Vysoká škola ekonomická v Praze

Fakulta informatiky a statistiky
Katedra informačních technologií

Studijní program: Aplikovaná informatika

Obor: Informatika

Tvorba webových aplikací v JSF PrimeFaces

BAKALÁŘSKÁ PRÁCE

Student : Jakub Macoun

Vedoucí : Ing. Jarmila Pavlíčková

Oponent : Mgr. Zbyněk Šlajchrt

2012

Prohlášení

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a že jsem uvedl všechny použité prameny a literaturu, ze které jsem čerpal.

V Praze dne 9. května 2012 .....

Jakub Macoun

Abstrakt

Práce se zaměřuje na tvorbu webových aplikací, primárně na technologii JSF s využitím frameworku PrimeFaces. Jejím cílem je popis obou těchto technologií a poté i jejich srovnání s nejužívanější technologií určenou pro webovou tvorbu, PHP. Způsob tvorby v JSF a PrimeFaces je v práci rozebrán podrobně a následně ukázán na příkladu.

Pokud má čtenář minimální znalosti z oblasti webové tvorby a rozumí principům objektově orientovaného programování, je schopen z textu pochopit základy práce s JSF a PrimeFaces a povšimnout si výhod, které tyto technologie přinášejí. Informace uvedené v práci jsou přínosné například z hlediska volby mezi dvěma nástroji pro webovou tvorbu, Javou a PHP. Informace uvedené v práci jsou podloženy funkční ukázkovou aplikací, jejíž zdrojový kód je součástí příloh.

Práce se nejprve věnuje současným webovým technologiím s důrazem na programovací jazyky PHP a Java. Následně popisuje tvorbu webových aplikací v Java EE a zaměřuje se na frameworky JSF a PrimeFaces. Nakonec je jako součást práce vytvořena ukázková webová aplikace a popsána její architektura.

Klíčová slova: webová aplikace, Java, Java EE, JSF, PrimeFaces, PHP, jQuery.

Abstract

The work is focused on creation of web applications, primary on JSF technology with usage of PrimeFaces framework. Intention of the work is description of both technologies and comparison of them to the most widely used web technology, PHP. The way of creation using JSF and PrimeFaces is described in detail and shown on an example.

Thanks to the work a reader is able to understand basics of work with JSF with PrimeFaces if he has basic knowledge of web technologies and object oriented programming. Also the reader is able to notice advantages afforded by these technologies. Information in the work is useful for example in case of choosing between two widely used web technologies, PHP and Java. It is practically shown in functional example application. Its source code is enclosed.

At first there are explained today's web technologies with PHP and Java programming languages highlighted. After that web creation using Java EE and frameworks JSF and PrimeFaces is focused on. At the final part the example application is created and its architecture is described.

Keywords: web application, Java, Java EE, JSF, PrimeFaces, PHP, jQuery.

Obsah

1 Úvod ... 6
1.1 Dosud publikované práce věnující se tématu ... 7
2 Moderní webové technologie ... 9
2.1 Databáze ... 9
2.2 JavaScript a jQuery ... 10
2.3 PHP ... 11
2.4 Java EE ... 15
3 Webové aplikace v Javě ... 18
3.1 Databáze a objektově relační mapování ... 18
3.2 Servlet ... 19
3.3 JavaServer Pages ... 20
4 JavaServer Faces ... 22
4.1 Struktura ... 22
4.2 Facelet ... 22
4.3 Konfigurace ... 23
4.4 Managed bean ... 24
4.5 JSF (faceletová) stránka ... 27
4.6 Lokalizace ... 30
4.7 Webové formuláře ... 31
4.8 Phase listener ... 35
5 Sada komponent PrimeFaces ... 36
5.1 Další odvětví PrimeFaces ... 37
5.2 Instalace ... 37
5.3 Komponenty ... 38
6 Ukázková aplikace TheShop ... 46
6.1 Uživatelské rozhraní ... 46
6.2 Struktura aplikace ... 48
7 Závěr ... 50
8 Terminologický slovník ... 51
9 Seznam literatury ... 57
10 Rejstřík ... 60
11 Přílohy ... 62
11.1 Diagram tříd ... 62
11.2 Ukázka uživatelského rozhraní ... 63

1 Úvod

Tvorbou webových aplikací se zabývám již řadu let a vytvořil jsem jich již několik. Často šlo o pouhé prezentace, jindy zas o robustnější projekty, jejichž výsledkem byly aplikace vykonávající nejrůznější funkce. Celou dobu jsem se zaměřoval na jednu technologii tvorby webových aplikací, konkrétně na programovací jazyk PHP. Poté, co jsem nastoupil jako student na Vysokou školu ekonomickou v Praze, jsem začal poznávat programování ze zcela nového pohledu, a to hlavně díky programovacímu jazyku Java, který jsem se zde začal učit. V poslední době jsem se začal zajímat o využití Javy jako prostředku k vytváření webových stránek a zjistil jsem, že je v mnoha ohledech lepší, a to jak z analytického hlediska, tak toho programového. Konkrétně mně zaujala technologie JavaServer Faces s použitím frameworku PrimeFaces. Poté, co jsem s těmito technologiemi začal pracovat, mi bylo jasné, že toto je směr, kterým se ve své budoucí tvorbě chci ubírat.

Tyto skutečnosti mě vedly k napsání této práce na téma „Tvorba webových aplikací v JSF PrimeFaces“. Jedním z jejích cílů je stručný náhled na současnou situaci webových technologií a způsoby jejich využití. Zvýšenou pozornost věnuji jazyku PHP jako standardu tvorby webových aplikací v dnešní době. Práce má za hlavní cíl vysvětlit principy frameworků JavaServer Faces a PrimeFaces. Tím dalším cílem je porovnání práce s PHP a práce s těmito dvěma frameworky.

Pro dosažení těchto cílů se budu kromě výše zmíněných témat věnovat také vytvoření ukázkové webové aplikace, jejímž účelem bude demonstrace technologií, kterým se tato práce věnuje. Výstupem práce tedy bude kromě informací o Java webových technologiích také funkční webová aplikace.

Čtenář této práce by měl mít základní informace o způsobu fungování internetu a o základních technologiích užívaných k tvorbě webových stránek. Základním předpokladem je znalost jazyků XML, XHTML, CSS, základní znalost protokolu HTTP a obecné webové terminologie. Čtenář by měl dále znát principy objektově orientovaného programování.

1.1 Dosud publikované práce věnující se tématu

Základním zdrojem informací o JSF a PrimeFaces je internet, který je plný webů obsahujících nejrůznější informace. Mimo internet ale existuje řada knižních publikací, které pojednávají o tématu z různých pohledů. Některé se věnují Javě jako programovacímu jazyku celkově, jiné jsou zaměřené konkrétně na její odvětví určené pro tvorbu webových aplikací.

1.1.1 Java EE 6 with GlassFish 3 Application Server [4]

Jde o jednu z velmi užitečných knižních publikací, která svého čtenáře uvede do problematiky Java Enterprise Edition (bude vysvětleno dále v práci) a ukáže mu nejpoužívanější způsoby tvorby webových aplikací. Autorem je David Heffelfinger a kniha byla poprvé vydána v roce 2010, informace v ní jsou tedy aktuální a konkrétně JavaServer Faces je věnována jedna kapitola obsahující ty nejdůležitější informace. Čtenář zde ale nenajde všechny informace o JSF, je proto vhodné použít jiné, nejlépe internetové zdroje pro doplnění pokročilých informací.

1.1.2 Webináře.cz – JavaServer Faces 2.0 [25]

Výborným úvodem do problematiky JSF je záznam online semináře na portálu webinare.cz. Seminář vede akademický pracovník FIT ČVUT Ing. Zdeněk Troníček, Ph.D. a postupně vysvětluje pomocí online prezentace základy JSF od začátku. Všechny části prezentace jsou doprovázeny příklady, takže jejich pochopení není problém.

1.1.3 PrimeFaces.org [18]

Největší množství informačních materiálů k frameworku PrimeFaces lze nalézt na jeho webových stránkách – primefaces.org. Kromě informací, které lze vyčíst z místních sociálních aplikací (např. forum), poslouží velmi dobře jako komplexní manuál na užívání frameworku uživatelská příručka [1], která je na stránkách ke stažení ve formátu PDF. Webový portál je navíc vybaven tzv. show casy, na kterých je užití jednotlivých částí frameworku demonstrováno.

1.1.4 Jiné akademické práce věnované tématu

O využití Javy jako webové technologie bylo napsáno několik bakalářských a diplomových prací. Mezi těmi vypracovanými na VŠE je jedna bakalářská práce, která se zabývá tvorbou webových aplikací pomocí frameworku JSF. Jde o práci s názvem Vývoj webo-

vých aplikací pomocí frameworku JavaServer Faces, autorem je David Henel a práce byla napsána v roce 2010.

Dále bylo na VŠE napsáno několik bakalářských a diplomových prací zabývajících se frameworky v Javě a jedna z těchto prací se částečně věnuje frameworku PrimeFaces, nicméně nejde o její hlavní zaměření. Jedná se o diplomovou práci s názvem Web-Based Information System for SME napsanou v roce 2011 Johanem Hornofem.

2 Moderní webové technologie

V případě tvorby dynamických stránek se programátor neobejde bez nějakého programovacího jazyka, který umožňuje tvorbu webových aplikací. Tyto jazyky obvykle generují HTML kód podle zadaných požadavků uživatele a podle dalších okolností, jako jsou například data v databázi. V této kapitole se budu věnovat těm dnes nejpoužívanějším programovacím jazykům a databázím, které jsou v současnosti používány. Mezi dnes nejčastější programovací jazyky patří PHP, Java, dále např. ASP.NET, Perl atd.

2.1 Databáze

Databáze není čistě webovou technologií, nicméně se bez jejího využití většina webových aplikací neobejde. Databázových produktů je k dispozici celá řada a každý z nich má různé vlastnosti a je využitelný pro jiné druhy projektů. Přenos dat a práce s databázemi na základní úrovni jsou prováděny pomocí jazyka SQL. V této podkapitole se budu věnovat jen některým nejčastějším typům.

2.1.1 MySQL

Databáze MySQL patří k těm nejpoužívanějším pro webové aplikace. Její správa není náročná a hlavně je poskytována zdarma. Poskytuje všechny typické služby, které jsou při práci s databází potřeba, její výkon je průměrný a je přizpůsobena pro většinu nejpoužívanějších programovacích jazyků. Hodí se tak pro malé až středně velké projekty.

2.1.2 PostgreSQL

PostgreSQL je konkurenčním projektem MySQL, obě databáze si jsou velmi podobné. PostgreSQL nicméně patří k těm méně oblíbeným a méně používaným, za některých okolností totiž dosahuje horších výkonů.

2.1.3 Oracle

Častým řešením je databáze od společnosti Oracle, která je v současnosti k dispozici ve verzi 11g. Je často využívána ve velkých komerčních projektech, a to díky plné podpoře skriptování a dalších funkcí, které open-source databáze (např. MySQL) v takovém rozsahu neposkytují. Databáze Oracle je také mnohem výkonnější.

2.2 JavaScript a jQuery

JavaScript je programovací jazyk fungující na straně klienta, jeho skripty tedy spouští webový prohlížeč. I přes podobu v názvu nemá JavaScript téměř nic společného s jazykem Java. Podobné pojmenování bylo zvoleno kvůli mírné podobě syntaxí obou jazyků, to je ale jediný výraznější společný rys. JavaScript je základem pro každou moderní a uživatelsky přívětivou webovou aplikaci, umožňuje například zobrazování dialogových oken, měnit obsah již načtených stránek apod.

Na základě JavaScriptu funguje technologie zvaná AJAX. Ta umožňuje interaktivní načítání specifických dat a jejich zpracování, například jejich zobrazení v již načtené webové stránce. Díky AJAXu lze dosáhnout velice kvalitního uživatelského rozhraní webové aplikace, je ale třeba vytvořit velké množství zdrojového kódu. Z toho důvodu vznikl JS framework jQuery.

jQuery je knihovna napsaná v JavaScriptu a umožňující snadnou manipulaci jak s AJAXem, tak s prostým JavaScriptem, a to vše velmi efektivně. Svou funkčností pokrývá veškeré akce, které lze dělat pomocí prostého JavaScriptu s tím rozdílem, že práce se zdrojovým kódem je jednoduší a výsledná akce bývá například doprovázena grafickým efektem apod.

jQuery samotné i tak ale neobsahuje zcela vše, co je třeba k vytvoření příjemného uživatelského rozhraní. Nebývá to ale problém, protože existuje mnoho rozšíření jQuery, která přidají například další využitelné komponenty. Jednou takovou komponentou může být interaktivní kalendář pro výběr data a času ve formuláři.

Syntaxe jQuery je rozmanitá, věnovat se ale budu jen tzv. selektorům, které jsou potřeba pro porozumění jedné z dalších kapitol. Selektor je nástroj, díky kterému lze vybrat některý z elementů v aktuálním HTML dokumentu a dále s ním pracovat. Jeho syntaxe je *$("...")* a místo tří teček uprostřed je třeba dosadit cestu k elementu. Způsobů, jak toto udělat, je několik a lze vybrat i více elementů jedním selektorem. Jeden z těchto způsobů je velmi podobný syntaxi CSS, která využívá určené znaky k výběru elementů k formátování. Například pro výběr elementu s atributem *id* s hodnotou *testElement* bude selektor vypadat takto: *$("#testElement")*. Lze vybírat i podle jiných atributů, např. *class*, vybraných elementů tak může být více než jeden. V takovém případě bude selektor v této po-

době: *$(".niceElement")*. Selektor bude odkazovat na všechny elementy třídy *niceElement*.

2.3 PHP

PHP je programovací skriptovací jazyk určený primárně pro tvorbu webových prezentací a aplikací fungující na straně serveru. Jeho vývoj začal v roce 1994, kdy Rasmus Lerdorf vytvořil jednoduchou webovou aplikaci v jazyce Perl, později C. Uživatelé tohoto systému měli požadavky na vylepšení, nakonec tak vznikl systém s názvem Personal Home Page Tools – zkráceně PHP. V pozdějších fázích vývoje byl tento systém přejmenován na Hypertext Preprocessor. V současnosti je PHP distribuován ve verzi 5.4.0.

PHP sice podporuje objektově orientované programování (hlavně od verze 5), nicméně jde stále primárně o skriptovací programovací jazyk. Vývojáři umožňuje definovat třídy a vytvářet objekty podobně jako ostatní objektově orientované programovací jazyky. Na rozdíl od nich ale nejsou tyto objekty uchovávány po delší dobu, ale jen po dobu provádění daného skriptu. Už jen kvůli této skutečnosti nelze plně využít výhod objektově orientovaného programování. Za hlavní výhodu PHP je považována jeho jednoduchost, je snadné se s tímto jazykem naučit i pro začínající programátory. Ve větším rozsahu je ale tato jednoduchost spíše na škodu, objem zdrojového kódu nepřiměřeně roste a program je náchylnější na chyby. PHP skripty jsou umístěny v souborech s příponou *php*.

Pro PHP existuje řada frameworků a dalších pomocných nástrojů. Některé jsou zaměřeny komplexně – snaží se pokrýt veškeré nejčastěji řešené problémy při tvorbě webové aplikace v PHP, některé se zas zaměřují jen na konkrétní část tvorby. Příkladem takového komplexního frameworku je například Zend. Konkrétně na objektově relační mapování je zaměřen například framework Doctrine.

V této podkapitole využiji svých několikaletých zkušeností s prací v tomto programovacím jazyce a nastíním, jak vypadá tvorba webové aplikace v PHP bez využití komplexně zaměřených frameworků. Účelem této podkapitoly je vyobrazení tvorby webových aplikací v PHP a její následné porovnání s tvorbou v JSF.

2.3.1 Vytvoření webové stránky

Vytvoření samostatné webové stránky v PHP je jednoduchá záležitost. Statický HTML kód stačí napsat přímo do souboru s PHP skriptem, programová část je oddělena tagy

<?php a *?>*. V programové logice mezi těmito tagy lze poté pomocí příkazu *echo* vygenerovat další HTML kód.

V takovém případě ale dochází k problému smíchání dvou vrstev aplikace a kompletnímu znepřehlednění zdrojového kódu. Pokud navíc takto vytvořená stránka využívá data z databáze, do hry vstupuje i persistentní vrstva.

Při tvorbě více podobných stránek navíc nastane další problém. V případě nevyužití některých z komplexních PHP frameworků je potřeba pro každou stránku vytvořit nový skript a do něj znovu vepsat HTML kód, který bývá podobný, jako na ostatních stránkách. Dochází tak k několikanásobnému opakování kódu.

2.3.2 Smarty a Doctrine

K vyřešení výše zmíněných problémů jsem při práci s PHP využíval šablonový systém Smarty a ORM framework Doctrine.

Hlavním přínosem Smarty je oddělení HTML kódu od aplikační logiky. Ta zůstává stále ve skriptech, zatímco HTML kód (případně jiný výstup – např. RSS) je umístěn v šablonách, tedy souborech s příponou *tpl*. Systém podporuje i zobrazení skriptem vygenerovaných dat na stránce. Například na iteraci polí je využíván Smarty tag *{section}*. Další výhodou šablon je fakt, že jednu šablonu lze použít na více stránkách. Odpadá tak časté opakování kódu, protože opakující se HTML je vepsáno v jedné šabloně, která se pak vkládá tam, kde je to potřeba.

Oddělení persistentní vrstvy od aplikační logiky zajišťuje Framework Doctrine. Ten je podobný Java frameworku Hibernate (viz dále), a to jak způsobem konfigurace a mapování jednotlivých persistentních tříd, tak zacházení s instancemi těchto tříd.

2.3.3 AJAX

Základním způsobem, jak pracovat s AJAXem, je prosté napsání JavaScriptové funkce, která čte výstup z PHP skriptu a ten poté reflektuje na danou část stránky. Tento způsob je poněkud nepohodlný a v samotném základu nenabízí tolik možností, jako jQuery.

jQuery lze v PHP aplikacích využívat poměrně dobře, v některých případech je jen potřeba obstarat patřičné generování výstupů PHP skriptem. Jde o výstupy generované například po odeslání webového formuláře, jež mají uživatele seznámit se stavem odeslání, tedy například potvrdit odeslání nebo naopak zobrazit patřičnou chybovou hlášku.

2.3.4 Formuláře

Formulář má v PHP aplikaci dvě vrstvy, obě většinou podobného rozsahu. První vrstvou je HTML kód formuláře, tou druhou skript (nebo jeho část), který má data z formuláře zpracovat.

Co se týče první vrstvy, je potřeba vytvořit HTML kód formuláře. To zpravidla není velký problém, pracnější činností je ale validace odesílaných dat.

Při zpracování dat z odeslaného formuláře je potřeba napsat kód různé náročnosti, rozsah práce záleží na tom, co chce programátor s daty dělat. Důležitým krokem před tímto zpracováním je validace dat, například kontrola vyplněných povinných polí formuláře. Tuto kontrolu je nutné provést (pokud je potřebná) vždy na straně serveru, v PHP skriptu. Pro uživatelskou přívětivost je ale vhodné tuto kontrolu provádět nejprve na straně klienta, JavaScriptem před odesláním. Uživatel tak nemusí čekat na odeslání formuláře jen proto, aby se dozvěděl, že je formulář vyplněný chybně a musí ho vyplňovat znovu. Tato dvojí kontrola způsobuje, že programátor při vytváření formuláře musí psát validační kód zdvojeně, jednou v JS a podruhé v PHP. Tato skutečnost programátora při práci velmi zdržuje.

Kontrolu pomocí JavaScriptu lze úplně vypustit v případě odeslání formuláře pomocí AJAXu. V takovém případě není velkým problémem kontrolu provádět jen na serverové straně a AJAXem poté odeslat výsledky kontroly zpět ke klientovi. Uživatel na klientské straně sice bude chvíli čekat na vyhodnocení formuláře, nemusí ale čekat na nové načtení celé stránky a v horším případě znovu vyplňovat všechna zadaná data.

2.3.5 Datové tabulky a seznamy

Tvorba datových tabulek či seznamů bývá v PHP časově velmi náročná. Vygenerování seznamu probíhá v několika fázích.

První fází je výběr dat (většinou z databáze) a jejich uspořádání do pole. Toto pole je následně odesláno do Smarty šablony.

V té je poté potřeba tento seznam zobrazit, je tedy potřeba v HTML vytvořit tabulku či jiný druh výpisu a v tom data zobrazit.

Samotné zobrazení dat často nestačí, je potřeba, aby bylo možné provádět s každým záznamem nějakou akci. Většinou jde o smazání či editaci daného záznamu. V tabulce se to

dá vyřešit například přidáním dalšího sloupce s patřičnými akčními tlačítky, které skriptu, jenž danou akci zprostředkovává, předají druh akce, který se má provést a unikátní identifikátor daného záznamu. Tento skript poté danou akci provede.

Časově nejnáročnější je ale naprogramovat stránkování. Drtivá většina datových seznamů se může skládat z velkého množství záznamů a je potřeba je proto rozdělit do stránek, aby nedošlo k zahlcení klientského prohlížeče. V takovém případě je potřeba provést zásah do všech výše zmíněných fází tvorby seznamu. Je nutné omezit výběr dat tak, aby byla vybírána jen data na dané stránce, v šabloně je potřeba zobrazit informaci o stránkování (počet stránek, aktuální stránka atd.). Je také potřeba synchronizovat výběr stránky s akcemi na jednotlivých záznamech.

2.3.6 Práce s databází

Díky Doctrine je programování přístupu k datům v databázi poměrně svižné. Webové aplikace v PHP jsou ale náročnější na databázi, a to díky faktu, že všechny objekty jsou po dokončení skriptu odstraněny. Při každém novém načtení stránky je tak potřeba některé údaje uložené v databázi znovu načíst. Zátěž databáze u větších aplikací tak bývá poměrně vysoká.

2.3.7 Server

Server, na kterém běží PHP aplikace, musí být patřičně softwarově vybaven. Základním požadavkem jsou samotné knihovny jazyka PHP. Dále je nutné na serveru provozovat software, který běh webových aplikací obstará. Nejrozšířenějším serverovým softwarem je Apache HTTP Server, který umožňuje jak rychlé prohlížení statického obsahu, tak řízení PHP skriptů.

2.3.8 Vývojová prostředí

Pro vývoj PHP aplikací není teoreticky nutné využít žádný specializovaný software. PHP skripty jsou v podstatě obyčejné textové soubory, které lze upravovat například ve Windowsové aplikaci Poznámkový blok. Takové řešení je ovšem pro programátora velmi nepraktické a existuje proto celá řada komerčních i nekomerčních softwarových produktů, které jsou specializovány pro vývoj v PHP.

Asi nejznámějším editorem je PSPad. Jde o textový editor se základními funkcemi podporujících programování jako je zvýrazňování syntaxe apod. Kromě PHP se specializuje i na jiné programovací jazyky.

Existují i produkty specializované jen na PHP a vybavené větším počtem funkcí než je tomu například v případě PSPadu. Většinou jde ale o komerční software, jsou to například vývojová prostředí phpDesigner, PHPEdit a další.

2.4 Java EE

V roce 1991, kdy počítačové technologie byly na úsvitu, začal menší tým pracovníků (tzv. „Green Team“) společnosti Sun pracovat na moderním programovacím jazyce, který by byl použitelný ve všech digitálních zařízených. Vznikla tak Java, multiplatformní open-source programovací jazyk, jehož rozmach nastal v roce 1995. Společnost Sun vyvinula Javu až do současné podoby, nicméně v roce 2010 byla koupena společností Oracle.

Samotná Java je programovacím jazykem, který lze využít k naprogramování různých druhů aplikací na různé platformy. Umožňuje tvorbu desktopových aplikací, serverových aplikací atd. V takovém případě jde o standardní edici, neboli Java Standard Edition. Na rozdíl od PHP, které je skriptovacím jazykem, lze tedy v Javě vytvářet aplikace, jejichž instance existuje dlouhodobě a lze mluvit o plné podpoře objektově orientovaného programování.

Na vývoj webových aplikací je zaměřená tzv. podniková edice, neboli Java Enterprise Edition. Je postavená na základě Java SE a má definované části tvořící jádro Java EE. Jde o programovací jazyk užívaný na straně serveru.

Tato kapitola se věnuje pouze základům programovacího jazyka Java standardní a podnikové edice, vývoj webových aplikací v Javě je detailně rozebrán v následující kapitole.

2.4.1 Vývojová prostředí

Pro rychlý vývoj aplikací v Javě existuje několik vývojových prostředí (zkráceně IDE) specializovaných právě na Javu. V současnosti jsou nejpoužívanějšími a nejznámějšími IDE NetBeans a Eclipse. Volba, které IDE používat, záleží na programátorovi, obě dvě jsou vysoce kvalitní a výběr je většinou otázkou „programátorského vkusu“ a zvyku.

NetBeans vznikl původně jako studentský projekt a byl později odkoupen k dalšímu rozvoji. Kromě základní podpory, jako je zvýrazňování syntaxe, našeptávání, sestavení aplikace apod. nabízí celou řadu dalších funkcí, jako je např. podpora SVN. Velkou nevýhodou NetBeans je jeho rychlost, jen samotné zapnutí a kompletní inicializace trvá na průměrném počítači několik minut.

Druhým vývojovým prostředím na kvalitativně stejné úrovní je Eclipse. Nabízí podobné možnosti jako NetBeans, jeden z rozdílů spočívá např. ve vysoké podpoře zásuvných modulů, na které se NetBeans příliš nesoustředí. Oproti NetBeans je rychlejší, nicméně trpí častějším výskytem chyb.

2.4.2 Aplikační servery

Aplikace v Javě EE vyžadují pro svůj chod tzv. aplikační server. Jde o zvláštní aplikaci naprogramovanou v Javě, která ovládá instance webových aplikací a přeposílá data klientskému webovému prohlížeči. Jde vlastně o obdobu Apache HTTP Serveru. Mimo toho musí být server vybaven Java softwarem.

Aplikační servery pro Javu jsou poskytovány buď zdarma, nebo na placenou licenci. Je jich několik druhů, v této podkapitole se ale budu věnovat jen některým z nich, a to těm neplaceným. V České republice je největším problémem aplikačních serverů jejich dostupnost. Java není pro tvorbu webových aplikací tolik rozšířená, jako například PHP. Všechny české hostingové společnosti se proto zaměřují právě na PHP a nabízejí servery vybavené jen Apache serverem a MySQL databází. Hostingových serverů podporující aplikace v Javě je velmi málo a často se jedná o dražší službu. Jiným řešením může být pořízení zcela vlastního serveru, který lze přizpůsobit veškerým potřebám, ať už jde o fyzický či virtuální server. Často jde ale o nákladnější záležitost a takové řešení nemá smysl, pokud je třeba zajistit jen chod běžné webové aplikace.

Nejznámějším aplikačním serverem je **GlassFish** vyvinutý společností Sun. Plně podporuje veškerou funkčnost podnikové edice. V české republice ale GlassFish hosting poskytuje jen minimální počet společností.

Druhým nejpoužívanějším aplikačním serverem je **Tomcat** od společnosti Apache. Jde o server, který běžně české hostingové společnosti podporující Javu nabízejí. Jeho nevýhodou je omezená podpora podnikové edice Javy, nepodporuje například tzv. Enterprise Java Bean (viz další kapitola). To ale při tvorbě webových aplikací nemusí nutně zname-

nat překážku. Tomcat je snadno konfigurovatelný a díky modulu Jakarta ho lze propojit s Apache HTTP Serverem. Oba servery jsou potom schopné spolupracovat.

Posledním příkladem je server **JBoss**. Je založený na Tomcatu, ale na rozdíl od něj má vyšší podporu podnikové edice, podporuje například EJB. Je určen pro větší projekty, takže tvoří dobrou alternativu pro případ, kdy není využit GlassFish, ale Tomcat je nedostačující.

3 Webové aplikace v Javě

Konkrétních způsobů, jak vytvářet webové aplikace v Javě, je několik. Tato kapitola se věnuje těm nejpoužívanějším z nich a jejich postupnému rozvoji. Pozornost je také věnována práci Javy s dalšími systémy, například s databází.

Jako informační zdroj pro tuto kapitolu posloužila publikace Java EE 6 with GlassFish3 Application Server.

3.1 Databáze a objektově relační mapování

Práce s databází pomocí ORM je v Javě standard, což je velký rozdíl oproti PHP. Základním ovladačem pro práci s databází je JDBC, který je využíván k realizaci spojení. Je navíc potřeba použít i ovladač určený pro konkrétní druh databáze.

Základem pro ORM v Javě je JPA. Konfiguruje se pomocí souboru *persistence.xml* a samotné mapování probíhá pomocí různých anotací, které je třeba vepsat přímo do tříd datového modelu. Tyto třídy mají kromě těchto anotací několik dalších specifických rysů, je například třeba vytvořit metody *equals()* a *hashCode()*, které slouží k identifikaci instance této třídy. Může totiž dojít k situaci, kdy se tyto objekty liší jen primárním klíčem, což je většinou číselná proměnná s názvem *id*. Metody zděděné ze třídy *Object* s tímto nepočítají a je proto třeba je přepsat. Dále je nutné, aby tato třída obsahovala konstruktor bez parametrů a aby každá persistentní proměnná měla svůj getter a setter. Ty můžou být privátní. Samotné JPA se ale příliš často nepoužívá, existuje totiž celá řada jeho rozšíření. Jedním z nich je framework zvaný Hibernate.

Hibernate je v dnešní době průmyslovým standardem pro aplikace v Javě. Základní konfigurace se nachází v souboru *hibernate.cfg.xml* umístěném v kořenovém balíčku aplikace. Hibernate také umožňuje mapování tříd pomocí anotací, častějším způsobem je však mapování pomocí konfiguračních XML souborů. Standardem je, že ke každé třídě datového modelu je přiřazen jeden konfigurační soubor, který obsahuje všechny informace potřebné k mapování dané třídy do databáze včetně datových typů jednotlivých proměnných (lze vytvářet i vlastní datové typy). Práce s daty pomocí Hibernate probíhá pomocí tzv. Hibernate session. Jde o instanci třídy *org.hibernate.Session*, která imple-

mentuje všechny metody potřebné pro práci s daty. Tato data navíc automaticky cachuje.

3.2 Servlet

Servlet je rozšíření aplikačního serveru, který má za úkol vyřizovat požadavky ze strany klienta a generovat odpovědi. V případě webových aplikací je častým typem HTTP servlet, kterému se budu dále věnovat. Základní třídou HTTP servletu je *javax.servlet.HttpServlet*.

3.2.1 Práce se servletem

HTTP servlet zpracovává čtyři druhy HTTP požadavků:

- GET,
- POST,
- PUT,
- DELETE.

Nejčastěji využívané typy požadavků jsou právě GET a POST. GET požadavky jsou generovány, když klientský uživatel zadá do adresy svého webového prohlížeče URL, POST požadavky jsou generovány většinou po odeslání webového formuláře. Servlet tyto požadavky vyřizuje pomocí přepisovatelných metod *doGet()* a *doPost()*.

Tvorba nového servletu probíhá jednoduchým způsobem. Třída servletu je potomkem třídy *javax.servlet.HttpServlet* a musí obsahovat metodu *doGet()*. Ta obstarává zobrazení webové stránky po zadání URL do klientského prohlížeče. Pokud bude stránka obsahovat formulář využívající POST požadavek, bude potřeba přepsat i metodu *doPost()*.

3.2.2 Sestavení servletové aplikace

Sestavení webové aplikace probíhá stejně jako u jiné Java aplikace. K sestavení může být použit Ant, Maven, atd.

Před samotným sestavením je ale potřeba webovou aplikaci nakonfigurovat, čehož lze dosáhnout dvěma způsoby. Konfiguraci lze provést buď anotacemi, anebo pomocí souboru *web.xml*. Nastavení pomocí XML dokumentu je častější. V konfiguraci je třeba uvést název třídy servletu a odkaz na ni, dále lze nastavit URL mapování.

Po dokončení konfigurace lze přistoupit k samotnému sestavení. Aplikace je komprimována do souboru typu WAR, ten obsahuje konfigurační soubory, Java třídy a soubory webu (stránky, CSS soubory, obrázky atd.). Zkompilovaný WAR soubor je poté nasazen na aplikační server.

3.2.3 Knihovna Tuckey URL Rewrite

Aplikace Apache HTTP Server umožňuje mimo jiné snadnou modifikaci URL adresy do takové podoby, aby byla snadno zapamatovatelná a optimalizovaná podle SEO standardů. Servlet samotný tuto možnost nabízí také, ale konfigurace není tak snadná jako v případě Apache. Právě kvůli tomu vznikla řada knihoven usnadňujících toto přepisování URL, jedna z nich je Tuckey URL Rewrite.

Knihovna potřebuje ke svému chodu jen konfigurační soubor *urlrewrite.xml* v balíčku *WEB-INF*, který obsahuje konkrétní data přepisování URL. Tento konfigurační soubor obsahuje řadu pravidel různých druhů definovaných různými XML elementy umístěných pod kořenovým elementem *<urlrewrite />*. Knihovna poté použije tuto konfiguraci a pomocí tzv. filtrů servletu přepisuje URL adresy na cesty ke konkrétním stránkám, aniž by to uživatel aplikace postřehl.

Knihovna je dostupná ke stažení na adrese *http://www.tuckey.org/urlrewrite*.

3.3 JavaServer Pages

Samotný servlet je dostačující pro zpracovávání formulářových dat, není ale vhodný pro tvorbu rozsáhlejších aplikací. V případě tvorby pomocí servletu není možné jednoduše oddělit HTML kód od aplikační logiky. HTML kód je generován patřičnými třídami a tvorba je o to náročnější. Ještě náročnější je pozdější údržba aplikace. Takový systém by se dal přirovnat k systému používanému při tvorbě webových aplikací v PHP bez systému Smarty.

Právě z těchto důvodů vznikly JavaServer Pages, zkráceně JSP. Hlavním přínosem JSP je oddělení aplikační logiky a výstupu pro klientský prohlížeč. Výstup je směsí statického HTML a dynamického obsahu, podobně jako v případě PHP Smarty šablony. Servlet má ale v tomto případě stále svůj vliv, JSP je totiž postaveno na jeho základě.

3.3.1 Struktura JSP projektu

Projekt v JSP a dalších odvozených technologiích (viz dále) má základní strukturu rozdělenou do dvou částí. Ta první obsahuje aplikační logiku, tedy beany (viz dále) a další Java třídy. Druhá část obsahuje stránky, které jsou použity k vytvoření výstupu pro klientský prohlížeč.

Ve výsledném WAR archivu jsou třídy business logiky a konfigurační soubory umístěny do složky *WEB-INF* a soubory stránek do kořenové složky archivu.

3.3.2 Ukázka

```
 <%@ page info="a hello world example" %>
<html>
<head><title>Hello, World</title></head>
<body bgcolor="#ffffff" background="background.gif">
<%@ include file="banner.html" %>
<table>
<tr>
<td width="150">   </td>
<td width="250" align="right"> <h1>Hello, World!</h1> </td>
</tr>
</table>
</body>
</html>
```

Toto je ukázka jednoduché stránky v JSP. Lze zde vidět statické HTML a dynamickou část. Dynamická část je označena zvláštními značkami (v tomto případě *<%@* a *%>*) či zvláštními XML tagy.

JSP se stal na dlouhou dobu standardem pro tvorbu webových aplikací v Javě. Umožňoval snadnou práci se servletem, oddělení jednotlivých vrstev aplikace a práce s JSP byla oproti práci s čistým servletem jednoduchá. Později vznikla řada frameworků využívajících JSP a značně usnadňujících práci s Javou jako nástrojem pro tvorbu webových aplikací. Nejvýraznějším z těchto frameworků je JavaServer Faces, zkráceně JSF.

4 JavaServer Faces

Účelem současné verze JSF (2.1) je co nejvíce usnadnit a urychlit vývoj webových aplikací. To se nejvíce projevuje právě v posledních verzích tohoto frameworku. První verze jednoduchostí neoplývají, za což byl framework tehdy kritizován.

Informace v této kapitole jsem čerpal převážně z publikace Java EE 6 with GlassFish 3 Application server [4] a z online semináře na téma Java Server Faces 2.0 vedeného Ing. Zdeňkem Troníčkem, Ph.D [25].

4.1 Struktura

Struktura aplikace je stejná, jako v případě JSP. Jediným rozdílem je, že soubory jednotlivých stránek mají příponu *.xhhtml*.

4.2 Facelet

Ve verzi 1.0 využívaly JSF jako hlavní technologii JSP. To působilo řadu komplikací už jen proto, že JSP je starší technologií než JSF.

Pro pozdější verzi proto Jacob Hookom vytvořil novou technologii, **facelet**. Tato technologie dosáhla velké oblíbenosti a stala se standardem pro JSF. Vedle JSP tak JSF aplikace využívají přednostně facelet.

Faceletové stránky jsou na rozdíl od JSP čisté XML. Na faceletové stránce se nenachází nic, co by odporovalo pravidlům správně formátovaného XML dokumentu. Faceletová stránka může obsahovat jak klasický HTML kód, tak XML tagy se specifickými vlastnostmi. Tyto dynamické XML tagy jsou vždy definovány jmenným prostorem. Nejtypičtějšími jsou jmenné prostory *http://java.sun.com/jsf/html* s prefixem *h* a *http://java.sun.com/jsf/core* s prefixem *f*. Tagy s prefixem *h* jsou určeny ke generování klasického HTML, prefix *f* mají tagy určené pro práci s jádrem JSF. Těmto a dalším jmenným prostorům se ještě budu věnovat později.

4.3 Konfigurace

Ve verzi 2.0 a novějších není prakticky potřebné provádět jakoukoliv konfiguraci, veškeré základní úkony lze v JSF řešit pomocí anotací. Nicméně v některých specifických případech, které konkrétně uvedu později, je potřeba konfiguraci provést. Konfiguraci obstarávají hlavně dva XML dokumenty, a to *web.xml* a *faces-config.xml*.

4.3.1 Soubor web.xml

Soubor *web.xml* obsahuje základní konfiguraci webové aplikace, včetně nastavení servletu.

Významným nastavením jsou kontextové parametry. Jedním z těchto parametrů je i vývojová fáze projektu, která ovlivňuje např. zobrazování chybových hlášení v aplikaci. Tato vlastnost se nastavuje pomocí kontextového parametru *javax.faces.PROJECT_STAGE* a přijímá hodnoty *production*, *development*, *unitTest* a *systemTest*.

Další důležitou vlastností, kterou lze ve *web.xml* nastavit, je určení domovské stránky, tedy indexu. Do pozice indexu lze nastavit kteroukoliv JSF stránku. Soubor indexu je určen elementem *<welcome-file />*, který obsahuje seznam jednotlivých indexů v elementu *<welcome-file />*, většinou však stačí určit jen jeden. Pokud toto nastavení chybí, JSF se pokusí najít index pod klasickými názvy, tedy *index.xhtml* nebo *index.jsf* v kořenové složce stránek.

Praktické využití často najde i nastavení chybových stránek. Lze nastavit zobrazení některé JSF stránky v případě výskytu Java výjimky nebo HTTP chyby. Každá chybová stránka se nastavuje pomocí elementu *<error-page />*, který obsahuje informaci o typu chyby a o JSF stránce k zobrazení.

4.3.2 Soubor faces-config.xml

Konfigurační soubor *faces-config.xml* je po vydání verze JSF 2.0 mnohem méně důležitý než *web.xml*. Díky anotacím není potřeba konfigurovat jednotlivé beany (viz dále) a ani provádět další dříve nutná nastavení. *faces-config.xml* je ale potřeba využít mimo jiné při tvorbě vícejazyčné aplikace.

4.4 Managed bean

Managed bean je základní druh Java třídy v JSF aplikacích. Jde o rozšíření třídy typu JavaBean, která má podobnou funkci a je využívána primárně v JSP. Managed bean je přizpůsobena JSF.

Managed bean vytvářejí základ pro zdynamičtění JSF stránek. Obstarávají pro ně data a vyřizují požadavky, většinou typu GET či POST. Lze je využít také pro navigaci ve webové aplikaci.

V dřívějších verzích JSF bylo třeba pro vytvoření managed bean provést konfiguraci pomocí *faces-config.xml*. V současnosti není tato konfigurace potřeba, stačí danou třídu označit anotací *@ManagedBean*. Beaně lze dát specifický název pomocí parametru anotace *name*, pokud není název určen pomocí tohoto parametru, je shodný s názvem dané třídy. Využití tohoto pojmenování vysvětlím později.

Třídy typu managed bean mají několik specifických vlastností. Musí obsahovat konstruktor bez parametrů, ve většině případů by měly implementovat rozhraní *Serializable* a k proměnným využívaným mimo danou beanu musí existovat gettery, případně settery. Settery nejsou potřeba např. při využití proměnné jako seznamu, který je potřeba vypsat na stránce. V takovém případě není potřeba hodnotu proměnné měnit, je potřeba z ní jen čerpat data pomocí getteru. Jiný případ to ale bude u webových formulářů, kde je potřeba i setter k nastavení hodnoty při odeslání formuláře.

4.4.1 Působnost

Managed bean má ještě jednu důležitou specifikaci, a to svou působnost. Působnost beany určuje, jak dlouho bude instance dané beany existovat a v jakém prostoru. Tato vlastnost se nastavuje pomocí anotací, každý druh působnosti má svou specifickou anotaci. Instance bean všech druhů působnosti jsou vytvářeny ve většině případů až v momentě, kdy k nim je přistupováno.

Výchozím druhem je působnost na požadavek, použije se v případě, že působnost není definována. Instance beany s touto působností existuje jen po dobu zpracování požadavku, v případě formulářů například po dobu zobrazení stránky a odeslání formuláře. Taková beana může být vhodná například právě pro zpracovávání webových formulářů. Působnost na požadavek se nastavuje pomocí anotace ***@RequestScoped***.

Podobně funguje i managed bean s působností na zobrazení, instance beany existuje jen po dobu zobrazení stránky, po jejím opuštění zaniká. Vzhledem k tomu, že po dobu zpracování požadavku již neexistuje, nemá beana s touto působností příliš velké uplatnění a používá se méně. Beana s tímto druhem působnosti je označena anotací ***@ViewScoped***.

Dalším druhem působnosti je působnost na session. Instance beany s touto působností je vytvářena pro každý klientský prohlížeč, který k webové aplikaci funguje a má životnost po celou dobu trvání sessiony. Beana tohoto druhu je tak ideální například k vytvoření systému uživatelských účtů a jejich přihlašování do aplikace. Pro nastavení působnosti na session je potřeba k třídě přidat anotaci ***@SessionScoped***.

```
@ManagedBean(name="user")
@SessionScoped
public class UserBean implements Serializable {

    public UserBean() {
    }
}
```

Dalším typickým druhem působnosti beany je působnost na aplikaci. Instance takové beany existuje jen jedna, a to po celou dobu, po kterou je webová aplikace spuštěna. Je tak vhodná například pro uložení menšího objemu dat, ke kterému je z aplikace často přistupováno a lze tak dosáhnout menších nároků na databázi. Oproti ostatním druhům působnosti lze v tomto případě navíc k anotaci *@ManagedBean* přidat parametr *eager* s hodnotou *true*. Ten způsobí, že instance beany s aplikační působností je vytvořena ihned při spuštění aplikace a ne až při prvním přístupu. Anotace pro tento druh působnosti je ***@ApplicationScoped***.

Lze vytvořit i vlastní druh působnosti se specifickými vlastnostmi. Tímto druhem se ale nebudu příliš zabývat, v drtivé většině případů jsou výše uvedené dostačující. Vlastní působnost lze nastavit pomocí anotace ***@CustomScoped*** se specifickou hodnotou parametru *value*.

Poslední možností, kterou lze využít, je nepřiřazení žádné působnosti. Instance bean bez působnosti jsou vytvářeny jen v případech, kdy je k nim přistupováno z jiných bean. K dosažení tohoto stavu je třeba využít anotaci ***@NoneScoped***.

4.4.2 Anotace @ManagedProperty

Často je potřeba, aby managed bean nekomunikovaly jen s JSF stránkami, ale i mezi sebou. Takové spojení je možné, a to právě pomocí anotace *@ManagedProperty* s povinným parametrem *value*, který určuje, s jakou beanou je třeba se spojit.

Tato anotace obstará přiřazení instance jedné beany do druhé. Odkazovaná beana ale nesmí mít kratší působnost než ta, ve které bude odkaz vytvořen. Není tedy možné v beaně s působností na aplikaci vytvořit odkaz na beanu s působností na session či požadavek.

Využití této anotace vysvětlím na příkladu. Jsou dvě třídy bean, jedna s působností na session (s názvem „user"), druhá s působností na aplikaci (s názvem „appData"). Cílem je vložit instanci beany *appData*, definované třídou *AppDataBean*, do beany *user*, definované třídou *UserBean*. V beaně s názvem *user* je tedy třeba vytvořit privátní proměnnou datového typu *AppDataBean*. Právě tato proměnná bude obsahovat instanci beany *appData*. Poté je potřeba přidat anotaci *@ManagedProperty* s parametrem *value*, jehož hodnota pomocí značky *#{}* (tzv. EL výraz) označí beanu, jejíž instance se má do proměnné vložit. Posledním, ale nutným, krokem je vytvoření setteru k této proměnné.

```
@ManagedBean(name="appData")
@ApplicationScoped(eager=true)
public class AppDataBean implements Serializable {
    public AppDataBean() {
    }
}
```

```
@ManagedBean(name="user")
@SessionScoped
public class UserBean implements Serializable {

    // V této proměnné bude uložena instance beany s názvem appData
    @ManagedProperty(value="#{appData}")
    private AppDataBean appData;

    public UserBean() {
    }

    // Setter je nutný k vložení instance beany do proměnné appData
    public void setAppData(AppDataBean appData) {
        this.appData = appData;
    }
}
```

4.5 JSF (faceletová) stránka

Soubor faceletové stránky má příponu *.xhtml* a jak bylo dříve zmíněno, jde o čistý XML dokument. Tyto soubory znázorňují to, co bude odesláno klientskému prohlížeči jako výstup a co uvidí uživatel, který prohlíží webové stránky. Lze do nich zapisovat jak obyčejné HTML, které ve většině případů zůstanou nezměněny, tak XML tagy určující dynamický obsah. Tyto dynamické tagy se nacházejí vždy v rámci zvláštních jmenných prostorů, budu se jim věnovat v jiné části této kapitoly. Výsledkem těchto dynamických tagů bývají JSF komponenty, které plní na stránce nějakou funkci. Jde například o formulářové komponenty, které mají funkci vstupních formulářových polí.

4.5.1 Data z bean

Nutností pokročilejší webové aplikace je možnost zobrazení dat zprostředkovaných beanami v JSF stránkách. Způsob, jakým toho lze dosáhnout, je velmi jednoduchý, stačí opět použít EL výraz. Tímto způsobem lze zobrazovat data ve stránkách, ať už jde o prostý text, seznamy nebo o jakýkoliv jiný typ dat.

Mezi složené závorky je třeba vypsat cestu k datům, která je potřeba zpracovat. Na prvním místě vždy bývá beana obsahující data, poté proměnná či metoda, která data vrací. Pokud jde o proměnnou, je nutné, aby daná proměnná měla v beaně svůj getter.

```
#{user.loggedUser.name.toLowerCase()}
```

V případě tohoto příkladu bude zobrazena proměnná *name* objektu, který je uchován v proměnné *loggedUser*. Ta je obsažena v beaně pojmenované *user*. Jméno je navíc díky použití metody *toLowerCase()* vypsáno malými písmeny.

4.5.2 Šablony

Většina webových aplikací má designové rozložení, které se snaží usnadnit navigaci po aplikaci a zároveň dává uživateli najevo, že se nachází pořád na jednom webovém prostoru. Takové aplikace jsou většinou rozloženy do několika částí, z nichž některé jsou společné pro všechny části webu a některé jsou zvláštně určeny pro konkrétní prohlíženou stránku. Aby nebylo nutné psát stejný faceletový kód pro každou stránku zvlášť, JSF je vybaveno speciálním druhem stránky nazývaným šablona. V PHP toto obstarává šablonový systém Smarty, ten je oproti JSF šablonám ovšem poněkud neobratný.

Šablona by měla obsahovat právě ten kód, který bude zobrazen na všech stránkách napojených na šablonu. Jde v podstatě o prostou faceletovou stránku s tím rozdílem, že obsahuje elementy *<ui:insert />*, které označují místa, kam se má vložit obsah konkrétní stránky. Tento element může mít klasický obsah a pokud nebude přepsán jinou stránkou, bude zobrazen právě ten. Důležitý je atribut *name*, který danou část šablony identifikuje.

Na každou šablonu lze napojit libovolný počet stránek. Stránka napojená na šablonu obsahuje element *<ui:composition />*, který o tomto napojení informuje jádro JSF. Tag tohoto elementu musí mít vyplněný atribut *template*, ten obsahuje cestu k šabloně. Pod sebou má dále elementy *<ui:define />* s atributem *name*. Ty zahrnují obsahy jednotlivých elementů *<ui:insert />* identifikovaných podle atributu *name*. Výsledným výstupem je tedy obsah šablony s některými částmi podle konkrétní stránky.

Ve vývojovém prostředí NetBeans je tvorba šablon velmi usnadněná. Soubor šablony lze vytvořit pod kontextovou nabídkou projektu *New -> Facelets Template*. NetBeans poté nabídne svému uživateli na výběr z několika nejtypičtějších uspořádání webové stránky a při vytvoření šablony zároveň obstará vygenerování patřičných CSS. Podobně lze vytvářet i stránky napojené na šablonu. Tuhle možnost nabízí také kontextová nabídka projektu pod *New -> Facelets Template Client*, kde je jen potřeba zadat cestu k již vytvořené šabloně.

4.5.3 Podpůrné zdrojové soubory

JSF nabízí usnadnění práce s podpůrnými zdrojovými soubory jako obrázky, CSS soubory apod. Standardně jsou tyto soubory umístěny do složky *resources* v kořenové složce WAR archivu. Zpravidla je vhodné jednotlivé druhy souborů rozdělit do podsložek, tedy například obrázkům vyhradit složku *images*, CSS souborům složku *styles* atd. Takto rozdělené soubory jsou nejen přehlednější, ale přístup k nim je i snadnější. Jak se k takto roztříděným zdrojovým souborům přistupuje, vysvětlím v následující podkapitole.

4.5.4 Jmenný prostor: h

Jedním z nejpoužívanějších jmenných prostorů pro dynamické JSF elementy je *h* nacházející se na *http://java.sun.com/jsf/html*. Jeho primárním úkolem je vygenerovat klasické HTML tagy jednodušším způsobem.

Každý tag ve jmenném prostoru *h* má několik dalších nastavitelných atributů, kterými lze výrazně ovlivnit vlastnosti generované komponenty. Klasickým příkladem atributu, který mají všechny tyto tagy, je *rendered*, který na své hodnotě přijímá boolean hodnotu. Ta určuje, zda se má komponenta na výstupu vůbec vykreslit či ne. Díky tomu lze dynamicky (pomocí dat z bean) určit, zda se má element zobrazit.

Asi nejdůležitějšími elementy v tomto jmenném prostoru jsou *<h:head />* a *<h:body />*. Ty obstarávají vygenerování základních HTML elementů *<head />* a *<body />*, kterým případně přiřadí další vlastnosti podle nastavení aplikace. Například místo *<head />* je nutné použít *<h:head />*, protože pokud bude jedna z faceletových stránek využívat AJAX, JSF potřebuje vložit do *<head />* elementu příslušné JavaScriptové knihovny. K tomu využívá právě element *<h:head />*.

Užitečnými tagy jsou také *<h:outputStylesheet />*, *<h:outputScript />* a *<h:graphicImage />*. Ty totiž umožňují rychlý přístup k podpůrným zdrojovým souborům. Jejich úkolem je vygenerovat odpovídající HTML tag, který reprezentuje odkaz na daný soubor určený pomocí atributů *library* a *name*. *library* říká, v jaké podsložce se soubor nachází, *name* jmenuje konkrétní soubor.

Častými elementy jsou *<h:outputText />* a *<h:outputFormat />*. Ty obstarávají vypsání prostého textu, často převzatého z beany. Tyto tagy text zpracují tak, aby byl vhodný pro zobrazení na stránce. Ohraničí ho patřičnými HTML tagy a HTML tagy v textu převedou na HTML entity (tzv. escapování) tak, aby nedošlo ke komplikaci při zobrazení textu.

Důležitým tagem je *<h:form />* a další s ním spojené, např. *<h:inputText />*. Elementy tohoto typu na stránce obstarávají vygenerování webových formulářů a jejich komponent, kterým se budu detailně věnovat později.

Pomocí tagů ve jmenném prostoru *h* je možno generovat i další HTML prvky, například tabulky. K tomu slouží tag *<h:dataTable />*, který na základě dat předaných z beany vygeneruje datovou tabulku. Podobný účel má například i tag *<h:panelGrid />*. S tímto tagem je spojeny i tag *<h:column />*.

4.5.5 Jmenný prostor: f

Ve jmenném prostoru *f*, nacházejícím se na *http://java.sun.com/jsf/core*, jsou tagy nástrojů JSF jádra. Jejich použití je různé, lze díky nim pracovat s AJAXem (*<f:ajax />*), ur-

čovat vlastnosti elementům, ve kterých se nacházejí (*<f:facet />*), obstarávat validaci webových formulářů atd. Tagům v tomto jmenném prostoru se budu věnovat ještě později.

4.5.6 Jmenný prostor: ui

Jmenný prostor *ui* s umístěným na *http://java.sun.com/jsf/facelets* má za úkol vytvářet základy uživatelského rozhraní aplikace. Tagy tohoto jmenného prostoru slouží například k vytváření dříve zmíněných šablon, mají ale i více funkcí. Například element *<ui:repeat />* slouží k iterování dat.

4.5.7 Jmenný prostor: c

Nacházejí se na *http://java.sun.com/jsp/jstl/core* a mají označení *c*. Jde o elementy, jenž jsou součástí JSTL a jsou využívány primárně v JSP, ale je možno je použít i v JSF. Nicméně fakt, že nejsou primárně určeny pro JSF, je častým původcem potíží s těmito elementy. Jejich používání proto není doporučováno.

Příkladem použití tohoto jmenného prostoru může být sestava elementů *<c:choose />*, *<c:if />* a *<c:otherwise />*. V podstatě jde o klasickou podmínku *if-else*.

4.6 Lokalizace

JSF přináší snadnou možnost lokalizování veškerých nedynamických textů. Je potřeba provést jen dva kroky, za prvé vytvořit patřičný soubor s texty v několika jazykových verzích a za druhé provést jednoduchou konfiguraci. JSF se poté samy postarají o zobrazení správné jazykové verze podle preferovaného jazyka klientského webového prohlížeče.

Veškeré texty jsou umístěny v několika *.properties* souborech, každý soubor představuje souhrn textů pro daný jazyk. Zároveň lze mít od jednoho jazyka více těchto souborů, například pro každou kategorii textů jeden. V případě české a anglické jazykové mutace tedy lze mít například tyto soubory:

- *messages_cs.properties*,
- *messages_en.properties*,
- *texts_cs.properties*,
- *texts_en.properties*.

Syntaxe zápisu jednotlivých textů odpovídá klasickému zápisu *.properties* souborů. Například: *wrongPassword=Špatné heslo!*

Druhým nutným krokem je konfigurace pomocí konfiguračního souboru *faces-config.xml*. Prvním elementem, který je třeba přiřadit pod element *<application />*, je *<locale-config />*. V něm je třeba nadefinovat výchozí a podporované jazyky, a to pomocí jednoho elementu *<default-locale />* a případně pomocí elementů *<supported-locale />*. Do obou těchto elementů je třeba vložit kód dané jazykové mutace (např. *cs*). Dále následuje seznam elementů *<resource-bundle />* (rovněž pod elementem *<application />*), které konfigurují jednotlivé *.properties* soubory s texty. Každý *<resource-bundle />* obsahuje další dva elementy. *<base-name />* obsahuje cestu k *.properties* souboru, *<var />* obsahuje název proměnné, do které budou texty uloženy.

Ve faceletu se k textům přistupuje podobně jako k proměnným v beanách s tím rozdílem, že místo názvu beany je v EL výrazu třeba použít název proměnné textů (nastavený v elementu *<var />* konfiguračního souboru). Příklad: *#{texts.hello}*

K textům ovšem lze přistupovat i uvnitř bean, případně jiných tříd. Jde o hodnoty uložené ve třídě *ResouceBundle*, takže je lze bez problémů získat. Je jen potřeba znát aktuální nastavení jazyka, které je uložené v kontextu faceletu.

Výběr správné jazykové mutace je sice obstaráván automaticky, někdy je ale třeba mít možnost změnit toto nastavení ručně. Stačí do faceletové stránky, případně do šablony, do elementu *<h:head />* vložit element *<f:view />* s atributem *locale* s hodnotou kódu jazyka požadované jazykové mutace. Tento kód může být brán z některé z bean, lze tedy dynamicky nastavit.

4.7 Webové formuláře

Tvorba webových formulářů pomocí JSF je relativně velmi jednoduchá. Díky JSF tagu *<h:form />* stačí provést minimum nastavení formuláře a facelet se při jeho generování postará o zbytek. Stejně tak není třeba používat obyčejné tagy pro vytvoření vstupních polí formuláře, v případě použití *<h:form />* to ani není žádoucí. Existuje celá řada tagů použitelných při vytváření formuláře. Patří mezi ně:

- *<h:button />*, *<h:commandButton />*,
- *<h:inputHidden />*, *<h:inputSecret />*, *<h:inputText />*, *<h:inputTextarea />*,

- *<h:selectBooleanCheckbox />*,
- *<h:selectManyCheckbox />*, *<h:selectManyListbox />*, *<h:selectManyMenu />*,
- *<h:selectOneListbox />*, *<h:selectOneMenu />*, *<h:selectOneRadio />*.

Jako obvykle má každý z elementů definovaných těmito tagy několik dalších rozšiřujících atributů. Tím nejdůležitějším je *value*. Ten určuje, kam se hodnota vstupního pole po odeslání formuláře uloží a jakou hodnotu bude mít pole předvyplněnou. Pokud bude tedy například vstupní textové pole formuláře vypadat takto: *<h:inputText value="#{user.username}" />*, při načtení formuláře bude mít pole předvyplněnou hodnotu, která je obsažena v proměnné *username* v beaně *user*. Po odeslání formuláře je tato hodnota uložena do té samé proměnné.

Všechny tyto formulářové proměnné musí mít své gettery a settery. Ty lze modifikovat pro individuální potřeby aplikace. Může například nastat situace, kdy je automatické doplňování výchozích hodnot do formulářových polí nežádoucí. Jedním z těch méně obratných řešení je vytvoření getteru tak, aby vždy vracel hodnotu *null*. Lepším řešením takového problému je nastavení *null* hodnot těmto proměnným po zpracování formuláře.

Dalším důležitým atributem je *actionListener*, který je součástí elementů *<h:button />* a *<h:commandButton />*. Ten určuje metodu, která bude spuštěna po odeslání formuláře a uložení všech hodnot do daných proměnných. Pokud bude tedy element odesílacího tlačítka vypadat takto: *<h:commandButton value="Odeslat" actionListener="#{user.login()}" />*, bude po odeslání formuláře zavolána metoda *UserBean.login()*.

Takové metody, které obsluhují formulářová data, mívají jako návratovou hodnotu objekt třídy *String*. Tento textový řetězec určuje, kam je uživatel po odeslání navigován. Je několik způsobů, jak lze tuto navigaci provést. Tím častějším je vložení prostého názvu stránky, kam má navigace proběhnout. Pokud má tedy navigace proběhnout na stránku *loginSuccessful.xhtml*, návratová hodnota bude *loginSuccessful*. Druhým, složitějším způsobem je vytvoření vlastního navigačního pravidla. Toho lze dosáhnout pomocí vložení elementu *<navigation-rule />* do konfiguračního souboru *faces-config.xml*. Pokud daná metoda vrátí *null*, navigace neproběhne a uživatel zůstane na původní stránce, ze které formulář odeslal.

4.7.1 Validace dat

JSF nabízí velmi snadný způsob validace formulářových dat, a to pomocí sady elementů nacházejících se ve jmenném prostoru *f*. Ty je potřeba přiřadit pod některý ze vstupních elementů, který je třeba validovat. Každému validovanému formulářovému prvku je většinou také dobré nastavit zobrazenou chybovou hlášku pro případ selhání validace. To se dá obstarat pomocí atributů *validatorMessage* a *requiredMessage*.

4.7.2 Vestavěné validátory

JSF nabízí celou řadu těch nejčastějších způsobů validace dat. Tím asi nejpoužívanějším je *<f:validateRequired />*, který určuje danou položku formuláře jako povinnou. Tento způsob validace jde u položky nastavit také pomocí atributu *required* s hodnotou *true*. V případě běhu aplikace na serveru Tomcat je to dokonce jediná možnost, *<f:validateRequired />* je totiž na Tomcatu nefunkční.

Dalším typickým způsobem validace dat je kontrola podle délky textu. K tomu slouží element *<f:validateLength />*, kterému je potřeba nastavit atributy *minimum*, *maximum* či obojí. Ty určují požadovanou délku zadávaného textového řetězce.

Podobnou funkci mají elementy *<f:validateLongRange />* a *<f:validateDoubleRange />*, jejichž úkolem je kontrolovat hodnotu zadávaného čísla, tedy zda se číslo nachází v daném rozsahu. Jako atributy také přijímají *minimum* a *maximum*.

Často je potřeba validovat data oproti regulárnímu výrazu, to v JSF umožňuje element *<f:validateRegex />*. Regulární výraz používaný ke kontrole vstupní hodnoty je třeba zadat jako hodnotu atributu *pattern*.

Dalším způsobem je validace na straně beany. V takovém případě je třeba použít element *<f:validateBean />* a v beaně zpracovávající formulář použít validační anotace přiřazené k daným proměnným. Jednou z takových anotací je například *@Size* s parametry *min* a *max*, které určují minimální či maximální délku textového řetězce. Tato validace na straně beany ovšem neposkytuje tolik možností, jako validace na straně faceletu. Neumožňuje například použití vlastních validátorů, proto se jí již ve své práci nebudu věnovat.

4.7.3 Vlastní validátory

JSF nabízí možnost vytvoření zcela vlastního validátoru. Někdy může být důvodem pohodlnost programátora, jindy zas JSF v dané situaci nenabízí žádný vhodný vestavěný validátor. Takovým vlastním validátorem může být například kontrola e-mailové adresy. Takové kontroly lze sice dosáhnout pomocí elementu *<f:validateRegex />*, programátor ale musí při každé takové kontrole znovu opisovat patřičný regulární výraz. Pro zjednodušení práce lze tedy vytvořit vlastní validátor.

Třída validátoru implementuje rozhraní *javax.faces.validator.Validator* a je určena anotací *@FacesValidator*, která má jako parametr identifikátor, textový řetězec, pomocí kterého je třeba později na validátor odkazovat. Samotná třída musí implementovat jen jednu metodu, a to *validate(FacesContext context, UIComponent component, Object value)*. Parametr *context* obsahuje kontext faceletu, *component* instanci validovaného vstupního prvku a *value* samotnou kontrolovanou hodnotu. Validace probíhá zavoláním této metody. Pokud metoda proběhne bez vyhození výjimky, validace proběhla v pořádku. Pokud jsou však validovaná data nekorektní, je třeba vyhodit výjimku *javax.faces.validator.ValidatorException* s parametrem obsahujícím JSF zprávu s konkrétní chybovou hláškou.

Po vytvoření třídy validátoru stačí už jen validátor použít ve formuláři v konkrétním vstupním poli pomocí elementu *<f:validator />*, kterému je třeba přiřadit atribut *validatorId*. Ten obsahuje identifikátor validátoru, který má být použit.

4.7.4 Konverze dat

Jeden z velmi užitečných prvků nabízených JSF je konvertor. Ten může mít využití ve formulářích, ale i jinde, např. při prostém zobrazení netextových dat mimo formulář. Konvertor umožňuje automatický převod textového řetězce do objektu jiné třídy než *String*, aniž by bylo třeba v beaně zpracovávající formulář programovat cokoliv navíc. Stejně tak umožňuje i převod z objektu do textového řetězce.

Podobně jako v případě validátorů nabízí JSF dva vestavěné konvertory. Jsou jimi *<f:convertDateTime />* a *<f:convertNumber />*. Ten první obstarává konverzi data a času, druhý zas převádí čísla a dá se využít například v případě zobrazení peněžní částky s měnou.

JSF nabízí možnost vytvoření vlastního konvertoru podobně jako tomu je v případě validátorů. Třída konvertoru implementuje rozhraní *javax.faces.convert.Converter* a je určena anotací *@FacesConverter*. Podobně jako validátor i tato anotace může mít jako parametr svůj identifikátor, na rozdíl od něj však není povinný. Místo identifikátoru stačí pouze nastavit třídu, pro kterou se má konvertor automaticky používat. K tomu slouží parametr *forClass*. Samotná třída musí implementovat dvě metody, a to *getAsObject(FacesContext context, UIComponent component, String value)* a *getAsString(FacesContext context, UIComponent component, Object value)*. Význam parametrů je stejný, jako v případě validátorů. Úkolem metody *getAsObject()* je převedení textového řetězce do vybraného objektu, druhá metoda převádí objekt zpět na textový řetězec. Konvertor může také vyhodit výjimku *javax.faces.convert.ConverterException*, která má stejné vlastnosti jako ValidatorException a ze strany uživatele vypadá stejně jako chyba validace.

4.8 Phase listener

JSF obsahuje zvláštní druh listeneru (neboli „posluchače“), který naslouchá určitým fázím, ve kterých se webová aplikace zrovna nachází a v případě změny této fáze provede nějakou akci podle implenetace listeneru. Phase listenery jsou vhodné například pro kontrolu, v jaké části aplikace se uživatel právě nachází a zda k ní má přístup.

Druhů fází je několik, jedna z těch nejvýznamnějších je ta s identifikátorem *RESTORE_VIEW*. Listener nastavený na naslouchání této fáze bude provádět naprogramovanou akci pokaždé, kdy uživatel webové aplikace požádá o zobrazení nové stránky.

Třída phase listeneru implementuje rozhraní *javax.faces.event.PhaseListener*. Cestu k ní je třeba přidat do souboru *faces-config.xml* pomocí elementů *<phase-listener />* umístěných pod elementem *<lifecycle />*. Implementují se 3 metody – *getPhaseId()* (určuje druh fáze k naslouchání), *beforePhase(PhaseEvent pe)* a *afterPhase(PhaseEvent pe)*. Poslední dvě metody obstarávají akci, která bude provedena před nebo po dané události.

5 Sada komponent PrimeFaces

Facelet nabízí spoustu komponent, které umožňují snadnou práci jak s HTML prvky, tak s daty. Jde ovšem o komponenty, které generují HTML ve své základní podobě. V dnešní době je ale třeba vytvářet moderní webové aplikace, které jsou snadno ovladatelné, designově dobře zvládnuté a umožňují uživateli rychlou interakci s aplikací. Nejvyužívanějším prvkem k dosažení takového uživatelského rozhraní bývá jQuery.

Pokud chce programátor vytvořit moderní aplikaci, má v JSF zpravidla dvě možnosti. Může použít klasické faceletové komponenty a vytvořit k nim potřebné CSS a jQuery skripty, díky kterým lze dosáhnout požadovaného výsledku. Taková práce bývá ale časově poměrně náročná a je zapotřebí velkého množství psaného kódu. Často pak dochází k chybám a naprogramování aplikace trvá dlouho. Tou druhou a mnohem efektivnější variantou je využití některé z předpřipravených sad komponent, které pro JSF existují. Mezi takové frameworky patří v současnosti například RichFaces, IceFaces a v neposlední řadě právě **PrimeFaces**.

Vývoj frameworku PrimeFaces začal v roce 2009 kdy byla vydána první verze, 0.9.0. Vývoj od té doby pokračoval a v současnosti je poslední stabilní verzí 3.2, nicméně v tuto chvíli se již pracuje na vývoji verze 3.3. Za vývojem stojí turecká společnost Prime Teknoloji specializující se mimo jiné na Javu EE a JSF.

PrimeFaces je aktivně vyvíjenou sadou komponent, která pokrývá téměř všechny požadavky moderní webové aplikace a přitom je práce s ní velmi jednoduchá. Většina komponent v PrimeFaces je založená na jQuery a nemalé množství z nich pracuje s AJAXem. Designové provedení všech komponent je velmi vkusné a jejich barevné provedení lze jednoduše změnit. Jediná nevýhoda, která se může při práci s tímto frameworkem projevit, je právě jeho neustálý vývoj. Některé verze obsahují drobné chyby, které jsou ale často v moderních aplikacích nepřijatelné. V pozdějších verzích bývají sice tyto chyby opravovány, někdy ale se změnou verze přichází také změna používání nějaké komponenty. Odbourání takových problémů při přechodu na novou verzi PrimeFaces někdy stojí nemalé úsilí. Nicméně poslední verze PrimeFaces 3.2 je stabilní a nejsou zaznamenány žádné výrazné problémy.

Kvalita této sady komponent je znát i na jejích referencích, používají ji ve svých aplikacích i společnosti jako jsou UniCredit, Raiffeisen Bank, Cisco, nVidia a řada dalších. PrimeFaces má také rozsáhlou komunitu ať jde o vývoj či o testování, sociální aplikace bují životem každým dnem, ať už jde o forum na webu či o skupinu na Facebooku.

Většina informací v této kapitole pochází z webových stránek PrimeFaces a z uživatelského manuálu k PrimeFaces 3.2, který je na webu ke stažení.

5.1 Další odvětví PrimeFaces

Základem PrimeFaces je zaměření na komponenty pro klasické webové aplikace (spouštěné na webových prohlížečích stolních počítačů, notebooků atd.). To je ale jen hlavní větev, je zde i několik dalších odvětví, která se specializují na další potřeby jiných druhů webových aplikací.

5.1.1 PrimeFaces Mobile

PrimeFaces Mobile jsou, jak název napovídá, odvětvím zaměřeným na mobilní platformy, jako jsou mobilní telefony či tablety. Komponenty jsou v tomto případě založeny na jQuery Mobile, jsou tedy přizpůsobeny mobilním zařízením a nehodí se pro ta desktopová. I přes to ale je rozsah nabízených komponent široký. V současnosti je toto odvětví v raném vývoji, aktuálně je framework vydáván pod verzí 0.9.2.

5.1.2 PrimeFaces Push

Zajímavým odvětvím je PrimeFaces Push. Ten usnadňuje vytváření aplikací, jejichž účelem je přenos dat v reálném čase, například chatu. K chodu aplikace využívající tuto technologii je třeba nezávislého serveru (tzv. PushServer), který ale funguje pouze na aplikačním serveru jetty.

Omezení na jeden druh aplikačního serveru spolu s faktem, že ne vždy má tvůrce aplikace možnost vytvářet další nezávislý server, způsobuje, že PrimeFace Push není široce využitelný.

5.2 Instalace

Pro používání PrimeFaces je potřeba připojit k aplikaci knihovnu frameworku stažitelnou z webových stránek na http://www.primefaces.org a není potřeba provádět žádnou další konfiguraci. Dalším krokem nutným k používání PrimeFaces komponent je ale při-

dání jmenného prostoru do faceletových stránek. Jde o jmenný prostor *p* nacházející se na *http://primefaces.org/ui*.

5.2.1 Stylování komponent

Všechny komponenty jsou předem nastylovány ve výchozím námětu, je možné si ale vybrat z několika jiných barevných námětů stažitelných z webových stránek PrimeFaces. Stažený námět je ve formě JAR souboru, který je třeba přidat do aplikace jako knihovnu. Po přidání kontextového parametru s názvem *primefaces.THEME* a názvem tématu jako hodnotou do souboru *web.xml* jsou všechny prvky nastylovány v duchu nového námětu.

Většinou je potřeba styl prvků výrazně změnit, protože pouhá změna barev nestačí. Ani to v PrimeFaces není velkým problémem, každá komponenta je definována několika CSS třídami, které lze přepsat vlastním stylem.

5.3 Komponenty

Většina komponent plně nahrazuje běžné JSF komponenty, které rozšiřuje. Často je tak využití typických JSF komponent minimální, podobně na tom jsou i prosté HTML tagy. Původní JSF komponenty sice lze využít, většinou to ale nemá smysl.

5.3.1 Charakteristika a základní atributy XML elementů

XML elementy PrimeFaces komponent mají některé společné vlastnosti například kvůli AJAXu a usnadnění práce s ním. Tyto vlastnosti lze pozorovat hlavně na možnostech konfigurace komponent pomocí atributů.

Jedním z takových atributů je *id*. Využívá se například při částečném vykreslování (viz dále), kdy je potřeba určit ty elementy, které je třeba aktualizovat AJAXem. Atribut *id* určuje identifikátor, ke kterému lze později přistupovat nejen v případě AJAXových volání. Připomíná atribut *id* u klasických HTML elementů, na rozdíl od něj však není třeba u JSF atributu *id* obstarat jeho unikátnost na celé vykreslené stránce. O unikátnost těch prostých elementů se postará JSF a ve faceletu se k elementům s JSF ID dá přistupovat pomocí cesty. Pokud bude například potřeba využít element s ID *el2*, která se nachází pod elementem s ID *el1*, bude potřeba využít této syntaxe: *:el1:el2*.

Stejně jako základní JSF komponenty, i elementy PrimeFaces komponent umožňují využití atributu *rendered*. Jeho význam se nemění, v PrimeFaces se dá ovšem velmi dobře využít při částečném vykreslování.

5.3.2 Částečné vykreslování

Částečné vykreslování je způsob, kterým PrimeFaces načítají data pomocí AJAXu s tím rozdílem, že data jsou načítána jen do vybraných částí webu. Jde tedy o klasické načítání dat AJAXem, ale práce s ním je maximálně usnadněna.

Tento způsob výpisu dat obstarávají v PrimeFaces komponenty, které jsou definovány elementy s atributem *update* či *autoupdate*. Jako hodnota se nastavuje prvek či prvky, které se mají po dané akci (specifické pro každou komponentu) překreslit. Jsou dvě možnosti, jak takové prvky určit, tím prvním je výpis cesty k elementu pomocí jeho ID. Jako druhá možnost se nabízí určení komponent pomocí klíčových slov uvedených znakem @. Tato klíčová slova jsou:

- *@this* – komponenta, která vyvolala částečné vykreslení (aktuální komponenta),
- *@parent* – komponenta, pod níž je zařazena aktuální komponenta,
- *@form* – formulář, v němž se aktuální komponenta nachází,
- *@none* – žádná komponenta,
- *@all* – celý dokument (nevyužívá se AJAXový požadavek).

Do budoucí verze PrimeFaces je navíc v plánu přiřazení dalšího klíčového slova, které bude schopné určovat komponenty k částečnému vykreslení pomocí jQuery selektorů.

5.3.3 Formuláře

Framework obsahuje rozsáhlé množství komponent určené k tvorbě formulářů. Díky nim lze vytvářet interaktivní formuláře založené na AJAXu s prvky typicky stylovanými v duchu ostatních PF komponent.

Kromě typických komponent s tagy elementů nacházejících se ve jmenném prostoru *p* je při tvorbě formuláře třeba využít klasický *<h:form />* element, který bude ostatní elementy formuláře obsahovat. Další formulářové komponenty jsou např:

- *<p:inputText />*, *<p:inputTextarea />*, *<p:editor />*, *<p:password />*,
- *<p:selectOneRadio />*, *<p:selectManyCheckbox />*,
- *<p:commandButton />*, *<p:button />*.

Tyto komponenty je v PrimeFaces formuláři třeba využít pro správné zpracování dat v nich obsažených, není vhodné je kombinovat s těmi klasickými obsaženými ve jmenném prostoru *h*. Některé komponenty rozšiřují ty původní ve jmenném prostoru *h* méně, některé více. Například komponenta *<p:inputTextArea />* rozšiřuje *<h:inputTextarea />*, stejnou komponentu ovšem rozšiřuje i komponenta *<p:editor />* s tím rozdílem, že je použit textový WYSIWIG editor. Znatelné rozšíření je znát například i na komponentě *<p:password />* (rozšiřuje *<h:inputSecret />*), kterou lze například nastavit pro automatickou kontrolu zadávaného hesla a informování uživatele aplikace o jeho složitosti (k tomu slouží atribut *feedback*).

Stejně jako u standardních JSF formulářů je většinou pro odeslání dat ve formuláři potřeba použít komponenty tlačítek, tedy v případě PF *<p:commandButton />* či *<p:buttton />*. Obě rozšiřují původní *h* komponenty se stejným názvem a mají tedy podobné vlastnosti jako ony. Na rozdíl od nich ale zpracovávají formulářová data AJAXem, data jsou odeslána bez načítání celého dokumentu. Často je vhodné u odesílacích tlačítek použít atribut *update* k provedení částečného vykreslení. Většinou je totiž třeba překreslit samotný formulář kvůli chybovým hláškám (viz následující podkapitola).

Problém může nastat v případě využití konvertorů ve formuláři, ty totiž nejsou zcela kompatibilní s AJAXem. Pokud dojde po prvním pokusu o odeslání formuláře k chybě ve validaci, ale konverze dat již proběhla, celý formulář přestane fungovat a je třeba znovu načíst celou stránku s formulářem. Toto lze v případě vlastních konvertorů vyřešit provedením kontroly úspěšnosti validace formuláře před samotnou konverzí.

Toto překreslení formuláře může způsobit problémy, konkrétně v případě, kdy je ve formuláři použita komponenta *<p:captcha />*, která obstarává kontrolu CAPTCHA (zprostředkovanou reCaptcha) před odesláním dat. Tato komponenta totiž při svém překreslení přestane fungovat a způsobí nefunkčnost celého formuláře. Toto lze vyřešit vynecháním CAPTCHA komponenty z částečného překreslování a překreslit jen potřebné chybové hlášky.

5.3.4 Notifikace

JSF samotné umožňuje efektivně zobrazovat informační či chybové hlášky, často bez potřeby složitého programování. Systém totiž v případě potřeby vypsání hlášky (vytvořené např. v beaně nebo validátoru) pouze vyhledá ve faceletu potřebný element hlášky

a ten nahradí daným textem. Framework PrimeFaces obsahuje několik komponent, které rozšiřují ty základní JSF komponenty obstarávající notifikace a vedle nich nabízí i další způsoby zobrazování těchto hlášek.

Základní komponenty jsou definovány elementy *<p:messages />* a *<p:message />*. Rozdíl mezi nimi spočívá v jejich způsobu použití. *<p:messages />* se používá pro zobrazení hlášek na jednom místě a často nepřiřazených k jiné komponentě (tzv. globálních). Naproti tomu elementy *<p:message />* jsou vždy přiřazeny ke konkrétní komponentě pomocí atributu *for* s hodnotou ID dané komponenty. Toto přiřazení může proběhnout buď při vytvoření instance hlášky v beaně nebo automaticky například při validaci. Dalším rozdílem od *<p:messages />* je absence atributu *autoUpdate*, takže je vždy nutno tyto hlášky manuálně překreslit (jako např. v přechozí podkapitole o formulářích).

Specialitou PF je komponenta zvaná Growl definovaná elementem *<p:growl />*. Její funkce je podobná jako v případě *<p:messages />*, ale liší se svým designem a způsobem zobrazování hlášek. Její design byl inspirován stejnojmenným systémem zobrazování notifikací na prostředí Mac OS X. Je vhodná pro zobrazování hlášek způsobených AJAXovou akcí, hlášku totiž zobrazuje stylem upozorňovacího okna, které překrývá zbytek stránky. Podle svého nastavení se hláška může zobrazovat jen určitý čas, než se automaticky skryje nebo ji může skrýt uživatel sám. Je vybavena atributem *autoUpdate*.

Nejen k notifikacím lze využít také dialogová okna. Tato komponenta je definována elementem *<p:dialog />*, který obsahuje to, co se má v okně zobrazit. Navíc lze díky několika atributům upravit různé vlastnosti tohoto okna, jako například modálnost, povolení změny velikosti apod. Chování dialogového okna lze ovládat pomocí JavaScriptových funkcí.

Zvláštní notifikační komponentou je *<p:ajaxStatus />*, která má za úkol informovat uživatele o probíhající AJAXové komunikaci. Pracovat s komponentou lze dvěma způsoby, každý z nich je ale ovládán událostmi vyvolanými AJAXovou akcí. Názvy těchto událostí jsou:

- *default* (výchozí stav),
- *prestart* (před začátkem akce),
- *start* (po začátku akce),
- *success* (po úspěšném zakončení akce),
- *error* (po chybovém zakončení akce),

- *complete* (po zakončení akce nehledě na výsledek).

Prvním způsobem, jak s komponentou pracovat, je vypsání nějakého obsahu v místě, kde se tag komponenty nachází. K tomu je zpravidla potřeba použít element *<f:facet />* s názvem události, pro kterou se má obsah v *<f:facet />*, vypsat. Druhou možností je vyvolání skriptu v případě události, ovládá se atributy elementu s touto strukturou: *on* + název události (např. *oncomplete*). Díky zkombinování událostí *start* a *complete* tak lze například po dobu vykonávání AJAXového požadavku zobrazit dialogové okno s informací o probíhající akci.

5.3.5 Rozvržení stránky

Několik komponent umožňuje snadno a rychle vytvořit celkové rozvržení stránky bez větší potřeby znalosti XHTML a CSS. Rozvržení lze přizpůsobit, lze určit, kde bude jaká část stránky umístěna, zda ji je možno skrýt atd. Toto předpřipravené rozvržení ale neodpovídá nepsaným standardům rozvržení moderních webových aplikací, proto není jeho využití velké.

PrimeFaces rozvržení lze vytvořit pomocí elementů *<p:layout />* a *<p:layoutUnit />*. Ten první vytvoří celé rozvržení a pomocí jeho atributů lze určit parametry tohoto rozvržení. Například atribut *fullPage* způsobí, že rozvržení bude pokrývat celou stránku tak, jak je zobrazena ve webovém prohlížeči. Element *<p:layoutUnit />* určuje jednotlivé části stránky, neboli jednotky a pomocí atributu *position* je nutno určit jeho pozici na stránce. Jako hodnota se tomuto atributu nastavuje anglický název světové strany, takže například pro umístění v horní části stránky je třeba nastavit hodnotu *north*. Lze nastavit i další parametry jako je například velikost jednotky v pixelech. K tomu slouží atribut *size*.

5.3.6 Menu

Framework nabízí dvě komponenty, díky kterým lze vytvořit plně přizpůsobitelné menu. Jde o komponenty určené elementy *<p:menu />* a *<p:menubar />*. Každou položku tohoto menu lze rozdělit do pod-menu, lze jí přiřadit hypertextový odkaz či provedení AJAXové akce. Tyto položky mohou být určeny pomocí elementů *<p:menuitem />* nebo programově pomocí beany.

Rozdíl mezi oběma komponenty je v orientaci vytvořeného menu. Menu vygenerované pomocí *<p:menu />* je orientováno vertikálně, zatímco to vygenerované pomocí *<p:menubar />* vertikálně. Menu komponenta by měla být vždy umístěna v rámci ele-

mentu *<h:form />* z důvodu možných akčních tlačítek na položkách. Samotnému menu lze nastavit několik dalších parametrů pomocí atributů. Například elementu *<p:menu />* lze přidat atribut *type*, který určí typ menu podle své hodnoty. Hodnota tohoto atributu může být *tiered*, *sliding* nebo prázdná pro běžné menu.

Pro přidání položek menu neprogramovým způsobem slouží elementy *<p:menuitem />* a *<p:submenu />* umístěné pod elementem menu. V případě *<p:menuitem />* jde o klasickou položku menu, které lze pomocí atributů nastavit několik vlastností jako je URL, AJAXová akce, ikona apod. Účelem elementu *<p:submenu />* je vytvoření pod-menu a je nutné mu nastavit atribut *label*, který obsahuje text zobrazený na patřičném tlačítku v menu. Pod tento element lze zařadit další elementy *<p:menuitem />* či *<p:submenu />*.

K programovému vytvoření struktury menu je třeba využít model umístěný v beaně. Tento model je instancí třídy *org.primefaces.model.MenuModel*, která implementuje metody potřebné pro přidávání položek (instance třídy *org.primefaces.component.menuitem.MenuItem*) či pod-menu (instance třídy *org.primefaces.component.submenu.Submenu*). Výsledný model je předán pomocí EL výrazu menu komponentě pomocí atributu *model*.

5.3.7 Datové seznamy

Tvorbu různých výpisů dat obsluhuje několik komponent a všechny umožňují použití funkcí jako je například automatické stránkování seznamů apod.

První z těchto komponent je určena elementem *<p:dataTable />* a rozšiřuje původní komponentu *<h:dataTable />*. Jde o datovou tabulku s různými nastavitelnými vlastnostmi.

Další komponentou je *<p:dataGrid />* rozšiřující původní komponentu *<h:panelGrid />*. Má podobné vlastnosti jako *<p:dataTable />* s tím rozdílem, že data nejsou uspořádána v tabulce do řádků, ale do panelů. Těch může být na jednom řádku více. Tato komponenta je vhodná například pro výpis dat, která nemají strukturu tabulky a je třeba jim dát specifický design.

5.3.8 Další užitečné komponenty

PrimeFaces nabízejí velké množství velmi užitečných komponent. V této podkapitole se budu věnovat jen některým z nich, některé komponenty ale mohou být v některých situacích vhodnější k využití než ty následující.

Díky komponentě určené elementem *<p:wizard />* lze vytvořit **průvodce**. Průvodce může fungovat například jako formulář rozdělený do několika kroků vyplňování. Každý z těchto pod-formulářů je při zvolení dalšího kroku průvodce zvlášť validován, takže uživatel může chyby v zadaných datech rovnou napravit. Data formuláře bývají odeslána až v posledním kroku. Jednotlivé části průvodce se tvoří pomocí elementu *<p:tab />*.

Často nepostradatelnou formulářovou komponentou je **kalendář** určený tagem *<p:calendar />*. Umožňuje pro uživatele snadné zadávání data a času pomocí interaktivního kalendáře, kde stačí jen navolit den, případně čas a framework obstará převod tohoto data a času do objektu třídy *DateTime*. Podobnou funkci má i *<p:colorPicker />*, který slouží jako **komponenta pro výběr barvy**. Ta podobným způsobem obstarává uživatelsky přívětivý výběr barvy ve formuláři.

Zvláštním formulářovým prvkem je *<p:autoComplete />*. Jde o klasické **textové pole** s tím rozdílem, že obsahuje **našeptávač**. Element obsahuje atribut *completeMehod*, do jehož hodnoty je třeba pomocí EL výrazu doplnit metodu, která vrací kolekci typu *List* a má jeden parametr typu *String*. Tato metoda se zavolá pokaždé, kdy uživatel přestane do pole na chvíli psát a je zadán určitý počet znaků. Parametr metody obsahuje dosud zadaný text a metoda vrací kolekci obsahující textové řetězce, které se zobrazí v našeptávači. Element této komponenty obsahuje mnoho atributů, kterým lze měnit její chování jako např. minimální počet znaků, který je třeba zadat k našeptávání apod.

Zajímavá komponenta je **Rating**, určená elementem *<p:rating />*. Jde o klasické hodnocení pomocí hvězdiček. Komponentě lze pomocí atributů elementu nastavit počet zobrazovaných hvězdiček, povolení vstupu (zda lze pomocí komponenty aktivně hodnotit nebo je určena jen k zobrazení hodnocení) atd. Rating je třeba umístit do formuláře a v případě potřeby aktivního hodnocení mu přiřadit pomocí atributu *value* proměnnou, kam bude hodnocení uloženo.

Několik komponent umožňuje generování **grafů**. Jde například *<p:pieChart />*, *<p:lineChart />* a řadu dalších. Každá komponenta generuje jiný druh grafu. Důležitým

atributem elementů grafů je *value*, do jeho hodnoty je třeba pomocí EL výrazu vložit objekt modelu. Tento objekt je instancí třídy modelu daného typu grafu, například v případě koláčového grafu jde o třídu *org.primefaces.model.chart.PieChartModel*. Způsob konstrukce modelového objektu se liší v závislosti na typu grafu.

Jednou z méně kvalitních komponent je **Lightbox** určený elementem *<p:lightbox />*. Umožňuje sice zobrazení jakéhokoliv obrázku či stránky v lightboxu včetně procházení, knihovna obstarávající tuto funkčnost se ale svou kvalitou nevyrovná konkurenčním knihovnám, jako je třeba Fancybox. Většinou je proto lepší místo využití této komponenty použít klasický způsob zavádění lightboxu.

Velice užitečnou komponentou je **DataExporter**, ke kterému se dá přistupovat skrze element *<p:dataExporter />*. Úkolem DataExporteru je export obsahu datové tabulky do dokumentu různého formátu. Díky DataExporteru tak lze například exportovat data z tabulky do dokumentu ve formátu MS Excel. Kromě Excelu jsou dále podporovány formáty PDF, CSV a XML. Element DataExporteru je třeba přiřadit pod akční tlačítko (*<p:commandButton />*) a důležitými atributy jsou:

- *type* – určuje typ exportovaného dokumentu,
- *target* – ID elementu, který data k exportu obsahuje
- *fileName* – název souboru, který bude vyexportován.

6 Ukázková aplikace TheShop

Pro účely demonstrace technologie JSF s použitím sady komponent PrimeFaces jsem vytvořil příkladovou webovou aplikaci nazvanou **TheShop**. Jde o jednoduchý e-shop, konkrétně o jeho část přístupnou běžnému uživateli aplikace, tedy nakupujícímu. Administrační rozhraní není v příkladu obsaženo, uživatelská část je k demonstraci dostačující. Zdrojový kód, který slouží jako souhrnný příklad k výše uvedené teoretické části, je připojen jako elektronická příloha.

6.1 Uživatelské rozhraní

Aplikace má strukturu typickou pro internetové obchody a je ve dvou jazykových mutacích, v češtině a angličtině. Umožňuje nákup produktu, přihlašování uživatelů, atd. K vytvoření základního rozvržení jsem použil komponenty PrimeFaces k tomu určené. Stránka aplikace je rozložena do čtyř částí, a to hlavičky, patičky, postranního panelu a obsahové části. Nejmenší funkci plní patička, obsahuje totiž jen základní informaci o aplikaci. O něco důležitější je hlavička, která obsahuje kromě hlavního nadpisu s odkazem na domovskou stránku také přihlašovací formulář s odkazem na nákupní košík a přepínač jazyků. Postranní panel obsahuje hlavní menu a produkty. Nejdůležitější ze všech je obsahová část uprostřed, ve které jsou umístěny hlavní informace zobrazované uživateli.

Náhled na uživatelské rozhraní lze vidět na přiložených obrázcích (přílohy 11.2).

6.1.1 Menu

Menu umístěné v horní části postranního panelu funguje jako hlavní komponenta umožňující navigaci po aplikaci. Obsahuje tři hlavní položky, které umožňují získat základní informace o fiktivním internetovém obchodu. V případě, že je přihlášen uživatel, je v menu i čtvrtá položka, která umožňuje správu uživatelského účtu.

6.1.2 Domovská strana

Domovskou stranu tvoří obsahová část uživatelského rozhraní. Ta obsahuje uvítání, dále pak náhled na tři náhodně vybrané produkty. Náhled obsahuje miniaturu obrázku produktu, který lze kliknutím zvětšit v lightboxu a zkrácený popis produktu.

Druhou část domovské stránky tvoří poslední tři komentáře k produktům. Ty mají navíc připojenou informaci o autorovi komentáře, datu jeho odeslání a hlavně o produktu, ke kterému se komentář vztahuje.

6.1.3 Produkty

Základní přehled produktů nabízených k prodeji se nachází v postranním panelu hned pod menu. Lze zde produkty vyhledávat s pomocí našeptávače a také je zobrazen seznam produktů rozdělený do kategorií. K vytvoření tohoto přehledu je využito menu typu tiered. Kliknutím na název produktu lze zobrazit jeho detail. Toto platí ve všech částech aplikace, kde je název nějakého produktu uveden.

Stránka s detailem produktu obsahuje jeho plný popis, obrázek, název kategorie, uživatelské hodnocení a cenu. Ve spodní části stránky je umístěný výpis všech komentářů k produktu. Přihlášení uživatelé navíc mají možnost přidat své vlastní hodnocení produktu, produkt okomentovat a přidat ho do svého nákupního košíku.

6.1.4 Uživatelé

Primárním účelem systému uživatelů je umožnění snadného vytváření objednávek produktů a jejich následné evidence. Vedlejší funkcí je podpora pro další funkce jiných částí systému jako například komentování či hodnocení produktů. U každého uživatele se eviduje e-mail a heslo (které jsou zároveň použity jako přihlašovací údaje), jméno a adresa.

Pro nového a neregistrovaného uživatele je v této části systému nejdůležitější nejprve hlavička webu, která obsahuje kromě přihlašovacího formuláře také odkaz na registraci. Ta vyžaduje CAPTCHA ověření a následné vyplnění registračního formuláře. V tom je třeba zadat e-mail, heslo a jméno. V případě úspěšné registrace je novému uživateli zaslán potvrzující e-mail. Ke správné funkci odesílání e-mailu je třeba, aby byl v aplikaci nakonfigurován správný SMTP server.

Po registraci je třeba se v hlavičce webu přihlásit. Po přihlášení tento formulář zmizí a místo něj se objeví odkaz pro odhlášení a odkaz na nákupní košík. Po přihlášení se zároveň objeví čtvrtá položka hlavního menu, kde lze uživatelský účet spravovat či přejít na seznam provedených objednávek.

6.1.5 Nákupní košík

Základní funkcí nákupního košíku v internetovém obchodě je shromáždění produktů, které se uživatel chystá koupit. V aplikaci TheShop se do něj produkty přidávají na stránce s detailem daného produktu (lze zvolit i množství požadovaného produktu) a zpřístupnit košík lze pomocí odkazu v hlavičce, který obsahuje také informaci o počtu položek v košíku.

Samotná stránka košíku obsahuje přehled jeho obsahu v datové tabulce. Ta umožňuje seřadit data podle názvu, počtu či celkové ceny za položku. Obsahuje také součtový řádek udávající celkový počet položek v košíku a celkovou cenu. Každý řádek představuje jeden produkt a obsahuje kontextové menu. Pomocí toho je možné danou položku z košíku odstranit či zobrazit detail o produktu. Tlačítkem umístěným pod tabulkou lze přejít k objednávce.

6.1.6 Objednávky

Vytvoření samotné objednávky je realizováno pomocí průvodce a probíhá ve dvou krocích. Tím prvním je volba způsobu doručení včetně možné změny doručovací adresy. Druhým krokem je rekapitulace, kde je zobrazena celá objednávka detailně a uživatel ji potvrdí a odešle.

Přehled uskutečněných objednávek se nachází v menu pod položkou správy uživatelského účtu. Jde o datovou tabulku obsahující údaje jako je datum odeslání, celková částka a stav vyřizování objednávky.

6.1.7 URL adresy

Všechny URL adresy v aplikaci jsou přizpůsobeny SEO standardům pomocí knihovny Tuckey URL Rewrite a odpovídají aktuální lokaci v aplikaci. Například adresa pro registrace účtů je */register*, pro nastavení uživatelského účtu */user/settings* apod.

6.2 Struktura aplikace

Celková logika aplikace je rozdělena do tří vrstev. Tou první jsou samotné faceletové stránky, následuje datový model a aplikační vrstva. Poslední dvě jsou detailně znázorněny na přiloženém diagramu tříd (příloha 11.1).

6.2.1 Faceletové stránky

Struktura vrstvy faceletu je poměrně jedndouchá. V kořenové složce se nacházejí všechny důležité stránky. Důležité jsou také složky *templates* (obsahuje hlavní šablonu) a *resources*, kde se nacházejí klasické podpůrné soubory jako CSS apod. Méně důležitou složkou je *errors*, která obsahuje stránky pro chybová hlášení.

Aplikace je přizpůsobena běhu na aplikačním serveru Apache Tomcat a pracuje s databází MySQL.

6.2.2 Datový model

S datovým modelem aplikace pracuje skrze framework Hibernate pro objektově relační mapování. Všechny jeho třídy včetně mapovacích XML souborů jsou umístěny v balíčku *ORM*. Kromě klasických tříd datového modelu jsou v tomto balíčku obsaženy také výčtové typy.

6.2.3 Aplikační vrstva

Aplikační vrstva se skládá z následujících balíčků:

- *controllers*,
- *util*,
- *helpers*,
- *validators*,
- *texts*.

Nejdůležitějším balíčkem je ***controllers***. Obsahuje všechny beany, třídy v tomto balíčku tedy řídí celou aplikaci.

V balíčku ***util*** se nachází jen jedna třída a tou je *HibernateUtil*. Ta obstarává práci s frameworkem Hibernate, tedy i práci s datovým modelem.

Balíček ***helpers*** obsahuje podpůrné třídy, které jsou v aplikaci využívané na různých místech. Příkladem takové podpůrné třídy je *Hasher*, obsahuje metody obstarávající převod textových řetězců (např. hesel) do hash formy.

Třídy v balíčku ***validators*** obstarávají validaci formulářových dat. Jde o klasické JSF validátory.

Důležitým balíčkem je ***texts***, a to i přes fakt, že neobsahuje žádné třídy. Jsou zde všechny lokalizované texty v souborech *.properties*.

7 Závěr

Způsobů tvorby webových aplikací je mnoho, stejně jako prostředků k vytvoření takové aplikace. Prostřednictvím většiny z nich lze téměř vždy dosáhnout požadovaného výsledku, jak je ale v mé práci vidět, většinou o výběru prostředku rozhoduje cesta k dosažní cíle.

Z obsahu práce lze vyvodit porovnání dvou nástrojů, PHP a JSF s použitím PrimeFaces. PHP je v současnosti nejrozšířenějším programovacím jazykem webových aplikací, a to díky své jednoduchosti. Znalosti PHP programátora nemusí být velké a přes to lze pomocí PHP dosáhnout různých výsledků. Na druhé straně použití Javy vyžaduje mnohem širší znalosti, proto je jako prostředek pro tvorbu webových aplikací využívána méně. Programátor s dobrou znalostí Javy je ovšem schopen vytvořit kvalitní webové aplikace v mnohem kratším čase a s menším úsilím vynaloženým na údržbu aplikace.

Frameworky JavaServer Faces a PrimeFaces se ukázaly jako vysoce kvalitní nástroj. Množství psaného kódu je nepoměrně menší než je tomu v případě PHP aplikací a navíc lze využít všechny výhody standardní Javy jako je například socketová komunikace. Java EE je využitelná nejen pro velké projekty, její použitelnost je univerzální. Jediný problém, na který u ní může tvůrce webové aplikace narazit, je menší dostupnost Java hostingů. V dostupnosti má jednoznačně výhodu PHP. Hostingové služby podporující PHP jsou poskytovány i zdarma. V případě výběru mezi PHP a Javy jako nástroji pro tvorbu webových aplikací, které mají plnit více funkcí, vítězí Java EE podpořená JSF s PrimeFaces, a to za předpokladu dobré znalosti obou technologií.

Příkladem vhodnosti Javy je vytvořená ukázková aplikace TheShop. Tento ukázkový internetový obchod byl vytvořen za poměrně krátkou dobu a díky využití Javy podpořené JSF s PrimeFaces nebylo zapotřebí psát nadměrné množství zdrojového kódu. Kdyby stejná aplikace se stejnými vlastnostmi měla být napsána v PHP, její tvorba by trvala odhadem třikrát až čtyřikrát déle a muselo by být napsáno podstatně více zdrojového kódu, což by mimo jiné způsobilo mnohem větší chybovost a její údržba by byla o mnoho náročnější.

8 Terminologický slovník

Termín	*Zkratka*	*Význam [zdroj]*
Active Server Pages .NET	ASP.NET	Programovací jazyk určený pro tvorbu webových aplikací od společnosti Microsoft. [vlastní definice autora]
Anotace v Javě		Příznak, díky kterému lze třídě, metodě či proměnné přiřadit nějakou informaci či funkci mimo běžný kód. Anotaci lze nastavit parametry. Anotace s parametrem vypadá například takto: *@Anotation(value="Hello world!")* [vlastní definice autora]
Ant		Program umožňující přizpůsobitelné sestavování aplikací vytvořených v Javě. [vlastní definice autora]
Aplikační logika		Vrstva aplikace obstarávající chod jádra aplikace, zpracování dat atd. [vlastní definice autora]
Asynchronous JavaScript and XML	AJAX	Technologie umožňující interaktivní načítání specifických dat a jejich zpracování, například jejich zobrazení v již načtené webové stránce. [vlastní definice autora]
Atribut XML elementu		Příznak XML elementu určující nějakou jeho vlastnost. [vlastní definice autora]
Boolean		Logický datový typ reprezentovaný jednou z hodnot pravda/nepravda. [vlastní definice autora]
C		Programovací jazyk určený původně pro potřeby systémů Unix. [vlastní definice autora]
Cache		Vyrovnávací paměť, jejímž účelem je zvýšení výkonu pomocí ukládání specifických dat náročných na přenos. [vlastní definice autora]
Cascade Style Sheet	CSS	Kolekce metod pro grafickou úpravu webových stránek. [27]
Completely Automated Public Turing Test To Tell Computers and Humans Apart	CAPTCHA	Způsob ověřování uživatele provádějícího vstup do webové aplikace. Účelem kontroly je ověřit, zda je uživatel skutečně člověk a ne bot (program). [vlastní definice autora]
Datový typ		Datový typ proměnné určuje, jak je proměnná uložena v paměti a jaké operace lze s proměnnou provádět. [2]

Desktopová aplikace		Aplikace běžící na straně pevného počítače (může být klientský). [vlastní definice autora]
Dialogové okno		Prvek grafického rozhraní aplikace, jehož účelem je informování uživatele (např. o skutečnosti, jež nastala). [vlastní definice autora]
EL výraz		Způsob předání dat do faceletové stránky při tvorbě v JSF. Syntaxe EL výrazu je ve formě *#{}*. [vlastní definice autora]
Enterprise Java Bean	EJB	Serverová komponenta umožňující další vrstvení aplikace v Java EE. [vlastní definice autora]
Extensible HTML	XHTML	Nástupce klasického HTML. Na rozdíl od něj dodržuje XML standardy, jde o XML dokument se specifickými vlastnostmi. [vlastní definice autora]
Extensible Markup Language	XML	Obecný značkovací jazyk, který byl vyvinut a standardizován konsorciem W3C. [3]
Framework		Softwarová struktura, která slouží jako podpora pri programování, vývoji a organizaci jiných softwarových projektů. Může obsahovat podpůrné programy, knihovny, API, návrhové vzory nebo doporučené postupy při vývoji. [22]
Getter a setter		Zvláštní druhy metod, které umožňují přístup k privátním proměnným. Český ekvivalent zní „přístupové metody“, ten se ale v odborné praxi příliš nepoužívá. [vlastní definice autora]
Hash forma		Data zpracovaná speciální šifrovací funkcí. Tato funkce je jednosměrná, data jednou zašifrovaná tedy již nejdou vrátit do původní podoby. [vlastní definice autora]
Hosting		Služba poskytující prostor k provozu webové aplikace. Většinou jde o zázemí s určitým diskovým prostorem a softwarovou výbavou nakonfigurované pro hromadný přístup uživatelů hostované webové aplikace. [vlastní definice autora]
HyperText Markup Language	HTML	Značkovací jazyk obsahující informace pro zobrazení webové stránky, kterému rozumí webový prohlížeč. V současnosti se již nevyvíjí, nástupcem je XHTML. [vlastní definice autora]
HyperText Transfer Protocol	HTTP	Základní protokol pro přenos dat webu mezi serverem a klientem. [vlastní definice autora]
Instance třídy		Instance třídy (v některých programovacích jazycích také objekt) je konkrétní datový objekt v paměti odvozený z nějakého vzoru (třídy) používaný v objektově orientovaných programovacích jazycích. [5]
Iterace		Opakování určitých dat v cyklu.

		[vlastní definice autora]
Java Archive	JAR	Spustitelný archiv obsahující zkompilované zdrojové soubory Java aplikace. [vlastní definice autora]
Java balíček		Část zdrojového kódu Java aplikace nebo Javy samotné obsahující specifický kód patřící pod jednu větev. [vlastní definice autora]
Java Persistance API	JPA	Java framework umožňující objektově relační mapování. [vlastní definice autora]
Jmenný prostor		Jmenné prostory slouží k tomu, aby logicky sdružovaly typy, které spolu souvisejí. [7]
JSP Standard Tag Library	JSTL	Sada tagů používaných pro práci s daty na JSP stránkách. [vlastní definice autora]
Klient		Ta strana síťové komunikace, která žádá informace. Typickým klientským softwarem je např. webový prohlížeč. [vlastní definice autora]
Knihovna		Sada tříd a dalších programových částí, která má za účel rozšířit funkce aplikace, ve které je knihovna nasazena. [vlastní definice autora]
Kontextová nabídka		Druh menu umožňující s komponentou uživatelského rozhraní aplikace. Často je toto menu zpřístupňováno pravým tlačítkem myši. [vlastní definice autora]
Lightbox		Webová komponenta umožňující zobrazení informace (většinou obrázku) ve speciálním dialogovém okně. Často je realizována pomocí jQuery. [vlastní definice autora]
Listener		Ta část programu, která má za úkol čekat na provedení nějaké akce. Při vykonání akce provede jinou akci, ta a způsob jejího provedení záleží na implementaci listeneru. [vlastní definice autora]
Maven		Program umožňující přizpůsobitelné sestavování aplikací vytvořených v Javě. [vlastní definice autora]
Metoda		Ta část definice třídy, která dává jejím instancím nějakou funkci. [vlastní definice autora]
Modálnost		Stav komponenty uživatelského rozhraní, kdy lze ovládat pouze danou modální komponentu a žádnou jinou. [vlastní definice autora]
Našeptávač		Webová komponenta přidružená většinou ke vstupnímu poli formuláře. Jejím úkolem je napovídat (našeptávat) hodnotu k zadání uživateli webové aplikaci při zadávání do pole. [vlastní definice autora]
null		NULL je obvykle označení pro žádnou hodnotu nebo nic. [10]

Objekt		Základní stavební prvek aplikací vytvořených v OOP jazyce. Je definován třídou, která tvoří návod k vytvoření objektu. [vlastní definice autora]
Objektově orientované programování	OOP	Objektově orientované programování vychází z myšlenky, že všechny programy, které vytváříme, jsou simulací buď skutečného nebo nějakého námi vymyšleného virtuálního světa. [13]
Objektově relační mapování	ORM	Programovací technika v softwarovém inženýrství, která zajišťuje automatickou konverzi dat mezi relační databází a objektově orientovaným programovacím jazykem. [11]
Open-source		Open-source (neboli otevřený) software má běžně dostupný zdrojový kód a to zdarma. Licence takového softwaru umožňuje jeho modifikace s dodržením autorských práv. [vlastní definice autora]
Perl		Interpretovaný programovací jazyk vytvořený Larry Wallem v roce 1987. S rozvojem internetu se Perl stal velmi populárním nástrojem pro tvorbu CGI skriptů. [14]
Persistentní vrstva		Vrstva aplikace obsluhující programovou část zabývající se daty. Může se skládat z datového modelu, konfigurace atd. [vlastní definice autora]
Pixel		Nejmenší jednotka digitální rastrové grafiky. [17]
Požadavek DELETE		Metoda DELETE požaduje, aby server smazal zdroj identifikovaný pomocí URL. [7]
Požadavek GET		Příkaz GET načte požadované informace, identifikované pomocí URL adresy, jen ze serveru. [8]
Požadavek POST		Metoda POST se používá k odeslání informací na server. Nejčastější použití je odeslání dat vyplněného formuláře na server. [8]
Požadavek PUT		Metoda PUT slouží k odesílání dat z klienta na server. Není tak běžná jako příkaz POST. Příjemce nesmí ignorovat libovolnou položku hlavičky Content-* (např. Content-Range) hlavičky, které nerozumí, nebo není implementována, a musí v tomto případě odpovědět 501 (Not Implemented). [7]
Prefix		Prefix v překladu znamená předpona. V případě XML jde o označení elementu pomocí daného prefixu a dvojtečky. [vlastní definice autora]
Primární klíč		Unikátní identifikátor záznamu v databázi. [vlastní definice autora]
reCaptcha		Zprostředkovatel CAPTCHA kontroly. [vlastní definice autora]
Regulární výraz		Regulární výraz slouží k vyhledání části řetězce, která není předem (úplně) známá nebo která může mít více podob.

		[29]
Rozhraní		Rozhraní je množina metod, která může být implementována třídou. Rozhraní pouze popisuje metody, jejich vlastní implementace však neobsahuje. [9]
Search Engine Optimization	SEO	Metodika vytváření a upravování webových stránek takovým způsobem, aby jejich forma a obsah byly vhodné pro automatizované zpracování v internetových vyhledávačích. [20]
Server		Stroj, který poskytuje data klientům. K tomu je vybaven jak hardwarově, tak softwarově. [vlastní definice autora]
Session		Instance klienta pracujícího s klient/server aplikací. Český ekvivalent pro session je „sezení“, tento výraz se však příliš nepoužívá. [vlastní definice autora]
Show case		Praktická ukázka programu či jeho části s uvedeným vysvětlujícím zdrojovým kódem. [vlastní definice autora]
Simple Mail Transfer Protocol	SMTP	Internetový protokol určený pro přenos zpráv elektronické pošty mezi přepravci elektronické pošty. [21]
Skript		Postupné instrukce pro provedení programu. [vlastní definice autora]
Socket		Prostředek meziprocesní komunikace. Procesy nemusejí být na jednom počítači, takže je vhodný pro síťovou komunikaci. [vlastní definice autora]
Structured Query Language	SQL	Dotazovací jazyk používaný pro komunikace s databází. Pomocí něho lze data získávat, měnit vlastnosti databáze atd. [vlastní definice autora]
Tag		Tag, neboli značka (český výraz se příliš nepoužívá), vymezuje v XML dokumentu určitá data. Tag označuje XML element. [vlastní definice autora]
Třída		Definice objektů vytvářených v aplikaci. Instancí tříd jsou objekty. [vlastní definice autora]
Uniform Resource Locator	URL	Řetězec znaků s definovanou strukturou, který slouží k přesné specifikaci umístění zdrojů informací (ve smyslu dokument nebo služba) na Internetu. [26]
Vyjímka		Událost v aplikaci, která mění její běžný chod, často se jedná o chyby v aplikaci. [vlastní definice autora]
Vývojové prostředí (Integrated Development Environment)	IDE	Software usnadňující práci programátorů, většinou zaměřené na jeden konkrétní programovací jazyk. [28]

Web Archive	WAR	Archiv obsahující zdrojové kódy webové aplikace v Java servletu. Soubor WAR se nasazuje na aplikační server, který ho dále zpracovává a obstarává chod aplikace v něm. [vlastní definice autora]
Webový formulář		Soustava vstupních polí na webové stránce. Data v nich lze odeslat a dále zpracovat (jak na straně klienta, tak serveru). [vlastní definice autora]
What You See Is What You Get	WYSIWIG	Zvláštní druh XHTML editoru, kdy uživatel nepracuje s XHTML kódem, ale upravuje přímo výsledný vzhled dokumentu. [vlastní definice autora]
XML element		Základní stavební prvek XML dokumentů. Je definován tagy a může se větvit do dalších XML elementů. [vlastní definice autora]

9 Seznam literatury

[1] ÇIVICI, Çağatay. PRIME TEKNOLOJI. *PrimeFaces User's Guide* [online]. [cit. 2012-04-26]. Dostupné z: http://primefaces.googlecode.com/files/primefaces_users_guide_3_2.pdf

[2] Datový typ. GYMNÁZIUM OLOMOUC-HEJČÍN. *Projekty SIPVZ* [online]. 2005 [cit. 2012-04-28]. Dostupné z: http://projektysipvz.gytool.cz/ProjektySIPVZ/Default.aspx?uid=66

[3] Extensible Markup Language. In: *Wikipedia*: *the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001 [cit. 2012-04-26]. Dostupné z: http://cs.wikipedia.org/wiki/Extensible_Markup_Language

[4] HEFFELFINGER, David. *Jave EE 6 with GlassFish 3 application server*: *a practical guide to install and configure the GlassFish 3 application server and develop Java EE 6 applications to be deployed to this server*. Birmingham, U.K.: Packt Open Source, 2010, 469 s. ISBN 978-1-849510-36-3.

[5] Instance třídy. In: *Wikipedia*: *the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001 [cit. 2012-04-26]. Dostupné z: http://cs.wikipedia.org/wiki/Instance_t%C5%99%C3%ADdy

[6] JAVA.NET. *GlassFish*: *Open Source Application Server* [online]. [cit. 2012-04-26]. Dostupné z: http://glassfish.java.net

[7] Katedra informatiky: Fakulta elektrotechniky a informatiky, VŠB-TUO. [online]. [cit. 2012-04-26]. Dostupné z: http://www.cs.vsb.cz/grygarek/PS/kotasek/htp04.htm

[8] Katedra informatiky: Fakulta elektrotechniky a informatiky, VŠB-TUO. [online]. [cit. 2012-04-26]. Dostupné z: http://www.cs.vsb.cz/grygarek/PS/kotasek/htp03.htm

[9] Naučte se Javu: Rozhraní. In: SEMECKÝ, Jiří. Interval [online]. [cit. 2012-04-26]. Dostupné z: http://interval.cz/clanky/naucte-se-javu-rozhrani/

[10] NULL. In: *Wikipedia*: *the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001 [cit. 2012-04-26]. Dostupné z: http://cs.wikipedia.org/wiki/NULL

[11] Objektově relační mapování. In: *Wikipedia*: *the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001 [cit. 2012-04-26]. Dostupné z: http://cs.wikipedia.org/wiki/Objektov%C4%9B_rela%C4%8Dn%C3%AD_mapov%C3%A1n%C3%AD

[12] ORACLE. *Oracle a Java* [online]. [cit. 2012-04-26]. Dostupné z: http://www.oracle.com/cz/technologies/java/index.html

[13] PECINOVSKÝ, Rudolf. Myslíme objektově v jazyku Java 5.0. Vyd. 1. Praha: Grada, 2004. ISBN 80-247-0941-4.]

[14] Perl. In: *Wikipedia*: *the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001 [cit. 2012-04-26]. Dostupné z: http://cs.wikipedia.org/wiki/Perl

[15] PHP. [online]. [cit. 2012-04-26]. Dostupné z: http://php.net

[16] PHP: Úvod do PHP. [online]. [cit. 2012-04-26]. Dostupné z: http://www.webtvorba.cz/php/uvod-do-php.html

[17] Pixel. In: *Wikipedia*: *the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001 [cit. 2012-04-26]. Dostupné z: http://cs.wikipedia.org/wiki/Pixel

[18] PRIME TEKNOLOJI. *PrimeFaces* [online]. 2012 [cit. 2012-04-28]. Dostupné z: http://www.primefaces.org

[19] RED HAT. *JBoss Community* [online]. 2012 [cit. 2012-04-28]. Dostupné z: http://www.jboss.org

[20] Search Engine Optimization. In: *Wikipedia*: *the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001 [cit. 2012-04-26]. Dostupné z: http://cs.wikipedia.org/wiki/Search_Engine_Optimization

[21] Simple Mail Transfer Protocol. In: *Wikipedia*: *the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2012-04-26]. Dostupné z: http://cs.wikipedia.org/wiki/Simple_Mail_Transfer_Protocol

[22] Software framework. In: *Wikipedia*: *the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001 [cit. 2010-04-30]. Dostupné z: http://en.wikipedia.org/wiki/Software_framework

[23] STACK EXCHANGE. *Stack Overflow* [online]. [cit. 2012-04-26]. Dostupné z: http://stackoverflow.com

[24] THE APACHE SOFTWARE FOUNDATION. *The Apache Software Foundation* [online]. [cit. 2012-04-26]. Dostupné z: http://www.apache.org

[25] TRONÍČEK, Zdeněk. WEBINÁŘE.CZ. *Java Server Faces 2.0*: *Videozáznam z webináře* [online]. [cit. 2012-04-26]. Dostupné z: http://www.webinare.cz/videozaznamy/videozaznam/57-java-server-faces-2-0.aspx

[26] Uniform Resource Locator. In: *Wikipedia*: *the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001 [cit. 2012-04-26]. Dostupné z: http://cs.wikipedia.org/wiki/Uniform_Resource_Locator

[27] Úvod do CSS [online]. [cit. 2012-04-26]. Dostupné z: http://www.jakpsatweb.cz/css/css-uvod.html

[28] Vývojové prostředí. In: *Wikipedia*: *the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001 [cit. 2012-04-26]. Dostupné z:

http://cs.wikipedia.org/wiki/V%C3%BDvojov%C3%A9_prost%C5%99ed%C3%AD

[29] WATZKE, David. Regulární výrazy. In: ABC Linuxu [online]. [cit. 2012-04-26]. Dostupné z: http://www.abclinuxu.cz/clanky/programovani/regularni-vyrazy

10 Rejstřík

A

AJAX, 10, 12, 13, 29, 36, 38, 39, 40, 41, 42, 43, 51
anotace, 18, 19, 23, 24, 25, 26, 33, 34, 35, 51
Ant, 19, 51
Apache, 14, 16, 17, 20, 49

B

bean, 16, 21, 23, 24, 25, 26, 27, 29, 31, 32, 33, 34, 40, 41, 42, 43, 49, 52

C

CSS, 6, 10, 20, 28, 36, 38, 42, 49, 51

D

databáze, 9, 12, 13, 14, 16, 18, 25, 49, 54, 55
Databáze, 9, 18
DELETE, 19, 54

E

Eclipse, 15, 16
EL výraz, 26, 27, 31, 43, 44, 45, 52

F

facelet, 22, 27, 28, 29, 31, 33, 34, 36, 38, 40, 48, 49, 52
faces-config.xml, 23, 24, 31, 32, 35
formulář, 10, 12, 13, 19, 20, 24, 27, 29, 30, 31, 32, 33, 34, 39, 40, 41, 44, 46, 47, 49, 53, 54, 56, 63

G

GET, 19, 24, 54
GlassFish, 16, 17

H

Hibernate, 12, 18, 49
hosting, 16, 50, 52
HTML, 9, 10, 11, 12, 13, 20, 21, 22, 27, 28, 29, 36, 38, 52
HTTP, 6, 19, 23, 52

J

Java, 3, 4, 6, 7, 8, 9, 10, 12, 15, 16, 18, 19, 20, 21, 23, 24, 36, 50, 51, 52, 53, 56
JavaScript, 10, 12, 13, 29, 41, 51
JDBC, 18
JPA, 18, 53
jQuery, 3, 4, 10, 12, 36, 37, 39, 53
JSP, 20, 21, 22, 24, 30, 53

K

konvertor, 34, 35, 40

M

Maven, 19, 53

N

NetBeans, 15, 16, 28

O

ORM, 11, 12, 18, 49, 53, 54

P

phase listener, 35
PHP, 3, 4, 6, 9, 11, 12, 13, 14, 15, 16, 18, 20, 27, 50
POST, 19, 24, 54
PUT, 19, 54

S

SEO, 20, 48, 55
server, 11, 13, 14, 15, 16, 17, 19, 20, 22, 33, 37, 47, 49, 52, 54, 55, 56
servlet, 19, 20, 21, 23, 56

T

Tomcat, 16, 17, 33, 49

U

URL, 19, 20, 43, 48, 54, 55

V

validátor, 33, 34, 35, 40, 49

W

WAR, 20, 21, 28, 56, 62
web.xml, 19, 23, 38
WEB-INF, 20, 21

X

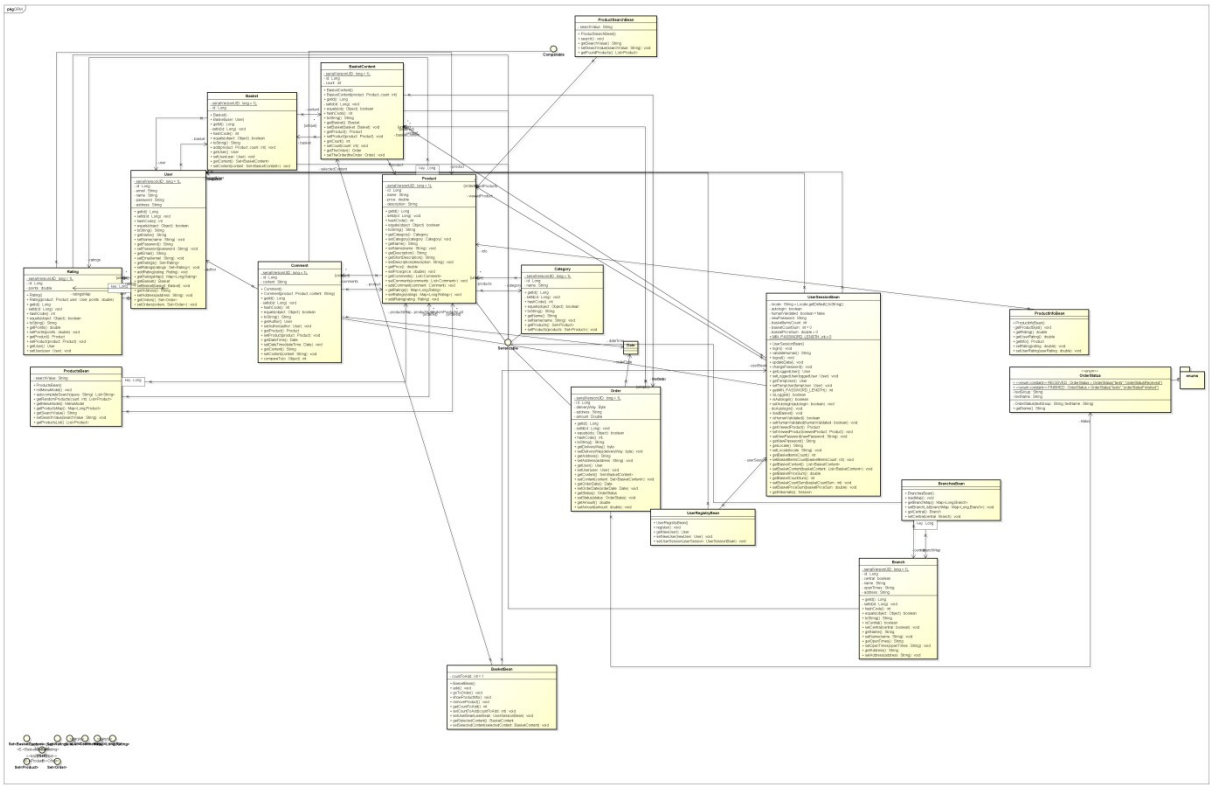
XHTML, 6, 42, 52, 56
XML, 6, 18, 19, 20, 21, 22, 23, 27, 38, 45, 49, 51, 52, 54, 55, 56

11 Přílohy

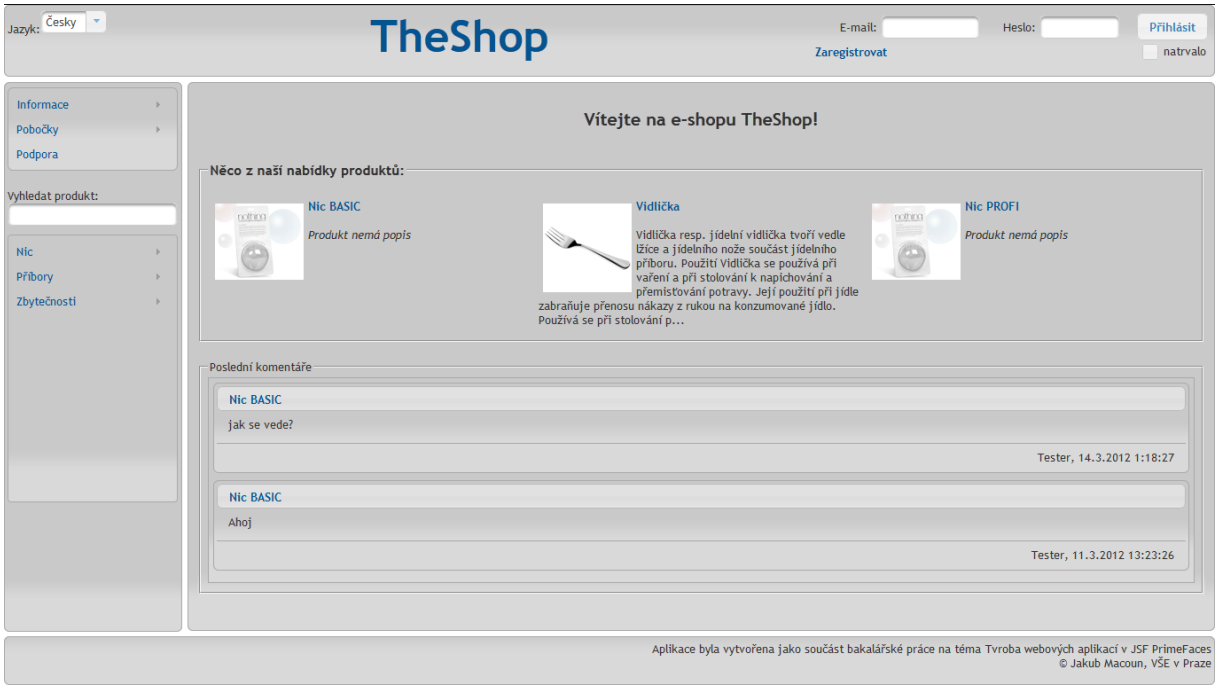
Následující přílohy obsahují podklady pro ukázkovou aplikaci TheShop:

- příloha 1 (také elektronická) – diagram tříd,
- příloha 2 – ukázka uživatelského rozhraní,
- příloha 3 (elektronická) – zdrojový kód,
- příloha 4 (elektronická) – soubor WAR.

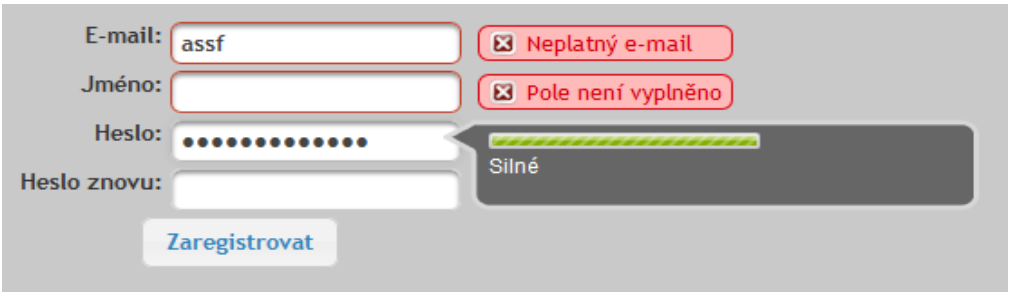
11.1 Diagram tříd



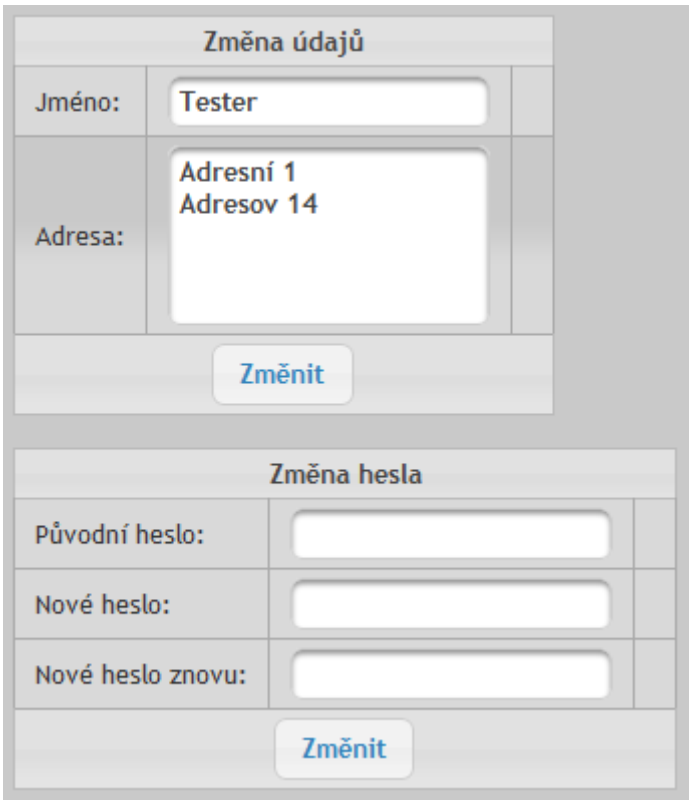
11.2 Ukázka uživatelského rozhraní



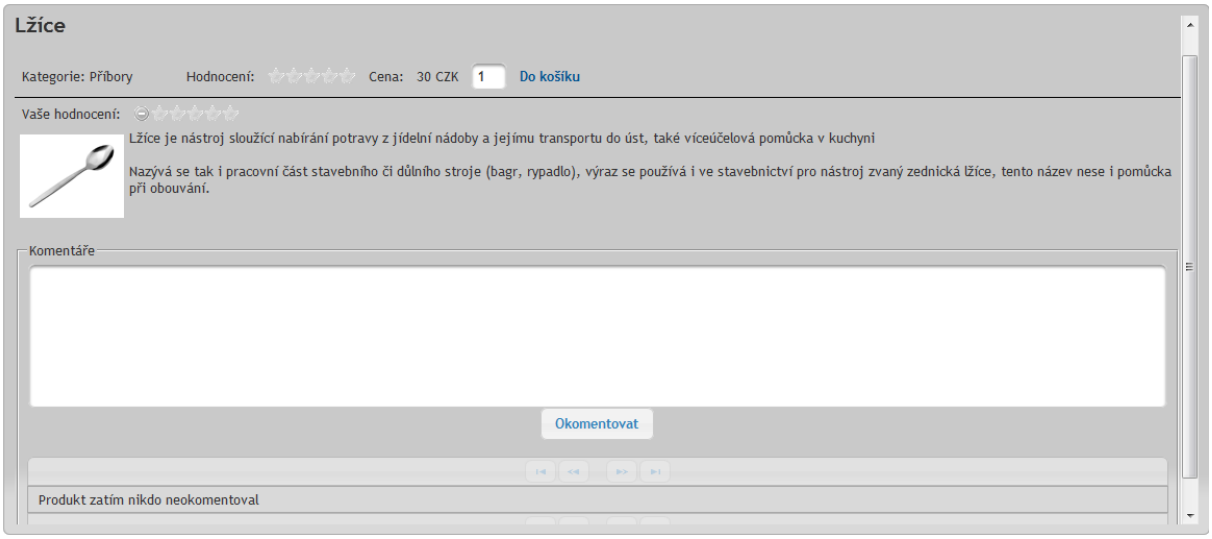
Příloha 11.2.1 – uživatelské rozhraní



Příloha 11.2.2 – registrační formulář



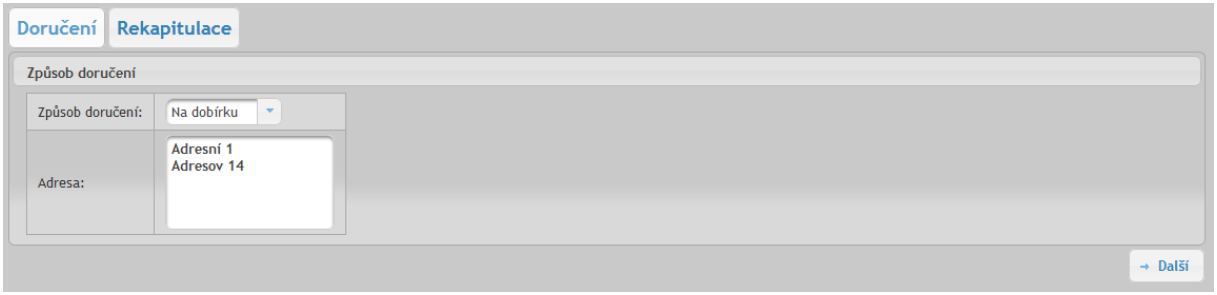
Příloha 11.2.3 – nastavení uživatelského účtu



Příloha 11.2.4 – detail produktu



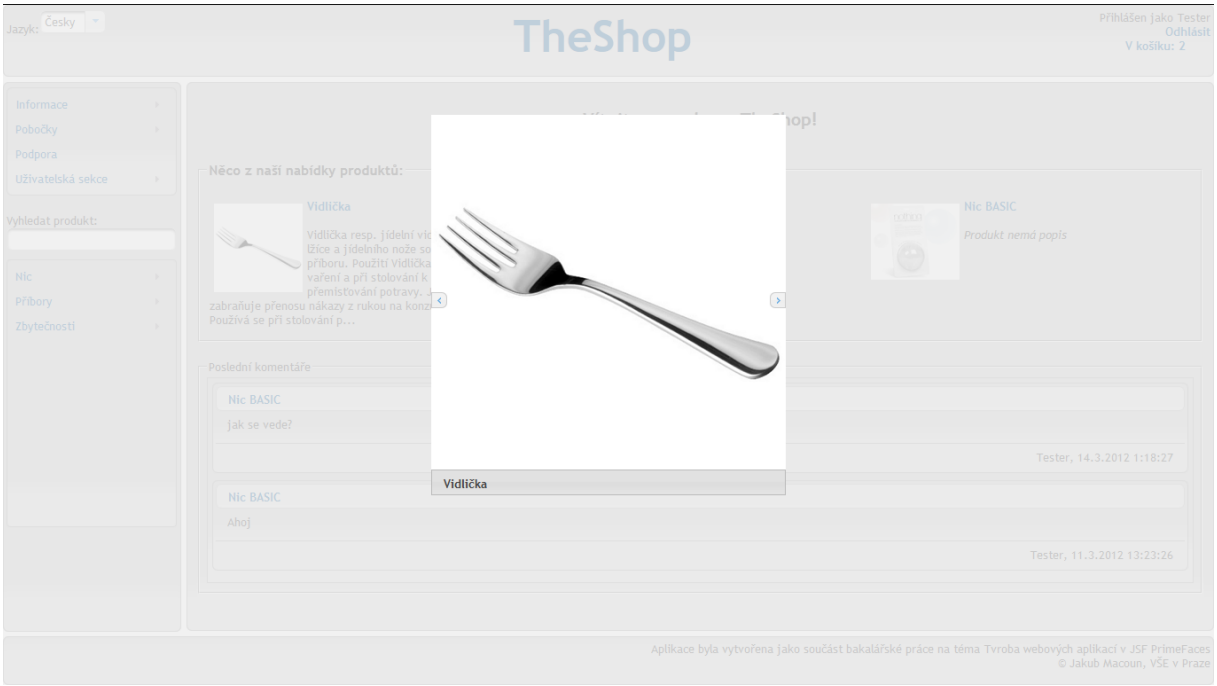
Příloha 11.2.5 – nákupní košík



Příloha 11.2.6 – objednávka



Příloha 11.2.7 – přehled realizovaných objednávek



Příloha 11.2.8 – lightbox na hlavní stránce