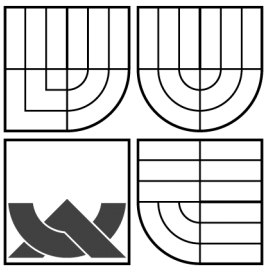
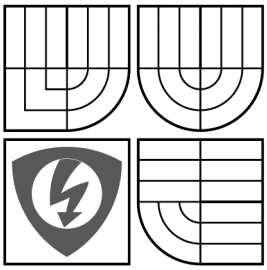


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

GWT A JINÉ MODERNÍ WEBOVÉ FRAMEWORKY

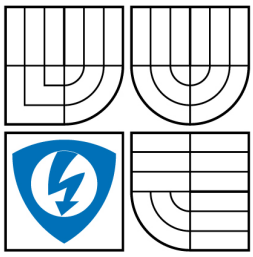
GWT AND OTHER MODERN WEB FRAMEWORKS

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE — TOMÁŠ HILL
AUTHOR

VEDOUCÍ PRÁCE — Ing. TOMÁŠ PELKA
SUPERVISOR

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Bakalářská práce

bakalářský studijní obor
Teleinformatika

Student:	Tomáš Hill	***ID:***	102052
Ročník:	3	***Akademický rok:***	2008/2009

NÁZEV TÉMATU:

GWT a jiné moderní webové frameworky

POKYNY PRO VYPRACOVÁNÍ:

Úkolem bakalářské práce je v prvé řadě analýza, popis a srovnání moderních webový frameforků postavených na technologii Java Script. V druhé části se student zaměří na framework GWT od firmy Google Inc. Úkolem studenta bude návrh webového rozhraní pro odevzdávání rozličných semestrálních projektů.

DOPORUČENÁ LITERATURA:

[1] COOPER, Robert , COLLINS, Charlie. GWT in Practice. 1st edition. [s.l.] : Manning Publications, 2008. 380 s. ISBN 978-1933988290.
[2] GUPTA, Vipul. Accelerated GWT : Building Enterprise Google Web Toolkit Applications. 1st edition. [s.l.] : Apress, 2008. 312 s. ISBN 978-1590599754.
[3] HOLDENER III, Anthony T. . Ajax : The Definitive Guide. 1st edition. [s.l.] : O\'Reilly Media, Inc., 2008. 980 s. ISBN 978-0596528386.
[4] DEWSBURY, Ryan. Google Web Toolkit Applications. 1st edition. [s.l.] : Addison-Wesley Professional, 2007. 608 s. ISBN 978-0321501967.

Termín zadání:	9.2.2009	***Termín odevzdání:***	2.6.2009

Vedoucí práce: Ing. Tomáš Pelka

prof. Ing. Kamil Vrba, CSc.
Předseda oborové rady

UPOZORNĚNÍ:

Abstrakt

Bakalářská práce se zabývá porovnáním moderních webových frameworků. Hlavní částí je popis JavaScriptového frameworku Gogole Web Toolkit a porovnání s dalšími webovými frameworky DoJo a MooTools. Webové frameworky mají za úkol usnadnit vývoj webových aplikací. V druhé části bakalářské práce je realizována webová aplikace ve frameworku Gogole Web Toolkit, která je určena pro správu různých školních projektů.

Klíčová slova

Ajax, Framework, GWT, Java, JavaScript, MVC, RPC, XHR

Abstract

This bachelor thesis deals with the comparison of modern web frameworks. The main aim is to describe JavaScript framework Google Web Toolkit and also compare it with other JavaScripts frameworks such as DoJo or MooTools. The web frameworks are designed to facilitate the development of web applications. In the next part of this thesis is carried out a web application with Google Web Toolkit framework, which is intended to manage various school projects.

Keywords

Ajax, Framework, GWT, Java, JavaScript, MVC, RPC, XHR

HILL, T. GWT a jiné moderní webové frameworky. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2009. 50 s. Vedoucí bakalářské práce Ing. Tomáš Pelka.

Prohlášení

Prohlašuji, že svoji bakalářskou práci na téma „GWT a jiné moderní webové frameworky“ jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.

V Brně dne 27.5.2009 ……

podpis autora

Poděkování

Děkuji vedoucímu bakalářské práce Ing. Tomáši Pelkovi za velmi užitečnou metodickou pomoc, cenné rady a připomínky při zpracování bakalářské práce.

OBSAH

1 Úvod

V této práci se budu zabývat porovnáním nejpoužívanějších webových frameworků především v jazycích Java, JavaScript a PHP, jejich možnostmi a architekturou. Dále tyto frameworky porovnám s Google Web Toolkit frameworkem (zástupce Java/JavaScript frameworků) a v poslední části bakalářské práce bude realizována konkrétní webová aplikace.

V kapitole 2 „Základní pojmy“ je úvod do problematiky webových frameworků, dále zde bude představena technologie Ajax a rozdíly mezi jazyky Java a JavaScript.

V následující kapitole 3 „JavaScript frameworky“ jsou podrobněji rozebrány jednotlivé frameworky jazyka JavaScript.

Kapitola 4 „PHP frameworky“ obsahuje přehled nejrozšířenějších frameworků a také obsahuje porovnání s JavaScript frameworky.

Popis funkcí, vlastností a způsob realizace vybrané webové aplikace, která je v této práci navržena pomocí frameworku Google Web Toolkit, je uveden v kapitole 5 „Návrh webové aplikace“.

Závěr této bakalářské práce je shrnut v kapitole 6 „Závěr“.

2 Základní pojmy

2.1 Návrhové vzory, webové frameworky

Při vývoji nejen webových aplikací se setkáme s problémy, které již byly mnohokrát vyřešeny. Tyto problémy se dají „standardizovat“, vytvořit určitý vzor nebo šablonu řešení daného problému. Tyto šablony, neboli návrhové vzory (někdy označované také jako „stavitelské“ nebo „architektonické“ vzory, viz [1]), byly vyvinuty za účelem udržení jistého standardu a přehlednosti kódu při vývoji ucelených informačních systémů a také k ulehčení samotného vývoje. Webový framework a návrhový vzor spolu úzce souvisí. Návrhový vzor je určitá forma abstrakce, typicky definuje vztahy mezi třídami a objekty, ale bez konkrétní implementace. Webový framework (dále jen framework) je vývojové prostředí, soubor softwarových nástrojů pro vývoj dynamických webových stránek, aplikací a služeb, které jsou konkrétní realizací abstraktního návrhového vzoru. Framework nám ale neposkytne řešení problému, je to jenom nástroj usnadňující dosažení našeho cíle, vyřešení daného problému. Záleží potom na typu frameworku, zda nám bude určovat strukturu řešení a do jaké míry tuto strukturu bude moci programátor ovlivnit [1]. Díky tomu se nemusíme soustředit až tak na technické detaily a můžeme se více věnovat vývoji aplikace po logické stránce.

Tyto sady nástrojů (frameworky) v dnešní době nejčastěji implementují návrhový vzor Model view controller a můžeme je rozdělit podle programovacího jazyka (PHP, Java, JavaScript, Perl, Python, Ruby, Smalltalk, C#, VB.NET atd.). Kromě programovacího jazyka můžeme frameworky ještě rozdělit na tzv. „push-based“ a „pull-based“. Push-based nejprve data zpracuje a potom je vloží do zobrazovací vrstvy k prezentaci výsledků. Naopak v pull-based architektuře si zobrazovací část „vytáhne“ (pull) vypočítané výsledky tak, jak zrovna potřebuje. Další důležité rozdělení frameworků a poté také výsledných aplikací je na „client-side“ a „server-side“. U server-side aplikací jsou prováděny veškeré operace a výpočty na serveru a koncovému uživateli (klientovi) je zobrazen pouze výsledek dané operace. Naopak u client-side aplikací jsou příslušné operace prováděny až na straně uživatele. Příkladem je zpracování JavaScriptového kódu webovým prohlížečem uživatele, které je klasickým zástupcem client-side přístupu.

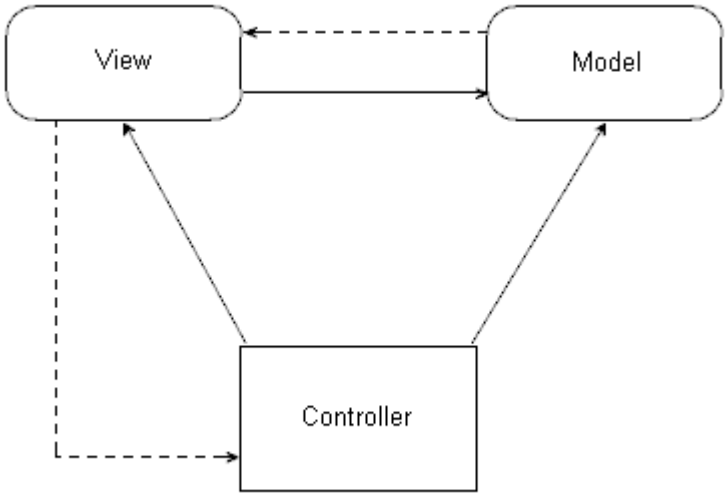
2.1.1 Návrhový vzor Model view controller

Model view controller (MVC) je návrhový vzor, který rozděluje aplikaci na dvě, resp. na tři části. Novější koncept vzoru, označován jako Model-2, rozděluje aplikaci na následující části [1]:

- **Model** – představuje tu část aplikace, která vykonává práci, pracuje s daty – řeší problém a poskytuje řešení. Tato ani jiná část neřeší problém uchovávání dat (např. do databáze)
- **View** (zobrazení) – zajišťuje prezentaci (zobrazení) dat koncovému uživateli, typicky „user interface“ – uživatelské rozhraní
- **Controller** (řadič) – řadič zajišťuje řízení předchozích dvou částí, modelu a zobrazení. Reaguje na pokyny uživatele a rozhoduje jak se má změnit nebo chovat model a co se má zobrazit pomocí zobrazení

Starší koncept vzoru, Model-1, sdružuje část View a Controller do jedné části.

Podstata MVC je jeho hlavní výhodou. Aplikace je rozdělena na tři části, každá část se zabývá určitým typem úkolů. Část model nemusí nic vědět o tom, jak data prezentovat uživateli. Prostě se stará o výpočty, řešení situace nebo problému. Obdobně část View pouze převezme výsledky od části Model a postará se o jejich zobrazení nebo předá Modelu příkaz od uživatele, koordinaci těchto dvou částí zajišťuje Controller. Schéma návrhového vzoru MVC je na obr. 1. 1.



Obr. 1. 1: Schéma návrhového vzoru MVC

2.1.2 Struktura frameworků

Strukturu frameworků můžeme rozdělit na jednotlivé části, většina frameworků obsahuje minimálně tyto části [1]:

- Návrhový vzor – jak již bylo řečeno, většina dnešních frameworků používá návrhový vzor Model view controller Model-2

- Kontrolní systém – každý framework má svůj systém kontroly a upozornění jak v části Model (např. kontrola kódu) tak i v části View (např. uživatel zadá špatné údaje)
- Vícejazyčná podpora – s rozšiřováním internetu vzrůstá potřeba podpory více jazyků – frameworky obsahují podporu pro vícejazyčné dialogy, obrázky atp.
- Podpora šablon – jednotlivé často opakované položky mohou být šablonami – odpadá tak nutnost neustále opakovat ten samý kód (např. menu na stránce, stejné pro všechny stránky)

Toto jsou nejčastěji obsažené části frameworků, nejsou však povinné, každý tvůrce frameworku si může zvolit, které části bude jeho framework obsahovat.

Velmi rozšířenou skupinou jsou tzv. perzistentní neboli stálé frameworky, které nabízejí propojení aplikace s datovým médiem, typicky s databází [1]. Tyto frameworky jsou s datovým médiem abstraktně provázány, aplikace je tak méně závislá na datovém médiu. Tzn. že se teoreticky nemusíme obávat při přechodu na jinou databázi jakýchkoliv závislostí na původní databázi. Tento typ frameworku tak dovoluje programátorovi soustředit se na samotný vývoj aplikace a framework obstará „konverzi“ mezi relačními databázovými prvky a objektově orientovanými prvky aplikace. Příkladem takového frameworku je Enterprise JavaBeans [1].

Další velmi používanou skupinou jsou tzv. „Utility frameworks“ neboli frameworky vyvinuté za jedním účelem. Příkladem je např. framework JUnit pro tvorbu testových aplikací.

2.2 Java vs. JavaScript

Jazyk Java byl vyvinut společností Sun Microsystems a jako součást platformy Microsystems’ Java Platform zveřejněn v roce 1995. Velká část jazyku Java byla odvozena od C a C++, ale na rozdíl od těchto jazyků byl zvolen jednodušší objektový model. Velká výhoda Javy spočívá v nezávislosti na platformě. Aplikace jsou kompilovány do bytového kódu, který je pak spuštěn v tzv. Java Virtual Machine (JVM). Pokud tedy existuje JVM pro požadovanou platformu (Windows, Unix, Mac atd.), lze na ní spustit kteroukoliv Java aplikaci.

JavaScript, navzdory svému názvu, není odvozen od jazyku Java, i když mají podobné syntaxe (jazyk C). JavaScript byl přejmenován ze svého původního názvu LiveScript díky dohodě firem Netscape a Sun Microsystems. JavaScript je v současné době používán nejvíce jako client-side skriptovací jazyk v dynamických webových stránkách.

Ve frameworku Google web toolkit (GWT) programátor používá k vývoji aplikace jazyku Java a GWT ji pomocí vestavěného kompilátoru převede do JavaScriptu, odpadá tak

pro uživatele nutnost mít nainstalován JVM potřebný pro běh Java aplikací a zároveň programátor může využít svých znalostí Javy a nemusí znát dopodrobna jazyk JavaScript. Použitím JavaScriptu jsou tyto moderní webové aplikace dostupnější většímu množství koncových uživatelů, stačí mít pouze zapnutou podporu JavaScriptu ve webovém prohlížeči, která je v současných prohlížečích implicitně zapnuta.

Tab. 2. 1 Srovnání jazyků Java a JavaScript

	Java	JavaScript
Kód	Statický	Dynamický
Základní filosofie	Třídy (class-based)	Prototypy (prototype-based)
Zpracování	Server	Klient

Pro pochopení schopnosti překladu z Javy do JavaScriptu následuje stručný popis filosofie obou jazyků [5].

V jazyce Java jsou vlastnosti objektů definovány svými třídami. Tyto třídy jsou pak nadefinovány při kompilaci na základě zdrojového kódu. Program tak nemůže přidávat za běhu další třídy a objekty ale může používat pouze ty, které byly nadefinovány před kompilací. Z tohoto důvodu se Java řadí mezi statické jazyky s klasickým objektově orientovaným programováním (OOP). Jazyk JavaScript má také podporu OOP, nicméně není to klasické OOP ale prototypově založené programování. Neumožňuje tvorbu klasických tříd tak jako v jazyce Java, místo toho lze definovat objekty a jejich prototypy a to přímo za běhu aplikace (skriptu). Dokonce lze za běhu měnit datové typy existujících objektů. Tyto typy jazyků se nazývají dynamické. Každý objekt má svůj prototyp, což je objekt, od nějž dědí vlastnosti a metody [5].

GWT tak může celkem snadno provést překlad ze statického jazyka do dynamického. Kompilátor provede statickou analýzu kódu a převede jej podle konvencí do jazyka druhého. [5]. Např. pokud budeme převádět třídu z Javy do JavaScriptu, vytvoříme na základě třídy funkci a její konstruktor. Tyto položky a náležící polymorfické metody zahrneme do výsledného prototypu jazyka JavaScript [6]. Opačný převod z jazyka dynamického do statického by nebyl tak jednoduchý a výsledná aplikace by byla omezena, protože nelze jednoduše převést dynamickou tvorbu objektů do statického kódu. Kompilátor by tak musel zohlednit všechny možné stavy chování aplikace napsané v dynamickém jazyce a to by bylo velice náročné.

2.3 Ajax

V posledních letech se na poli moderních webových stránek začínají prosazovat tzv. dynamické aplikace, dynamický obsah, který je označován jako Web 2.0, web druhé generace. Tyto aplikace se také označují jako Rich Internet Applications (RIA). Rozvoj směřem k RIA odstartovala v roce 2005 firma Google, která představila své služby Google Mail a Google Maps umožňující na svou dobu pro web nevídané funkce. Jejich webové aplikace se svým ovládáním a funkcemi podobaly stolním offline aplikacím [2]. Těchto funkcí Google dosáhl nasazením technologie Ajax. Ajax ale není novou technologií, jde pouze o název spojující již existující technologie v celek (Asynchronous JavaScript and XML).

2.3.1 Struktura Ajaxu

Ajax představuje spojení těchto základních částí:

- HTML a CSS
- XML, XMLHttpRequest (XHR)
- JavaScript

Html a Css jsou základními stavebními kameny webových stránek, které není potřeba představovat.

XML je tzv. značkovací jazyk a je určený primárně pro výměnu dat mezi aplikacemi nebo pro uchování dokumentů. Modul XMLHttpRequest umožňuje výměnu dat (nejen ve formátu XML) mezi klientem a serverem.

JavaScript nebyl donedávna příliš používán díky různým implementacím v různých webových prohlížečích. Nebylo tak možné zaručit, že se JavaScriptový kód bude chovat stejně ve všech prohlížečích [2]. Na konci devadesátých let, kdy utichla válka webových prohlížečů, se vlastnosti prohlížečů ustálily a JavaScript se v malé míře začal používat, např. pro kontrolu údajů ve formuláři před odesláním dat na server. Své největší slávy se dočkal až ve spojení s XML, resp. s modulem XMLHttpRequest (XHR).

Ajax odstranil nutnost načítat celou webovou stránku při sebemenší změně. XMLHttpRequest zajišťuje výměnu dat na pozadí a JavaScript obstará překreslení pouze té části stránky, u které je to nutné. Tato základní vlastnost Ajaxu může být využita tak, že se aplikace skutečně chovají jako stolní offline aplikace. Jednoduchým příkladem může být hlasování v anketě, kdy se data o hlasování odešlou a následně se uživateli zobrazí aktuální stav ankety po jeho hlasování, bez nutnosti načítat mnohdy několik set kB velkou stránku.

V současné době jsou ale již k dispozici propracované nástroje a služby, které mohou částečně konkurovat klasickým stolním aplikacím, jako např. Google Docs (online alternativa kancelářských balíků Microsoft Office nebo OpenOffice.org).

3 JavaScript frameworky

V tabulce 3. 1 je uveden přehled frameworků, které budou v této kapitole následně popsány.

Tab. 3.1 Přehled JavaScript frameworků

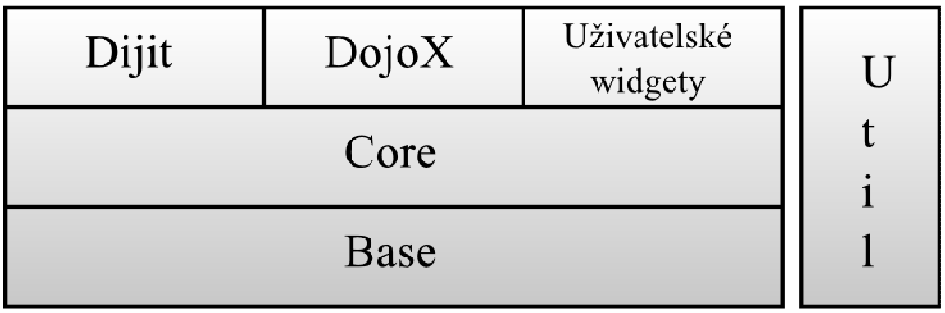
	DoJo	GWT	MooTools
Architektura	Base, Core, Util, Dijit	Kompilátor Java->JavaScript, Mód zobrazení, Emulace JRE, GWT Web UI	Core (Core, Native…) More (Fx, Utilities…)
Licence	AFL / BSD	Apache 2.0	MIT
Vývoj od r.	2004	2006	2006
Rozšiřitelnost (widgety)	Ano	Ano	Ano
Specifika	Možnost výběru licence Široké spektrum možností	Programování v Java, výsledná aplikace je pak zkompilována do JavaScriptu	Možnost sestavit si vlastní jádro a redukovat tak velikost frameworku
Použití	Svou rozsáhlostí je určen spíše zkušenějším programátorům k realizaci větších projektů	Základní možnosti - vhodné pro začátečníky, dobře zpracovaná online dokumentace včetně praktických příkladů	Zaměřen na OOP -> vhodné pro programátory zvyklé programovat v OOP

3.1 DoJo

3.1.1 Struktura DoJo

Dojo toolkit se skládá z následujících částí (obr. 3. 1) [3]:

- Base (jádro, základ)
- Core (jádro)
- Dijit, DojoX, uživatelské widgety
- Util



Obr. 3. 1 Struktura DoJo toolkit

Base

Základem DoJo je Base. Base je kompaktní, vysoce optimalizovaná knihovna, která tvoří základ pro všechny další části tohoto toolkitu. Vše z knihovny Base je dostupné na nejvyšší úrovni toolkitu jako dojo.*, jinak Base knihovna je zabalená jako dojo.js a má velikost cca. 30kB [3]. Knihovna je navržena tak, že k načtení základů DoJo dojde pouhým vložením souboru dojo.js do stránky. Tzn. pouhým načtením tohoto souboru je detekováno prostředí, vyrovnají se rozdíly mezi prohlížeči a načte se jmenný prostor DoJo. Volitelnou možností je pak ještě automatická analýza widgetů a další automatické zpracování inicializačních úkonů.

Widget nese název pro již připravené části stránek, které se pouze vloží do stránek. Většinou jejich kód není třeba nijak upravovat.

Core

Core by se dalo nazvat nástavbou Base. Jádro obsahuje další možnosti pro analýzu (schvalování) widgetů, pokročilé animační efekty, správu cookies, podporu více jazyků, ošetření problému s tlačítkem „zpět“ (viz. 3. 1. 2 Možnosti Dojo) atd. Tyto a všechny další možnosti nejsou natolik abstraktní, aby mohly být zařazeny do Base. Tzn. všechno co není obsaženo v Base, je součástí Core a to jmenného prostoru dojo.fx nebo dojo.data.

Dijit

Dijit (nebo také Dojo Widget) je soubor widgetů, hotových částí stránek, které ve většině případů nemusí být upravovány ani kouskem JavaScriptového kódu. Widgety můžeme rozdělit do kategorií, jako aplikační (všeobecné), návrhové, formulářové a uživatelské. Pokud tedy programátor navrhuje vlastní widget a bude se držet standardů a konvencí pro Dijit, může se tento widget použít ve kterémkoliv jiném DoJo projektu, samozřejmě pokud v něm najde uplatnění.

DojoX

V této části se nachází všechny widgety, které jsou ve fázi vývoje a mohou tak být značně nestabilní. Stejně tak se zde mohou nacházet dlouho dobu poměrně stabilní a užitečné widgety. U každého widgetu je přiložen soubor readme, kde je uvedeno v jaké fázi vývoje widget je a příp. i verze widgetu.

Util

Util je samostatná jednotka a má za úkol testování a „kompilování“ JavaScriptového kódu. Toto „kompilování“ spočívá v kompresi nástrojem ShrinkSafe a optimalizaci kódu do vrstev, což není nic jiného než generování JavaScriptových souborů a vrstvení do adresářů. Výsledkem těchto operací pak je optimalizovaný a menší kód.

3.1.2 Možnosti DoJo

DoJo nabízí vylepšené nástroje pro manipulaci s DOM objekty. DOM je zkratka pro Document Object Model a je to specifikace pro uspořádání objektů na webové stránce. Samotný JavaScript obsahuje základní možnosti pro operace s DOM objekty a DoJo tyto možnosti rozšiřuje a zjednodušuje je dalšími příkazy, namátkou dojo.setSelectable umožňuje zakázat označování a kopírování textu ze stránky; dále pak nástroje pro změny atributů DOM objektů-změna rámečku atd.

DoJo sice obsahuje mechanismy pro vyrovnání rozdílů mezi prohlížeči, ale pokud bychom potřebovali z jakéhokoliv důvodu detekovat určitý prohlížeč, DoJo nabízí na toto další sadu nástrojů (dojo.isOpera, dojo.isMozzila, dojo.isIE atd.). Nechybí podpora pro ukládání a manipulaci s cookies (dojo.cookie).

Dynamicky měnící se stránky pomocí JavaScriptu nemají ze strany prohlížečů tzv. podporu tlačítka zpět. Tzn. pokud je na stránce provedena nějaká akce pomocí JavaScriptu (např. zobrazení dodatečných informací k produktu), není tato změna prohlížečem detekována a při použití tlačítka zpět se vrátíme na předchozí stránku místo aby se „vrátila“ JavaScriptová operace. DoJo nabízí řešení v podobě sady nástrojů dojo.back.functions, kterými lze toto ošetřit. Použití tohoto ošetření ale není vždy vhodné a vždy záleží na typu obsahu stránky.

Ajax

Technologie AJAX implementovaná v toolkitu DoJo je připravena k používání s JSON (JavaScript Object Notation), v podstatě se jedná o převod mezi JavaScriptovými objekty a textovými řetězci (dojo.fromJson, dojo.toJson). JSON se v současné době stává nepsaným standardem a alternativou k XML pro výměnu dat v Ajaxových aplikacích [3].

DoJo používá speciální třídu Deferreds která zjednodušuje operace spojené s asynchronní komunikací. Deferreds umožňuje spojovat více volání a ošetření chyb, umožňuje také jednoduše celý proces komunikace zrušit. Deferreds tak zavádí srozumitelnou formu požadavků a odpovědí při asynchronní komunikaci [3].

DoJo podporuje také IFRAME přenos, což je obdoba XHR. XHR je navržen pro přenos drobných dat na pozadí a nehodí se tak pro stahování/nahrávání souborů z/na server. Toto řeší IFRAME – nabízí přenos souborů nebo velkých objemů dat na pozadí.

Další možnosti

DoJo toolkit disponuje také modulem pro podporu více jazyků. Pokud tedy autor bude chtít prezentovat aplikaci ve více jazycích, modul i18n (zkratka pro Internationalization) mu zajistí dobré prostředí pro realizaci. Jednotlivé jazykové překlady jsou roztříděny v adresáři nls (Native Language Support), který je umístěný v daném modulu, který chceme opatřit více

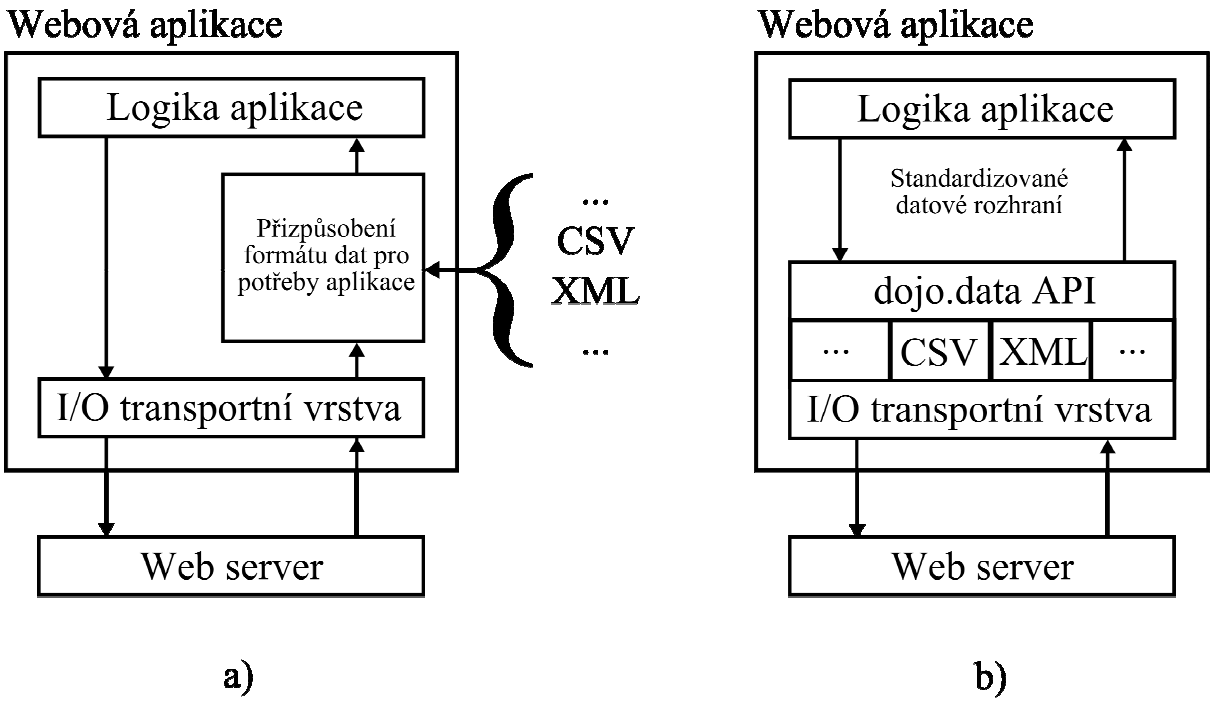
jazyky. Jazyky jsou pak rozděleny dále podle zkratek jazyků dle dokumentu RFC 3066 (en – angličtina, cs – čeština atd.). Během zavádění aplikace se dojo.locale dotáže prohlížeče na jazyk a podle toho pak zvolí příslušný překlad. Pokud daný jazyk nenalezne, zvolí výchozí překlad (umístěný v kořenu adresáře nls). V souvislosti s používáním více jazyků DoJo také nabízí nástroje pro podporu různých národních měn, formátů dat a čísel (moduly dojo.currency, dojo.date a dojo.number).

Nedílnou součástí moderních webových aplikací je také podpora drag-and-drop a příblížení se tak k desktopovým aplikacím. Tuto možnost DoJo zaštiťuje souborem nástrojů dojo.dnd.*, pomocí tohoto souboru nástrojů a kaskádových stylů (CSS) může programátor definovat objekty a případně omezovat jejich možnosti „pohybu“ (drag-and-drop) po obrazovce.

Další schopností DoJo je vytvářet jednoduché animace. Základní animační schopnosti obsahuje již Jádro (Base) a tytou jsou rozšířeny pomocí dojo.fx (Core). Animaci pomocí Jádra zajišťuje třída _Animation, která mění neustále parametry objektu a tím dochází k animaci (dojo.fadeIn, dojo.fadeOut, dojo.animateProperty). Rozšiřující efekty dojo.fx jsou Slide (klouzání) a Wipe (stírání).

Přístup k datům

Tradiční způsob přístupu k datům vyžadoval zpracování těchto dat do podoby vhodné k dalšímu zpracování vlastní aplikací (např. různé vstupní datové typy atd.). DoJo přináší řešení v podobě vrstvy mezi logikou aplikace a formátem dat (viz. obr. 3. 2). Dojo.data modul poskytuje standardizované prostředky pro přístup k datům a jeho sada API (Application programming interface) poskytuje k těmto datům přístup stejně, ať již jsou vzdálená nebo místní. Programátor se tak nemusí zabývat způsobem provedení stahováním dat, nahráváním dat na server nebo udržování spojení se serverem [3].



Obr. 3. 2 a) tradiční způsob práce s daty; b) datová abstrakce pomocí dojo.data API

Simulace tříd a dědičnosti

JavaScript má základní možnost vytváření tříd, resp. má možnost vytvářet konstruktorem new libovolný počet instancí daného vzoru (viz. 2. 2 Java vs. JavaScript). Tato vlastnost byla využita pro simulaci tříd. Dojo.declare je funkce pro simulaci tříd a hierarchii dědičnosti.

Dijit – widgety

Dijit nabízí programátorovi sadu hotových řešení, která nevyžadují žádné nebo jen minimální programování. Největší výhodou Dijit je zapouzdření komponentů uživatelského rozhraní do samostatných widgetů - z toho plyne minimální nutnost programování. Další výhodou je pokud programátor navrhne widget dle standardů, může se poté použít i v dalších projektech. Widgety jsou v DoJo děleny do několika kategorií:

- Formulářové – widgety používané ve spojení s formuláři – různá tlačítka, textboxy, posuvníky, comboboxy atd.
- Návrhové – nahrazují složité CSS programování pro vytvoření komplexního návrhu stránky
- Aplikační – neboli všechny ostatní widgety; sem patří různá menu, toolbary, bublinové nápovědy, průběhové lišty (progress bar) atd.

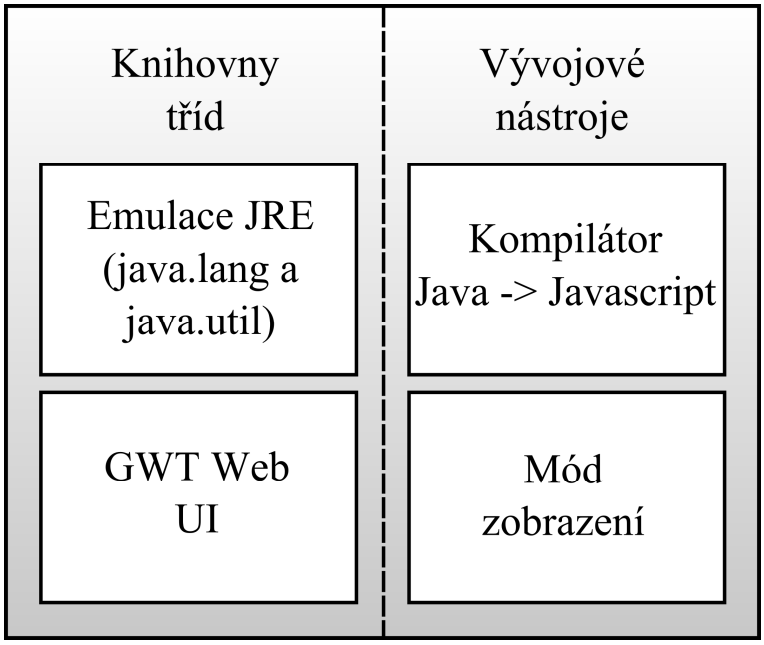
3.2 Google Web Toolkit

GWT je dalším zástupcem JavaScript frameworků. Jeho hlavní odlišnost od ostatních JavaScriptových frameworků je, že návrh aplikace je realizován v jazyce Java a GWT následně kód převede pomocí vestavěného kompilátoru do jazyka JavaScript, který je nativně podporován všemi moderními webovými prohlížeči.

3.2.1 Struktura GWT

GWT se skládá z částí (viz. obr. 3. 3)

- Knihovny tříd
  - Emulace Java platformy (JRE)
  - Knihovny pro tvorbu uživatelského rozhraní (GWT Web UI)
- Vývojové nástroje
  - Kompilátor z Javy do JavaScriptu
  - Mód zobrazení



Obr. 3. 3 Struktura GWT

Kompilátor z Java do JavaScript

Tento kompilátor (com.google.gwt.dev.GWTCompiler) zajišťuje převod z Java do JavaScriptu, podobně jako Java kompilátor kompiluje zdrojový kód do bytového kódu. GWT kompilátor ale pracuje trochu odlišně od klasických Java kompilátorů. GWT kompilátor kompiluje pouze to, co se v daném modulu (aplikaci) používá. To umožňuje při programování

používat rozsáhlé knihovny a kompilátor do výsledného kódu zahrne pouze ty třídy a metody, které jsou skutečně používány a nepoužívané funkce tak zbytečně nezabírají místo [4].

Kompilátor poskytuje tři druhy překladu. Ačkoliv nejsou pojmenovány, daly by se tyto druhy překladu nazvat jako matoucí, klasický a podrobný. Výchozím překladem je matoucí styl; kompilátor vyprodukuje kód značně nečitelný pro programátora, cílem matoucího stylu je co nejmenší kód a používá se až pro finální aplikaci. Klasický překlad je mnohem čitelnější a podrobný styl překladu rozšiřuje klasický překlad o plné názvy tříd. Tyto plné názvy tříd jsou pak součástí názvů JavaScriptových metod [4], to umožňuje snadné dohledání části kódu napsaného v Javě zpětně z již přeloženého JavaScriptu. Tyto dva módy se tak používají především při vývoji aplikace.

Kompilací také vzniknou čtyři výstupy, každý pro jednu z podporovaných platforem (Internet Explorer, Opera, Firefox a Safari) [12]. Při načítání se ze serveru stáhne malý soubor, který určí o jakou platformu se jedná a následně se načte stránka pouze pro danou platformu a ušetří se tak čas nutný pro načtení aplikace, než kdyby se načítaly všechny verze aplikace a použila se jenom jedna.

Emulace JRE

Tím že výsledná aplikace neběží v prostředí JVM nebo JRE (Java Runtime Environment - rozšíření JVM o další knihovny) ale je přeložena do JavaScriptu a je spouštěna v prostředí webového prohlížeče, vyvstává logicky nutnost zahrnout některé základní JRE třídy do výsledného JavaScriptového kódu. Tento problém řeší emulace prostředí JRE, resp. základních tříd a metod z balíčků java.lang a java.util, které jsou pak zakomponovány do výsledného JavaScriptového kódu.

Mód zobrazení – Hosted a Web

Hosted mode je používán výhradně při vývoji aplikaci v prostředí GWT, koncept aplikace se zobrazí ve speciálním prohlížeči přímo v jazyce Java (tzn. v Java Virtual Machine), nedochází tedy ke kompilaci kódu do JavaScriptu a programátor má možnost průběžně kontrolovat výsledky své práce bez nutnosti kompilace.

Naopak pro konečnou podobu aplikace je určen mód Web, kdy je kód zkompilován do JavaScriptu a následně otevřen v klasickém webovém prohlížeči.

GWT Web UI

Nástroje pro tvorbu uživatelského rozhraní (UI – User Interface) jsou rozděleny do dvou kategorií, widgety a panely. Widgety jsou ovládací prvky určené ke komunikaci s uživatelem

(různé textová pole pro psaní textu, tlačítka atp.). Panely jsou tzv. kontejnery, do kterých jsou widgety umístěny. Tyto kontejnery určují rozmístění ovládacích prvků na stránce.

GWT poskytuje základní soubor asi třiceti widgetů a panelů určených k tvorbě stránek. Mimo tento soubor je možné také importovat widgety a panely vytvořené jinými programátory, které jsou volně dostupné na internetu, např. kalendáře, kalkulačky, panely pro kreslení atd. [4] To výrazně usnadňuje práci při tvorbě stránek a naplňuje definici frameworků – nedělat znovu to, na co již existuje řešení.

3.2.2 Komunikace se serverem

Základní potřebou AJAX aplikace je komunikace se serverem. GWT nabízí dvě možnosti výměny dat se serverem, Remote Procedure Calls (vzdálená volání procedury) a klasické požadavky zajišťované třídou RequestBuilder [4].

RequestBuilder

Třída RequestBuilder je v podstatě jenom zapouzdření objektu XMLHttpRequest (XHR) pro práci s ním v jazyce Java. Odpovědí serveru na dotaz ze třídy RequestBuilder je vždy čistý text ve formě XML, HTML, JSON nebo prostého textu.

Remote Procedure Calls (RPC)

Jelikož RPC je obecné označení pro vzdálené volání procedur, bylo označeno používání RPC v souvislosti s GWT zkratkou GWT-RPC. GWT-RPC je propracovaný nástroj pro přenos Java objektů mezi klientem a serverem. GWT-RPC zajišťuje veškeré nutné kroky pro přenos objektů a odstínění od struktury HTTP, XHR a také serializací a deserializací objektů [4]. Serializací se rozumí převedení objektu Java do kódu vhodného pro přenos, analogicky deserializace značí opačný proces.

3.2.3 Další možnosti

Pro mnohé aplikace může být nepostradatelná potřeba lokalizace, GWT řeší lokalizace aplikace dvojím způsobem, statickým a dynamickým. Statická lokalizace se provádí při kompilaci aplikace pomocí konstant a zpráv. Konstanty jsou stálé fráze, které se nemění. Zprávy na rozdíl od konstant obsahují parametr, kam se podle výskytu zprávy doplní odpovídající text. Dynamická lokalizace je řešena pomocí slovníku, kdy jednotlivé fráze překladu jsou vkládány do stránek dynamicky. To velmi zatěžuje server a používá se vyjímečně. Dynamický překlad se použije, pokud programátor nechce nebo nemůže implementovat statické metody překladu do již existující aplikace.

Jako většina konkurenčních frameworků GWT také obsahuje podporu historie, resp. ošetření tlačítka zpět (viz. 3. 1. 2). Dále integruje JUnit, což je mezi programátory oblíbený nástroj pro testování kódu.

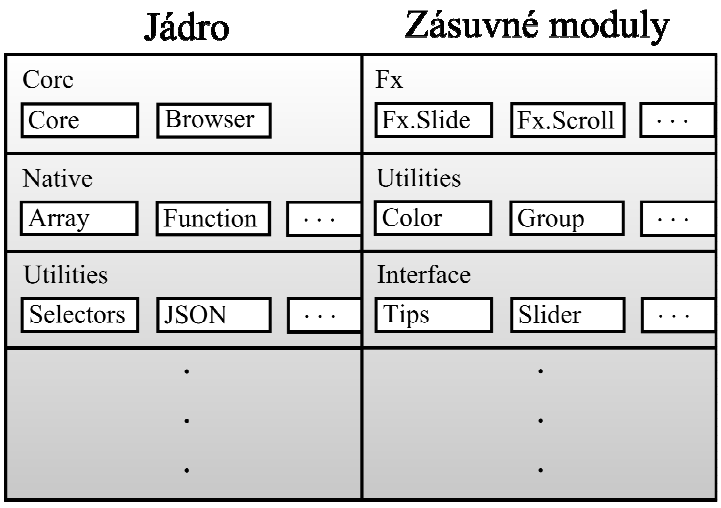
Specifickým nástrojem v GWT je JavaScript Native Interface (JSNI). Tento nástroj umožňuje volat JavaScriptový kód přímo z Javy, což programátor využije zejména tehdy, pokud bude chtít použít cizí JavaScriptový kód ve své aplikaci a nemá k dispozici tento kód v jazyce Java [4].

3.3 MooTools

Framework MooTools vznikl na základě knihovny pro jiný JavaScriptový framework - Prototype. Programátor této knihovny pak po roce vývoje vydal v roce 2006 vlastní framework, Mootools [7]. MooTools se v mnoha směrech inspiroval od Prototype, ale v mnoha směrech je odlišný. Hlavní odlišností a také důvodem vývoje nového frameworku bylo, že Prototype nijak neusnadňoval a nerozšiřoval JavaScriptový prototyp Element (na rozdíl např. od typu Array nebo Function) [7], takže byl programátor frameworku Prototype nucen používat standardní syntaxe, od kterých se chtěl používáním frameworku odprostit. Naopak stejná je filosofie obou frameworků, oba jsou objektově orientovanými frameworky a zaměřují se na pozměňování původních prototypů. To s sebou přináší hlavně vytvoření nových metod pro práci s těmito prototypy.

3.3.1 Struktura MooTools

MooTools se dělí na dvě hlavní části, jádro (Core) a zásuvné moduly (More). Obě tyto hlavní části dále obsahují podskupiny a ty potom už jednotlivé komponenty (viz. [7], [8]).



Obr. 3.5 Struktura MooTools

Jádro obsahuje základní funkce a implementace nutné pro běh samotného frameworku a některé volitelné části (např. JSON modul, Fx moduly, Ajax moduly nebo moduly pro manipulaci s CSS). Zásuvné moduly rozšiřují možnosti frameworku např. o pokročilé animační efekty nebo o další prvky stránky (Slider, Scroller atd., viz. [8])

Jednou z předností MooTools je jeho modulárnost. Přímo na webových stránkách frameworku [8] si každý může sestavit jádro i zásuvné moduly podle svých potřeb. Každý si tak sestaví tyto dvě hlavní části podle potřeb aplikace a redukuje tak velikost výsledného frameworku.

3.4 Ostatní frameworky a knihovny

V této podkapitole se nachází přehled ostatních známých JavaScriptových frameworků. Tato kapitola nemá za cíl přinést konečný výčet JavaScriptových frameworků, ale zmínění těch známých a používaných frameworků, které nebyly podrobněji probrány výše.

Prototype

Framework Prototype má podobně jako DoJo široké spektrum možností a někdy je také nazýván jako nástavba JavaScriptu [9]. To dokazují mnohé knihovny které jsou nástavbami frameworku Prototype (např. již zmíněná knihovna Moo).

Script.aculo.us

Script.aculo.us je knihovna, která je nástavbou frameworku Prototype. Tato knihovna rozšiřuje Prototype o další možnosti, v prvé řadě však o vizuální efekty jako jsou různé přechody, animace atp.

Echo

Echo je dalším JavaScriptovým frameworkem pro vývoj RIA. Echo nabízí tři verze frameworků: Echo1 je původní verze a již není vyvíjena a není doporučena k používá. Echo2 je v současnosti hlavní verze frameworku určená k běžnému použití a oproti Echo1 přináší nové možnosti. V době psaní této práce je k dispozici beta verze frameworku Echo3, v současnosti určena k testování nových možností a výhod nové verze [10].

3.5 Shrnutí

Webové frameworky jsou zcela závislé na programovacím jazyku, proto se nedá říct že jeden JavaScript framework umí něco, co jiný JavaScript framework neumí. Rozdíly jsou spíše ve filozofii daného frameworku, tzn. jak danou konkrétní věc implementovat a usnadnit programátorům práci. Kritéria výběru frameworku v rámci jednoho programovacího jazyka by se daly shrnout do těchto bodů:

- Požadované možnosti – existují frameworky „vše v jednom“, tzn. pokrývají celou řadu oblastí, pro které jsou aplikace určeny (animace, drag-and-drop, práce s databázemi atd.) a frameworky, které jsou zaměřeny pouze na úzkou cílovou skupinu uživatelů
- Komunita uživatelů – silná základna uživatelů frameworku usnadňuje vývoj aplikace, uživatelé se setkávají na vývojářských fórech a vyměňují si své zkušenosti a poznatky
- Dokumentace – kvalitní dokumentace je základem pro poznání frameworku a užitečným pomocníkem při řešení potíží
- Licenční podmínky – používáním frameworku nesmějí být porušeny jeho licenční podmínky, proto jsou také kritériem výběru frameworku

4 PHP frameworky

Tak jako JavaScriptové nebo Java frameworky mají i PHP frameworky za cíl ulehčit programátorům rutinní práci. Je těžké porovnat vzájemně JavaScriptové a PHP frameworky, neboť tyto frameworky jsou postaveny na zcela odlišných programovacích jazycích. Zatímco JavaScript je jazyk klientský (výpočty jsou prováděny na straně klienta), PHP je jazyk serverový (kód je spouštěn na straně serveru a klient dostane jenom výsledek celé akce). U jazyka PHP jsou tedy kladeny vysoké nároky na server, u JavaScriptu se výpočetní výkon rozloží na klienty. Výběr technologie pro vývoj RIA aplikací je tak nejvíce postaven na zkušenostech programátora s jednotlivými technologiemi a na schopnostech serveru implementovat danou technologii. Tato kapitola tak obsahuje shrnutí nejznámějších PHP frameworků a nastíní jejich možnosti.

4.1 CakePHP

CakePHP framework je vyvíjen od roku 2005 a nabízen pod MIT licencí. Mezi jeho základní vlastnosti patří podpora MVC architektury, obsahuje různé „Pomocníky“ (Helpers) [11] pro práci s Ajaxem, JavaScriptem a Html, podporuje přístupové seznamy (Access Control Lists), různé bezpečnostní nástroje, sadu nástrojů CRUD pro práci s databázemi (CRUD – create, read, update, delete) atd. CakePHP podporuje verze jazyka PHP 4 i 5.

4.2 Zend framework

Zend framework je nabízen pod BSD licencí a je vyvíjen od roku 2005. Na rozdíl od CakePHP Zend podporuje pouze PHP verze 5. Podobně jako CakePHP podporuje MVC architekturu, přístupové seznamy, pomocníky, lokalizaci (i18n) a obsahuje další nástroje a možnosti.

4.3 Srovnání frameworků

Oba výše zmíněné frameworky jsou v současné době nejspíše nejznámějšími a největšími PHP frameworky, oba obsahují široké spektrum nástrojů a možností, liší se pouze v detailech a samozřejmě v použité syntaxi. Oba frameworky mají širokou komunitní základnu, dobře zpracovanou dokumentaci s příklady použití.

5 Návrh webové aplikace

Pro realizaci webové aplikace ve frameworku GWT byl zvolen informační systém pro správu projektů. Jedná se o systém pro zadávání, odevzdávání a hodnocení projektů.

Informační systém zahrnuje tři základní rozhraní pro komunikaci s uživatelem: rozhraní student, učitel a administrátor. Každé rozhraní má svoje specifické menu a každé z rolí jsou tak přístupné pouze některé funkcionality systému v závislosti na dané roli (např. student nemůže zadávat projekty atp.). Přístup do aplikace je zabezpečen na základě uživatelského jména a hesla. Správu uživatelů a jejich rolí zajišťuje role administrátor.

Funkce dostupné všem rolím:

- Změna modulu – pokud má přihlášený uživatel přiřazeno více rolí, může mezi těmito rolemi libovolně přepínat a využívat tak funkce dostupné dané roli
- Změna hesla

Funkce dostupné roli student:

- Zobrazení přidělených projektů a jejich detailů danému uživateli
- Nahrávání, stahování a mazání uživatelových souborů náležících k danému projektu
- Komunikace se zadavatelem (učitelem) daného projektu pomocí komentářů
- Zobrazení doporučené literatury k danému projektu
- Odevzdání projektu
- Opětovné odevzdání projektu v případě že bude zadavatelem vrácen k přepracování
- Zobrazení hodnocení projektu

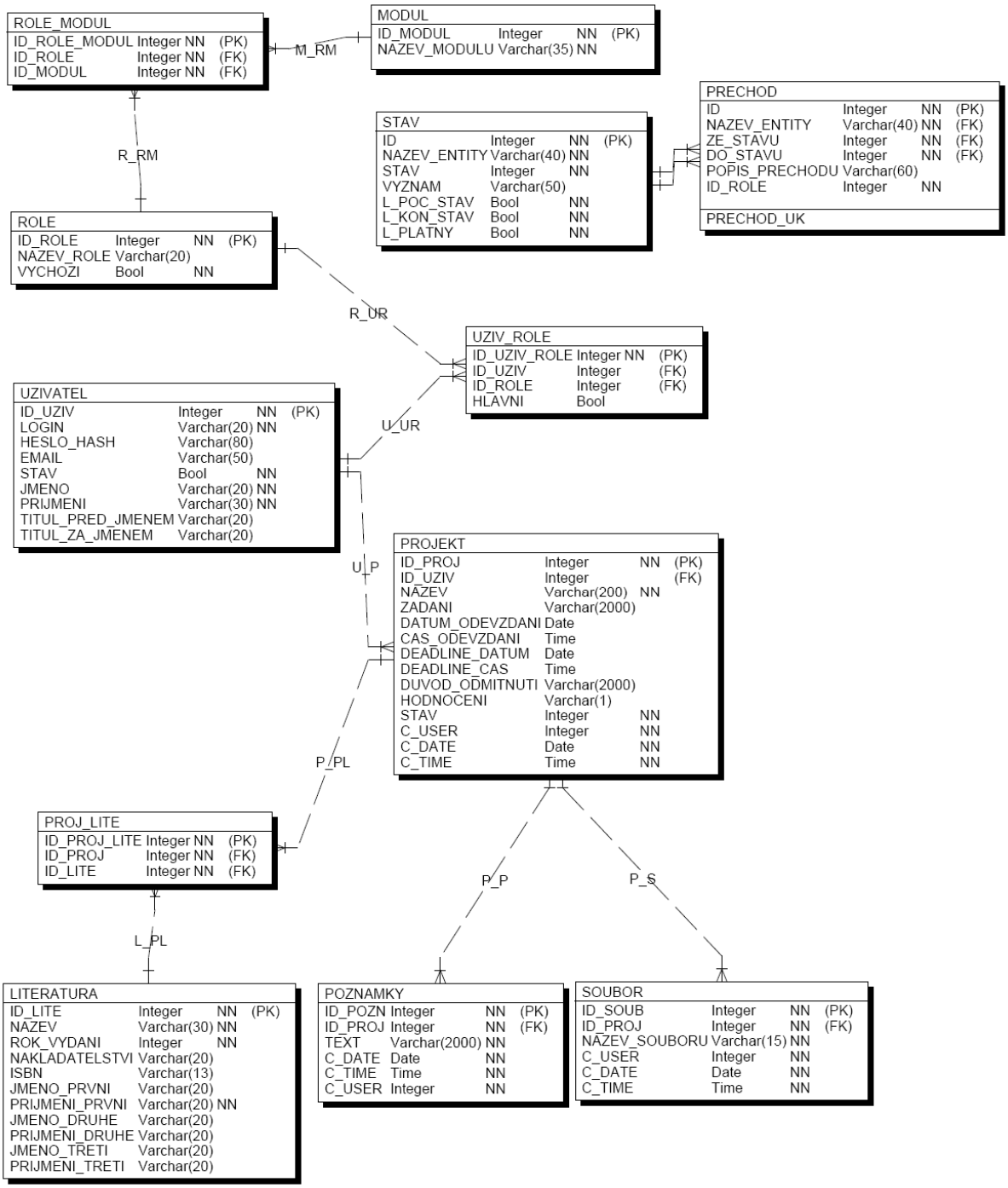
Funkce dostupné roli učitel:

- Zadávání projektů, jejich editaci a přidělení konkrétnímu uživateli (studentovi)
- Správa literatury – vkládání nové literatury a editace literatury uložené v systému
- Komunikace se studenty pomocí komentářů u projektu
- Stahování souborů nahraných k projektům
- Hodnocení projektů
- Vrácení projektů k přepracování

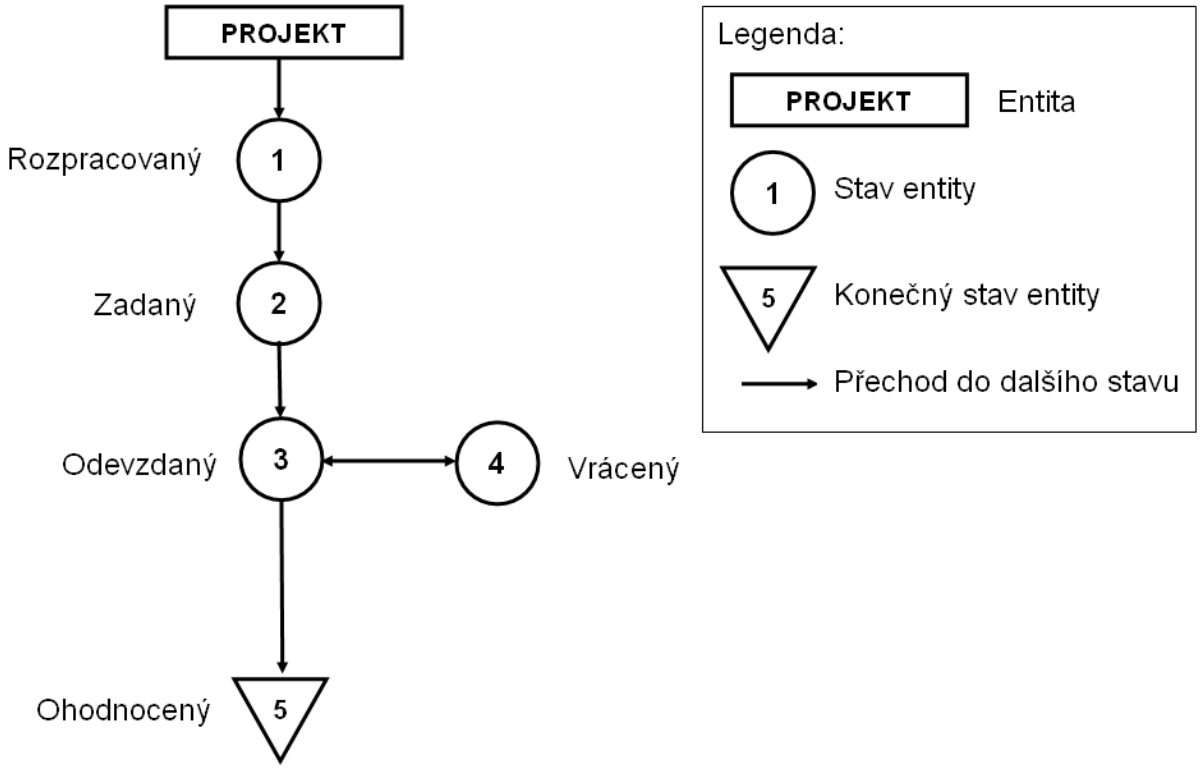
- Hromadné ohodnocení projektů známkou nedostatečně pokud nebyly odevzdány v řádném termínu

5.1 E-R model, stavový model

Na obr. 5.1 je znázorněný E-R model, jenž definuje strukturu dat v databázi včetně vyznačených datových typů jednotlivých atributů, primárních a cizích klíčů a relací mezi entitami. Na dalším obr. 5.2 je znázorněný stavový model hlavní entity PROJEKT, který definuje všechny možné stavy a přechody mezi stavy entity PROJEKT.



Obr. 5.1 E-R model



Obr. 5.2 Stavový model entity PROJEKT

Hlavní entitou v E-R modelu je entita PROJEKT se kterou jsou vazbami 1:N propojeny entity POZNAMKY, SOUBOR a PROJ_LITE. Entita PROJ_LITE je tabulka použitá z důvodu normalizace a nahrazuje vazbu M:N mezi entitami PROJEKT a LITERATURA. Obdobným způsobem jsou řešeny vazby M:N v celém E-R modelu. Entity UZIVATEL, ROLE, MODUL a vazební entity UZIV_ROLE a ROLE_MODUL zajišťují ochranu před neoprávněným použitím funkcí v servletu. Např. role student nemá povoleno hodnotit projekty, proto kdyby se o toto nějakým způsobem pokusil uživatel s právy studenta, nebude mu akce povolena. Obdobně je ochráněn přechod stavů entit z jednoho stavu do druhého pomocí entit STAV a PRECHOD. Proto není možné provést přechod ze stavu 1 – Rozpracovaný do stavu 3 – Odevzdaný. Tento záznam v databázi není a aplikace přechod nepovolí. Stejně tak se kontroluje zda má uživatel přidělena práva pro daný přechod.

Jak je patrné ze stavového modelu, učitel nejdříve projekt připraví, uloží a poté jej zadá k vypracování studentovi. Před přepnutím stavu do „Zadaný“ je ještě možné projekt jakkoliv upravovat, pokud je projekt zadaný, nelze vlastnosti projektu již upravovat. Student projekt vypracuje, nahraje soubory na server a projekt odevzdá. Poté již učitel může projekt ohodnotit nebo může projekt vrátit studentovi k přepracování.

5.2 Programové vybavení

Pro vývoj webové aplikace pro správu projektů byly použity programy a komponenty uvedené v tab. 5.1

Tab. 5.1 Programové vybavení

Program	Verze	Použití
Eclipse SDK	3.4.1	Vývoj aplikace
Google Web Toolkit	1.5.3	Vývoj aplikace
MySQL	5.0.51a-24	Databázový server
MySQL Connector/J	5.1.7	Propojení aplikace a databázového serveru
Java SE	1.6.0_12-b04	Vývoj aplikace
Apache TomCat	6.0.18	Webový server s podporou Javy
commons fileupload	1.2.1	Java knihovna pro podporu nahrávání souborů na server
commons io	1.4	Java knihovna pro podporu operace se soubory
gwtswfext	1.0	Zapouzdření JavaScript knihovny SWFUpload do prostředí Java
SWFUpload	1.0	JavaScript knihovna pro podporu nahrávání souborů na server v klientské aplikaci (využívá technologii Adobe Flash)
jBCrypt	0.2	Hash hesla

5.3 Struktura projektu

Při vývoji aplikace je nutné dodržet určitou strukturu souborů [13], aby bylo možné aplikaci zkompilovat a používat v prohlížeči. Pomocí aplikace `applicationCreator`, která je součástí balíku GWT, lze přesně tuto strukturu vytvořit pomocí jednoduchého příkazu, např. pro vývojové prostředí Eclipse:

```
applicationCreator -eclipse MyApplication -out MyApplication \
 cz.myapplication.client.MyApplication
```

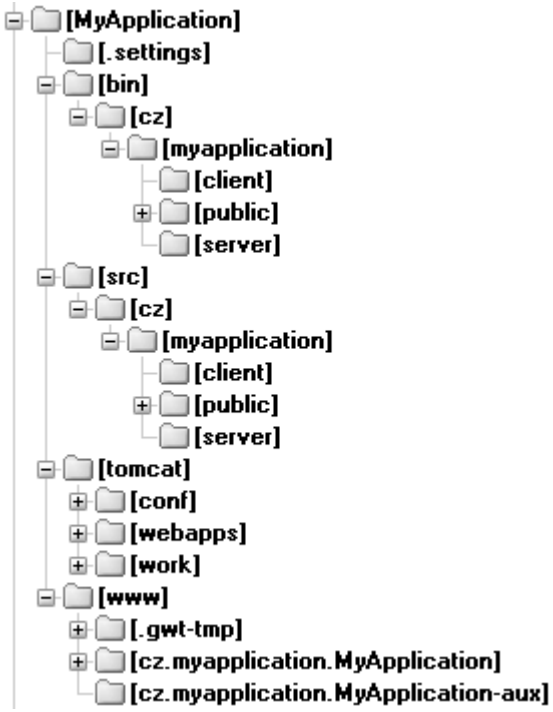
Pro vývoj aplikace lze použít kterékoliv Java IDE, potom je příkaz pro vytvoření souborové struktury odlišný:

```
applicationCreator -out MyApplication \
cz.myapplication.client.MyApplication
```

Pro vývojové prostředí Eclipse je navíc připraven generátor projektu do tohoto vývojového prostředí. Projekt pak lze v IDE Eclipse jednoduše importovat:

```
projectCreator -eclipse MyApplication -out MyApplication
```

Po provedení příkazů je vytvořena struktura souborů a adresářů, viz. Obr. 5.3



Obr. 5.3 Struktura projektu GWT

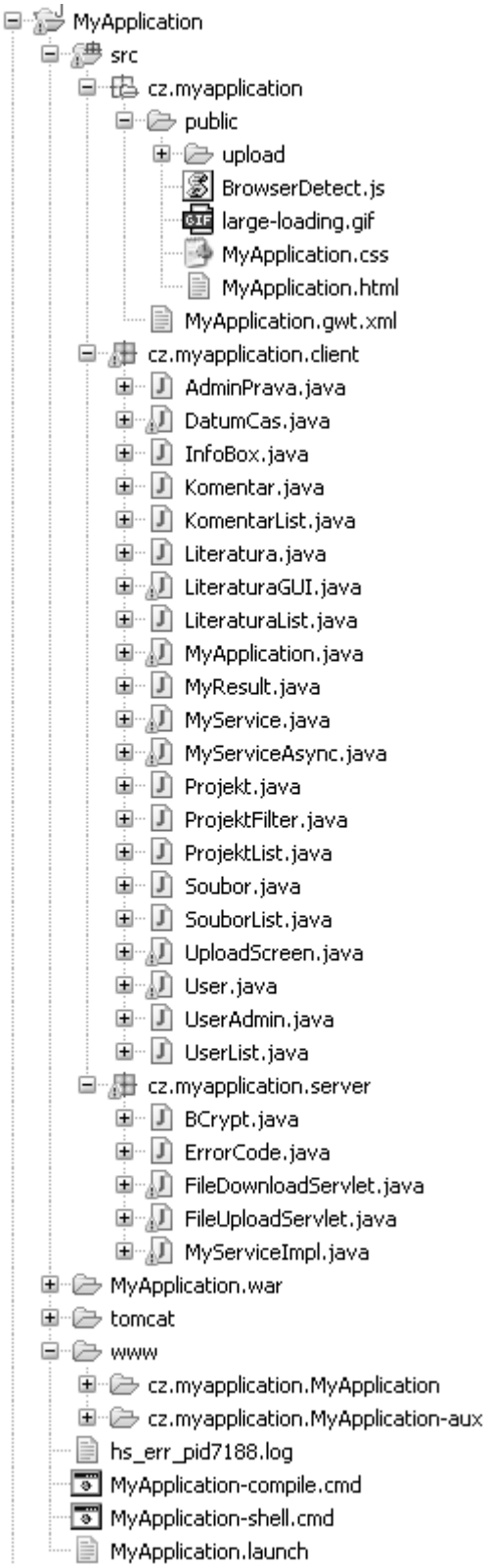
Obsah adresářů:

- bin
    - Soubory .class – zkompilované zdrojové kódy v jazyce Java (bytecode). Tyto soubory jsou nutné pro běh serverové části (servletu), která je standardně spuštěna na serveru s podporou servletů (např. Apache TomCat).
    - Vnitřní struktura je rozdělena na klientskou, veřejnou a serverovou část
- src
    - Zdrojové kódy v jazyce Java
- tomcat
    - Nastavení konkrétní instance serveru Apache TomCat pro běh v hosted módu při vývoji webové aplikace
- www
    - Soubory pro umístění webové aplikace na server

    - Kompilované soubory z jazyka Java do jazyka JavaScript, pro každý podporovaný prohlížeč frameworkem GWT se zde nachází jeden soubor s optimalizovaným kódem pro daný prohlížeč.
    - Tento adresář je vytvořen až po konečné kompilaci, nikoliv při vytváření projektu

Vnitřní struktura adresáře src, (viz. obr. 5.4):

- public
    - Statické prvky webové aplikace, např. html soubory, kaskádové styly, obrázky nebo knihovny JavaScript třetích stran
- client
    - Klientská část aplikace – zde jsou umístěny všechny objekty, které jsou používány v klientské části
    - Hlavním souborem je ten, který nese název jako samotný projekt – v tomto případě tedy `MyApplication.java`. Jeho struktura bude popsána dále.
    - Všechny zde přítomné soubory jsou v konečné fázi zkompilovány do jazyka JavaScript
- server
    - Soubory v této části běží na serveru a díky nim můžeme se serverem komunikovat pomocí RPC (asynchronního volání).
    - V této části neprobíhá kompilace do JavaScriptu, pouze se provede kompilace do bytového kódu (Java) a ten se poté umístí na server.



Obr. 5.4 Struktura adresáře src

5.4 Struktura aplikace (client)

Jak již bylo řečeno, základním souborem je `MyApplication.java`. Tento soubor je také nazýván „entry point class“ neboli „vstupní třída“. Pojmenování toho souboru stejně jako název celého projektu je pouze konvencí, proto musíme do zvoleného souboru (třídy) implementovat rozhraní `EntryPoint`, aby GWT mohl tuto výchozí třídu identifikovat:

```
public class MyApplication implements EntryPoint {
```

Tímto frameworku dáme na vědomí, že v této třídě se nachází funkce `onModuleLoad()`. Po spuštění aplikace (načtení stránky v prohlížeči) je volána právě tato funkce. Pak již má programátor naprostou volnost, může celou aplikaci napsat v této jediné funkci a značně zdrojový kód znepřehlednit, nebo může „předat řízení“ aplikace voláním dalších funkcí.

V mém informačním systému volám ve funkci `onModuleLoad()` funkce `prihlaseniInit()` a `prihlaseniLoad()`, první jmenovaná funkce zajistí inicializaci globálních prvků nutných k přihlášení a funkce `prihlaseniLoad()` zajistí zobrazení prvků na obrazovce. Dále ve funkci `onModuleLoad()` volám `ucitelInit()`, `studentInit()` a `administraceInit()`. Tyto funkce opět zajistí inicializaci globálních prvků, v tomto případě se inicializuje nabídka menu pro tři různé role v databázi. Až na pár vyjímek dále jsou v aplikaci používany funkce a lokální proměnné, proto již další inicializace nejsou potřeba. Po úspěšném ověření uživatele pomocí jména a hesla se načte podle jeho hlavní role příslušné menu a uživatel se může pohybovat v systému.

Klientská část je celá složena z tzv. panelů a widgetů. Základním panelem na stránce je `RootPanel`, do kterého se umisťují další panely a widgety. Teoreticky je tedy možné vnořit do sebe nekonečné množství panelů a vytvořit tak požadovaný vzhled aplikace. Panely se dělí na dva základní typy: `HorizontalPanel` a `VerticalPanel`. Do obou panelů se mohou opět umisťovat další panely bez rozdílu typu. Jediný rozdíl v těchto dvou panelech je ten, že u `HorizontalPanel` se vkládané prvky řadí vedle sebe a u `VerticalPanel` naopak pod sebe. Widgety a panely jsou statické prvky. Veškeré akční události jsou odchytávány pomocí „Event Listener“, který musíme danému prvku nadefinovat. Např. pro událost při stisku tlačítka musíme definovat:

```
tlacitko.addClickListener(new ClickListener() {
   public void onClick(Widget sender) {
            //požadovaná akce
        }
});
```

Tímto jsme definovali tlačítku nový „odposlouchávač“-Listener. Jedná se o Listener, který hlídá stisk tlačítka a pokud se tak stane, je volána funkce `onClick(Widget sender)` ve které se provede požadovaná akce. K dispozici je velké množství těchto Event Listener, např. stisk klávesy, stlačení klávesy, uvolnění klávesy, přejetí myší přes objekt atd. K dispozici je také obecný Listener, kde si tvůrce aplikace může pomocí DOM naprogramovat libovolný Listener podle svých potřeb.

5.4.1 Asynchronní volání

Pro komunikaci se serverem v prostředí JavaScriptu se využívá vzdálené volání procedury RPC. Pro odeslání a zpracování požadavku není na rozdíl od klasické HTML technologie nutné překreslovat celou stránku, celý proces se děje na pozadí. Aby toto bylo možné, je nutné dodržet následující body [12].

V první řadě je nutné vytvořit na straně klienta rozhraní pro komunikaci se službou na serveru. Vytvořené rozhraní musí rozšiřovat rozhraní `RemoteService`:

```
public interface MyService extends RemoteService {
```

Do tohoto rozhraní se umisťují pouze hlavičky metod (služeb na serveru), které budou volány ze strany klienta, např.

```
public User authenticateUser(String user, String pass);
public String[] test();
//atd.
```

Na straně serveru je potřeba vytvořit třídu, která bude vyřizovat požadavky na samotné služby. Podle konvencí by měla třída mít název podle rozhraní na straně klienta s příponou `Impl`, tedy `MyServiceImpl`. Třída musí rozšiřovat třídu `RemoteServiceServlet` a implementovat námi definované rozhraní `MyService`:

```
public class MyServiceImpl extends RemoteServiceServlet implements
MyService {
```

Tato třída se ve struktuře nachází v balíčku server, tzn. nebude kompilována do JavaScriptu ale pouze do bytového kódu jazyku Java. Z toho vyplývá, že ve třídách na straně serveru lze použít bez omezení všechny možnosti jazyku Java a není potřeba se řídit podle omezených možností kompilátoru z jazyka Java do jazyka JavaScript. Struktura třídy `MyServiceImpl` bude podrobněji rozebrána v kapitole věnované serverové části.

Aby aplikace na straně klienta nebyla blokována při čekání na odpověď od serveru a volání služeb se stalo plně asynchronními, je třeba vytvořit další rozhraní, které bude očekávat odpovědi serveru. Zde je již nutné striktně dodržet následující pravidla:

- Rozhraní musí mít stejný název jako rozhraní použité pro odesílání požadavků serveru s příponou `Async`, tedy `MyServiceAsync`
- Obě rozhraní se musí nacházet ve stejném balíčku, zde v `cz.myapplication.client`
- Rozhraní `Async` musí obsahovat stejné metody s tím rozdílem, že neobsahují žádnou návratovou hodnotu, tzn. `void`, a obsahují jako poslední parametr objekt `AsyncCallback`

Rozhraní `MyServiceAsync` může tedy vypadat takto:

```
public interface MyServiceAsync {
  public void authenticateUser(String user, String pass, AsyncCallback<User>callback);
  public void test(AsyncCallback<Object[]>callback);
  //atd.
```

Služby jsou nyní připraveny pro vzdálené volání. Jako poslední krok stačí v klientské aplikaci vytvořit instanci rozhraní služby:

```
MyServiceAsync svc = (MyServiceAsync) GWT.create(MyService.class);
```

Dále ošetřit zpětné volání jednotlivých služeb, kde se při úspěšném volání zavolá funkce `onSuccess(User result)` která také obdrží ze serveru požadované hodnoty v objektu `result` a to typu, kterého jsme definovali (v tomto případě `User`). Pokud volání úspěšné nebude, zavolá se funkce `onFailure(Throwable ex)`.

```
final AsyncCallback<User> callback = new AsyncCallback<User>()
  {
    public void onSuccess (User result)
       {
         //úspěšné volání
       }
    public void onFailure (Throwable ex)
       {
          //neúspěšné volání
       }
  };
```

Nyní již můžeme jednoduše volat službu podle potřeby, např. následující příkaz vložit do bloku, který ošetřuje stisk tlačítka

```
svc.authenticateUser(loginTxtBox.getText(),    passwordTxtBox.getText(),
callback);
```

Při přenášení objektů sítí pomocí RPC je podmínkou, aby tyto objekty byly serializované. Objekt se serializovatelný, pokud splňuje následující podmínky:

- Objekt implementuje rozhraní `IsSerializable` nebo `Serializable`
- Není finální
- Obsahuje prázdný konstruktor např. `private User {};`

Tyto pravidla platí pro vlastní objekty, všechny jednoduché datové typy jako např. `char` nebo `int` jsou standardně serializovatelné.

5.5 Struktura aplikace (server)

Jak bylo v předchozí kapitole zmíněno, balíček na serveru (`cz.myapplication.server`) obsahuje pouze jazyk Java, není kompilován do JavaScriptu. Základní třídou je `MyServiceImpl`, která implementuje všechny používané služby na bázi RPC. V dalším textu bude rozebrána struktura tohoto souboru.

Aby bylo možné z prostředí jazyka Java komunikovat s databází, resp. zadávat příkazy a pracovat s databázovými daty, je nutné použít tzv. connector, který zprostředkuje komunikaci s příslušnou databází. Pro implementaci connectoru a pro samotnou práci s příkazy databáze je nutné importovat potřebné knihovny:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
```

Použitý connector se bude lišit v závislosti na použité databázi. Při použití databáze MySQL byl použit MySQL Connector/J verze 5.1.7. Pro správnou funkci aplikace v hosted módu je nutné umístit connector do adresáře GWT kde je také umístěna naše aplikace (nadřazený adresář adresáři `MyApplication`). Při znalosti umístění databáze, uživatelského jména a hesla k databázi se připojíme do databáze:

```
private Connection conn = null;
private String url = "jdbc:mysql://127.0.0.1/projekty";
private String userDbRoot = "root";
private String passDbRoot = "r00tr00t";
```

```
Class.forName("com.mysql.jdbc.Driver").newInstance();
conn = DriverManager.getConnection(url, userDbRoot, passDbRoot);
```

Proces přihlášení do databáze je aplikován při přihlášení uživatele a poté jsou již vykonávány příkazy jednotlivých služeb podle potřeb uživatele.

5.6 Bezpečnost

5.6.1 Autentizace uživatele

Při vstupu do systému je vyžadováno uživatelské jméno a heslo. Pokud je přihlášení úspěšné, uloží se na serveru do kontejneru `HashMap` k příslušné session ID uživatelské jméno, jednotlivá ID přidělených rolí a uživatelovo ID v databázi:

```
HttpSession session = getSession();
HashMap pole = getHashMapFromSession("loginMap", session);
pole.put("login", uzivatelDb);
pole.put("pravaInt", pravaInt);
pole.put("uzivatelId", idUzivatel);
```

Termínem session ID se rozumí jedinečný identifikátor, který je každému uživateli přidělen na základě prvotní komunikace se serverem a zůstává stejný po celou dobu komunikace se serverem. Tento identifikátor včetně kontejneru je uložen v paměti serveru až do vypršení časovače nečinnosti (timeout) nebo dokud uživatel neprovede odhlášení. Z kontejneru jsou pak podle potřeby při každém dotazu na server ověřovány údaje, zda je uživatel přihlášen a jaké má přidělené role. Pomocí funkce `checkRigts` je pak prováděno ověření, zda má uživatel oprávnění k požadované akci:

```
ErrorCode kontrolaUzivatele = checkRights("požadovaná akce");
if (!kontrolaUzivatele.getErrorBool()) {
        //uživatel má k akci oprávnění
}
else {
        //uživatel nemá k akci oprávnění
}
```

Funkce vrací objekt typu `ErrorCode` a pomocí metody `getErrorBool` zjistíme zda je akce povolena (false) nebo ne (true). Funkce `checkRights` se připojí do databáze a z entit ROLE, ROLE_MODUL a MODUL zjistí pomocí údajů z kontejneru zda je uživatel oprávněn vykonat danou akci a podle toho nastaví návratový objekt `ErrorCode`.

Vlastní heslo pro přihlášení do systému je zabezpečeno hashováním. Při procesu přihlášení je vytvořen hash hesla zadaného uživatelem a je porovnáván s hashem hesla uloženým v databázi. Pokud hashe souhlasí, je uživatelovi povolen přístup do systému. V projektu byla použita knihovna BCrypt, konkrétně implementace do jazyka Java jBCrypt [14]. Pro použití je nutné importovat samotnou knihovnu BCrypt:

```
import cz.myapplication.server.BCrypt;
```

a dále již můžeme využívat příkazy pro práci s knihovnou:

```
BCrypt.hashpw("heslo", BCrypt.gensalt()); //vytvoření hash (nové heslo)
BCrypt.checkpw("hash1","hash2"); //porovnání dvou hash řetězců, funkce
                                 //vrací true pokud hashe souhlasí
```

Použití šifrované formy hesel je důležité z důvodu bezpečnosti, neboť v případě nešifrovaných hesel by mohl administrátor z databáze jednoduše přečíst hesla jednotlivých uživatelů.

5.6.2 Šifrování přenosu dat

Při přenosu dat počítačovou sítí může dojít k odposlechu paketů a útočník tak může získat uživatelské jméno a heslo nebo jiná data přenášená mezi klientskou aplikací a serverem. Proto je namístě zajistit ochranu před takovýmto útokem.

Pro zabezpečení aplikace byl zvolen protokol SSL [18], který zapouzdřuje použitý komunikační protokol šifrováním pomocí zvoleného šifrovacího algoritmu. Jako šifrovací algoritmus byl zvolen algoritmus RSA. Samotné šifrování je prováděno na základě veřejných a soukromých klíčů. Při zahájení komunikace je serverem zaslán klientovi certifikát serveru, který slouží pro ověření identity serveru a také si klient se serverem vymění veřejné klíče, pomocí kterých bude probíhat šifrování. Dešifrování pak probíhá na základě soukromých klíčů.

Na server tedy musí být umístěn certifikát pro ověření identity serveru. Generování certifikátu lze provést příkazem

```
$JAVA_HOME/bin/keytool -genkey -alias tomcat -keyalg RSA
```

kde `$JAVA_HOME` označuje domovský adresář prostředí Java. Vygenerovaný certifikát v souboru `.keystore` je nutné přesunout do adresáře se serverem TomCat, do adresáře `/usr/share/tomcat6/`. Jako další krok je nutné povolit zabezpečenou komunikaci prostřednictvím HTTP connectoru. To lze provést v konfiguračním souboru serveru TomCat, v souboru `server.xml`:

```
<Connector port="8180" protocol="HTTP/1.1" \
```

```
               connectionTimeout="20000" \
               redirectPort="8443" />
<Connector port="8443" protocol="HTTP/1.1" SSLEnabled="true" \
      maxThreads="150" scheme="https" secure="true" clientAuth="false" \
      sslProtocol="TLS" keystorePass="certifikatheslo" />
```

V prvním connectoru definujeme standardní, nezabezpečený port pro přístup k serveru (port 8180) a také port, na který má dojít k přesměrování, pokud server přijme požadavek na zabezpečenou komunikaci HTTPS (port 8443). V druhém connectoru již definujeme samotný port pro zabezpečenou komunikaci prostřednictvím SSL. Zde definujeme různé parametry, mj. také heslo pro načtení certifikátu určeného pro ověření identity serveru.

Pokud vyžadujeme, aby aplikace vždy prováděla veškerou komunikaci zabezpečeně pomocí SSL, je nutné toto nastavit v konfiguračním souboru `WEB-INF/web.xml` archivu WAR (jeho struktura je popsána v kapitole 5.9):

```
 <security-constraint>
   <web-resource-collection>
     <web-resource-name>Entire Web App</web-resource-name>
     <url-pattern>/*</url-pattern>
   </web-resource-collection>
   <user-data-constraint>
     <transport-guarantee>CONFIDENTIAL</transport-guarantee>
   </user-data-constraint>
 </security-constraint>
```

Při přístupu k aplikaci pomocí nezabezpečeného protokolu HTTP dojde automaticky k přesměrování uživatele na zabezpečené stránky aplikace, konkrétně na port definovaný v connectoru nastaveném v souboru `server.xml` webového serveru Apache TomCat.

5.7 Struktura služeb, vykonávání dotazů

Každá služba v první fázi vykoná ověření uživatele (funkce `checkRights`), poté vykoná dotaz, zpracuje případné výsledky a vrátí podle typu služby požadovaný návratový objekt. Pro přehlednost uvádím pouze vybrané části služby, zdrojový kód celé služby, resp. celé aplikace, je uveden v příloze v elektronické verzi na přiloženém CD. Příkazy pro vykonání SQL příkazu UPDATE:

```
String dotaz = "UPDATE PROJEKT SET NAZEV = ‘" + promenna + "’";
PreparedStatement ps = conn.prepareStatement(dotaz);
```

```
ps.executeUpdate();
ps.close();
```

V proměnné `PreparedStatement ps` je uložen připravený dotaz včetně parametrů připojení do databáze (proměnná `conn`, kterou jsme definovali dříve při prvotním připojení k databázi). Metodou `executeUpdate()` je provedeno samotné připojení k databázi a vykonání příkazu a metodou `close()` je spojení uzavřeno. Tato struktura je platná pro SQL příkazy typu UPDATE, INSERT a DELETE. Pro příkaz SELECT je struktura mírně odlišná:

```
dotaz = "SELECT * FROM UZIVATEL";
PreparedStatement ps = conn.prepareStatement(dotaz);
ResultSet result = ps.executeQuery();
while (result.next()) {
  //zpracování výsledků dotazu
}
result.close();
ps.close();
```

Protože příkaz SELECT vrací data z databáze, je využita metoda `executeQuery()`, data jsou uloženy do proměnné typu `ResultSet` a následně jsou tyto data procházena po jednotlivých řádcích pomocí `while (result.next())`, kde jsou s daty prováděny požadované operace, např. uložení dat do pole a následný transport ke klientovi.

5.8 Operace se soubory

Součástí webové aplikace je také nahrávání souborů na server a jejich stahování. Pro nahrávání souborů na server bylo zvoleno rozšíření GWT gwtswfext, což je zapouzdření JavaScriptové knihovny SWF Upload do prostředí GWT [15, 16]. SWF Upload je hotové řešení výběru souboru na straně klienta pomocí technologií Adobe Flash a JavaScript. Samotný proces nahrávání dat na server zajišťuje metoda POST ze sady příkazů standardu HTML. V prostředí GWT pak už stačí nastavit parametry knihovny SWF Upload a cílové umístění souboru na serveru. Konkrétně v třídě `UploadScreen` na straně klienta, která zajišťuje také import knihovny SWF Upload, se provedou veškerá nastavení a omezení jako např. maximální velikost souboru, povolené typy souborů, nastavení časovače timeout atd. V balíčku server je vytvořen nový servlet `FileUploadServlet`, který ošetřuje příjem souboru a jeho uložení na disk serveru. Soubor je na serveru uložen do adresáře `/tmp/nazev_souboru` a pro zajištění jedinečnosti souboru je přidána přípona s uživatelským jménem přihlášeného uživatele. Jakmile je celý soubor nahrán do adresáře `/tmp/`, dojde u klienta ke spuštění klasického vzdáleného volání (RPC), soubor je uložen přímo do databáze a

smazán z dočasného adresáře `/tmp/`. Před uložením do databáze je samozřejmě odstraněna přípona přihlášeného uživatele. Při stahování souboru ze serveru je postup podobný, ale opačný. Voláním RPC se soubor uloží z databáze do dočasného adresáře `/tmp/` a u klienta pak dojde ke stažení souboru pomocí HTTP metody GET. Soubor je z dočasného adresáře odstraněn pokud uživatel požádá o stažení dalšího souboru, z aplikace se odhlásí nebo pokud vyprší časovač neaktivity (timeout). Servlet zajišťující stahování souborů ze serveru se nazývá `FileDownloadServlet` a je umístěn opět v balíčku server.

5.9 Umístění aplikace na server

Server Apache Tomcat využívá pro umístění souborů na server formát WAR. Jedná se o archiv celé webové aplikace v pevně dané struktuře [17]. Do archivu se umístí všechny soubory, které byly vygenerovány kompilátorem GWT ve složce `www/cz.myapplication.MyApplication`. Dále je nutné vytvořit složku `WEB-INF`, která obsahuje složky `classes`, `lib` a soubor `web.xml`. Složka `classes` obsahuje zkompilované zdrojové kódy v jazyce Java. Zde je nutné přesně dodržet souborovou strukturu, v našem případě tedy `WEB-INF/classes/cz/myapplication/client` a `WEB-INF/classes/cz/myapplication/server`. Složka `lib` je určená pro externí knihovny a `JAR` soubory, které aplikace používá ke své činnosti. V našem případě tedy do složky `lib` umístíme soubory `mysql-connector-java-5.1.7-bin.jar`, `gwt-servlet.jar`, `commons-fileupload-1.2.1.jar` a `commons-io-1.4.jar`. Prvním souborem je MySQL Connector/J zajišťující propojení servletu s databází. `Gwt-servlet.jar` zajišťuje samotný chod GWT servletů a poslední dva soubory jsou nutné pro operaci se soubory, tedy nahrávání souborů na server a stahování souborů ze serveru. Soubor `web.xml` obsahuje důležité nastavení aplikace, mj. cestu k jednotlivým servetům a také časovač pro vypršení relace (session timeout). Dokument má strukturu podle standardu XML a je k nahlédnutí v příloze na CD. Příklad kofigurování a mapování servletu v souboru `web.xml`:

```
<servlet>
   <servlet-name>MyService</servlet-name>
   <servlet-class>cz.myapplication.server.MyServiceImpl</servlet-class>
 </servlet>
 <servlet-mapping>
   <servlet-name>MyService</servlet-name>
   <url-pattern>/myService</url-pattern>
 </servlet-mapping>
```

Celá výše popsaná struktura se zabalí libovolným komprimačním programem do formátu `nazev_aplikace.war`. Tento soubor již lze nahrát na server Apache Tomcat a aplikace je připravena k použití.

6 Závěr

V první části bakalářské práce byly zpracovány základy problematiky moderních webových frameworků a byly porovnány hlavní JavaScriptové frameworky. Dále byly stručně porovnány JavaScriptové a PHP frameworky. Tato práce je zaměřena na GWT a JavaScriptové frameworky, a jelikož nelze přímo porovnánat JavaScript frameworky s PHP frameworky, byly zmíněny PHP frameworky, jejich možnosti a porovnání, pouze okrajově.

V druhé části této práce byl proveden návrh E-R modelu databáze pro správu projektů, na kterém je následně založeno fungování celé aplikace. Databázový model umožňuje kromě samotné správy projektů také správu uživatelů, přidělování více rolí jednotlivým uživatelům a také umožňuje jednotlivým rolím povolovat konkrétní operace v rámci aplikace, jako je např. zadávání projektu, změna projektu, hodnocení projektu atd. Na základě tohoto abstraktního modelu databáze je realizována webová aplikace pro správu projektů pomocí frameworku Google Web Toolkit. Systém umožňuje zadávání projektů, jejich odevzdání a hodnocení, případně vrácení k přepracování. Systém dále umožňuje nahrávání souborů na server a jejich následné stahování, komunikaci mezi zadavatelem projektu a uživatelem, který projekt vypracovává. Webová aplikace také obsahuje různorodé ochrany jako je např. kontrola délky zadaných dat do formulářových polí dle délky odpovídajících proměnných v databázi, kontrola zadání vyžadovaných neprázdných položek projektu, kontrola správného zadání formátu data a času a také kontrola změny dat při editaci položky jiným uživatelem. Systém také umožňuje změnu modulu, resp. změnu položek menu podle přidělených rolí konkrétnímu uživateli a to bez nutnosti opětovného přihlašování. Samozřejmostí je možnost změny hesla a také kompletní správa uživatelů včetně možnosti zablokovat účet. V místech aplikace, kde lze očekávat vyšší počet položek z databáze, typicky seznam projektů, je implementováno stránkování s možností volby počtu položek na stránku. Pro snadné vyhledávání konkrétních dat je také umožněno filtrovat požadovaný obsah podle mnoha kritérií.

Pro posílení bezpečnosti byla použito šifrování dat mezi klientem a serverem pomocí SSL, avšak při nasazení tohoto zabezpečení došlo k odstavení nahrávání souborů na server. Tento problém je způsoben pravděpodobně technologií Adobe Flash, která vykazuje chyby při použití tzv. self-signed certifikátu, tzn. certifikátu, který je podepsán (ověřen) svým tvůrcem. Tento problém by byl vyřešen ověřením certifikátu cerfitikační autoritou (např. VeriSign) nebo používáním nezabezpečeného spojení pro nahrávání souborů na server, což by však vyžadovalo zásah do samotné aplikace.

Po pochopení filosofie GWT se tento framework stává mocným nástrojem pro vývoj webových aplikací. Ať už jde o jednoduchou implementaci menu do aplikace nebo import rozšíření třetích stran, které rozšiřují možnosti GWT nebo ještě více zjednodušují samotné

programování jako např. rozšíření SWFUpload, díky kterému je možné jednoduchým způsobem implementovat odesílání souborů na server. Výborným způsobem je také zpracováno asynchronního volání RPC, které funguje naprosto bez problémů a jeho zpracování v aplikaci je naprosto jednoduchou záležitostí. Dalším argumentem pro používání GWT je velice silná základna uživatelů a oficiální fórum Googlu, kde lze často rychle a jednoduše nalézt odpověď na problémy při programování a které jsem sám nejednou použil. Výbornou pomůckou je také přehledný tutoriál na kterém lze snadno pochopit základní funkce a filosofii GWT. Naopak velikou nevýhodou GWT je absence podpory pro výměnu souborů mezi serverem a klientem. Tuto nevýhodu lze částečně řešit např. již zmíněným rozšířením SWFUpload, ale vždy je nutné samotnou výměnu souborů provádět přes klasické HTTP metody GET a POST.

Při programování komplexních aplikací, jako je např. aplikace pro správu projektů zpracovaná v rámci této práce, je také nutné zvolit jednotnou metodiku pojmenování funkcí, proměnných a samotných tříd pro snadnou orientaci ve velkém množství napsaného zdrojového kódu.

Literatura

[1] CHOPRA, Vivek, et al. *Beginning JavaServer Pages.* Indianapolis (Indiana) : Wiley Publishing c2005. 1262 s. ISBN 0-7645-7485-X.

[2] HARMON, James Earl. *Dojo : Using the Dojo JavaScript Library to Build Ajax Applications*. Indiana : Prentice Hall PTR, 2008. 350 s. ISBN 978-0-132-35804-0.

[3] RUSSELL, Matthew. *Dojo : The definitive guide*. 1st edition. Sebastopol (California) : O'Reilly Media, 2008. 486 s. ISBN 978-0-596-51648-2.

[4] HANSON, Robert, TACY, Adam. *GWT in Action : Easy Ajax with the Google Web Toolkit*. [s.l.] : Manning Publications, c2007. 597 s. ISBN 978-1-933-98823-8.

[5] POWEL, Thomas, SCHNEIDER, Fritz. *JavaScript 2.0 : The Complete Reference*. 2nd edition. [s.l.] : McGraw-Hill/Osborne , c2004. 976 s. ISBN 0-07-225357-6.

[6] *Compiling Java to JavaScript : A Conversation with Scott Blum by Frank Sommers* [online]. c1996-2007 , December 21, 2006 [cit. 2008-11-26]. Dostupný z WWW: <http://www.artima.com/lejava/articles/java_to_javascript.html>.

[7] NEWTON, Aaron. *MooTools Essentials : The Official MooTools Reference for JavaScript™ and Ajax*. [s.l.] : Apress, c2008. 276 s. ISBN 978-1-4302-0983-6.

[8] *MooTools - a compact javascript framework* [online]. 2006 [cit. 2008-11-28]. Dostupný z WWW: <http://mootools.net/>.

[9] ZAMMETTI, Frank. *Practical JavaScript™, DOM Scripting, and Ajax Projects*. USA : Apress, c2007. 546 s. ISBN 978-1-59059-816-0.

[10] GOLDING, David. *Beginning CakePHP : From Novice to Professional*. [s.l.] : Apress c2008. 344 s. ISBN 978-1-4302-0977-5.

[11] ALLEN, Rob, LO, Rick. *Zend Framework in Action*. [s.l.] : Manning Publications, c2007. 425 s.

[12] *Google Web Toolkit 1.5* [online]. c2008 [cit. 2008-11-21]. Dostupný z WWW: <http://code.google.com/docreader/#p=google-web-toolkit-doc-1-5>.

[13] *Project Structure* [online]. c2008 [cit. 2009-04-12]. Dostupný z WWW: <http://code.google.com/docreader/#p=google-web-toolkit-doc-1-5&s=google-web-toolkit-doc-1-5&t=DevGuideProjectStructure>.

[14] *JBCrypt - strong password hashing for Java* [online]. 2008 [cit. 2009-04-22]. Dostupný z WWW: <http://www.mindrot.org/projects/jBCrypt/>.

[15] *Gwtswfext - Google Code : GWT SWFUpload extension* [online]. c2009 [cit. 2009-05-12]. Dostupný z WWW: <http://code.google.com/p/gwtswfext/>.

[16] *SWFUpload* [online]. c2007 [cit. 2009-05-12]. Dostupný z WWW: <http://swfupload.org/>.

[17] *Web Application Archives* [online]. 2002 [cit. 2009-05-14]. Dostupný z WWW: <http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/WCC3.html>.

[18] *Apache Tomcat 6.0 : SSL Configuration HOW-TO* [online]. c2008 [cit. 2009-05-16]. Dostupný z WWW: <http://tomcat.apache.org/tomcat-6.0-doc/ssl-howto.html>.

Seznam použitých zkratek

AFL - Academic Free License

AJAX - Asynchronous JavaScript and XML

API - Application programming interface

BSD - Berkeley Software Distribution

CRUD - Create, read, update and delete

CSS - Cascading Style Sheets

DOM - Document Object Model

GWT - Google Web Toolkit

HTML - HyperText Markup Language

HTTP – Hypertext Transfer Protocol

HTTPS - Hypertext Transfer Protocol Secure

i18n – Internationalization

IDE – Integrated development environment

JDBC – Java Database Connectivity

JRE - Java Runtime Environment

JSNI - JavaScript Native Interface

JSON - JavaScript Object Notation

JVM - Java Virtual machina

MIT - Massachusetts Institute of Technology

MVC - Model View Controller

NLS - Native Language Support

OOP - Object-oriented programming

RIA - Rich Internet Application

RPC - Remote procedure call

RSA – Rivest, Shamir, Adleman

SSL – Secure Sockets Layer

UI - User interface

WAR – Web application archive

XHR - XMLHttpRequest

XML - Extensible Markup Language