženy jedné VerticalLayout komponentě. Mají jednotný styl a velikost. Tlačítko Zpět je oddělené od kontejneru pro položky menu aby nedocházelo k

vzájemnému ovlivnění zarovnávání komponent.
Po dokončení implementační části úkolu jsem pokračoval s úpravou vzhledu. Všechny styly jsou napsané pomocí *Sass* stylusu. Vaadin Framework má sám o sobě spoustu předdefinovaných palet vzhledů které jsou perfektně optimalizovány, avšak problém nastává u specifických požadavků na vzhled.
PDA simulátor se vzhledově kompletně liší od stylu Vaadin frameworku. Například vzhled *Button* komponenty je tvořen z pevně vysokých obrázků, které nejdou upravit přidáním stylu na komoponentu. Pro upravení buttonu jsem musel použít metodu *setPrimaryStyleName*, která přepíše všechny defaultní styly Vaadin frameworku a umožní nastylovat celé tlačítko znovu.

Závěr

PDA simulátor byl mým prvním úkolem po nástupu do praxe. Nejtěžší na tomto úkolu bylo zvyknout si na formátování Vaadin komponent v různých rozloženích (*Layout*). Po čase práce s tímto frameworkem jsem se naučil spoustu triků jak dosáhnout požadovaného vzhledu.

5.2.2 Využití Custom Layout komponenty

Zadání

Vaadin Framework využívá GWT pro překlad kódu do webového rozhraní. Nevýhoda je množství generovaného HTML kódu i pro velmi jednoduché formuláře. *CustomLayout* komponenta umožňuje vytvořit si vlastní HTML šablonu. Mým úkolem bylo otestovat práci s touto Vaadin komponentou a zhodnotit její použití.

Postup řešení

Při vytváření *VEditorTabPartner* záložky v agendě Partneři, se mi nepodařilo správně zarovnat Vaadin komponenty přesně podle požadavů. Můj mentor mi poradil, abych zkusil otestovat funkce *CustomLayout* komponenty, která mi umožní psát čistý HTML kód.
Pro testovací účely této komponenty jsem dostal za úkol předělat tuto záložku.
Ve Vaadin dokumentaci jsem si našel jednoduchou ukázku použití *CustomLayout* komponenty. Pro testovací účely jsem si vytvořil jednoduchou HTML šablonu. Šablona se bude chovat jako tělo *CustomLayout* komponenty.

```
<div>
    Test Label:
    <div location="testLabel"></div>
</div>
```

Výpis 1: CustomLayoutTest.html

Atribut *location* označuje místo, kam se má vložit testovací *Label* komponenta. Při vkládání do *CustomLayout* komponenty, se musí nadefinovat totožný *location* identifikátor. Díky tomu se Vaadin komponenta vloží na správnou pozici v šabloně. V konstruktoru *CustomLayout* komponenty se uvádí název nebo cesta k šabloňe. Vaadin defaultně hledá šablonu v adresáři templates.

```
class VEditorTabPartner extends VEditorTab {
    ...
    Label myLabel = new Label("Everything␣works");
    CustomLayout content = new CustomLayout("CustomLayoutTest.html");

    public VEditorTabPartner() {
        content.addComponent("testLabel", myLabel);
        ...
    }
}
```

Výpis 2: CustomLayout komponenta

Po tomto kroku už bylo vše jednoduché. Nepotřeboval jsem žádné Layout komponenty. Java kód i výsledný HTML kód byl mnohonásobně kratší a o poznání jednodušší. První verze *VEditorTabPartner* záložky se skládala z HTML *div* tagů s využitím CSS stylů pro správné zarovnávání. Po konzultaci s mentorem jsem dostal za úkol předělat momentální řešení *VEditorTabPartner* záložky. Motivace využití *CustomLayout* komponenty je, umožnit v budoucnu zákazníkovy upravovat rozmístění vstupů v šabloně pomocí HTML. Aby byl kód šablony co nejjednodušší, musel jsem omezit co použití CSS stylů při formátování komponent.
Abych dosáhl co nejpřesnějšího požadovaného vzhledu bez použití CSS stylů, použil jsem HTML tabulku.
Každá Vaadin komponenta má defaultně *Caption* atribut. *Caption* atribut nastavuje nadpis komponenty. Generuje se nad svým vlastníkem jako jednoduchá *Label* komponenta. Bohužel se nedá nastavit pozice nadpisu. *VEditorTabPartner* je formulář, který přidává firemní partnery do databáze aplikace. Podle návrhu designu se mají popisky vstupních

polí formuláře zobrazovat nalevo od jejich vlastníků. *FormLayout* Vaadin komponenta řeší tento problém. Vzhledem k použití *CustomLayout* komponenty, je zmíněné řešení nežádoucí.
Dalším řešením by bylo vytvářet ke každé komponentě zvlášť *Label* komponentu. Takové řešení je zbytečně pracné. Nakonec jsem vytvořil třídu dědící z *CustomLayout* komponenty a přepsal metodu *AddComponent*. Tato upravená metoda zjistí typ vkládané komponentu a pokud je daná komponenta instancí třídy *TextField*, odstraní její caption a vytvoří z něj *Label* komponentu. Nově vytvořenou *Label* komponentu spolu s textovím polem přidá do výsledné HTML šablony.
V šabloně pak musím mít nadefinované kam se má vygenerovaná *Label* komponenta vložit s hodnotou v atributu location: *<id Komponenty>-label*.

Závěr

CustolLayout komponenta má své vyžití především při vytváření složitějších částí uživatelského rozhraní. Je pomocníkem hlavně tam kde Vaadin komponenty nestačí. Je to nezbytný doplněk Vaadin frameworku při tvorbě webových aplikací.

5.2.3 Formulář přihlášení

Zadání

Aplikace měla ze začátku dva kroky přihlášení. Prvním formulářem se přihlašovala firma a druhým samotný uživatel. Mým úkolem bylo sjednotit formuláře přihlášení do jednoho. Použít funkční desing formuláře přihlášení. Upravit vzhled.

Postup řešení

Výsledný formulář přihlášení by měl obsahovat tři textové pole pro přihlášení uživatele. První pro název firmy a další dva s jménem a heslem samotného uživatele.
Jako řešení jsem zvolil jeden hlavní Vertikální Layout, který obsahoval dva další Vertical Layouty pro obrázek a pro formulář přihlášení. Layout pro formulář přihlášení se skládal z dalších dvou *VerticalLayout* komponent. Jeden pro pole formuláře a druhý pro ovládací prvky (tlačítka). Tuto strukturu zanoření jsem zvolil z důvodu oddělení jednotlivých skupin komponent, pro snadnější manipulaci a zarovnávání. Všechny tyto komponenty mají nastavenou šířku na 100%. Pouze *Panel* komponenta, která zaobaluje kořenový *Layout*, má nastavené rozměry na pevno. Účelem bylo, aby se při změně šířky hlavního panelu, přizpůsobili ostatní komponenty uvnitř panelu.
Textové pole jsem zaobalil do *Placeholder* komponenty, která uvnitř textového pole zobrazuje text. Ten zmizí po začátku psaní do pole

Část s ovládacími prkvy obsahuje navíc odkazy na registraci a nápovědu při potížích s přihlášením. Nakonec jsem přidal pár CSS stylů, které doladily celkový vzhled přihlašovacího okna.

Závěr

Ačkoliv se tento úkol zdál velmi jednoduchý, upravit rozlozložení prvků dle požadovaného designu zabralo nejvíce práce. Výsledný formulář přihlášení působí uživatelsky přívětivějším dojmem než předchozí řešení.

5.2.4 Editor kódu skladových akcí

Zadání

Uživatel má možnost nastavit si detailní monitorování skaldu. Může přiřadit různé akce v případě nedostatku sledovaného materiálu. V případě vyvolání události se provede nadefinovaná akce a odešle se E-mail na uživatelem nastavené adresy. Akce se definují formou menších skriptů.
Mým úkolem bylo vytvořit vyskakovací okno pro editaci skriptů.

Postup řešení

Využil jsem Vaadin doplněk: komponentu *AceEditor*. *AceEditor* zvýrazňuje syntaxi nastaveného programovacího jazyka. Programovací jazyk se dá měnit pomocí roletky. Ve spodní části editačního okna se nachází tlačítka pro uložení a zrušení provedených změn.
Nejprve jsem si vytvořil instanci *VerticalLayout* komponenty jako kořenové rozložení pro Editor. Ten se skládá z vrchní *VerticalLayout* komponenty pro samotný editor a roletky pro výběr jazyka, která má nastavenou hodnotu *ExpandRation* = 1, a Horizontálního roložení ve spodní části kořenové *Layout* komponenty pro ovládací tlačítka „Uložit“ a „Zrušit“. Roletku pro výběr jazyka jsem zaobalil do komponenty *FromLayout*, aby se nadpis komponenty zobrazoval na pravé straně od komponenty. Protože *HorizontalLayout* dává komponentám stejně široký prostor, zarovnáním jednoho tlačítka doprava a druhého doleva jsem dosáhl, že obě tlačítka jsou zarovnané na střed vedle sebe.
Poté jsem pokračoval s implementováním funkčnosti. Roletce vybírající programovací jazyk jsem přidal *ValueChangeListener*, který při změně hodnoty nastaví *AceEditoru* jinou syntaxi vybraného jazyka.
Talčítku „Uložit“ jsem přidal *ButtonClickListener*, který po kliknutí na tlačítku uložil text editoru a typ jazyka do instance *CodeAsData*, která nastavení uložila do databáze,
a uzavře okno editoru. Tlačítko *Zrušit* pouze uzavře okno editoru.

Závěr

Nastavení roletky pro výběr jazyka byla nejtěžší část úkolu. Aby nedocházelo k chybám při ukládání změn, nesměl jsem opomenout aktualizovat všechny proměnné, které jsou se změnou jazyka spojené.

5.2.5 Do PDA simulátoru přidat uživatelské rozhraní pro objednávky

Zadání

V android aplikaci přibyla nová agenda Export objednávek. Dostal jsem za úkol přidat tuto agendu i do PDA simulátoru.

Postup řešení

Nejprve jsem vytvořil novou třídu pro tuto agendu s názvem *OrderPanel*, která dědí ze třídy *TopPanel*. Jako kořenové rozložení jsem zvolil *VerticalLayout* komponentu a dovnitř jsem přidal *TextField* pro číslo objednávky, *TextArea* pro adresu a *HorizontalLayout* zaobalující dvě tlačítka pro expedici zvolené objednávky.
Nakonec jsem nově vytvořenou agendu přidal do menu PDA simulátoru.

Závěr

Na PDA simulátoru jsem již předtím pracoval a přidat novou agendu pro mě nebyl problém. Při plnění tohoto úkolu jsem navíc opravil chybu v agendě vypisující sumy odvedené práce. Byla to má vlastní chyba, které jsem si předtím nevšiml. V komponentě DatePicker jsem špatně nastavoval datum denní části.

5.2.6 VEditorTabMaterial přeprogramovat s využitím CustomLayout komponenty

Zadání

Rozložení prvků ve formuláři přidávající materiál bylo nešikovné a nepřehledné. Úkolem bylo upravit rozložení komponent, tak aby působily jednoduše a přehledně. Při této příležitosti jsem měl použít *CustomLayout* komponentu, která v budoucnu bude použita na většinu částí aplikace.

Postup řešení

Stejně jako u desktopových aplikací, GIRITON aplikace nemá *roztahovatelnou* výšku. V případě překročení výšky prohlížeče, se komponentém bez pevné výšky nastaví minimální možná velikost. Původní rozložení formuláře zakrývalo spodní tabulku výpisu materiálu. Musel jsem dbát na rozložení komponent formuláře a využít chytře vertikální

prostor.
Nejprve jsem si na papír rozvrhl rozložení prvků. *VEditorTabMaterial* komponenta se skládá ze dvou sloupců. Jeden obsahuje formulář pro přidávání nového materiálu a druhá obsahuje komponentu s náhledem fotografie materiálu a polem pro přidání. Jako výchozí prvek jsem použil HTML tabulku o dvou sloupcích. Na formulář v prvním sloupci kořenové tabulky jsem použil další čtyřsloupcovou tabulku. Ta umožní snadnou organizaci jednotlivých textových polí a prvků. Kde bylo potřeba, nastavil jsem colspan atribut. *Colspan* atribut roztahuje buňku tabulky přes více buněk.
V Java kódu jsem vymazal všechny komponenty spravující rozložení. Zbylé komponenty jsem přidal do upravené *CustomLayout* komponenty a přiřadil jim patřičné location identifikační popisky. Nakonec jsem upravil zarovnání a velikosti jednotlivých textových polí formuláře.

Závěr

Zvolit správné rozložení prvků byla nejtěžší část úkolu. Několikrát jsem prohazoval prvky, dokud jsem nedosáhl použitelného rozložení.

5.2.7 Vytvoření defaultního stylu tlačítek a jejich barevné variace.

Zadání

Nevýhodou Vaadin frameworku je pevná výška *Button* komponenty a velmi malá paleta možností úpravy dosavadního stylu. Pro unikátnost aplikace bylo potřeba vytvořit vlastní vzhled pro všechny druhy tlačítek, tak aby se dal použít na různých místech bez nutnosti velkých úprav.

Postup řešení

Sada zahrnuje styl pro tlačítko s ikonou, tlačítko v tabulce, v tabulce s ikonou, obyčejné tlačítko, obyčejné tlačítko s ikonou a 4 barevné variace pro každý druh: oranžová, modrá, zelená a šedá.
Styly komponent ve Vaadin frameworku se definují pomocí CSS pravidel a *Sass* stylusu, který umožňuje dědičnost a mnoho dalších bonusů, které defaultní CSS neumí. Aby se neduplikoval kód, využil jsem dědičnosti *Sass* stylusu při vytváření těchto tlačítek.
Nejprve jsem si vytvořil Abstraktní verzi tlačítka bez barvy, jen se základními pravidly. Od tohoto tlačítka jsem postupně dědil ve všech verzích tlačítek. Když jsem měl základní pravidla pro zarovnávání a velikost nadefinované, vytvořil jsem zvlášť třídy pro barevné variace těchto tlačítek. V barevných variacích pouze měním barvy pozadí a ohraničení, ostatní parametry jsou zděděné z abstraktních verzí tlačítek.

Pro snadný přístup ke všem stylům tlačítek, jsme s kolegou vytvořili Statickou třídu *ScssUtils*, ve které jsou nadefinovány všechny dostupné třídy stylů.

Závěr

Trošku problematické bylo nastavení textu tlačítek, aby se zarovnával na střed.

5.2.8 Optimalizace přepínání agend

Zadání

Informační systém GIRITON se skládá z různých agend. Tyto agendy se po otevření ukládají do postranní lišty, aby mezi nimi mohl uživatel snadno přepínat a vrátit se k rozdělané práci. Přepínání mezi otevřenými agendami trvalo dlouhou dobu. Čas přepnutí byl v řádech sekund. Mým úkolem bylo prozkoumat možnosti jak optimalizovat přepínání jednotlivých agend.

Postup řešení

Tento problém je způsoben sestavováním a vytvářením požadované agendy při každém dotazu nebo přepnutí. Jako řešení jsem zvolil uchovávání všech otevřených agend ve stejném Rozložení a nepoužívané agendy posunout mimo zorné pole aplikace. Pomocí přidávání a odebírání *Active* CSS třídy, která nastaví celé agendě normální polohu.
Alogirtmus nejprve projde centrální *Layout* komponentu a zjistí, jestli už náhodou není požadovaná komponenta v zarovnání obsažena. Pokud ano, odstraní všem komponentám *Active* třídu a nastaví jí jen požadované komponentě. Pokud ne, přidá tuto komponentu do layoutu a nastaví jí *Active* třídu a všem ostatním jí odstraní.
Přepínání se zrychlilo mnohonásobně, ale bylo potřeba prozkoumat i další řešení tohoto problému. Další řešení bylo místo změny pozice agend, měnit *z-index* atribut. U tohoto řešení bylo potřeba vytvořit clonu, která by oddělovala zásobu agend a aktuální komponentu. Rychlost se nezměnila.
Poslední a konečné řešení bylo využití *CssLayout* komponenty. *CssLayout* komponenta umožňuje přepsat *protected* metodu *getCss*, která umožňuje nastavit CSS pravidla při generování HTML. Pokud aktuálně vykreslovaná komponenta je naše vybraná komponenta, nastaví se pravidlo visible na hodnotu true. Pokud to je jakákoliv jiná komponenta v *CssLayout* komponentě, nastaví se visible pravidlo na *false*. V algoritmu se místo přidávání *Active* třídy pouze nastaví momentálně vybraná komponenta. Aby se změny provedly, *CssLayout* komponentě se nastaví povinné překreslení.
Výsledná rychlost přepínání se nyní pohybuje ve stovkách milisekund.

Závěr

Tento úkol mě velice zaujal a věřím, že nabytou zkušenost využiji v budoucnu. Osobně si myslím, že tento typ optimalizace je využíván ve spoustě velkých webových aplikací

5.2.9 Integrační testy přidání a odebrání materiálu

Integrační testy

Celkový systém velmi komplexní a ruční testování funkčnosti všech částí systému je nadlidský úkol. Proto bylo potřeba vytvořit testy, které automaticky po každém commitu úprav nebo přidání částí kódu do aplikace, otestují již hotové části, zda provedená změna neovlivnila jiné části systému.
Pro integrační testy jsem používal knihovnu *SeleniumHQ* [6], pro tvorbu webových testů. Integrační testy se provádí vždy na čisté databázi, proto při testování úpravy záznamů v databázi přes uživatelské rozhraní, je potřeba nejprve vytvořit testovací data.
Pro snadnější a rychlejší psaní testů, je nadefinovaná statická metoda, která automaticky vytvoří instanci webového rozhraní pro test a přihlásí se na účet testovací firmy.
Pro snadný výběr agendy z popup menu je nadefinovaná statická metoda pro výběr požadované agendy. Aplikace obsahuje vlastní editor na tvorbu reportů. Jedná se o *drag and drop* editor s paletou komponent. Test je vytvořen tak aby otestoval drag and drop funkci tohoto editoru.

Zadání

Formulář na vkládání nového materiálu do databáze obsahuje mnoho textových polí. Ruční testování nepřipadá v úvahu. Test by měl ověřit funkčnost vkládání a odstranění materiálu v databázi pomocí uživatelského rozhraní.

5.2.10 Postup řešení

Vložení materiálu

Na začátku testu se provede zkouška počtu záznamů materiálů v tabulce. Jelikož se vždy vytváří nová instance firmy, tabulka s materiály by měla být prázdná. Po otevření příslušné agendy s materiály, se postupně pomocí ID atributů označí jednotlivá textová pole formuláře. Do polí se pošlou testovací data nadefinovaná jako konstantní atributy příslušné testovací třídy.
Součástí formuláře je přidání příslušných kategorií, pod které bude nový materiál spadat. Protože test pracuje s prázdnou databází, musí se vytvořit testovací kategorie materiálu. Z důvodu přehlednosti kódu jsem oddělil výběr a vytvoření kategorie materiálu do nové

privátní metody. Po otevření modálního okna s výběrem kategorií, se pomocí ID atributu stiskne tlačítko pro přidání nové kategorie. Po jeho zobrazení se označí textové pole názvu kategorie a nahrají se testovací data.
Formulář se potvrdí a test počká, do doby než se objeví nově vytvořená kategorie. Pokud by se tak nestalo do 15 vteřin, test selže. Jakmile se objeví nově přidaná kategorie, pomocí Xpath dotazu se vybere a otestuje se její název, zda odpovídá vloženým datům.
Vybraná data se potvrdí tlačítkem pro uložení změn a test vyplní zbývající textová pole formuláře. Následně se formulář potvrdí a počká se na informační modální okno o úspěchu nebo neúspěchu operace.
Na konci testu se provedou *Assert* zkoušky zda je v databázi vložen nový záznam v tabulce materiálu.

Odstranění materiálu

Druhá část testu testuje následné odstranění materiálu z databáze přes uživatelské rozhraní. Na začátku se provede zkouška a počtu záznamů v databázi. Očekává se pouze jeden záznam. V opačném případě test selže.
Pomocí *XPath* dotazu se vybere materiál z tabulky materiálů a následně se vybere a stiskne tlačítko pro odstranění. Po potvrzovacích modálních oknech se vlákno uspí na 5 vteřin, aby se nové změny stihly projevit v databázi. Upravená data se znovu z databáze nahrají a provede se zkouška na počet záznamů v tabulce materiálů. Očekává se nulová hodnota, v jiném případě test selže.

Závěr

S psaním integračních testů jsem se setkal poprvé. Jejich důležitost jsem předtím nevnímal. Po zkušenosti s tímto úkolem, pohlížím na vývoj informačních systémů z třosku jiného pohledu. Chápu důležitost průběžného testování vyvíjené apliakce a získané zkušenosti v budoucnu určitě využiji dále.

5.2.11 Integrační testy report editoru

Zadání

První test by měl otestovat vkládání prvků z palety komponent na plochu editoru. Následně uložit změny a otestovat, zda se projevlily v databázi.
Druhý test by měl prověřit funkci odstranění reportu z databáze pomocí uživatelského rozhraní systému.

Postup řešení

Editace reportu

Abych mohl otestovat editování reportu, musel jsem nejprve vytvořit prázdný report v databázi. Jeho nově vytvořenou instanci jsem si uchoval. Poté jsem přes instanci webového prohlížeče pomocí *XPath* dotazu vyhledal nově vložený prvek v tabulce reportů. Označil jej a otevřel editovací okno. Nově otevřené modální okno má specifickou Vaadin třídu. Pomocí ní jsem se odkazoval dále na paletu komponent reportu.
Pro simulaci přetáhnutí komponenty, jsem použil třídu *Actions* z knihovny Selenium testů.
Třída *Actions* umožňuje nadefinovat přesné pohyby myši po obrazovce. Dokáže nasimulovat držení stisknutého tlačítka po dobu nezbytně nutnou. Aby se veškeré akce provedly, musí se zavolat metoda na jejich zkompilování a poté metoda na jejich provedení. Přes *XPath* dotazu jsem označil první komponentu v paletě komponent a pustil jsem výše nadefinovanou akci. Po úspěšném otestování „dropnuté" komponenty, jsem si uložil počet komponent na vykreslovacím plátně reportu před a po akci, a hodnoty porovnal. Pokud se hodnoty rovnaly, test byl neúspěšný.
Report jsem uložil a na pár vteřin uspal vlákno vykonávajícího testu, aby se stihly provést změny v databázi. Po uplynulém čase jsem si tento report znovu vytáhl z databáze. Porovnal jsem, zda jsou rozdílné *timestamp* atributy instance čerstvých dat z databáze a instance, kterou jsem si uchoval těsně po vytvoření reportu. Timestamp atribut zaznamenává poslední úpravu daného záznamu. Pokud by se *timestamp* atributy rovnaly, námi provedené změny se neprojevily v databázi a test je neúspěšný.

Odstranění reportu

Druhý integrační test testuje úspěšné odstranění reportu z databáze. Tento test se spouští po provedení prvního testu. Díky předchozímu testu je již testovací záznam reportu vytvořen.
Na začátku testu se provádí *Assert* zkouška, která předpokládá jeden záznam reportu v databázi. Pokud v databázi není žádný záznam, test selže.
Pomocí *XPath* dotazu jsem označil první řádek v tabulce reportů. Tlačítku na vymazání reportu jsem nastavil ID a pomocí něj jsem si tlačítko na vymazání našel a stisknul. Po potvrzení modálních vyskakovacích oken jsem otestoval pomocí *Assert* zkoušky, že počet záznamů v tabulce je roven nule. V opačném případě test selže.

Závěr

Tento úkol prohloubil mé dosavadní znalosti programování integračních testů. Technologie *SeleniumHQ* se dá využít nejen na testování webových aplikací. Díky možnosti definování vlastních akcí, nemá tento nástroj hranice. Jediný problém Selenium testů je ne-

šťastně řešené vyčkávání na odpověď od serveru.

5.2.12 Jasper Reporty

Zadání

GIRITON systém disponuje modulem umožňující tvorbu jednoduchých reportů dat z databáze. K tomu využívá volně dostupnou knihovny JasperReports library. Dostal jsem za úkol spolupracovat na vytvoření základní sady reportů, pomocí nástroje iReport designer od vývojářů JasperSoft s.r.o. Jednalo se především o grafické zobrazení dat za posledních 30 dní v podobě spojnicového nebo sloupcového grafu.

Postup řešení

Po seznámení s nástrojem *iReport desingner*[4] jsem přistoupil k tvorbě prvního reportu. Report má zobrazit spojnicový graf znázorňující množství odpracovaných úkolů za posledních třicet dní. Protože report obsahuje pouze graf, vyčistil jsem pracovní plochu návrháře od nepotřebných sekcí.
Nejdůležitější části reportu jsou data která bude zobrazovat, proto jsem vytvořil nový datový zdroj databázového typu. K němu jsem vytvořil dvě výstupní proměnné *COLUMN_1* a *COLUMN_2*, které slouží jako ukazatelé na výsledná data z SQL dotazu. Proměnné poté mohu použít k naplnění vytvořené šabloby reportu daty. Pokračoval jsem vytvořením řady grafu. Ose X jsem nastavil data z proměnné *COLUMN_1* reprezentující posledních 30 dní a ose Y data z proměnné *COLUMN_2*. Proměnná *COLUMN_2* reprezentuje počet provedených úkolů v jednotlivých dnech. Graf navíc umožňuje úpravu hodnot a popisků pomocí jednořádkových výrazů s použitím platných pravidel programovacího jazyka Java. Toho jsem využil při úpravě formátu popisků v jednotlivých dnech.
Posledním krokem je vytvoření samotného SQL dotazu na databázi. Ačkoli prostředí iReport návrháře nabízí rozhraní pro připojení k databázi, z důvodu specifického nastavení databáze, bylo jednodušší otestovat správnost SQL dotazu přímo v systému GIRITON. Nahrál jsem tedy nově vytvořený report do aplikace a spustil zpracování reportu. Výsledkem byla prázdná stránka bez grafu. Systém GIRITON nabízí funkční uživatelské rozhraní pro úpravu reportů a tak jsem mohl opravit chyby v dotazu rovnou v aplikaci.
Druhý pokus byl úspěšný, ale v grafu chyběla některá data. V případě víkendů a svátků je výroba pozastavená a v databázi nejsou pro tyto dny žádná data. Vynechané dny v grafu působí matoucím dojmem. Pomocí SQL dotazu není možné přidat chybějící dny. Jedním řešením by bylo vytvořit v databázi proceduru, která se postará o doplnění chybějících hodnot. Z důvodu zachování modulárnosti systému, je procedura nozouvým

řešením problému.
Kolega mi poradil o možnosti použití scripletů [5]. Scriplet je třída, jejíž metody se pouští v určitých fázích sestavy reportu. Reportu se pouze nastaví který, scriplet má použít a knihovna JasperReports se postará o zbytek. Pro testovací účely jsem si vytvořil jednoduchý scriplet, který před sestavením reportu změní text nadpisu. Nadpis se změnil podle očekávání. Rozhodl jsem se tedy zkusit přidat chybějící dny pomocí scripletu. Metoda *AfterInitialize* se pustí po získání dat z databáze. Ty jsou v scripletu jednoduše dostupná. Vytvořil jsem *for* cyklus s 30ti iteracemi procházející 30 dní. V každé iteraci se provádí kontrola, zda se aktuální datum iterace nachází v datech z databáze. V případě neshody se do nových kolekcí přidá nulová hodnota pro aktuální datum v iteraci.

```
LocalDate fromDate = DateTime.Now().dayOfWeek() − DateTime.Now().minusDays(30);
int dateIt = 0, valueIt = 0;

for(int i=0; i<30; i++) {
    if (fromDate == dates[dateIt++]) {
        newDates.Add(fromDate);
        newValues.Add(values[valueIt++]);
    }
    else {
        newDates.Add(fromDate);
        newValues.Add(0);
    }
    fromDate = fromDate.plusDays(1);
}
```

Výpis 3: Algoritmus přidávající chybějící dny do výpisu.

Po skončení cyklu, jsem staré kolekce nahradil novými a provedl test. Výsledný graf nyní obsahoval všech 30 dní.
Nakonec jsem dostal za úkol upravit report tak aby prováděl výpis v zadaném období. V případě nenastavení hodnot se provede výpis za posledních 30 dní. Do stávající reportu jsem přidal dvě proměnné *FromDate* a *ToDate*, na které se knihovna JasperReports dotáže před provedením SQL dotazu. Dále jsem upravil algoritmus a bylo hotovo.
Ostatní reporty jsem vytvořil stejným způsobem s využitím již hotového scripletu.

Závěr

Než mi kolega pověděl o možnosti využití scripletu, uběhlo několik hodin hledání alternativních řešení výše zmiňovaného problému. Naučil jsem se používat další užitečný

nástroj, který v budoucnu jistě využiji. Hlavně jsem si z tohoto úkolu vzal ponaučení. Pokud má problém složité řešení, někdo ho už zjednodušil.

5.2.13 Modul desktopových upozornění

Zadání

Informační systém GIRITON obsahuje interní chat. Mým úkolem bylo doimplementovat funkci vyskakovacího okna v případě příchodu nové zprávy. K tomu účelu jsem měl použít Vaadin doplněk *DesktopNotification widget* [**?**], který podporuje vyskakování okna mimo prostředí webového prohlížeče.

Postup řešení

Po přidání závislosti do POM souboru, jsem si na stránkách modulu našel vzorový příklad k použití. Veškerou práci zastává třída *DesktopNotifier*. Pro správné fungování deskotových uporoznění, je třeba požádat uživatele apliakce o povolení. Podle ukázkového příkladu jsem nastavil třídu *DesktopNotifier* a spustil apliakci.
Hned ze začátku se objevilo několik problémů. Modul vyhazoval vyjímku *MethodWasCalledBeforeRoundtipException*. Kromě vzorového příkladu, nemá doplněk žádnou dokumentaci. Podíval jsem se tedy rovnou do zdrojových kódů modulu. Po bližším prozkoumání, jsem zjistil že zmíněná vyjímka je vyhazována ve dvou případech. V případě *null* hodnoty proměnné *isAllowed*, která značí povolení desktopových notifikací, a v případě *null* hodnoty proměnné *isSupported*, která značí zda je tato funkce podporovaná webovým prohlížečem. Ve vzorovém příkladě jsem opomenul část nastavení proměnné *isSupported*. Do konstruktoru jsem přidal chybějící část a nyní vše fungovalo.
Text příchozí zprávy jsem nastavil do těla upozornění.
Těsně před požádavkem o povolení je uživatel pomocí modálího okna upozorněn na zobrazení žádosti o povolení desktopových oznámení. Přišlo mi zbytečné tuto zprávu zobrazovat. Po konzultaci jsem dostal za úkol odstranit tento mezikrok. Protože *DesktopNotifier* jinou možnost nepodporuje, rozhodl jsem se upravit zdrojové kódy modulu.
Protože je modul veřejně dostupný, stáhl jsem si jeho zdrojové kódy. Mezikrok se dá velmi jednoduchou úpravou kódu vynechat. Zkusil jsem provést patřičné změny. Bohužel bez úspěchu. Žádost o povelení oznámení se nezobrazila. Po přečtení dokumentace k použití desktopových oznámení jsem pochopil důležitost mezikroku. Žádost může být vyvolána pouze v případě uživatelem vyvolané udalosti. Například klepnutím na tlačítko OK v mezikroku. Napadlo mě využít tlačítko odesílající zprávy chatu.
V třídě *DesktopNotifier* jsem si vytvořil vlastní metodu, která připojí *onClick* posluchač eventů na komponenty v parametrech metody. Event provede žádost o povolení deskto-

pových oznámení. První parametr metody přijímá selektor, druhý parametr přijímá typ selektoru (element, třída, ID elementu). Metoda je typu native a překládána pomocí GWT nástroje. Native příznak znamená že tělo metody je napsané v jiném jazyce než Java. V tomto případě je tělo napsané v čistém javascriptu.
Aby se event úspěšně připojil k tlačítku, musí se metoda připojující *onClick* event volat až po úspěšném vytvoření chatovacího okna. Z tohoto důvodu se event nastaví až při první příchozí nebo odchozí zprávě, a až v následujícím odeslání zprávy je vyvolána žádost o povolení.
Na závěr jsem vytvořil *README* soubor s užitečnými odkazy a informacemi o použití desktopových oznámení.

Závěr

Ačkoliv mi splnění úkolu zabralo skoro tři dny, naučil jsem se využívat nestandartní nástroje a prostředky. Naučil jsem se využívat desktopová oznámení z prostředí prohlížeče pomocí javascriptu. Vyzkoušel jsem si použití nástroje GWT a native metod. Úkol samotný mě velmi zaujal.

6 Znalosti získané v průběhu studia uplatněné v průběhu odborné praxe

V průběhu odborné praxe jsme uplatnil především znalosti oběktově orientovaného programování. Umožnili mi efektivně navrhnout modely tříd jejich rozhraní. Dále jsem uplatnil znalost programovacího jazyka Java, který se učí v třetím vyučovacím semestru oboru Informatiky a výpočetní techniky. Programoval jsem v něm po celou dobu odborné praxe. Uplatnil jsem i znalost dotazovacího databázového jazyka SQL. Nejvíce jsem tuto znalost uplatnil při vytváření sestav reportů.
Z předmětu Vývoj internetových aplikací jsem zde využil nabyté znalosti HTML, kaskádových stylů, XML dokumentů, XPath a základy javascriptu.
Pro snadnější pochopení vyvíjené apliakce mi pomohly znalosti získané v předmětu Vývoj informačních systémů.

Sourh uplatněných předmětů

- Programovací jazyky I (PJ I)
- Algoritmy I, II (ALG I,II)
- Úvod do databázových systému (UDBS)
- Vývoj informačních systémů (VIS)
- Vývoj internetových aplikací (VIA)

7 Znalosti scházející v průběhu odborné praxe

V průběhu odborné praxe jsem se setkal se spoustou nových technologií, ale nechyběly mi žádné základní znalosti. Znalosti získané ve škole mi umožnil osvojit si nové technologie velmi snadno a rychle. Jednou z těchto technologií je Vaadin framework. Znalost tohoto frameworku byla nezbytná k vykonávání odborné praxe. Základy jsem se musel naučit ještě před nástupem do praxe.

8 Závěr

V momentě když jsem se dozvěděl, že jsem byl přijat na praxi, hlavou mi proběhla myšlenka: *„Konečně budu dělat něco ve svém oboru.“*. Odborná praxe mi dala velkou zkušenost do života. Kromě nových technologií a nástrojů, jsem si vyzkoušel práci v týmu a párové programování. Poznal jsem nové kamarády a osvojil se nové postupy vývoje aplikací. Velkým přínosem pro mě byla práce s verzovacím SVN systémem, bez kterého se v týmu pracuje velmi těžko. V průběhu praxe jsem řešil složité i jednoduché úkoly. Pokud jsem si nevěděl s něčím rady, kolegové mi ochotně pomohli a poradili. Praxe ve firmě GIRITON system splnila mé očekávání.

9 Reference

[1] Marko Grönroos, *Book of Vaadin:Vaadin 7 Edition* 2nd Revision, Vaadin Ltd, 2014
Vaadin je nová technologie. Knižní podoba publikace zatím byla vydaná, proto jsem používal PDF verzi.
`https://vaadin.com/book`

[2] *Vaadin community forum* komunitní fórum
`https://vaadin.com/forum`

[3] DesktopNotifications *Desktop Notifications for Vaadin* - zdrojové kódy widgetu
`https://github.com/wolfie/DesktopNotifications`

[4] Jaspersoft Corporation, *iReport Ultimate Guide* version 0113-UGI50-6
`http://community.jaspersoft.com/documentation/ireport-ultimate-guide`

[5] *JasperReports - Reports Scriplet* - Ukázkový příklad použití scripletu
`http://community.jaspersoft.com/documentation/ireport-ultimate-guide`

[6] *SeleniumHQ Browser Automation, oficiální dokumentace*
`http://docs.seleniumhq.org/docs/`

[7] *Syntactic awesome stylesheets, oficiální dokumentace CSS preprocesoru*
`http://sass-lang.com/`

[8] *Stránky společnosti GIRITON Systems s.r.o.*
`http://www.giriton.cz/`

[9] *Dzone Refcardz, pomocné karty různých technologií*
`http://refcardz.dzone.com/`

[10] Petr Hatina, *Java seriál* - výuka programovacího jazyka Java
`http://www.linuxsoft.cz/article.php?id_article=244`