Vysoká škola báňská – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Absolvování individuální odborné praxe

Individual Professional Practice in the Company

2014 Tomáš Blahut

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Tomáš Blahut**

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

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA * Fakulta elektrotechniky a informatiky * OSTRAVA *

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

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 na tomto místě poděkoval firmě GIRITON Systems s.r.o. a především Ing. Janu Gřeši MSc. za možnost absolvování odborné praxe, vytvoření výborného pracovního prostředí, vstřícný přístup a pomoc při plnění zadaných úkolů. Dále pak pánu Mgr. Miloši Kudělkovi, Ph.D. za udělené rady a připomínky při tvorbě této práce.

Abstrakt

Cílem této bakalářské práce je popsat mé působení v rámci odborné praxe ve firmě GIRITON Systems s.r.o. V práci nejprve informuji o oblasti odborného působení firmy, dále se zabývám technologiemi a nástroji, se kterými jsem během praxe pracoval a také uvádím příklady zadaných úkolů s popisem jejich řešení. V závěru práce hodnotím své znalosti a absolvovanou odbornou praxi jako celek.

Klíčová slova: Odborná praxe, Java, Vaadin, Sass, Selenium, Robotium, Jasper Reports

Abstract

The aim of this bachelor thesis is to describe my work within proffesional practice in the company GIRITON Systems Ltd. The paper first informs about the company's area of expertise, later it desribes used technologies and tools, with whom I worked during practise. Further I mention examples of given tasks along with description of their solution. In the conclusion I evaluate my own knowledge and professional practice as a whole.

Keywords: Professional practice, Java, Vaadin, Sass, Selenium, Robotium, Jasper Reports

Seznam použitých zkratek a symbolů

AJAX	–	Asynchronous JavaScript and XML
API	–	Application Programming Interface
AWT	–	Abstract Window Toolkit
CSS	–	Cascading Style Sheets
CSV	–	Comma-separated values
DOM	–	Document Object Model
EJBQL	–	Enterprise Java Beans Query Language
GUI	–	Graphical User Interface
HTML	–	HyperText Markup Language
ID	–	Identity
JPA	–	Java Persistence API
JPQL	–	Java Persistence Query Language
JRXML	–	Jasper Reports XML
POJO	–	Plain Old Java Object
POM	–	Project Object Model
SASS	–	Syntactically Awesome Style Sheets
SQL	–	Structured Query Language
SWT	–	Standard Widget Toolkit
UML	–	Unified Modeling Language
URL	–	Uniform Resource Locator
XML	–	Extensible Markup Language

Obsah

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Ze dvou možných variant závěrečné bakalářské práce, které Vysoká škola báňská - Technická univerzita Ostrava nabízí, jsem si zvolil absolvování individuální odborné praxe. K této variantě jsem se přiklonil z toho důvodu, že jsem si chtěl vyzkoušet práci na problémech z reálného prostředí a získat cenné pracovní zkušenosti. Věřím, že mi tyto zkušenosti budou do budoucna více ku prospěchu, než vypracování teoretické práce.

V této práci nejprve popisuji firmu GIRITON Systems s.r.o., kde jsem odbornou praxi vykonal a své pracovní zařazení v dané firmě. V dalších kapitolách se věnuji technologiím a nástrojům, se kterými jsem se během praxe setkal. Spolu s teoretickým popisem konkrétní technologie uvádím příklady úkolů a jejich řešení. Práce obsahuje také posouzení mých znalostí v průběhu praxe. V samotném závěru hodnotím dosažené výsledky a přínosy absolvované odborné praxe.

2 GIRITON Systems s.r.o.

Informace o firmě GIRITON Systems s.r.o. jsem převzal z jejího portfolia na firemních internetových stránkách. Uvádím zde pouze oblasti působení relativní k mé práci ve firmě.

2.1 O firmě

GIRITON Systems s.r.o. je mladá a dynamická firma zabývající-se tvorbou zakázkových informačních systémů. V současné době se firma zaměřuje obecně na tvorbu podnikových informační systémů používajících mobilní terminály, tablety a chytré telefony pro automatický i ruční sběr dat a obousměrnou komunikaci. V neposlední řadě firma vytváří, dodává a provozuje zakázkové aplikace na podporu firemních procesů, a to jak klasické, tak Cloudové aplikace přístupné odkudkoli po internetu [1].

2.2 Řízení výroby

Společnost se věnuje vývoji informačních systémů pro řízení a monitorování především výrobních procesů v reálném čase. Mezi funkce systému patří například podpora variant produktů, šarží, pokročilých statistik a reportů či vlastní akce vykonávané při vybraných událostech. Součástí produktu je volitelně i docházkový systém, díky kterému je možno tvořit mzdové podklady zaměstnanců nejenom v úkolové, ale také v časové mzdě [1].

2.3 Sběr dat ve výrobě

Produkty pro sběr dat nachází uplatnění nejčastěji ve firmách výrobního a zpracovatelského průmyslu. V těchto odvětvích pomáhají sbírat data především z následujících agend.

- Odvádění výroby - provedené operace, použitý materiál, rozpracovanost výrobních dávek, stav plnění zakázek.
- Pohyb materiálu - naskladňování nebo vyskladňování jednotlivých položek materiálu, hlídání minimálního množství skladu atd.
- Šarže - který pracovník se podílel na výrobě konkrétního produktu, z jakých přesně materiálů byl produkt sestaven.

Data jsou sbírána mobilními terminály. Pomocí jednoho terminálu může data vkládat jeden či více uživatelů a to formou snímání čárových kódů či bezdotykových RFID čipů. Data jsou okamžitě přenášena po bezdrátové síti a dávají tak přesný obraz o aktuálním stavu výroby [1].

2.4 Docházkové systémy

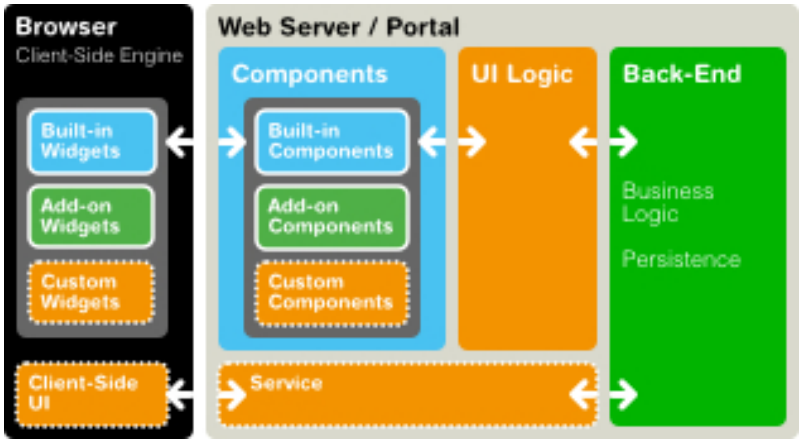
Firemní docházkové systémy umožňují jednoduše sledovat, evidovat a vyhodnocovat docházku zaměstnanců. Systém je tvořen mobilními terminály, které pomocí bezdrátové komunikace předávají data o docházce k dalšímu zpracování. Uživatelé jsou identifikováni pomocí magnetických karet, čárových kódů, bezkontaktních RFID čipů, či pomocí osobního hesla [1].

2.5 Mé pracovní zařazení

Ve firmě GIRITON Systems s.r.o. jsem zaujímal pozici Java programátora. V průběhu praxe jsem pracoval na firemním informačním systému, kde jsem se nejčastěji pohyboval v agendách zabývajících se celkovým procesem výroby od objednávky až po hotový produkt. Náplní mé práce byla tvorba a editace uživatelských rozhraní, implementace aplikační a doménové logiky, psaní integračních testů pro webového a mobilního klienta informačního systému nebo také například vytváření šablon tiskových sestav pro knihovnu JasperReports.

3 Vaadin framework

Popis samotného Vaadin frameworku a jeho součástí, v této práci zmíněných(GUI, datový model a témata), jsem převzal z knihy Book of Vaadin. Vaadin je framework pro vývoj Java webových aplikací, navržen za účelem zjednodušit tvorbu a údržbu webových uživatelských rozhraní vysoké kvality. Umožňuje opomenout prostředí internetu a vytvářet uživatelská rozhraní stejně jako pro desktopové aplikace za pomocí standardních nástrojů programovacího jazyka Java jako jsou AWT, Swing nebo SWT. Serverově orientovaná část frameworku se stará o správu uživatelského rozhraní v prohlížeči a AJAX komunikaci mezi webovým prohlížečem a serverem. Při použití Vaadinu se programátor nemusí přímo zaobírat technologiemi prohlížeče jako jsou HTML a JavaScript.



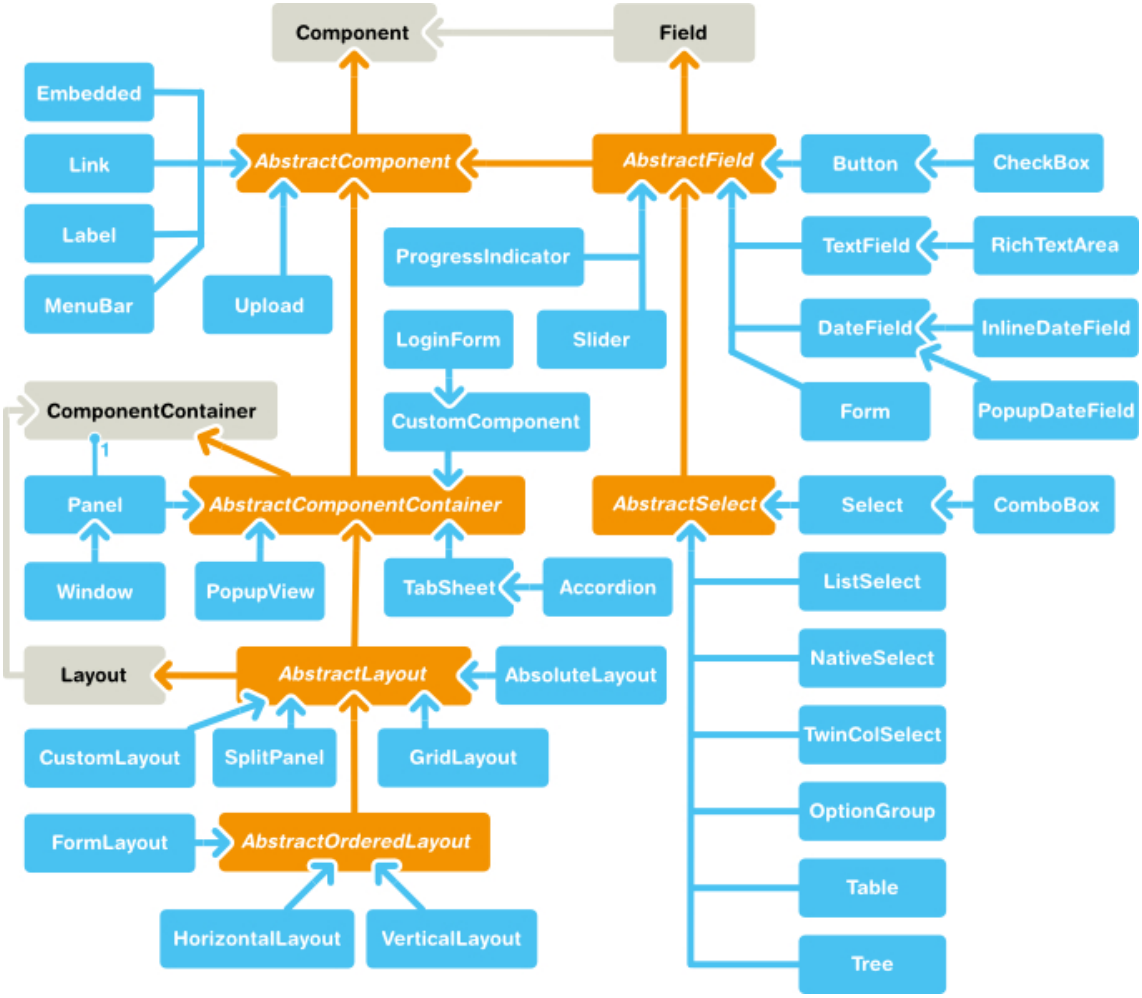
Obrázek 3.1: Architektura Vaadin aplikace.
Obrázek převzat z Book of Vaadin.

Obrázek 3.1 zobrazuje základní architekturu webové aplikace vytvořené pomocí Vaadinu. Ta se skládá z server-side frameworku a client-side enginu. Client-side engine běží uvnitř webového prohlížeče jako JavaScript kód, který vykresluje uživatelská rozhraní a odesílá požadavky na server. Na straně serveru funguje logika ovládání uživatelského rozhraní spolu s business logikou systému.

Kromě vývoje serverově orientovaných aplikací umožňuje Vaadin vytvářet také widgety, které běží výhradně na straně klienta v internetovém prohlížeči. K tomuto využívá Google Web Toolkit, který slouží jako překladač z Javy do JavaScriptu, který následně běží v prohlížeči. Vaadin jasně separuje strukturu uživatelského rozhraní od jeho vzhledu a umožňuje jejich oddělený vývoj. Toto je realizováno pomocí tzv. Témat, která definují vzhled aplikace prostřednictvím Sass nebo CSS kaskádových stylů a volitelně také HTML šablon. Součástí frameworku jsou také kompletní výchozí témata Runo, Reindeer a Chameleon [2, strany 23-25].

3.1 Uživatelská rozhraní

Vaadin poskytuje komplexní sadu komponent uživatelského rozhraní a také umožňuje definovat komponenty vlastní. Komponenty lze rozdělit na dvě skupiny: ty, které slouží k interakci s uživatelem a layout komponenty, jejichž prostřednictvím jsou jiné objekty umisťovány na specifická místa v uživatelském rozhraní. Obrázek 3.2 ilustruje hierarchii tříd a rozhraní komponent uživatelského rozhraní. Rozhraní jsou zobrazena šedě, abstraktní třídy oranžově a regulérní třídy modře.



Obrázek 3.2: Hierarchie komponent uživatelského rozhraní Vaadin frameworku. Obrázek převzat z Book of Vaadin.

Rozhraní *Component* a třída *AbstractComponent* představují vrchol celé hierarchie. Komponenty uživatelského rozhraní, které slouží pro zobrazení informací a interakci s uživatelem mají společného předka třídu *AbstractField*. Výjimkou jsou komponenty, které nejsou vázány na datový model Vaadin frameworku. Ty dědí přímo ze třídy *AbstractComponent*.

Rozložení komponent v okně prohlížeče je řízeno skrze tzv. layout komponenty, stejně jako tomu je v běžných Java UI nástrojích pro desktopové aplikace. Předek těchto komponent v hierarchii na obrázku 3.2 je reprezentován abstraktní třídou *AbstractLayout*. Kromě standardních layoutů ve Vaadinu existuje ještě třída *CustomLayout*, která umožňuje definovat rozložení komponent uživatelského rozhraní pomocí XHTML šablony [2, strana 104].

3.1.1 Má práce

Práce na uživatelském rozhraní webového klienta informačního systému patří k činnostem, kterým jsem během odborné praxe věnoval podstatnou část pracovní doby. Mezi aktivity spojené s touto prací patřila tvorba nových a editace stávajících struktur uživatelského rozhraní pomocí objektů Vaadin frameworku a dále případné definování vzhledu komponent uživatelského rozhraní skrze *Sass* kaskádové styly.

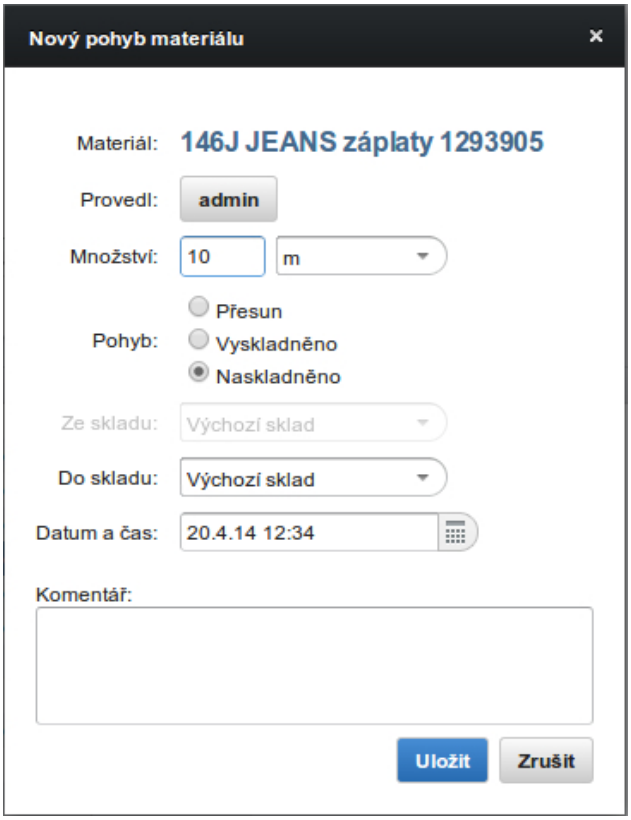
V závislosti na složitosti požadovaného výsledku jsem vytvořil jednoduchý grafický návrh rozložení komponent a jejich chování například při změnách rozměrů okna prohlížeče. Vaadin poskytuje širokou škálu objektů pro vzájemné uspořádání komponent. Díky tomuto téměř vždy existuje několik různých řešení daného problému, ze kterých jsem volil nejvhodnější hierarchii vzhledem k možnostem frameworku. V dalším kroku jsem tuto hierarchii naprogramoval a ladil dle výsledků zobrazených přímo v internetovém prohlížeči.

Z celé své práce na uživatelském rozhraní informačního systému bych zde uvedl několik příkladů, které dobře popisují velkou část oblasti tvorby struktury uživatelského rozhraní ve Vaadin Frameworku a také definici způsobu vykreslování komponent na straně klienta.

Příklad 3.1
Jeden z příkladů uživatelských rozhraní, která jsem vytvořil, je dialog pro pohyb materiálu. Tento dialog spadá do skladové agendy a jeho úkolem je umožnit uživateli přidávat nové záznamy o pohybu konkrétního materiálu a editovat záznamy stávající. Tak jako u každého Vaadin UI se struktura dialogu skládá z objektů definujících vzájemné umístění elementů uživatelského rozhraní a komponent sloužících k interakci s uživatelem. Výpis 1 v přílohách znázorňuje zdrojový kód struktury uživatelského rozhraní dialogu. Pro jeho stavbu jsem použil několik druhů komponent pro vzájemné uspořádání elementů uživatelského rozhraní.

- VerticalLayout - rozložení, které komponenty skládá svisle pod sebe dle pořadí, v jakém jsou do něj přidány.
- HorizontalLayout - stejné jako VerticalLayout s tím rozdílem, že komponenty umisťuje vodorovně.
- FormLayout - rozdělí komponenty do dvou vedle sebe situovaných sloupců, kde v prvním jsou nadpisy komponent a ve druhém samotné komponenty. Výsledek vzbuzuje dojem formuláře.

Dialog obsahuje také většinu standardních komponent Vaadin frameworku pro interakci s uživatelem. Patří zde například *TextField* pro ruční zadávání textu, *PopupDateField* pro výběr data a času, *ComboBox* a *OptionGroup* pro výběr jedné možnosti z předem definované množiny hodnot, *Label* pro zobrazení statického textu a *Button* představující klasické tlačítko. Objekt doménové logiky, který reprezentuje přesun materiálu obsahuje reference na přesouvaný materiál a aktéra, který přesunutí učinil, dále pak datum a čas, kdy byl přesun proveden, o jaký typ přesunu se jedná a jaké sklady jsou zahrnuty. Výsledný dialog je zobrazen na obrázku 3.3.



Obrázek 3.3: Dialog pro pohyb materiálu

■

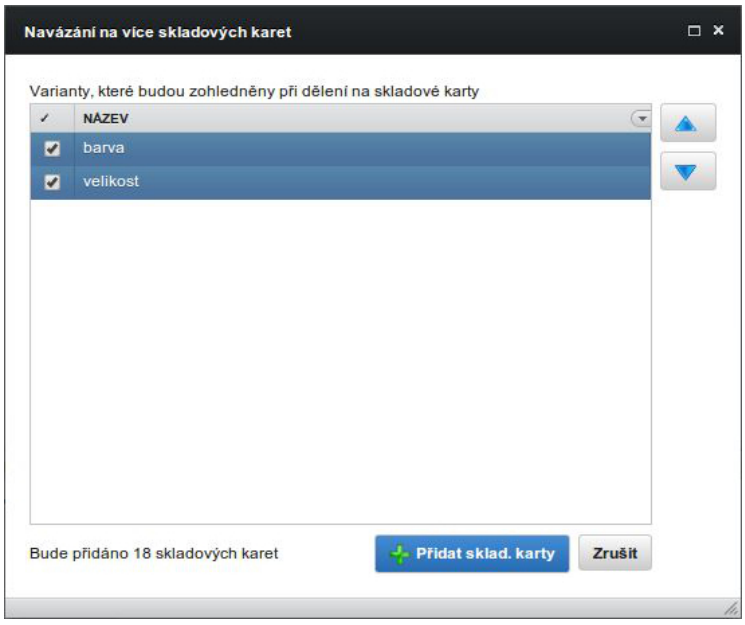
Příklad 3.2
Jako další uživatelské rozhraní, které jsem vytvořil, bych uvedl dialog pro generování skladových karet. Skladové karty jsou vytvářeny pro jednotlivé výrobní procesy dle jejich dostupných variant například barvy. Oproti předchozímu příkladu se zde budu věnovat více vlastnostem komponent uživatelského rozhraní než jeho struktuře.

K sestavení výsledného uživatelského rozhraní nestačí pouhá definice jeho struktury. Nezbytné je specifikovat také jeho chování pomocí vlastností komponent, které toto rozhraní tvoří.

Mezi základní nastavení patří:

- Rozměry komponenty - Výšku a šířku lze stanovit použitím metod *setWidth()* a *setHeight()* a to pomocí procentuálního vyjádření anebo konkrétní hodnoty a jednotky. Zvláštní případy nastavení rozměrů reprezentují metody *setSizeFull()*, která nastaví komponentě výšku i šírku na 100% a *setSizeUndefined()*. Druhá z uvedených metod říká komponentě, aby své rozměry přizpůsobila prostoru, který zabírá její obsah.
- Odsazení - Vaadin framework dovoluje specifikovat odsazení komponenty od okrajů jejího kontejneru skrze metodu *setMargin()*. Dále je možné odsadit jednotlivé komponenty od sebe navzájem metodou *setSpacing()*.
- Zarovnání - Metoda *setComponentAlignment()* poskytuje programátorovi schopnost zarovnat komponenty uživatelského rozhraní v rámci obalujícího kontejneru. Existuje devět standardních pozic od levého horního rohu po pravý spodní okraj.
- Expandování - Komponenty uživatelského rozhraní Vaadin frameworku jsou schopné dynamicky měnit svou velikost vzhledem k rozměrům layoutu, ve kterém jsou obsaženy. K tomuto slouží metoda *setExpandRatio()*.

Mezi další vlastnosti komponent uživatelského rozhraní patří například nastavení ikony a vlastního stylu pro vykreslování komponenty na straně klienta. Obrázek 3.4 ilustruje dialog sestavený pomocí zdrojového kódu ve výpisu 2 v přílohách.



Obrázek 3.4: Dialog pro generování variant výrobního procesu

■

Příklad 3.3
Ze všech Vaadin implementací *Layout* rozhraní třída *CssLayout* poněkud vyčnívá. Kromě běžných funkcí umožňuje také silnou kontrolu nad stylováním komponent, které se v něm nachází. Situace, kdy se *CssLayout* nabízí jako vhodné řešení, je například potřeba naskládat několik komponent na sebe. Přesně takovýto problém jsem řešil v průběhu odborné praxe. Požadavek zněl, umístit přes komponentu *ProgressBar*, sloužící k zobrazení postupu v nějakém procesu, textový indikátor tohoto postupu.

CssLayout a jeho komponenty jsou na straně klienta sestaveny do jednoduché DOM struktury skládající se z *div* elementů. Ze všech dostupných layoutů se proto v prohlížeči na straně klienta vykresluje nejrychleji. Metoda *getCss()* je volána pro každou komponentu, kterou *CssLayout* obsahuje. Návratová hodnota je textový řetězec s *CSS* pravidly pro konkrétní komponentu [2, strana 238]. Řešení spočívalo v umístění *Label* komponenty relativně k jejímu kontejneru nezávisle na sousedních elementech. Tohoto efektu jsem docílil pomocí *CSS* pravidla *position: absolute*. Komponenty lze poté umístit přes sebe použitím atributu *z-index*. Na obrázku 3.5 je vidět využití takto vytvořeného *ProgressBaru* uvnitř tabulky.

STAV	PRŮBĚH	POSLEDNÍ ÚKOL PROVEDE
nový	4/8	2014-02-26 13:00:46
nový	4/8	2014-02-26 13:00:49
nový	4/8	2014-02-26 13:00:53
nový	5/8	2014-02-25 13:56:06
nový	2/10	2014-02-28 13:41:34
nový	0/10	

Obrázek 3.5: Progress bar s textovým indikátorem vytvořený pomocí CssLayoutů

■

3.1.2 Témata

Vaadin odděluje vzhled uživatelského rozhraní od jeho logiky pomocí tzv. *Témat*. Témata v sobě mohou nést *Sass* nebo *CSS* soubory, definice vlastních HTML layoutů a jakoukoliv další nezbytnou grafiku. Aplikační zdroje uchovávané v tématu jsou přístupné také skrze kód aplikace v podobě *ThemeResource* objektů, což je umožňuje zobrazit v ikonách komponent anebo přímo jako obsah komponenty *Image*. Vlastní témata jsou umístěna pod adresářem *VAADIN/themes* v projektu webové aplikace. Tato pozice je pevně daná - adresář *VAADIN* obsahuje statické prostředky, které obsluhuje Vaadin servlet. Jméno adresáře, ve kterém je téma uloženo definuje název samotného tématu a je následně využito v anotaci *@Theme*, pomocí které se aplikaci nastaví konkrétní téma. Vlastní témata musí dědit z některých vestavěných témat Vaadin frameworku a obsahovat soubor styles.css buďto pro *Sass* anebo prosté *CSS* témata. [2, strany 261-262].

Příklad 3.4
U vlastních Vaadin témat nastává problém v okamžiku, kdy je potřeba nastavit nestandardní výšku komponentám pro interakci s uživatelem konkretně například tlačítku anebo komponentám jako jsou *ComboBox* a *CheckBox*. Tyto komponenty jsou na straně klienta vykreslovány pomocí obrázků, jejichž výška zůstává stejná bez ohledu na výšku komponenty. Jedním z úkolů, na kterých jsem během praxe pracoval, bylo vytvoření nového a kompletního řešení *Sass* stylů pro tlačítka v aplikaci.

Zdrojový kód v příloze 4 demonstruje pouze malou část celkové množiny stylů. Ta obsahuje řešení pro vykreslování tlačítek i jiných komponent v různých variantách, které se od sebe navzájem liší například barvou nebo cíleným umístěním komponenty. K vytvoření zde prezentovaných stylů jsem využil *Sass*. Ty oproti klasickým *CSS* stylům poskytují programátorovi několik nových možností jako je zanořování selektorů anebo dědičnost. Většina komponent Vaadin frameworku je v DOMu na straně klienta tvořena několika do sebe zanořenými elementy. Konkrétně tlačítko tvoří obalující element *div*, dva elementy *span* a pokud má tlačítko nastavenou ikonu také element *img*. Při tvorbě stylů je nutné jejich identifikátory cílit na jednotlivé elementy tvořící danou komponentu. Na obrázku 3.6 jsou zobrazena tlačítka vykreslena podle vestavěného tématu Reindeer(vlevo) a pomocí stylů, které jsem vytvořil(vpravo).

Obrázek 3.6: Porovnání stylů tlačítek

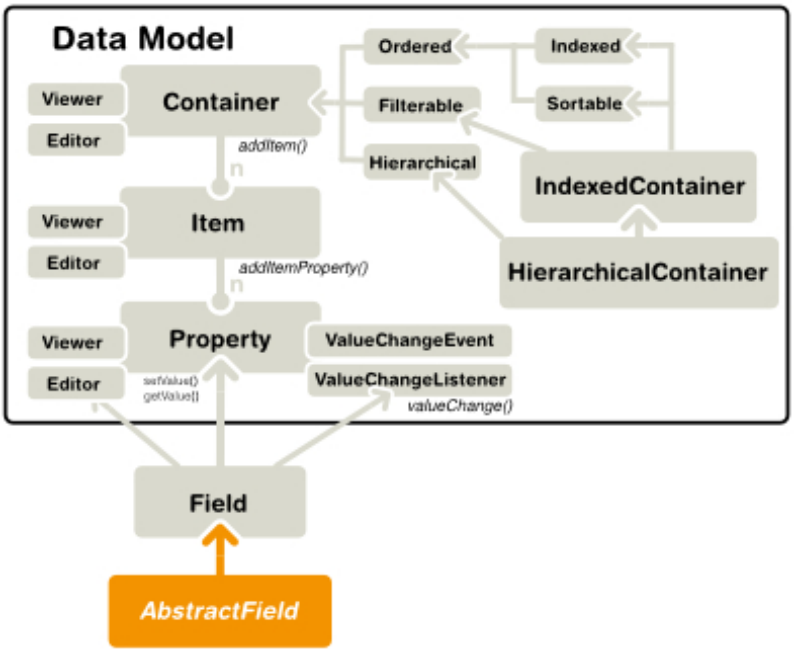
■

3.2 Datový model

Pojem datový model zde nepředstavuje model databáze nebo jiného persistentního uložiště dat. V kontextu Vaadin frameworku se jedná o způsob uložení dat v objektech uživatelského rozhraní. Jelikož tento termín využívají přímo vývojáři Vaadin frameworku, rozhodl jsem se jej zachovat i ve své práci.

Datový model je jedna z klíčových součástí Vaadin frameworku. Model umožňuje vázat komponenty uživatelského rozhraní přímo na data, která zobrazují a případně dovolit jejich editaci. Existují tři úrovně zanoření v hierarchii datového modelu: *property*, *item* a *container*. Ve srovnání s relačními databázemi by objekt *container* představoval tabulku, *item* by reprezentoval jeden řádek tabulky a *property* její sloupec. Datový model je realizován jako množina rozhraní. Nedefinuje reprezentaci dat, nechává ji plně na implementaci kontejneru. Je intenzivně využíván základními komponentami uživatelského rozhraní, zvláště pak tzv. *Field* komponentami. Klíčovou vlastností všech vestavěných *Field* komponent (například *TextField* nebo *PopUpDateField*) je, že samy dokáží udržovat svá data anebo být svázány s externím datovým zdrojem. Uložená hodnota je vždy dostupná skrze

rozhraní *Property*. S jedním datovým zdrojem může být svázáno více než jedna komponenta zároveň [2, strana 286]. Vztahy mezi rozhraními figurujícími v datovém modelu znázorňuje obrázek 3.7.



Obrázek 3.7: Vztahy mezi rozhraními v datovém modelu.
Obrázek převzat z Book of Vaadin

Datový model má mnoho důležitých a užitečných vlastností, jako je podpora pro upozornění na změny hodnoty *Property*. Zvláště pak kontejnery mají k dispozici velké množství pomocných rozhraní včetně těch, která umožňují indexování, třídění a filtrování dat. Také *Field* komponenty poskytují množství funkcí spojených s datovým modelem například buffering, validaci dat a lazy loading. Vaadin nabízí několik vlastních implementací rozhraní datového modelu [2, strana 287].

3.2.1 Má práce

Další důležitou součástí mé odborné praxe byla práce s datovým modelem Vaadin frameworku. Mezi činnosti spojené s touto prací patřila manipulace s POJO objekty a jejich kolekcemi, práce s režijními objekty aplikační logiky jako jsou posluchači událostí, práce s datovými kontejnery pro tabulky a stromy a vytváření nových komponent.

Příklad 3.5
Panel pro sběr a zobrazování systémových zpráv napříč informačním systémem je ideální příklad práce s datovým modelem Vaadin frameworku. Tento panel se skládá z tabulky, stromu, textového pole a skupiny *CheckBoxů*. Tabulka zobrazuje jednotlivé záznamy systémových hlášení. Textové pole slouží pro podrobné výpisy zpráv, které hlášení obsahují.

Strom, jež zobrazuje adresářovou strukturu souborů, v nichž byla hlášení vyvolána, spolu s *CheckBoxy* umožňuje filtrování zobrazených záznamů v tabulce.

Zdrojový kód v příloze 5 představuje proces vytvoření datového kontejneru, jeho přiřazení tabulce a naplnění tabulky daty. Sloupce jsou ve výsledné tabulce reprezentovány jako *Property* objekty kontejneru. *Property* lze do kontejneru přidat metodou *addContainerProperty()*, do které programátor předá tři parametry: objekt identifikující danou *Property* v rámci kontejneru, datový typ hodnoty této *Property* a její výchozí hodnotu. Tabulce je datový kontejner přiřazen pomocí metody *setContainerDataSource()*, řádky tabulky neboli *Itemy* kontejneru lze přidat skrze metodu *addItem()*. Každá buňka tabulky je od ostatních jednoznačně odlišena identifikátory objektů *Item* a *Property*. Tato ID mohou být jakékoliv objekty. Pro *Item* je Vaadin schopen použít své výchozí hodnoty, kde jako identifikátory řádků tabulky používá celá čísla začínaje od nuly. ID *property* musí programátor vždy specifikovat ručně. ■

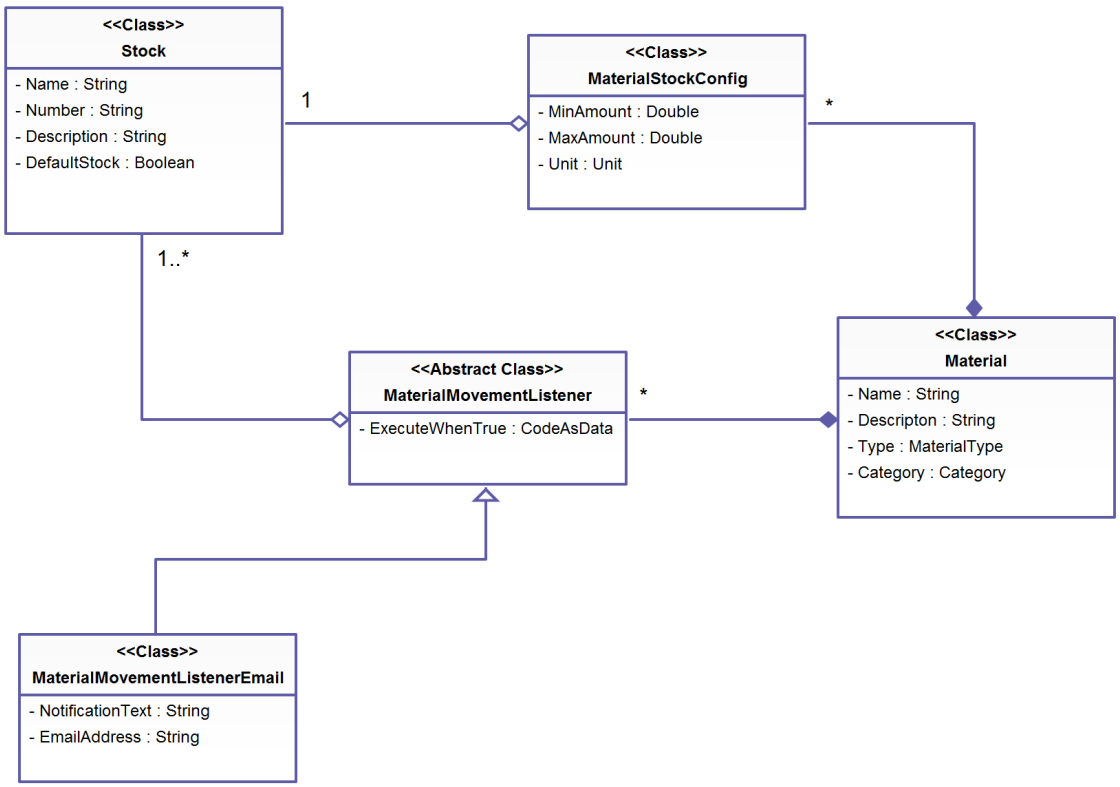
Příklad 3.6
Mezi výsledky mé práce patří také komponenta pro vyhledávání specifických záznamů napříč informačním systémem. Tato komponenta je vždy úzce svázána s konkrétní tabulkou, jejíž záznamy jsou filtrovány dle zadaných kritérií. Uživatelské Rozhraní komponenty je dynamické, mění se dle datového typu hodnot ve sloupci tabulky, podle kterého si uživatel přeje vyhledávat. Mezi podporované datové typy patří textové řetezce (*String*), číselné datové typy (*Integer, Double, Short, Long* a *Float*) a datum a čas (*LocalDate, LocalDateTime* a *Date*). Komponenta je vytvořena tak, že lze množinu podporovaných datových typů bez obtíží rozšířit.

Pro vyhledávání dle textových řetězců se komponenta chová jako textové pole bez jakýchkoliv kritérií aplikovaných na zadanou hodnotu. Je-li v komponentě nastaven číselný datový typ, skládá se ze dvou textových polí akceptující pouze číselné hodnoty. V případě vyhledávání dle data a času je uživatelské rozhraní podobné jako u číselných datových typů s tím rozdílem, že namísto textových polí se komponenta skládá ze dvou *PopUpDateField* komponent, které umožňují uživateli zadat datum a čas ručně nebo je vybrat pomocí rozvinovacího kalendáře. V případě, že chce uživatel filtrovat podle sloupce, jehož hodnoty jsou nepodporovaného datového typu, chová se komponenta stejně, jako by bylo nastaveno vyhledávání dle textových řetězců. Rozdíl je v implementaci použitého filtru, kde je namísto hodnoty objektu v konkrétní buňce tabulky použita jeho textová reprezentace získaná metodou *toString()*.

K výběru požadovaných záznamů využívá tato komponenta filtrů aplikovaných na datový kontejner tabulky. Filtr v příloze 6 je v komponentě využíván pro vyhledávání dle data a času. Aby objekt mohl vystupovat jako filtr musí implementovat rozhraní *Filter* a definovat jeho dvě metody. První z nich: *appliesToProperty()* slouží k určení, na které *Property* kontejneru(sloupce tabulky) se má filtr aplikovat. V uvedeném příkladě je filtr aplikován pouze na *Property*, která představuje uživatelem vybraný sloupec tabulky. Druhá metoda *passessFilter()* provádí samotnou restrikci řádků tabulky. Pokud pro daný

item kontejneru vrátí hodnotu *true* bude tento *item* zobrazen i po dokončení procesu filtrace. Zde se ověřuje, zda je datum a čas filtrované *property* každého *itemu*(řádku tabulky) v kontejneru mezi hodnotami *dateFrom* a *dateTo*. Tyto dvě hodnoty představují mezní data získaná od uživatele pomocí komponent *PopupDateField*. ■

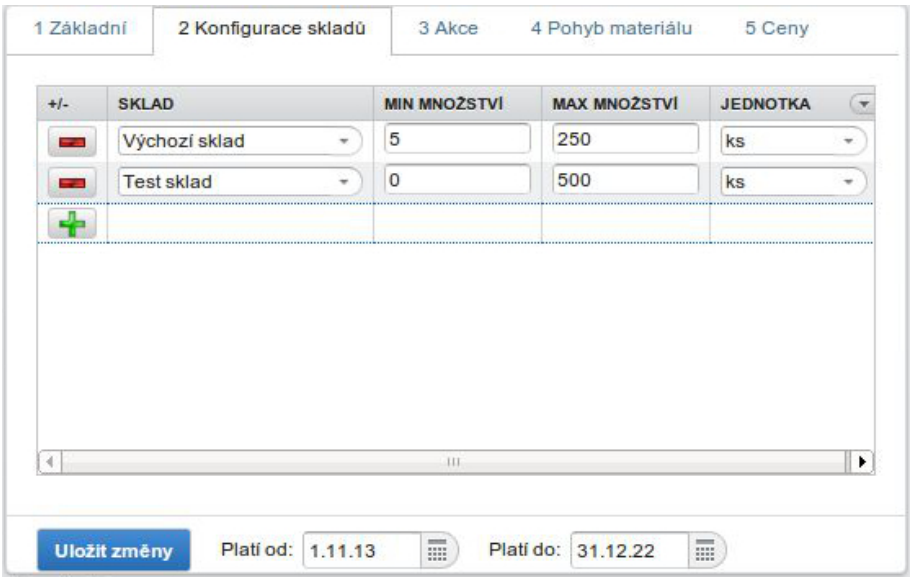
Příklad 3.7
V průběhu odborné praxe jsem také implementoval část doménové logiky z oblastí skladů a materiálů. Konkrétně se jedná o objekty, které slouží k odposlouchávání událostí přesunu materiálu mezi sklady a konfiguraci materiálu na skladě. Vzájemnou provázanost tříd, které jsem vytvořil a stávajících objektů doménového modelu znázorňuje třídní diagram na obrázku 3.8.



Obrázek 3.8: Třídní diagram doménových objektů z oblasti materiálu a skladů

Instance třídy *MaterialStockConfig* představují konfiguraci materiálu na určitém skladu. Mezi atributy této třídy patří minimální a maximální povolené množství materiálu na skladu a jednotka, ve které je toto množství uváděno. V systému v současné době figuruje jeden potomek abstraktní třídy *MaterialMovementListener*. Jeho primární účel je upozornit zainteresovanou osobu emailem při přesunech materiálu. Potomci třídy *MaterialMovementListener* na události přesunu materiálu reagují pouze, je-li splněna podmínka reprezentovaná atributem *ExecuteWhenTrue*.

Součástí tohoto úkolu byla také tvorba uživatelského rozhraní pro manipulaci se zde uvedenými objekty. Na obrázku 3.9 je zobrazeno uživatelské rozhraní pro manipulaci s objekty *MaterialStockConfig*. To je stejně jako rozhraní pro editaci objektů *MaterialMovementListener* tvořeno tabulkou, kde jednotlivé řádky představují instance těchto objektů.



Obrázek 3.9: Grafické uživatelské rozhraní pro editaci konfigurací materiálu

■

4 Integrační testy

Součástí mé odborné praxe bylo psaní integračních testů pro různé funkce jak webového klienta informačního systému, tak i mobilních aplikací pro platformu Android. Testy simulovaly interakce běžného uživatele s aplikací skrze její uživatelské rozhraní. K vytváření testů jsem použil dva nástroje: *Selenium* a *Robotium*. Popis těchto nástrojů jsem převzal z jejich oficiálních webových prezentací.

4.1 Selenium

Selenium je sada nástrojů pro automatizaci webových prohlížečů napříč více platformami. Poskytuje vysoce flexibilní funkce sloužící k vyhledávání prvků uživatelského rozhraní uvnitř objektové reprezentace HTML dokumentu a porovnávání očekávaných výsledků testů se skutečným chováním aplikace [6].

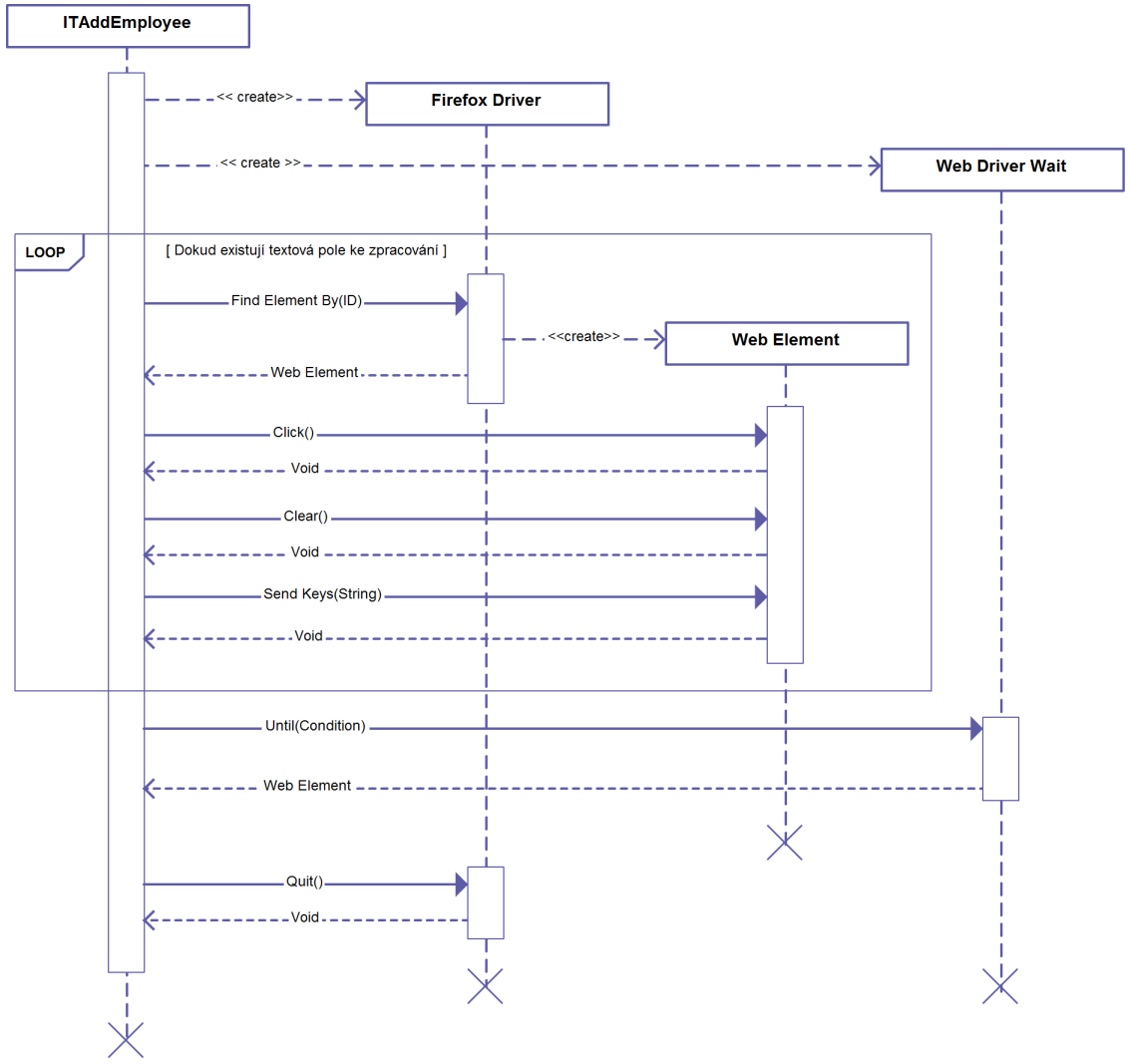
Nástroj Selenium jsem využil při testování webového klienta informačního systému. Cílem těchto testů bylo ověřit funkčnost přidávání a mazání požadovaných objektů jako jsou objednávky, zaměstnanci nebo výrobní příkazy. V jedné testovací třídě může být obsaženo několik samostatných testů, které by měly být navzájem nezávislé a jsou vykonávány každý samostatně. Test je reprezentován metodou, která je označena anotací *@Test*. Před provedením samotného testu je vždy nejprve volána speciální metoda, jejíž účel je připravit prostředí nezbytné k správnému průběhu testu. Tato metoda je identifikována anotací *@Before* a standardně se nazývá *setUp()*.

Selenium využívá k provádění testů objekt typu *RemoteWebDriver*, který reprezentuje a ovládá webový prohlížeč. Prostřednictvím tohoto objektu programátor například specifikuje URL adresu testované aplikace, spravuje okno a cookies prohlížeče nebo nastavuje maximální dobu, po kterou se má čekat na načtení testované aplikace. Z dostupných implementací objektu *RemoteWebDriver* jsem využil *FirefoxDriver*, který představuje internetový prohlížeč Mozilla Firefox. Další objekt nezbytný pro tvorbu testů je *WebDriverWait*. Ten například umožňuje v průběhu testu vyčkat na vykreslení konkrétního prvku uživatelského rozhraní nebo na přítomnost daného elementu v objektové reprezentaci HTML dokumentu. Stejně jako na začátku každého testu je i na jeho konci volána speciální metoda. Je označena anotací *@After*, měla by nést název *tearDown()* a slouží pro uvolnění prostředků potřebných k průběhu testu.

K manipulaci s prvky uživatelského rozhraní slouží instance třídy *WebElement*. Ty se dají získat dvěma způsoby:

- Metoda *findElementBy()* - Vstupní bod vyhledávání může být přímo objekt *RemoteWebDriver* nebo jiný *WebElement*. Prvky objektového modelu dokumentu jsou vyhledávány dle kritérií specifikovaných třídou *By*. Mezi tato kritéria patří ID elementu, název *CSS* třídy, název tagu elementu nebo XPath řetězec. Výsledek vyhledávání může být i kolekce všech *WebElement* objektů, které odpovídají zadaným podmínkám.

- Metoda *until()* - Na rozdíl od předchozího způsobu, je tato metoda volána na instanci objektu *WebDriverWait*. Primární účel metody until je pozastavit průběh testu dokud není splněna určitá podmínka nebo nevyprší stanovený časový limit. Podmínka je stejně jako u předchozího způsobu specifikována prostřednictvím třídy *By*. V případě, že není podmínka splněna ve stanovenou dobu, test končí neúspěšně. Metodu *until()* jsem nejčastěji využil při čekání na vykreslení konkrétního elementu uživatelského rozhraní. V případě úspěchu vrací tato metoda *WebElement* objekt, který reprezentuje požadovaný prvek webové stránky.



Obrázek 4.1: Sekvenční diagram průběhu selenium testu

Diagram na obrázku 4.1 znázorňuje průběh testu pro přidání nového zaměstnance do systému. Jako první se vytvoří objekty *FirefoxDriver* a *WebDriverWait*. Dále se inicializují

objekty *WebElement*. K nalezení těchto elementů je využita metoda *findElementBy()*, kde kritériem pro vyhledávání je ID požadovaných textových polí. Proces vepsání hodnoty do textového pole se skládá ze tří kroků. Na požadované textové pole se nejdříve klikne, poté se smaže jeho současná hodnota a vepíše nový text. V závěru testu se prostřednictvím metody *until()* čeká na zobrazení potvrzení o uložení. Není-li toto potvrzení zobrazeno v daném časovém limitu, test končí neúspěchem.

4.2 Robotium

Robotium je framework pro tvorbu instrumentačních testů na platformě Android s plnou podporou nativních a hybridních aplikací. Robotium umožňuje vývoj efektivních testů s minimální znalostí testované aplikace. Framework dokáže automaticky zpracovávat více Android aktivit, zvyšuje čitelnost již vytvořených testů oproti standardním instrumentačním testům a zvyšuje rychlost jejich průběhu [7].

Tento nástroj jsem využil k testování aplikací pro platformu Android, které jsou součástí informačního systému pro řízení výroby. Struktura instrumentačního testu platformy Android se oproti testu napsanému pomocí nástroje Selenium moc neliší. Jedna třída může stejně jako u Selenia obsahovat více testů, které jsou opět prováděny ve třech krocích. Hlavní rozdíl je v identifikaci metod představujících tyto kroky. Android za tímto účelem nevyužívá anotace ale názvy metod. Metody volané na začátku a konci každého testu jsou deklarovány ve třídě *ActivityInstrumentationTestCase2*, ze které vlastní testovací třídy dědí. Nazývají se *setUp()* a *tearDown()*. Každý test je reprezentován metodou, jejíž název začíná klíčovým slovem *test*, její viditelnost musí být *public* a návratová hodnota *void*. Jednotlivé testy jsou prováděny v abecedním pořadí.

Hlavní zbraní Robotium frameworku je třída *Solo*. Její instance je pro každý testovací případ vytvářena zvlášť uvnitř metody *setUp()*. Pomocí této instance programátor kompletně ovládá uživatelské rozhraní testované aplikace. Na začátku testu, uvedeném v příloze 7, se nejdříve provede ověření správnosti zobrazené aktivity pomocí metody *assertCurrentActivity()*. Následně jsou získány potřebné prvky uživatelského rozhraní, se kterými se dále pracuje. K jejich nalezení jsem využíval metody objektu *Solo* jako jsou *getButton()* nebo *getView()*. Po zadání přihlašovacích údajů a stisku tlačítka OK se čeká stanovený časový limit na zobrazení úvodní obrazovky aplikace. Pokud objekt *Solo* v časovém limitu nalezne specifikovaný element uživatelského rozhraní, test končí úspěšně. Po skončení testu jsou uzavřeny veškeré otevřené aktivity.

5 Jasper reporty

Popis knihovny JasperReports jsem převzal z její oficiální webové prezentace. Knihovna JasperReports je volně šiřitelný nástroj pro generování tiskových sestav kompletně vytvořen v jazyce Java. Je schopný využít libovolný zdroj dat, z něhož sestaví dokumenty, které mohou být prohlíženy, tisknuty nebo exportovány do různých formátů včetně HTML, PDF, XLS, DOC a OpenOffice formátů. Pro generování dokumentů jsou využívány XML šablony [3].

5.1 Šablony

Mým úkolem ve vztahu ke knihovně JasperReports byla tvorba JRXML šablon tiskových sestav. Finální dokument vygenerovaný pomocí těchto šablon představoval různé statistiky z oblasti procesu výroby nebo docházky zaměstnanců. Ke své práci na tomto úkolu jsem využíval jednak nástroj iReport, dodávaný přímo společností Jaspersoft a také jednu z komponent informačního systému, která nástroj iReport napodobuje.

Pro každou šablonu tiskové sestavy je nutné specifikovat zdroj dat, který bude využit k naplnění vygenerovaného dokumentu. Kromě XLS nebo CSV souboru je možné vstupní data specifikovat dotazem na databázi, čehož jsem sám využíval. Z pestré palety dotazovacích jazyků jsem si vystačil s SQL a EJBQL. Samotná šablona definuje strukturu výsledné tiskové sestavy pomocí široké škály objektů. Mezi základní prvky patří:

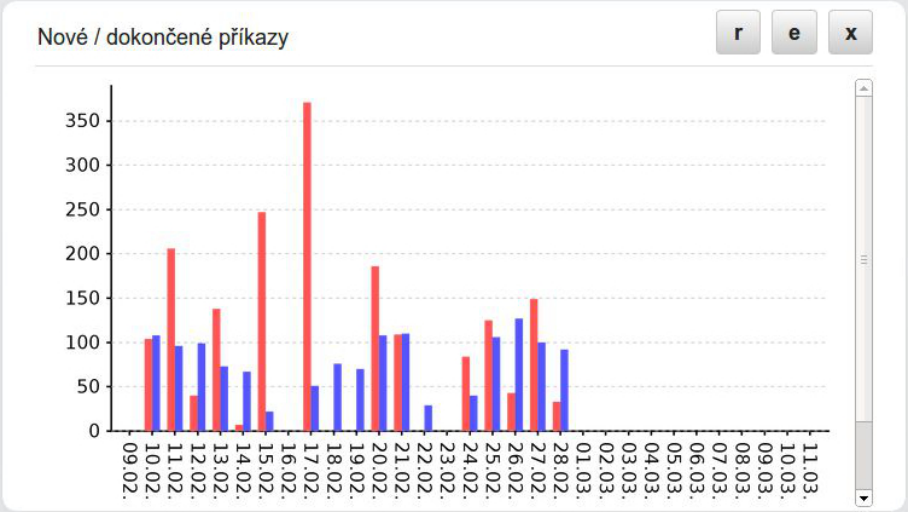
- Pole - Pole jsou prvky, které představují mapování mezi zdrojem dat a šablonou sestavy. Počet polí v šabloně není nijak omezen. Struktura výsledné množiny záznamů poskytnuté zdrojem dat by měla odpovídat všem deklarovaným polím v šabloně.
- Parametry - Parametry jsou reference na objekty, které jsou do sestavy předány během operace jejího plnění. Jsou užitečné pro předávání dat, která nemohou být zahrnuta ve zdroji dat sestavy. Parametry lze také použít v databázovém dotazu a měnit tak výslednou množinu záznamů ve zdroji dat.
- Textová pole, statický text - Textová pole a statický text jsou elementy reportu sloužící pro zobrazení textu. Na rozdíl od statického textu je hodnota textového pole výraz. Výrazy jsou utvářeny kombinací parametrů a polí reportu spolu se statickými daty. K psaní výrazů je ve výchozím nastavení používán jazyk Java.
- Základní grafické prvky - Zde patří například kruh nebo obdélník.

Šablony, které jsem vytvořil obsahovaly kromě polí a parametrů také standardní spojnicové ,sloupcové a koláčové grafy. Data zobrazená grafem jsou definována pomocí řad. Každá řada využívá výrazy pro specifikaci hodnot na osách X a Y. V těchto výrazech jsem použil hodnoty polí reportu s upraveným výstupním formátem, takže graf zobrazoval přímo zdrojová data sestavy.

Jedním z problémů, které jsem při práci na sestavách řešil, byla neúplnost množiny dat vrácené databázovým dotazem. V databázi neexistovaly záznamy pro konkrétní dny, které bylo ale potřeba v grafu také zahrnout. K řešení jsem využil *ChartCustomizer* objekty, které poskytují úplnou kontrolu nad grafy v Jasper reportech. Konkrétnímu grafu je lze přiřadit skrze atribut *Customizer class*. Metoda *customize()* ve zdrojovém kódu na příloze 8 prochází osu X grafu pro každou jeho řadu. Pokud narazí na chybějící den, přidá ho a hodnotu osy Y nastaví na nulu. Ostatní hodnoty pouze překopíruje do nového zdroje dat, který je v samotném závěru grafu nastaven.

5.2 Plovoucí panely

Statistiky vygenerované pomocí šablon, které jsem během odborné praxe vytvořil mohou být zobrazeny v podobě plovoucích panelů na domovské obrazovce informačního systému. Z šablony reportovací engine vygeneruje sestavu, která je vyexportována do HTML souboru. Tento soubor je poté znovu přečten a jeho obsah je použit jakožto hodnota Vaadin komponenty *Label*. Ta má nastaven speciální mód obsahu, což jí umožňuje interpretovat HTML kód stejně, jako to dělá běžný prohlížeč. Obrázek 5.1 zobrazuje jeden z těchto plovoucích panelů. Konkrétně se jedná o statistiku výrobních příkazů za posledních 30 dní. Viditelná je i funkce výše zmíněného *ChartCustomizer* objektu. V okamžiku vytvoření reportu nebyla v databázi data za měsíc březen. V grafu jsou ale tyto dny zahrnuty s nulovou hodnotou, přestože nejsou obsaženy ve zdroji dat sestavy. Kromě tohoto procesu jsem také naprogramoval logiku pozicování plovoucích panelů dle aktuální velikosti okna prohlížeče.



Obrázek 5.1: Plovoucí panel zobrazující statistiku výrobních příkazů

6 Ostatní technologie

V této kapitole se krátce věnuji několika dalším technologiím a nástrojům, se kterými jsem se během svého působení na odborné praxi setkal. Popis těchto technologií a nástrojů jsem převzal z jejich oficiálních internetových stránek.

6.1 Apache Subversion

Subversion je open source systém pro správu verzí zdrojových kódů založen firmou CollabNet, Inc. v roce 2000. Subversion je vyvíjen jako Apache Software Foundation projekt a je charakterizován svou spolehlivostí při uchovávání cenných dat, jednoduchostí svého modelu a také schopností uspokojit potřeby široké škály uživatelů a projektů od jednotlivců po rozsáhlé podnikové operace. Mezi vlastnosti Apache Subversion patří:

- **Atomické commity** - Žádná část commitu nenabývá účinku dokud nebyl celý commit úspěšně dokončen.
- **Uzamykání souborů** - Subversion podporuje(ale nevyžaduje) uzamykání souborů. Uživatel může být poté varován v případě, že se více lidí snaží upravit tentýž soubor najednou.
- **Interaktivní řešení konfliktů** - Tento mechanizmus je přístupný skrze několik API, takže klienti(jako jsou grafické editory zdrojového kódu) mohou nabízet interaktivní řešení konfliktů v závislosti na jejich rozhraní [5].

S tímto nástrojem jsem pracoval neustále. Využil jsem jej v případě, kdy jsem potřeboval aktualizovat lokální kopii zdrojových kódů, nahrát změny v lokální kopii projektu na server a případně řešit konflikty v souborech, na kterých kromě mé osoby pracoval ve stejnou dobu ještě někdo další.

6.2 Apache Maven

Apache Maven je nástroj pro správu softwarových projektů. Je pokusem aplikovat vzory na infrastrukturu sestavování projektu, za účelem zlepšení produktivity a pochopení daného projektu. Poskytuje standardizovaný způsob sestavování aplikací a jednoznačně definuje z čeho se má projekt skládat. Objektový model projektu (Project Object Model - POM) je základní stavební kámen Maven projektu. Jedná se o XML soubor, který obsahuje informace o projektu a konfigurační detaily, které Maven využívá k jeho sestavení [4].

Má práce s nástrojem Maven spočívala v úpravách POM souborů u modulů, kde bylo potřeba přidat závislosti na jiných modulech nebo knihovnách. Závislost je definována pomocí elementu *<dependency>* obsahující další tři elementy jednoznačně identifikující požadovaný modul nebo knihovnu. První je *<groupId>*, které vzájemně odlišuje jednotlivé projekty v repozitáři. Dále *<artifactId>* představuje jméno jar souboru a element *<version>* určuje jeho verzi.

7 Zhodnocení vlastních znalostí

7.1 Znalosti získané při studiu

V průběhu odborné praxe jsem měl příležitost uplatnit své znalosti především z oblasti programování. Konkrétně programovací jazyk Java, se kterým jsem se poprvé seznámil při studiu na Vysoké škole báňské v kurzu Programovací jazyky I, jsem využil u každého z úkolů, na kterých jsem pracoval. Další z důležitých schopností, které jsem během studia získal, je schopnost číst a vytvářet UML diagramy spojené s vývojem softwaru, jako je statický diagram tříd. Tyto znalosti mi pomohly lépe se orientovat v třídní hierarchii informačního systému, na kterém jsem pracoval a lépe pochopit zadané úkoly.

Naprosto nezbytná byla obecná znalost internetových technologií jako jsou HTML a CSS. S těmito technologiemi jsem se seznámil již před studiem na univerzitě, avšak mé znalosti zde byly prohloubeny a rozšířeny například o Sass, JavaScript a XPath. Tohoto jsem využil při vytváření kaskádových stylů a psaní testů webového klienta informačního systému. V poslední řadě bych zmínil dotazovací jazyk SQL, který jsem využil například při vytváření šablon tiskových sestav.

7.2 Scházející znalosti v průběhu praxe

Mezi své nedostatky během práce na zadaných úkolech bych považoval slabou znalost frameworků pro objektově relační mapování a dotazovacích jazyků s nimi spojených. Oblast mapování mezi objekty doménové logiky a relačními databázemi byla v bakalářském programu několikrát předmětem studia, avšak řešení spočívalo v ruční implementaci. Informační systém, na kterém jsem v průběhu praxe pracoval, využívá implementaci JPA frameworku Hibernate. S touto technologií jsem se během studia setkal v kurzu Java technologie, ale vzhledem k obsáhlé osnově předmětu byly pokryty pouze základy. Ty se v praxi ukázaly jako nedostatečné.

8 Závěr

Absolvování odborné praxe pro mne představuje zkušenost, které si velice cením. Měl jsem příležitost nahlédnout do prostředí firmy zabývající se vývojem softwaru a být součástí procesu vývoje komplexního informačního systému. Seznámil jsem se s několika pro mě novými technologiemi a vyzkoušel si práci v týmu. Odborná praxe mi také poskytla příležitost vyhodnotit stav svých dosavadních znalostí při práci na zadaných úkolech. Ve většině případů jsem využil alespoň částečně znalosti nabyté během studia na vysoké škole a také zjistil, ve kterých oblastech zaostávám. Celkově hodnotím absolvovanou praxi velmi pozitivně a při opětovném rozhodování bych znovu zvolil tuto variantu bakalářské práce.

Tomáš Blahut

9 Literatura

[1] GIRITON SYSTEMS S.R.O. *GIRITON* [online]. 2014 [cit. 2014-04-27]. Dostupné z: www.giriton.cz

[2] GRÖNROOS, Marko. VAADIN LTD. *Book of Vaadin: Vaadin 7 Edition - 2nd Revision* [online]. 2014 [cit. 2014-03-30]. Dostupné z: https://vaadin.com/download/book-of-vaadin/vaadin-7/pdf/book-of-vaadin.pdf

[3] JASPERSOFT CORPORATION. *Jaspersoft Community: JasperReports Library* [online]. 2014 [cit. 2014-03-30]. Dostupné z: http://community.jaspersoft.com/project/jasperreports-library

[4] THE APACHE SOFTWARE FOUNDATION. *Apache Maven Project* [online]. 2014 [cit. 2014-03-30]. Dostupné z: http://maven.apache.org/

[5] THE APACHE SOFTWARE FOUNDATION. *Apache Subversion* [online]. 2014 [cit. 2014-03-30]. Dostupné z: http://subversion.apache.org/

[6] *SeleniumHQ: Selenium Documentation* [online]. 2014 [cit. 2014-03-30]. Dostupné z: http://docs.seleniumhq.org/docs/

[7] *Robotium: User scenario testing for Android* [online]. 2014 [cit. 2014-03-30]. Dostupné z: https://code.google.com/p/robotium/

A Zdrojové kódy

```
public class VAddMaterialWindow extends Window
{
  private final Label labelMaterial = new Label();
  private final HorizontalLayout materialContent = new
     HorizontalLayout(labelMaterial);

  private final Button btnChooseActor = new Button("Vyberte
     aktera:");
  private final HorizontalLayout actorContent = new
     HorizontalLayout(btnChooseActor);

  private final TextField textAmount = new TextField();
  private final ComboBox comboUnit = new ComboBox();
  private final HorizontalLayout amountContent = new
     HorizontalLayout(textAmount, comboUnit);

  private final OptionGroup radioBtns = new OptionGroup("Pohyb:");
  private final ComboBox comboStockFrom = new ComboBox("Ze
     skladu:");
  private final ComboBox comboStockTo = new ComboBox("Do
     skladu:");
  private final PopupDateField dateField = new
     PopupDateField("Datum a cas:");
  private final FormLayout formWrapper = new
     FormLayout(materialContent, actorContent, amountContent,
     radioBtns, comboStockFrom, comboStockTo, dateField);

  private final TextArea textComment = new TextArea("Komentar:");
  private final Button btnSave = new Button("Ulozit");
  private final Button btnCancel = new Button("Zrusit");
  private final HorizontalLayout btnsContent = new
     HorizontalLayout(btnSave, btnCancel);

  private final VerticalLayout wrapper = new
     VerticalLayout(formWrapper, textComment);
  private final VerticalLayout content = new
     VerticalLayout(wrapper, btnsContent);

  public VAddMaterialWindow()
  {
      setContent(content);
      ...
  }
}
```

Výpis 1: Zdrojový kód uživatelského rozhraní dialogu pro pohyb materiálu

```
    private static class VVariationsWizardWindow extends Window {
       ...
           private final Table table = new Table();

           public VVariationsWizardWindow() {
              btnUp.setIcon(Icons16Utils.arrowUp);
              btnAdd.setPrimaryStyleName(ScssUtils.MY_BLUE_BUTTON);

              middle.setSizeFull();
              middle.setExpandRatio(left, 1.0f);
              middle.setSpacing(true);

              right.setSizeUndefined();

              bottom.setWidth("100%");
              bottom.setComponentAlignment(labelCounting,
                  Alignment.MIDDLE_LEFT);

              this.setWidth("550px");
              this.setHeight("500px");
              this.setContent(content);
       }
    }
```

Výpis 2: Zdrojový kód uživatelského rozhraní dialogu pro skladové karty

```
Label label = new Label();
ProgressBar bar = new ProgressBar();
CssLayout cssLayout = new CssLayout() {
  @Override
  protected String getCss(Component c) {
    if (c instanceof Label) {
      return "position: absolute; "
        + "text-align: center; "
        + "top: 0; "
        + "left: 0; "
        + "bottom: 0; "
        + "right: 0; "
        + "margin: auto;"
        + "font-weight: bold;";
    } else if (c instanceof ProgressBar) {
      return "margin-bottom: 1px; vertical-align: middle;";
    }
    return null;
  }
};

bar.setWidth("100%");
bar.addStyleName(ScssUtils.MY_BLUE_PROGRESS_BAR);
cssLayout.addComponents(bar, label);
cssLayout.setSizeFull();

CssLayout cssWrap = new CssLayout() {
  @Override
  protected String getCss(Component c) {
    if (c instanceof CssLayout) {
      return "position: relative;";
    }
    return null;
  }
};

cssWrap.addComponent(cssLayout);
cssWrap.setSizeFull();
```

Výpis 3: Umístění komponent na sebe pomocí CssLayoutu

```
@mixin AbstractButton {
     .AbstractButton,
     .AbstractButton.v-has-height,
     .AbstractButton.v-has-width {
       margin: 0;
       padding: 5px 10px;
       border-radius: 3px;
       -webkit-border-radius: 3px;
       -moz-border-radius: 3px;
       border: 1px solid #C6C6C6;
       background-color: #DFDFDF;

       white-space: nowrap;
       text-align: center;
       font-weight: bold;
       overflow:hidden;

       -webkit-box-sizing: border-box;
       box-sizing: border-box;
       -webkit-font-smoothing: antialiased;
     }
}

@mixin coloredButtons {
     @include AbstractButton;

   .blueButton,
     .blueButton.v-has-height,
     .blueButton.v-has-width,
     .blueButton.v-has-height.v-has-width {
     @extend .AbstractButton;

       color: white;
       border-color: #488dd5;
       background: rgb(70,141,213);
       background: -moz-linear-gradient(top, rgba(70,141,213,1)
          0%, rgba(40,108,178,1) 100%);
       background: -webkit-linear-gradient(top,
          rgba(70,141,213,1) 0%,rgba(40,108,178,1) 100%);
       background: linear-gradient(to bottom, rgba(70,141,213,1)
          0%,rgba(40,108,178,1) 100%);
     }
}
```

Výpis 4: Ukázka hierarchie Sass stylů pro tlačítka

```
public class VPanelErrorReader {
  private final String LOGGER_NAME = "Logger name";
    private final String LOGGER_DATE = "Datum";
    private final String LOGGER_LEVEL = "Level";
    private final String LOGGER_MESSAGE = "Zprava";
    private final Object[] visibleColumnsIds = {LOGGER_NAME,
        LOGGER_DATE, LOGGER_LEVEL, LOGGER_MESSAGE};

  private final Table tableErrors = new Table();

  public VPanelErrorReader() {
       IndexedContainer tableData = new IndexedContainer();
       tableData.addContainerProperty(LOGGER_NAME, String.class,
          null);
       tableData.addContainerProperty(LOGGER_DATE, Date.class,
          null);
       tableData.addContainerProperty(LOGGER_LEVEL, Level.class,
          null);
       tableData.addContainerProperty(LOGGER_MESSAGE,
          String.class, null);
       tableErrors.setContainerDataSource(tableData);

       List<LogRecord> logData =
          ModuleBus.getService(LoggerService.class).getLogEvents();

       for (LogRecord l : logData) {
              Object itemId = tableErrors.addItem();
              tableErrors.getContainerProperty(itemId,
                 LOGGER_NAME).setValue(l.getLoggerName());
              tableErrors.getContainerProperty(itemId,
                 LOGGER_DATE).setValue(new Date(l.getMillis()));
              tableErrors.getContainerProperty(itemId,
                 LOGGER_LEVEL).setValue(l.getLevel());
              tableErrors.getContainerProperty(itemId,
                 LOGGER_MESSAGE).setValue(l.getMessage());
       }
       tableErrors.setVisibleColumns(visibleColumnsIds);
  }
}
```

Výpis 5: Naplnění tabulky daty

```
public class MyDateBetween implements Filter {

      @Override
      public boolean passesFilter(Object itemId, Item item) {
         Object value =
             item.getItemProperty(propertyId).getValue();
         if (value instanceof LocalDateTime) {
            LocalDateTime ldtVal = (LocalDateTime) value;
            return (ldtVal.isEqual(dateFrom) ||
                ldtVal.isAfter(dateFrom))
                   && (ldtVal.isEqual(dateTo) ||
                       ldtVal.isBefore(dateTo));
         } else if (value instanceof LocalDate) {
            LocalDate ldVal = (LocalDate) value;
            return (ldVal.isEqual(dateFrom.toLocalDate()) ||
                ldVal.isAfter(dateFrom.toLocalDate()))
                   && (ldVal.isEqual(dateTo.toLocalDate()) ||
                       ldVal.isBefore(dateTo.toLocalDate()));
         }
         return false;
      }

      @Override
      public boolean appliesToProperty(Object propertyIdParam) {
         return propertyId != null &&
             propertyId.equals(propertyIdParam);
      }
}
```

Výpis 6: Ukázka filtru datového kontejneru Vaadin frameworku

```
public class MainTest extends
    ActivityInstrumentationTestCase2<Main> {
  private Solo solo;
  private Main activity;

  @Override
  protected void setUp() throws Exception {
    super.setUp();
    setActivityInitialTouchMode(false);
    activity = getActivity();
    solo = new Solo(getInstrumentation(), activity);
  }

  private void testLogin() {
    solo.assertCurrentActivity("Selhalo overeni spravnosti
        zobrazene aktivity.", Main.class);

    Button userBtn = solo.getButton("Login:");
    String userCaption = userBtn.getText().toString();
    Button pwdBtn = solo.getButton("Heslo:");
    String pwdCaption = pwdBtn.getText().toString();
    Button numberOneBtn = solo.getButton("1");

    solo.clickOnView(userBtn);
    solo.clickOnView(numberOneBtn);
    assertEquals("Selhala kontrola jmena uzivatele",
        userBtn.getText(), userCaption + numberOneBtn.getText());

    solo.clickOnView(pwdBtn);
    solo.clickOnView(numberOneBtn);
    assertEquals("Selhala kontrola hesla uzivatele",
        pwdBtn.getText(), pwdCaption + "*");

    solo.clickOnButton("OK");

    boolean elementVisible = solo.waitForText("Vyrobni prikaz:",
        1, DEFAULT_WAIT_TIME);
    assertTrue("Selhala kontrola zobrazeni panelu po
        prihlaseni.", elementVisible);
  }

  @Override
  protected void tearDown() throws Exception {
    solo.finishOpenedActivities();
  }
}
```

Výpis 7: Robotium test pro přihlášení do aplikace.

```
public class GraphMissingValuesCommonFiller extends
    JRAbstractChartCustomizer {
  public static final String DATE_FROM_PARAMETER_NAME =
      "dateFrom";
  public static final String DATE_TO_PARAMETER_NAME = "dateTo";
  public static final String DATE_PATTERN = "dd.MM.";

  @Override
  public void customize(JFreeChart jfc, JRChart jrc) {
    LocalDate dateFrom = new LocalDate().minusDays(30);
    LocalDate dateTo = new LocalDate();

    CategoryPlot categoryPlot = (CategoryPlot) jfc.getPlot();
    DefaultCategoryDataset oldDataset = (DefaultCategoryDataset)
        categoryPlot.getDataset();
    DefaultCategoryDataset newDataset = new
        DefaultCategoryDataset();

    List seriesIdentifiers = oldDataset.getRowKeys();
    List xAxisValues = new ArrayList(oldDataset.getColumnKeys());
    Collections.sort(xAxisValues);

    String currentSeries;
    LocalDate currentDay;
    for (int i = 0; i < seriesIdentifiers.size(); i++) {
      currentSeries = (String) seriesIdentifiers.get(i);
      currentDay = dateFrom;

      while (currentDay.isBefore(dateTo) ||
          currentDay.isEqual(dateTo)) {
        try {
          Number yAxisValue = oldDataset.getValue(currentSeries,
              currentDay);
          newDataset.addValue(yAxisValue, currentSeries,
              currentDay.toString(DATE_PATTERN));
        } catch (UnknownKeyException ex) {
          newDataset.addValue(0, currentSeries,
              currentDay.toString(DATE_PATTERN));
        } finally {
          currentDay = currentDay.plusDays(1);
        }
      }
    }
    categoryPlot.setDataset(newDataset);
  }
}
```

Výpis 8: Třída doplňující chybějící dny v posloupnosti osy X grafu.