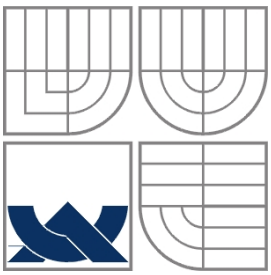
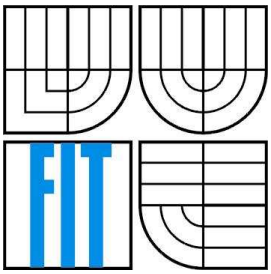


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

VZTAH KLIENT-SERVER U WEBOVÝCH APLIKACÍ

CLIENT-SERVER RELATIONSHIP IN WEB APPLICATIONS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

PETR KUČERA

VEDOUCÍ PRÁCE
SUPERVISOR

ING. SVATOPLUK ŠPERKA

BRNO 2011

Abstrakt

Tato práce zkoumá možnosti přesunu aplikační logiky ve webových aplikacích ze serverů na klienty s cílem snížit zátěž serverů. Zkoumá činnosti, které mohou být prováděny na straně klientů v oblasti generování HTML kódu ze strukturovaných dat a operací nad daty bez nutnosti komunikace se serverem. Popisuje implementaci JavaScriptového frameworku, který převádí poznatky do praktického využití. Jeho využití demonstruje na jednoduchém informačním systému. Výsledky provedených měření ukazují, že frameworku se cíl podařilo splnit a že touto cestou je možné vytvářet webové aplikace kladoucí důraz na snížení serverové zátěže.

Abstract

This paper conducts a research in the field of transferring application logic to clients in web applications in order to reduce server load. It examines actions that can be performed on the client side in area of generating HTML code from structured data and data manipulation operations eliminating the need to communicate with server. It describes implementation of a JavaScript framework which turns this knowledge into practical use. It shows its usability on a simple information system. The results of the measurements show that the goal has been achieved and that the framework shows the way in which web applications with a strong emphasis on reducing server load can be made.

Klíčová slova

Webové aplikace, server, klient, zátěž, logika, HTML, JavaScript, framework, JSON, AJAX

Keywords

Web applications, server, client, load, logic, HTML, JavaScript, framework, JSON, AJAX

Citace

Kučera Petr: Vztah klient-server u webových aplikací, bakalářská práce, Brno, FIT VUT v Brně, 2011

Vztah klient-server u webových aplikací

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Svatopluka Šperky.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Petr Kučera
17. 5. 2011

Poděkování

Tímto bych chtěl poděkovat vedoucímu bakalářské práce, Ing. Svatopluku Šperkovi, za pomoc, odborné rady, vedení a veškerý čas, který mi věnoval.

Obsah

1 Úvod

V současném trendu stále se webové aplikace a služby těší čím dál větší oblibě. S rozvojem internetu a stále větším množstvím uživatelů, kteří jej mohou využívat, také vznikají nové, větší požadavky na tyto aplikace. Dávno jsou pryč časy, kdy byl internet jen pro vyvolené a s větším počtem uživatelů – klientů roste také zatížení serverů. Webové aplikace také nejsou pouze doménou prostředí internetu, ale jsou čím dál více populární i pro podnikové systémy v sítích intranet.

Motivací k této práci bylo prozkoumat možnosti přesunu zátěže ze serverů na klienty u webových aplikací a vytvořit nástroj umožňující vývojářům využít tyto poznatky v praxi. Usnadnit jim tak práci při tvorbě aplikací, které kladou důraz na snížení zatížení serveru. Protože se jedná o webové aplikace, nástroj by měl být navržen pro tenkého klienta (internetový prohlížeč ve standardním nastavení, tzn. stav po instalaci, bez nutnosti instalace přídavných zásuvných modulů, doplňků apod.) Měl by být vytvořen s přenositelným kódem, tak, aby byl kompatibilní s většinou běžně používaných prohlížečů a operačních systémů.

Problémem při přesunování aplikační logiky na strany klientů je odhalení vykonávané logiky útokům. Na stranu klientů lze tak bezpečně přesunout pouze ty operace, které nejsou rizikové a jejichž případné napadení by neohrozilo uživatele či samotnou aplikaci. V této práci se zaměřuji především na možnosti generování HTML kódu klientem z dat, která obdrží ze serveru ve strukturované formě, a také na možnosti takových operací, které s těmito daty pracují a eliminují potřebu komunikovat se serverem.

V první části práce jsou popsány základní technologie používané při tvorbě webových aplikací, stejně jako popis samotné webové aplikace a její architektury.

Následující část obsahuje teoretickou analýzu problematiky. Popisuje vytvoření obslužné aplikace, která řídí činnost logiky na straně klienta. Navrhuje možnosti generování HTML kódu, přizpůsobení generování potřebám vývojáře, operace nad přijatými daty jako řazení, vyhledávání, které nevyžadují komunikaci se serverem a další funkce, které je možné na obslužné aplikaci vykonávat.

Další kapitola pak popisuje implementaci takové aplikace v jazyce JavaScript, která implementuje některá z navrhovaných řešení. Závěrečné kapitoly se věnují praktickému testování na demonstračním informačním systému a ukazují měření, která byla provedena k ověření přínosu aplikace.

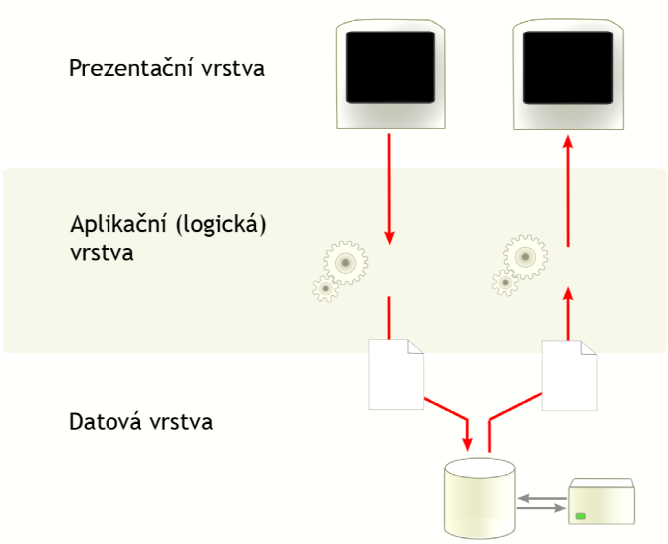
2 Základní pojmy

2.1 Webová aplikace

Webové aplikace jsou aplikace typu klient-server, které využívají tzv. tenké klienty. Tzn. klientská část aplikace (internetový prohlížeč) neobsahuje aplikační logiku, ta je vykonávána na webovém serveru a na klienty odeslána ve formátech podporovaných prohlížeči, např. HTML/XHTML. Výhoda webových aplikací spočívá především v možnosti aktualizace aplikace. Při použití tlustých klientů (speciálních aplikací, které je nutné mít nainstalované na straně klienta) je často nutné při aktualizace serverové části aplikace, aktualizovat také klientskou část u každého uživatele.

2.1.1 Architektury webových aplikací

Pro webové aplikace jsou typické vícevrstvé architektury. To znamená, že aplikace tvoří několik vzájemně spolupracujících vrstev, které mohou být umístěny na různých stanicích a které spolu komunikují přes definované rozhraní. Pro webové aplikace bývá nejtypičtější třívrstvá architektura (viz Obrázek 2.1).



Obrázek 2.1 Třívrstvá architektura (zdroj: wikipedia.org)

Prezentační vrstva slouží k zobrazení výstupu aplikace a pro zobrazení grafického uživatelského rozhraní. Tato vrstva je typicky reprezentována prohlížečem.

Aplikační (či logická vrstva) obsahuje jádro aplikace, samotnou logiku, všechny funkce, zpracování dat. Reprezentuje ji aplikační (webový) server.

Datová vrstva slouží k uchovávání dat, jejich opětovnému získávání a udržování konzistence. Typicky tuto vrstvu obsluhuje databázový server.

Nevýhodou tohoto modelu je skutečnost, že prezentační vrstva pouze staticky zobrazuje obsah vygenerovaný a zaslaný z vrstvy aplikační. S rozvojem webových aplikací dochází ke stavu, kdy aplikaci využívá velké množství uživatelů, ale webový server zůstává v podobě jednoho, či několika málo stanic. Tím rostou hardwarové nároky na aplikační i na databázové servery.

Tato práce hledá způsoby přesunutí některé logiky z aplikační vrstvy na klienty. Navrhuje vytvoření mezivrstvy mezi prezentační a aplikační vrstvou, která bude se vykonávat v prohlížeči a bude nahrazovat některé funkce, které provádí server a tím se pokusí snížit zatížení webového serveru.

2.2 Webové technologie

2.2.1 HTML, XML, XHTML

Jazyk, který popisuje prohlížeči, jak zobrazit data, se nazývá HTML (Hypertext markup language = Značkovací jazyk pro hypertext). Je to jeden z hlavních jazyků, který se při tvorbě webových stránek používá.

Jazyk využívá značek, které tvoří příkazy. Např.:

```
<p align="center">Odstavec zarovnaný na střed</p>
```

Značky se uzavírají do ostrých závorek a tvoří ohraničení celku, který se označuje jako *element*. Elementy mohou obsahovat parametry (v příkladu `align`), které elementu dávají dodatečné vlastnosti. Těmto parametrům říkáme *atributy* (např. atribut `id` identifikuje element).

Struktura každého HTML dokumentu se skládá ze značek (elementů), jak ukazuje následující příklad.

```
<html>
   <head>
      <title>Název dokumentu</title>
   </head>
   <body>
      <p>Obsah</p>
   </body>
</html>
```

Element `<html>` je nadřazen všem ostatním elementům, tvoří jakýsi kořenový prvek. Element `<head>` označuje tzv. hlavičku stránky. Zde najdeme např. titulek dokumentu – element `<title>`. Obecně lze říci, že obsah hlavičky se na stránce nezobrazuje, ale nějakým způsobem ji charakterizuje. Element `<body>` označuje tzv. tělo stránky. Ten naopak obsahuje vše, co se má na stránce zobrazit, v tomto případě jediný odstavec s textem.

HTML vznikl ze značkovacího jazyka SGML (Standard Generalized Markup Language = Standardizovaný zobecněný značkovací jazyk) a v roce 1986 se stal mezinárodním standardem.

V současné době je nejnovějším standardem HTML 4.01 z roku 1999. (Předcházející část podkapitoly převzata z [9].)

Jeho následovníkem se stal jazyk XHTML (Extensible hypertext markup langure = rozšiřitelný hypertextový značkovací jazyk), využívající univerzální jazyk XML.

XML (Extensible Markup Language = Rozšiřitelný značkovací jazyk) je obecný značkovací jazyk používaný jak k vytváření konkrétních značkovacích jazyků (jako v případě XHTML), tak pro serializaci dat.

Současně je aktuální verze XHTML 1.1, práce na verzi XHTML 2.0 byly zastaveny. Od roku 2007 se připravuje nová specifikace jazyka HTML, označovaná jako HTML5, a současně její XML varianta XHTML5, které představují řadu novinek pro vývoj komplexních webových aplikací jako je podpora pro audio, video, prohlížečem řízená tvorba grafiky pomocí JavaScriptu, datové úložiště, geolokace – systém pro určení geografické polohy uživatele, systém „drag & drop“ („přetahování“ tažením myši) a další.

2.2.2 JavaScript

JavaScript je multiplatformní, objektově orientovaný, interpretovaný jazyk, který se používá k vytvoření dynamického obsahu u jinak statických stránek. Dnes je velmi oblíbeným nástrojem k vytváření animací a grafických efektů. Jazyk je většinou interpretován prohlížečem, na straně klienta, z čehož plynou určitá bezpečností omezení i rizika. Např. JavaScript nemůže přistupovat k souborovému systému uživatele.

JavaScript poprvé představila společnost Netscape v roce 1995 ve svém prohlížeči. Původně měl pomoci k lepší integraci Java apletů do HTML stránek. Později se však z jazyka stal oblíbený nástroj pro přidání interaktivity do webových stránek, většinou zcela bez použití Javy.

V dnešní době je JavaScript standardně podporován ve více než 99% [7] programů určených k prohlížení webových stránek. Nutno ovšem podotknout, že podle pravidel přístupnosti webu se dlouhou dobu JavaScript považoval za nepřípustný a tvůrce webu musel zajistit plnohodnotnou funkční variantu bez jeho použití [2]. V současné době dochází ke zmírnění tohoto postoje. Nové pravidla přípustnosti webu již neukládají povinnost mít verzi bez použití JavaScriptu, pouze kladou důraz na maximální kompatibilitu se stávajícími a budoucími prohlížeči, včetně pomocných technologií [3]. K tomuto kroku přispěl zejména rozvoj technologií AJAX a RIA.

2.2.3 RIA

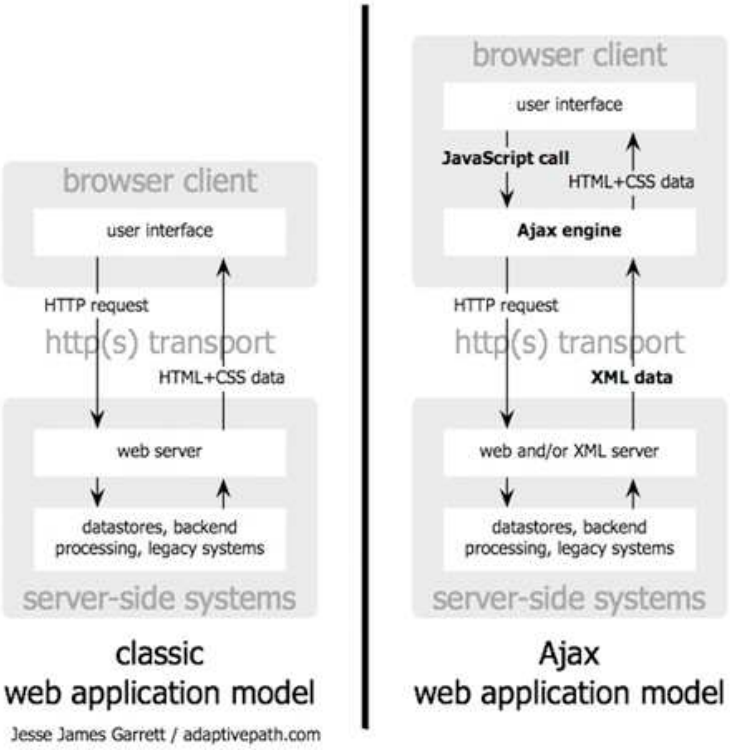
RIA (Rich internet application = Bohatá internetové aplikace) je webová aplikace, která se svojí funkcionalitou blíží desktopovým aplikacím. Toho bývá dosaženou použitím dodatečných zásuvných modulů, které je nutné instalovat do prohlížeče, aby takové aplikace mohli fungovat. Např. aplikace využívající práci s videem jsou nuceny takové platformy využít. Nově připravovaný standard HTML5

však již přináší podporu práce s videem integrovanou přímo do prohlížeče. Nejpoužívanějšími platformami pro RIA aplikace jsou Adobe Flash, Java a Microsoft Silverlight.

2.2.4 AJAX

AJAX (Asynchronous JavaScript and XML = Asynchronní JavaScript a XML) je soubor několika technologií, používaných k vytváření interaktivních webových aplikací, které umožňují komunikaci a výměnu dat mezi klientem a serverem bez nutnosti opětovného načítání stránky. Termín AJAX poprvé použil Jesse James Garrett [1] a definoval ho jako soubor následujících bodů:

1. Standardy HTML, XHTML a CSS pro zobrazení informací
2. Dynamické zobrazení a interakce pomocí DOM (Document object model = objektový model dokumentu) – objektově orientované reprezentace HTML či XML dokumentu
3. XHR (XMLHttpRequest) pro asynchronní komunikaci se serverem
4. JavaScript, který všechno spojuje



Obrázek 2.2 Schéma činnosti AJAX (zdroj: adaptivepath.com)

2.2.5 JSON

JSON (JavaScript object notation = JavaScriptový objektový zápis) je způsob zápisu dat určených pro datový přenos. Vstupem mohou být libovolná data (číslo, řetězec, pole, objekt) v libovolné hierarchii, výstupem je vždy řetězec. JSON se hojně používá ve webových aplikacích při komunikaci mezi klientem a serverem, zejména při použití technologie AJAX, ať už v síti internet nebo vnitropodnikových sítích intranet.

Ve srovnání s dalším velmi používaným způsobem formátu přenášených dat, XML, JSON nepřenáší metadata. Díky této vlastnosti a díky absenci značek a atributů je na přenos stejné informace zapotřebí výrazně méně dat než v případě XML. Další výhodou při použití ve webových aplikacích je jednoduchá interpretace dat, jedná se vlastně přímo o korektní zápis dat v jazyce JavaScript, takže není nutné používat dodatečné nástroje pro zpracování. Navíc v moderních prohlížečích jsou implementovány nativní syntaktické analyzátory pro JSON, které zvyšují rychlost jeho zpracování i bezpečnost jeho použití.

3 Analýza problému

3.1 Bezpečnost vykonávání aplikační logiky na straně klienta

Důležitým hlediskem při tvorbě této práce bylo bezpečnostní hledisko. Protože navrhovaný framework je psán v jazyce JavaScript, interpretovaném jazyce, jehož otevřený kód je přenášen sítí na klienta, není možné v otevřené podobě přenášet takovou logiku aplikace, u které by mohlo dojít, při jejím pozměnění k ohrožení bezpečnosti aplikace či uživatelských dat. V následujících kapitolách budou popsány možnosti přesunu logiky bez vystavení aplikace bezpečnostním rizikům.

Další možností je použití uzavřeného kódu, jako např. Java aplety, Flash, apod. Tento přístup ovšem vyžaduje dodatečné doplňky instalované v prohlížeči, což nemůže být zaručeno u všech klientských stanic.

3.2 Segmentace dokumentu

Jednou z možností jak snížit zátěž serveru a také snížit objem přenášených dat může být segmentace dokumentu. Při klasickém pojetí webové aplikace uživatelská akce vyvolá požadavek, který server zpracuje a pošle na klienta odpověď v podobě nové stránky. Ta je pak kompletně překreslena prohlížečem. Přitom v některých případech dojde ke změně pouze v malé části dokumentu. Řešením je použitím technologií AJAX k asynchronnímu požadavku na server, vygenerovat na straně serveru pouze změněnou část dokumentu a následně změnit příslušnou část DOM pomocí JavaScriptu tak, aby došlo k překreslení obsahu prohlížečem.

Na straně klienta v prohlížeči musí běžet *obslužná aplikace*, která spravuje zachycení akcí uživatele, následně tvorbu XHR požadavků, komunikaci se serverem, zpracování příchozích dat a správné začlenění těchto dat do dokumentu.

Problémem v tomto přístupu je identifikace části, kterou chceme změnit. Běžně se řeší tak, že porovnáním identifikátoru elementů se najde odpovídající část a nahradí se novými daty. Je však nutné důsledně dodržovat v celé aplikaci precizní označování elementů.

3.3 Generování HTML kódu

Další možností je přesun režie spojené s generováním HTML kódu na stranu klienta. Ve spojení s předchozí možností to umožňuje posílat pouze změněnou část dokumentu a navíc čistě ve formě strukturovaných dat (např. formátu XML, či JSON). Výše zmiňovaná obslužná aplikace, dále jen

kontrolér, pak vygeneruje HTML kód pro přijatá data a následně ho začlení do dokumentu. Tento proces můžeme rozdělit na několik částí. *Zachycení akce* uživatele, *komunikace se serverem*, který odešle odpověď, zpracování přijatých dat - *vygenerování HTML kódu* a jeho *začlenění do zobrazené stránky*.

3.3.1 Zachycení akce

Kontrolér musí být schopen zachytit akci uživatele a rozpoznat, jaká akce se má vykonat. Existují dva přístupy, které je možno použít.

Prvním je vytvořit pro všechny akce, které mají být obslouženy kontrolérem, speciální označení, např. přiřazením obslužné funkce na události typu „onclick“ či „onsubmit“, popř. použitím speciálního formátu atributu „href“. Tato možnost zaručuje, že kontrolér bude pracovat pouze s těmi akcemi, které budou explicitně definovány.

Druhým způsobem je pak vytvořit výše zmíněné akce pro všechny elementy schopné vytvořit požadavek a přenechat na kontroléru rozhodnutí, zda tuto událost obsloužit „klasickým způsobem“, kdy v odpovědi webového serveru bude opět kompletní stránka, nebo jestli aplikovat procesy pro vlastní vytvoření HTML kódu, změny segmentu stránky, apod.

Použití druhého způsobu považuji za univerzálnější řešení, navíc tato funkčnost navázání obslužných funkcí na jednotlivé elementy může být součástí kontroléru samotného a být aktivována při jeho inicializaci a tedy nezatěžuje vývojáře webové aplikace. Druhým kladem je to, že takto může vývojář aplikace vytvořit stejný kód, který bude fungovat i v případě, že funkce kontroléru nebudou k dispozici, např. budou zakázány na klientské stanici.

3.3.2 Komunikace se serverem

Způsob, jak rozlišit obsloužení přijaté zprávy kontrolérem, je možné zjistit analýzou přijatých dat. Aby bylo možné správně interpretovat přijatá data a vygenerovat HTML kód, je nutné vytvořit a dodržovat formát zprávy. V této práci navrhuji následující strukturu zasílaných zpráv:

- Identifikátor objektu
- Identifikátor nadřazeného objektu
- Datový typ objektu popisující jak budou data interpretována
- Požadovaná akce určující, jak bude s vygenerovaným kódem naloženo
- Struktura dat obsahující názvy, popisy sloupců
- Samotná data
- Vlastnosti objektu obsahující podrobnější informace o vzhledu a interpretaci dat

Výše uvedená struktura se může nepatrně měnit v závislosti na datovém typu objektu, např. některé datové typy (prostý text) nemusí pracovat se strukturou dat, apod.

Kontrolou přijatého formátu zprávy pak může kontrolér rozhodovat, jestli se jedná o běžnou stránku či o data vyžadující zpracování.

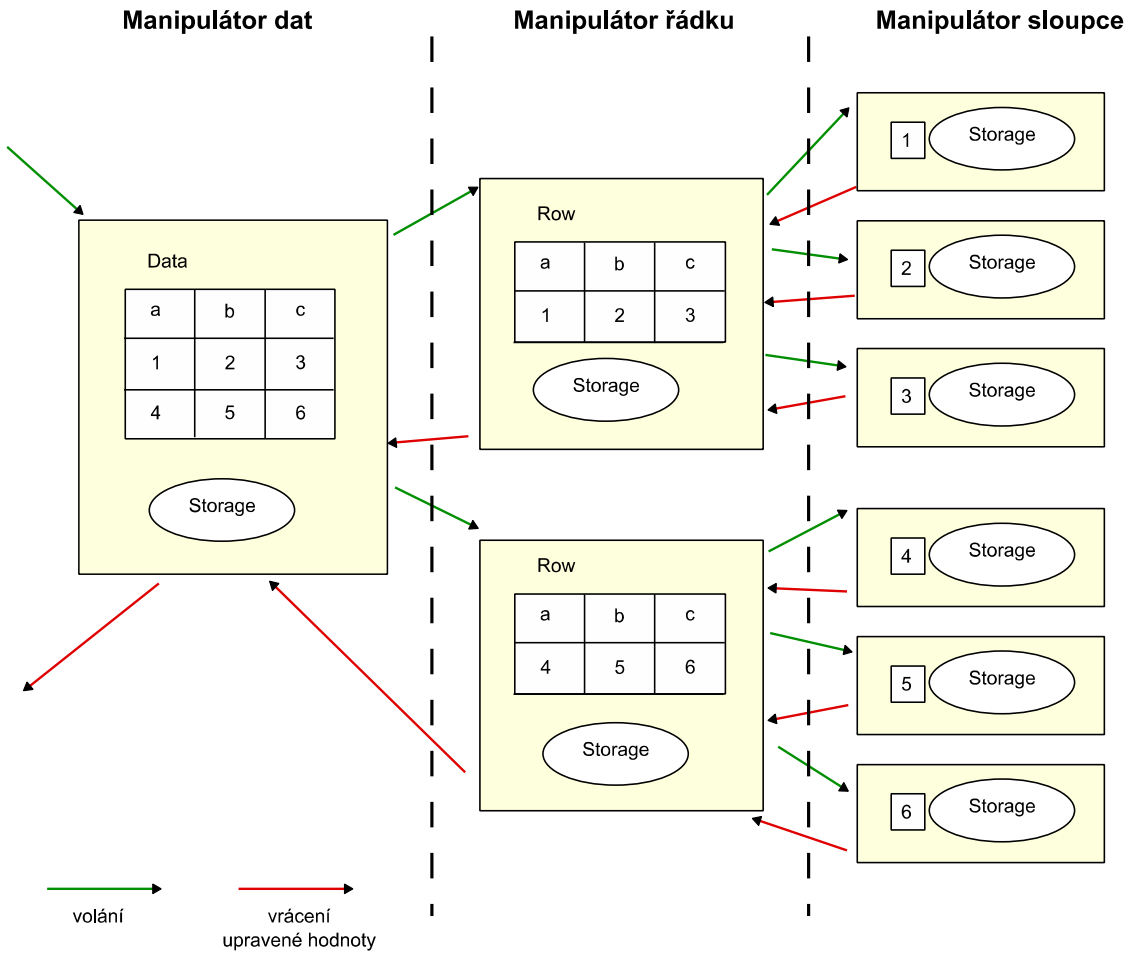
3.3.3 Možnosti přizpůsobení generování kódu

Po zpracování přijaté zprávy nastává fáze samotného generování HTML kódu. Každý datový typ má jinou HTML reprezentaci a také je nutné umožnit vývojáři, aby definoval vlastní pravidla při vytváření kódu. K tomuto účelu navrhuji systém uživatelsky definovatelných funkcí, tzv. *datových manipulátorů*, a *událostí*, které umožňují měnit podobu HTML výstupu.

Při popisu tohoto systému budu používat „tabulkový model“ generovaných dat. Jelikož každý datový typ může mít jinou strukturu dat a jejich použití, zaměřím se v popisu systému na typy, které zobrazují taková data, která se dají definovat tabulkou, resp. daty v podobě, v jaké jsou uloženy v relačních databázích. Generování kódu právě takovýchto prvků (kdy se v cyklu generují desítky prvků – řádků/položek) na klientu má stěžejní význam při přenosu zátěže ze serverů.

3.3.3.1 Systém datových manipulátorů

Proces generování HTML kódu je rozdělen na 3 části, každá část má svůj manipulátor. Kontrolér obsahuje výchozí manipulátory dat, které se implicitně používají, pokud uživatel tyto výchozí manipulátory nenahradí definováním vlastních. Manipulátory mají hierarchické uspořádání. Nejvyšší řád má manipulátor dat, následně manipulátor řádku a nejnižší má manipulátor sloupce. Výchozí manipulátory vždy spouštějí manipulátory nižších řádů (viz Obrázek 3.1). Pokud uživatel tento řetěz přeruší definováním vlastního manipulátoru, přebírá tím i řízení nad jejich voláním, implicitně se již nespouštějí.



Obrázek 3.1 Schéma hierarchie a činnosti datových manipulátorů

Manipulátor dat

Spouští se pouze jedenkrát pro celý proces generování, vytváří obalový prvek elementu. Řídí činnost manipulátorů nižších řádů a jeho výsledkem je kompletní HTML kód pro daný element, který je připraven na začlenění do dokumentu.

Pro každý řádek přijatých dat je volán manipulátor řádku, kterému manipulátor dat předává k dispozici jak data příslušného řádku, tak i prostor pro ukládání dat a proměnných v kontextu přesahujícím jednotlivé volání řádkových manipulátorů (v obrázku objekt `Storage`).

Manipulátor řádku

Manipulátor řádku přistupuje k jednotlivým řádkům dat, které dostává k dispozici od nadřazeného manipulátoru a vrací jejich HTML reprezentaci. Umožňuje uživatelsky definovat podobu jednoho záznamu, použít logiku pro ovlivnění vzhledu.

Výchozí manipulátor vytváří HTML kód, který obaluje jednotlivé řádky a řídí volání manipulátorů sloupců.

Vývojář má možnost definovat vlastní manipulátor pro každý sloupec, který je popsán ve struktuře dat. Řádkový manipulátor vyhledá definované sloupcové manipulátory, a pokud existují, tak při zpracování daného sloupce je volán místo implicitního manipulátoru tento uživatelem definovaný.

Stejně jako manipulátor dat, předává sloupcovému manipulátoru jak hodnotu samotnou, tak i prostor pro vlastní data.

Manipulátor sloupce

Manipulátor sloupce obsahuje modifikace, které se týkají všech hodnot v tomto sloupci. Implicitní manipulátor sloupce pouze vrací předanou hodnotu, kterou nijak nemodifikuje.

3.3.3.2 Systém událostí

Další možností, jak vývojář může ovlivnit vykonané akce a podobu HTML kódu je systém událostí. Kontrolér implementuje zjednodušený systém událostí, tedy možnost zaregistrovat si obslužnou funkci, která má být vykonána, pokud na určeném objektu nastane žádaný stav. Jsou možné dva přístupy.

První možností je umožnit vývojáři definovat si vlastní typy událostí. Při tomto přístupu je ovšem nutné, aby vývojář zasahoval větší měrou do kódu kontroléru. Potřebuje proto podrobnější znalost jeho fungování a také je tento přístup pro něj pracnější.

Druhou možností je nabídnout vývojáři sadu již vytvořených událostí, které jsou v kontroléru zakomponovány a kterých může jednoduše využít. Pokud se navrhne dostatečně velký výběr událostí, bude práce s nimi pohodlnější a vývoj rychlejší. Měl by však být kladen důraz také na optimalizaci, přílišné množství volání událostí může mít neblahý vliv na rychlost vykonávání operací kontroléru, což může na koncového uživatele aplikace působit nepříjemně.

3.3.4 Začlenění do dokumentu, typy akcí

Po dokončení generování HTML kódu je nutné, aby kontrolér data správně integroval do dokumentu. V navrženém systému zprávy k tomu slouží položky typu akce a identifikátoru nadřazeného objektu popsané v části *3.3.2*. V návrhu uvažuji následující možné akce:

1. Přidání objektu do nadřazeného (otcovského) objektu. Při této akci se vyhledá otcovský objekt a vygenerovaný kód se připojí k již stávajícímu obsahu. Tohoto lze využít k tomu, že v aplikaci si koncový uživatel může zobrazit např. více pohledů na stejná data nebo si zobrazit různé výběry dat pro srovnání apod. Dalším možným zlepšením může být přidání bližšího určení místa, kam přidávaný element umístit. Pro tento účel mohou např. sloužit přídavné identifikátory prvků, před které se má prvek vložit nebo za které se má připojit.
2. Nahrazení obsahu rodičovského objektu. Při této možnosti se vyhledá rodičovský objekt a celý jeho obsah je nahrazen vygenerovaným kódem.
3. Odstranění elementu či některé jeho části. Tato možnost může sloužit jednak k odstranění celého elementu z rodičovského, ale může být použita i např. k odstranění pouze určité části. Identifikace části by mohla být provedena porovnáním vygenerovaného HTML kódu a část elementu, která by takovému elementu odpovídala, by byla odstraněna.

3.3.5 Přírůstková data

Popisovaný koncept kontroléru také umožňuje tzv. přírůstková data. To jsou taková data, která se nemají zobrazit samostatně, ale mají být přidány k již existujícím, zobrazeným datům. Aby bylo možné poslat např. jeden řádek k již zobrazené tabulce bez nutnosti zasílat opětovně celou tabulku. Toho může být dosaženo přidáním nějakého zvláštního typu zprávy, ale v mém návrhu je to řešeno tak, že se před samotným procesem generování HTML kódu provede ověření typu rodičovského elementu a v případě, že tyto typy se shodují, provede se sjednocení přijatých dat s daty uloženými v kontroléru a následně se znovu vygeneruje HTML kód pro rodičovský element a v dokumentu nahradí ten původní.

3.4 Operace nad daty

Pokud použijeme výše uvedené principy, kdy jsou ve zprávách zasílány pouze strukturovaná data a nikoliv HTML kód, objem přenášených dat se značně sníží. V těchto případech vzniká prostor, aby se ve zprávě nezasílal pouze výtažek všech záznamů, ale aby se odeslala kompletní data, která jsou k dispozici. V takovém případě, má-li kontrolér k dispozici kompletní data, může s nimi provádět operace, které by v opačném případě vyžadovali zpracování serverem. Uživateli se zobrazí pouze výtažek dat, stejně jako při klasickém způsobu, ale kontrolér může udržovat na pozadí kompletní data a některé operace nad těmito daty vykonávat bez nutnosti komunikace se serverem. Následující kapitoly popisují operace, které se takto dají vykonávat. V popisu opět budu uvažovat „tabulkový model“ dat.

3.4.1 Stránkování

Z důvodů přehlednosti informací se většinou záznamy nezobrazují všechny současně, ale zobrazují se vždy po určitém počtu záznamů na jedné straně. Aby kontrolér nabídnul přístup ke všem záznamům, musí implementovat systém pro stránkování.

U každého objektu, který kontrolér udržuje v paměti, musí být uložen údaj o počtu záznamů zobrazených na jedné straně, stejně jako údaj o aktuálně zobrazené straně. Kontrolér musí nabízet obslužné funkce pro práci se stránkami, minimálně přechod na uživatelem určenou stranu. Dalšími možnostmi jsou funkce, které usnadňují práci se stranami, jako přechod na další, předchozí, první či poslední stranu, a také funkce pro změnu počtu záznamů zobrazených na jedné straně.

Kontrolér pak může nabízet automatické prostředky pro generování takových ovládacích prvků, které budou navázány na proces generování HTML kódu, ale měl by nabízet i podporu pro uživatelem definované ovládací prvky. Kontrolér by pak měl umožňovat propojení ovládacích prvků s příslušnými akcemi objektu, ale také zpětnou vazbu z objektu na tyto prvky, např. při přechodu na jinou stranu aktualizovat údaj o aktuální prohlížené straně. K tomuto účelu je možno použít výše

zmiňovaný systém událostí (viz *3.3.3.2*). Např. vygenerování HTML kódu spustí událost určitého typu, na kterou bude moci vývojář zaregistrovat obslužnou funkci, která aktualizuje číslo aktuálně zobrazené strany, které se nachází v jiném elementu stránky.

3.4.2 Řazení

Kontrolér může nad daty provádět operace řazení, ať už jednoduché řazení – podle jedné položky, či vícenásobné. U objektu musí být uložen údaj o sloupci, resp. sloupcích, podle kterých je aktuální pohled seřazen a údaj o sestupném či vzestupném pořadí.

Protože pro řazení různých druhů informací se používají různá pravidla (řazení čísel, dat, slov) je nutno znát povahu řazených dat. Buď uchovávat ve struktuře objektu u každého sloupce informaci, o jaká data se jedná, nebo před samotným řazením provést analýzu hodnot, které se ve sloupci nacházejí. V obou případech je nutné nejdříve určit porovnávací funkci, která bude při řazení použita.

Kontrolér pak musí nabídnout podobně jako v případě obsluhy stránkování vlastní, automaticky generované ovládací prvky (v případě tabulky je vhodná možnost využít záhlaví tabulky), ale i možnost zaregistrovat akce k externímu ovládání.

3.4.3 Vyhledávání

Operace vyhledávání se může provádět nad všemi daty či nad jejich vybranou podmnožinou, zpravidla hodnotami z konkrétních sloupců. Možností, jak uchovávat informaci o tom, který řádek zobrazit a který ne, je více. Jedním může být pomocné pole, ve kterém by byly uloženy indexy řádků zobrazených či nezobrazených. Nevýhoda této možnosti se však projeví v momentě, kdy nad takto vybranou podmnožinou budeme chtít provádět další operace – např. řazení. Při takových operacích řádky dat změní pořadí a uložené indexy by tak neodkazovaly na původní data. Řazení by pak muselo být speciálně upraveno.

Další možností může být příznak uložený u každého řádku označující, zda se má řádek zobrazit. Zde odpadá problém, který se vyskytoval u první možnosti v případě řazení, ovšem má tu nevýhodu, že při všech dalších operacích pracujeme vždy s celými daty. I kdyby zobrazená množina byla jeden, dva záznamy, operace se musí provést nad všemi záznamy a při fázi generování kódu pak zobrazovat jen řádky s příznakem. Navíc to komplikuje získávání údajů o počtu záznamů, stránkování, atd.

Podle mého názoru nejlepší možností je uchovávat data ve dvou strukturách – jedné struktuře pro *viditelná data* (tj. data, která budou zobrazena) a druhé struktuře pro data, která zobrazena nejsou (dále budu používat termín *skrytá data*). Kompletní data se získají sjednocením těchto dvou množin. Tímto se odstraňují výše uvedené nedostatky předchozích možností. Proces vyhledávání jednoduše rozhoduje, do které množiny se zkoumaný řádek přiřadí. Veškeré následné operace (řazení, stránkování) se provádějí naprosto stejně, jako by se prováděli nad kompletní množinou, tedy není

potřeba žádná úprava jejich funkce. Navíc se pracuje právě jen s takovou množinou, která odpovídá podmínkám vyhledávání a nedochází tak k nadbytečným operacím.

3.4.4 Filtrování

Základní princip filtrování je stejný jako u vyhledávání. Jde tedy o rozhodování, jestli řádek zařadit do struktury pro viditelná data. Ovšem umožňuje dva přístupy v závislosti na míře univerzálnosti, které chceme dosáhnout. Pokud by se funkce filtrů omezila pouze na logický součin podmínek pro právě jeden sloupec, tedy pouze typu $(a < X) \wedge (b > Y) \wedge (c < Z)$, mohlo by filtrování použít možnost prohledání podmínek sekvenčně. Následující porovnávání by se vždy provádělo již nad omezenou množinou dat a procházelo by menší množství dat.

V případě, že kontrolér bude podporovat i výrazy logického součtu typu $(a < X) \vee (a > Y) \vee (b > Z)$ již tuto možnost použít nelze. Vyhodnocování pak sestaví celkovou podmínku a jedním průchodem této podmínce podrobí každý jednotlivý řádek dat.

3.4.5 Problémy při práci s rozsáhlými daty

Přenesení všech záznamů však nemusí být vždy správným řešením z hlediska snížení zátěže serveru. Přenesení statisíců záznamů za účelem zobrazení několika málo řádku rozhodně nepřispěje k zamýšlenému snížení. Taková možnost je výhodná pouze v určitém datovém rozsahu. Zároveň je to ovšem ovlivněno uživatelskou aktivitou. Čím více operací uživatel nad těmito daty provádí, tím je výhodnější zaslat data kompletní (podrobnější informace a měření jsou uvedeny v kapitole *6 Testování*). Jako řešení navrhuji určitý druh umělé inteligence na straně serveru, která má k dispozici údaje o množství dat uložených v databázi a zároveň statistické informace o činnosti uživatelů a na základě těchto informací pak rozhoduje, zda bude přínosnější přenášet data celá nebo pouze jejich části.

Kontrolér na straně klienta pak musí umět rozpoznat, zda přijatá data má k dispozici kompletní, čehož se dá dosáhnout jednoduchým příznakem u zprávy. V případě kompletních dat může provádět výše zmíněné operace nad daty sám bez nutnosti komunikovat se serverem. V opačném případě každá operace vytvoří požadavek na server a kontrolér jen zobrazí obdržená data.

3.4.6 Aktuálnost dat

Problémem tohoto přístupu však zůstává aktuálnost dat. V aplikacích typu klient-server bývá obvyklé, že s aplikací pracuje současně více uživatelů, kteří mají právo data měnit. Při klasickém přístupu se při každé akci pracuje na straně serveru vždy s aktuálními daty. Pokud tyto funkce supluje kontrolér, pak pracuje vždy s daty starými z okamžiku prvotního požadavku. Změny provedené později se tak u uživatele neprojeví.

Možným řešením je nastavení časového limitu platnosti dat. Kontrolér si udržuje čas prvotního požadavku a porovnáním s nastaveným časovým limitem pak určí, jestli je nutné data aktualizovat nebo může pokračovat s původními daty.

Dalším řešením může být periodické dotazování serveru na aktuálnost dat, resp. na změny. To by ovšem způsobilo vyšší režii serveru spojenou s udržováním rozdílu v datech a s vyšším počtem spojení. Pokud by server udržoval časové značky všech změn dat, nad kterými připojení klienti pracují, mohl by následně aktualizovat data pomocí přírůstkových dat (viz *3.3.5*). Nebylo by pak nutné přenášet celý obsah znovu. V tomto případě ovšem nejsem zcela přesvědčen, že by se úspora vyrovnala oné vyšší režii.

Obecně lze tedy říci, že zasílání kompletních dat na klienta a přenechání operací nad nimi kontroléru není příliš vhodné u aplikací, kde dochází k častým změnám dat a zároveň se v nich pracuje s rozsáhlými daty.

3.5 Uložení stavu aplikace

Tento problém se vyskytuje obecně u aplikací využívající technologie AJAX či jiných skriptů, které mění obsah stránky.

Prvním problémem je uložení současného stavu do URL tak, aby uživatel mohl tento stav uložit a později se k tomuto stavu vrátit. Protože prohlížeče z bezpečnostních důvodů neumožňují měnit URL bez obnovení stránky (např. aby nemohlo dojít k záměně za stránky, které vypadají po vizuálně stránce velice podobně a snaží se nalákat uživatele, aby poskytl své přihlašovací údaje – tento způsob útoku je známý jako „*phising*“), k uložení stavů se využívá část URL zvaná fragment, která odkazuje na část dokumentu, a proto její změna nevyvolá obnovení stránky. Aplikace proto musí být schopna aktuální stav zakódovat do fragmentu URL a zpětně být schopna obnovit stejný stav z takového fragmentu.

Dalším problémem je funkce tlačítek prohlížeče zpět a vpřed. Použitím výše uvedeného postupu změny fragmentu URL sice prohlížeč ukládá jednotlivé změny jako položky historie a lze mezi nimi procházet, ale už není schopen uložit i stávající stav dokumentu. Aplikace si proto změnu stavu dokumentu musí obsloužit sama.

3.6 Správa jazykových mutací

Existuje několik přístupů, jak ve webových aplikacích řešit problematiku vícejazyčnosti. V podstatě se řešení této problematiky dají zobecnit tak, že aplikace používá nějaký druh symbolického zápisu výrazu, který je vyhledán ve slovníku s aktuálně zvolenou jazykovou mutací a nahrazen jeho překladem. Toto je režie, která se provádí několikrát v průběhu generování jedné stránky.

Je to jedna z funkcí, kterou lze vykonávat na straně klienta. Kontrolér si načte a uloží jazykový slovník. To lze provést v inicializační fázi, před samotným spuštěním, ale i za chodu kontroléru. Takto lze jednoduše přepínat jazykové mutace za chodu aplikace pouhým odesláním nového slovníku. Ze strany serveru pak přichází symbolické zápisy výrazů, které se při procesu generování HTML kódu překládají dle slovníku, který má kontrolér k dispozici.

3.7 Kontrola vstupních dat

Kontrola vstupních dat uživatele je další oblast, ve které se kontrolér může uplatnit. Zejména kontrola formulářových prvků, která může odhalit chyby v zadávání ještě před odesláním, jednak méně zatěžuje server a jednak je i pro uživatele pohodlnější (nedochází k obnovování formuláře či nutnosti zadávat údaje opětovně).

Následující body předpokládají typy vstupních polí formulářů podle návrhu konceptu HTML5 [4], tj. pole typu text, číslo, datum, datum a čas, heslo, email.

a) kontrola předepsaného formátu, který je možno definovat např. regulárním výrazem uloženým v atributu elementu, všech prvků. Např. v případě textu je možno využít omezení množiny znaků u uživatelského jména nebo minimální počet znaků hesla, apod.
b) kontrola minimální a maximální přípustné hodnoty u polí typu číslo, datum, datum a čas
c) kontrola vyplnění povinných položek, které jsou definovány atributem „required“
d) kontrola shodnosti polí heslo a ověření hesla

3.8 Novinky HTML5

Některé kontroly formulářových polí zmíněné v předchozí kapitole jsou již postupně implementovány do nejnovějších verzí nejpoužívanějších prohlížečů (Internet Explorer, Firefox, Google Chrome) v souvislosti s podporou pro připravovanou specifikaci HTML5. Kromě zmíněných kontrol se ve specifikaci HTML5 objevují i další body, které usnadňují funkce kontroléru a obdobných aplikací.

Koncept HTML5 představuje novou podobu objektu *History*, která umožňuje ukládání současného stavu dokumentu do paměti prohlížeče včetně „stavového objektu“, ve kterém je možné ukládat uživatelská data. Lze ukládat i změny URL adresy a nadpisu dokumentu bez nutnosti znovunačtení stránky. Tímto řeší výše zmiňované problémy, jak umožnit používání tlačítek zpět a vpřed v navigaci prohlížeči při používání technologií AJAX, stejně jako možnost uložení stávajícího stavu do záložek k pozdějšímu přístupu.

Dalšími novinkami, které koncept představuje, jsou lokální úložiště dat. Jedná se o úložiště, které jsou ukládány na klientské stanici v souborovém systému a jsou tak k dispozici v offline režimu i při opětovném navštívení webu. V prvé řadě je to úložiště typu klíč – hodnota *localeStorage* [5], které

se dá použít pro ukládání stavu klientských aplikací v podobě serializovaných dat. Dalším typem je pak lokální databáze *IndexDB* [6] pro uložení rozsáhlejších datových struktur s rychlejší implementací přístupu k datům a operací s nimi.

4 Vlastní implementace

4.1 Volba implementační platformy

Framework je napsán v jazyce JavaScript, čímž splňuje podmínku přenositelnosti kódu i podmínku podpory v běžných prohlížečích bez nutnosti instalace dodatečných doplňků. Rozhodl jsem využít volně šiřitelný JavaScriptový framework jQuery [8]. Tento framework obsahuje nástroje pro manipulaci s DOM a také poskytuje abstrakci některých funkcí, které mají různou implementaci v rozdílných prohlížečích.

Jako formát přenášených zpráv jsem zvolil JSON díky jeho malé režii na objem přenášených dat i rostoucí vestavěné podpoře jeho zpracování v moderních prohlížečích.

4.2 Formát zpráv

Podoba formátu zasílaných zpráv (viz Tabulka 4.1) vychází z analýzy uvedené v kapitole *3.3.2 Komunikace se serverem.* Obsah položek `structure` a `data` je proměnný v závislosti na datovém typu přenášeného objektu.

id	řetězec
className	řetězec
type	řetězec
settings	objekt **parentId**: řetězec **actionType**: řetězec **userRights**: objekt **move**: boolean **resize**: boolean **sort**: boolean **filters**: objekt **page**: číslo **listing**: číslo **search**: řetězec **searchIn**: pole řetězců **sort**: řetězec
structure	různé
data	různé

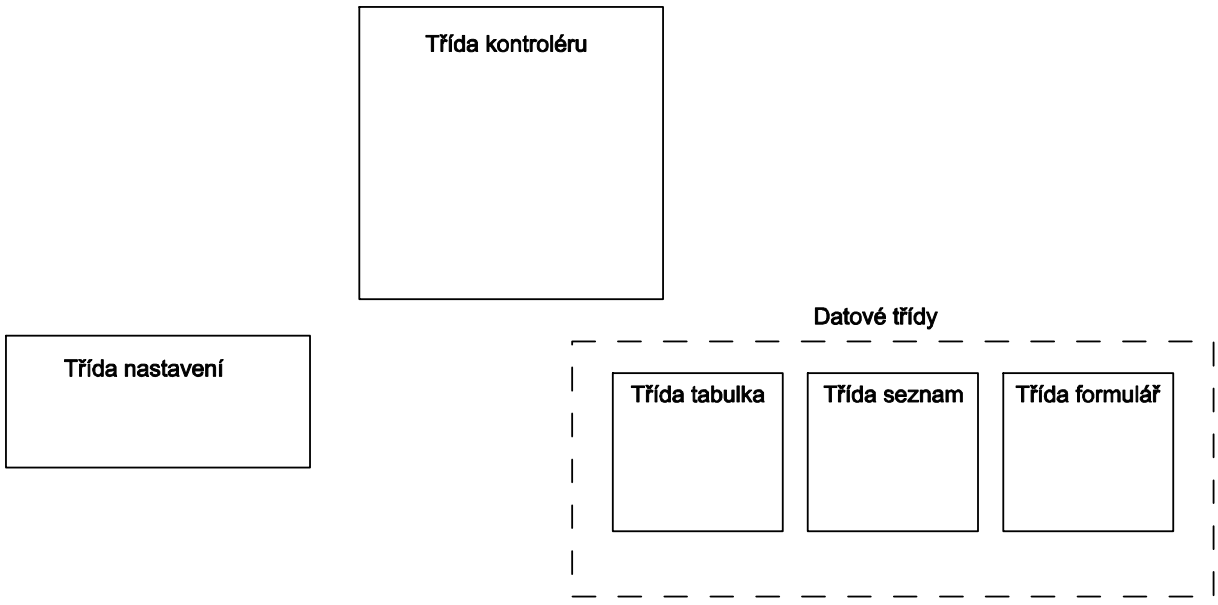
Tabulka 4.1 Návrh struktury zasílaných zpráv

Návrh frameworku obsahuje 4 datové typy. Typ tabulka, seznam, formulář, prostý text. Typy tabulka a seznam používají data v tabulkové podobě. Strukturou dat se rozumí počet sloupců, jejich identifikátory a jejich textová reprezentace. Samotná data pak reprezentuje dvojrozměrné pole hodnot. Tento přístup jsem zvolil kvůli tomu, že odpovídá systému uložení informací v relačních databázích. Pro webový server je tak velice jednoduché taková data připojit do zprávy, aniž by musel s nimi manipulovat.

Typ prostý text strukturu dat nepoužívá a jako data přenáší řetězec. Typ formulář má vlastní schéma dat. Obsahuje pole elementů, které formulář tvoří. Elementy jsou definovány jako objekty. Elementy pak obsahují atributy, které odpovídají skutečným HTML atributům, které tyto prvky mohou nabývat. Podrobnější schéma v kapitole *4.6.4 Třída pro formulář*.

4.3 Struktura frameworku

Framework se skládá ze tří skupin tříd (viz. Obrázek 4.1)



Obrázek 4.1 Struktura frameworku

První skupinu tvoří třída kontroléru, který funguje jako řídící prvek celé aplikace. Třída povoluje jedinou instanci. Kontrolér je jádrem celé aplikace, udržuje v paměti kolekce instancí ostatních tříd (nastavení, datových tříd), kontroluje chod všech procesů.

Další skupinu tvoří třídy datových objektů. Třídy datových objektů rozlišují různé typy zobrazení dat jako je tabulka, seznam, formulář, prostý text, apod. Datové třídy poskytují rozdílné implementace pro daný typ, přičemž zachovávají jednotné rozhraní, se kterým může kontrolér pracovat. Framework je navržen tak, aby šlo jednoduše – přidáním nové datové třídy – vytvořit nový způsob interpretace dat a rozšířit tak možnosti frameworku.

Poslední skupinu tvoří třída nastavení, která slouží k uchování podrobných informací o jednotlivých instancích datových tříd. Každá instance datové třídy má právě jednu instanci třídy nastavení.

4.4 Popis činnosti frameworku

Framework je tvořen jediným externím JavaScriptovým souborem, který se přidává do stránky. Po jeho přidání je nutné kontrolér inicializovat – vložit parametry běhu. Po této inicializaci je připraven k činnosti. Činnost můžeme rozdělit na fázi inicializační a fázi chodu aplikace.

Ve fázi inicializační dochází při spuštění kontroléru k obnově uloženého stavu. Stav uložený v URL je zpracován a kontrolér provede všechny operace tak, aby obnovil stav, při kterém byla tato URL uložena. Fáze inicializace kromě samotného spuštění kontroléru a obnovení stavu představuje i operace a procesy, které se provádějí ihned po spuštění (jsou vloženy přímo do kódu stránky). Může se jednat o operace načtení objektů nastavení, načtení jazykového slovníku, apod.

Ve fázi chodu aplikace pak dochází ke zpracování uživatelských akcí, komunikaci se serverem, generování HTML kódu, řazení objektů, vyhledávání v objektech, atd.

4.5 Třída kontroléru

Třída kontroléru `jSeCR_Controller` je jádrem celého frameworku. Řídí všechny operace, vytváří a udržuje instance ostatních tříd.

Třída je implementována jako singulární objekt, vytvořený pomocí literálu objektu. Tím je zajištěna kromě jedinečnosti instance také oddělení jmenného prostoru od ostatního kódu, který se může ve webové aplikaci vyskytovat.

4.5.1 Inicializace

Pro načtení kontroléru je nutné spustit inicializační metodu `init`, která jako argument přijímá objekt s nastavením kontroléru.

devMode	boolean – ladící mód
urlRoot	řetězec – adresa kořenového adresáře webové aplikace
processHashMilisecLimit	číslo – počet milisekund mezi XHR požadavky se stejnou URL v případě automatického obnovení stavu kontroléru (aby nedocházelo k duplicitám)

Tabulka 4.2 Položky nastavení kontroléru

Inicializační metoda po zpracování argumentu spustí proces obnovy stavu z aktuálně nastavené URL. Pokud v URL najde zakódovanou sekvenci příkazů (viz kapitola *4.5.6 Ukládání stavu aplikace*), sekvenci zpracuje a pokusí se obnovit předchozí stav. Součástí obnovy stavu mohou být jednak požadavky na vyžádání dat ze serveru a jednak operace, které byly s daty provedeny na straně klienta. V případě požadavku na server započne komunikace se serverem za účelem získání potřebných dat. Protože tento proces je asynchronní, zpracování sekvence pokračuje. Pokud se v sekvenci nachází operace s objektem, který ještě nebyl ze serveru získán, jsou tyto operace ignorovány. Po přijetí požadavku od serveru a vložení kódu do stránky se znovu spustí zpracování URL s parametrem, který zakazuje odesílání požadavků na server (aby nedošlo k zacyklení) a vykonají se ty operace, které byly dříve ignorovány kvůli nepřítomnosti objektu.

Protože může dojít k situaci, kdy ve stavu URL je uložena akce na získání stejných dat, která se následně ihned po inicializaci kontroléru provádí znovu (např. za účelem výchozího zobrazení dat), kontrolér si uchovává historii všech požadavků na server a pokud nově vzniklý požadavek je na stejná data již získána obnovováním stavu z URL a před časem kratším, než je časový limit (nastavitelný v parametrech při inicializaci), pak se tento požadavek nevykoná, aby nedocházelo ke zbytečně duplicitním komunikacím. Tento časový limit se nevztahuje na požadavky vzniklé uživatelskou akcí, pouze na případ automatické obnovy stavu.

Posledním krokem inicializační metody je navázání obslužné metody kontroléru `getUrl` na všechny odkazy vyskytující se v dokumentu. Tato metoda nahradí standardní chování prohlížeče asynchronním požadavkem a následným zpracováním příchozích dat.

4.5.2 Načtení nastavení

Další operace, kterou je možno zařadit do inicializační fáze je načtení objektů nastavení. Každý prvek má řadu unikátních vlastností, které ovlivňují jeho podobu. Principem návrhu je oddělení těchto vlastností do objektu nastavení, které se načítají do kontroléru po jeho inicializaci (např. vložením souboru do hlavičky stránky) tak, aby se tato data načetla pouze jednou na začátku a při posílání dalších požadavků, které obsahují data k zobrazení a kterých se předpokládá větší množství, už se používaly pouze odkazy na tyto vytvořené objekty. Proto kontrolér nabízí metody, jak uložit soubor takových objektů nastavení do paměti. Při načtení samotného datového objektu automaticky vyhledává dříve načtené prvky v kolekci objektů nastavení a ty pak používá.

Metoda pro načtení položek nastavení přijímá jeden argument, a to pole objektů nastavení, které budou podrobně popsány v kapitole *4.7 Třída nastavení*.

4.5.3 Systém správy jazyka

Dalším procesem inicializační fáze může být definování slovníku jazykové mutace. Kontrolér nabízí metodu k nastavení slovníku `setLangDef`, která přijímá pole typu klíč – hodnota, kde klíč je

symbolický zápis výrazu používaný v aplikaci a hodnota představuje jeho překlad v daném jazyce slovníku.

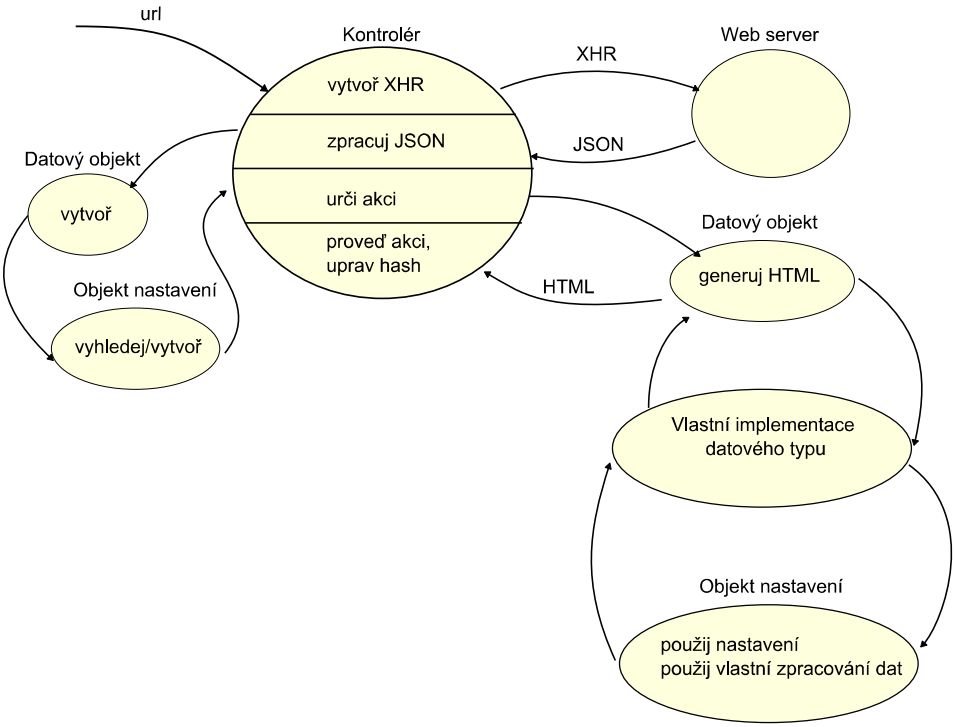
K použití slovníku pak slouží metoda `lang`, která očekává jeden parametr – symbolický zápis výrazu. Ten vyhledá ve slovníku a v případě nálezu vrací jeho překlad, v opačném případě generuje výjimku a vrací nezměněný argument.

4.5.4 Správa výjimek

Kontrolér implementuje jednoduchou správu výjimek. Procesy řízené kontrolérem odchytávají jak výjimky generované samotným JavaScriptem (typu index mimo rozsah pole, apod.) tak i výjimky generované vykonávanými metodami kontroléru. To slouží jako nástroj k ladění aplikace. Výjimky generované vlastními metodami používají formát „*název_metody(): Popis chyby*“. Všechny zachycené výjimky jsou zpracovány pomocnou ladící metodou kontroléru, která v případě, že kontrolér je inicializován se zapnutým přepínačem `devMode`, vypíše výjimku spolu s časovou značkou do nově vytvořeného elementu, který je připojen k tělu dokumentu.

4.5.5 Získávání dat a generování HTML kódu

Kontrolér řídí procesy zachycení uživatelské akce, komunikace se serverem, zpracování příchozích dat, generování HTML kódu a začlenění do stránky. Proces popisuje následující schéma (Obrázek 4.2).



Obrázek 4.2 Schéma činnosti kontroléru při získávání dat a generování HTML kódu

Při kliknutí uživatele na odkaz, se spustí vykonávání metody `getUrl`. Tato metoda přijímá 3 argumenty. První povinný argument je URL cíle, druhý je nepovinný příznak, zda se jedná o automatické volání z metody kontroléru či o uživatelskou akci – ve výchozím stavu uživatelská akce, třetí je nepovinný přepínač, zda požadavek maže všechny předchozí uložené stavy v URL. Kontrolér umí rozlišovat, jestli požadavek je hlavní tzn., že při obnovení stavu není nutné načítat předchozí požadavky, ale pouze tento poslední. Požadavky, které nejsou hlavní, se ukládání do URL fragmentu všechny a při obnovení jsou načteny každý v novém požadavku.

V dalším kroku je vytvořen XHR požadavek, který je zpracován na webovém serveru a v jeho odpovědi se nachází data k zobrazení. Kontrolér spouští metodu zpracování příchozích dat. Pokud se nejedná o data ve formátu JSON, došlo k situaci, kdy byl přijat text, který není určen pro zpracování kontrolérem, ale pro vykreslení prohlížečem. V takovém případě, kontrolér skriptem přesměruje adresu prohlížeče na danou URL. Tímto se sice znovu odešle stejný požadavek podruhé, ale nenalezl jsem jinou metodu, jak změnit adresu v prohlížeči a zároveň zobrazit přijatý obsah (kontrolér nevyužívá žádné vylepšení specifikace HTML5, kde by toto možné bylo). Situace, že by vývojář část aplikace vytvářel s podporou pro kontrolér a část klasickým způsobem, ovšem nepovažuji za příliš časté.

V případě, že v odpověď je určena ke zpracování (tj. je ve formátu JSON), provede se vytvoření instance datového objektu podle typu, který je uveden ve zprávě. Kontrolu položek příchozích dat pak obstarává konstruktor konkrétní datové třídy, jelikož každý typ může mít jiné požadavky na povinné položky zprávy. Kontrolér pouze zjistí, zda uvedený typ má zaregistrován jako podporovaný datový typ a volá příslušný konstruktor. V průběhu tvorby datového objektu je volána metoda kontroléru, která vyhledává v kolekci nastavení, zda pro tento objekt (na základně `id` nebo `className` objektu) již neexistuje načtené nastavení. Pokud ne, vytvoří se nová instance třídy nastavení s výchozími hodnotami. Pokud vše proběhne v pořádku, datový objekt je vytvořen. Pokud zpráva obsahuje dodatečné informace v položce `settings` (viz Tabulka 4.1), pak uloží tyto hodnoty k objektu. Touto cestou je možné i dodatečně přepsat hodnoty, které se načetly v inicializační fázi. Takto vytvořený objekt si kontrolér uloží do kolekce datových objektů.

Po uložení objektu kontrolér určí, jaká akce bude provedena na základě hodnoty v nastavení (ať už se nacházela ve zprávě nebo v souboru nastavení). Metoda reprezentující danou akci pak volá metodu datového objektu `generateHtml`, kterou všechny podtřídy datových typů dědí. Pak nastává samotné generování kódu, které se liší na implementaci každého datového typu (viz kapitola *4.6 Třídy datových typů*). Kontrolér obdrží vygenerovaný kód a provede příslušnou akci.

Kontrolér implementuje možnosti přidání k otcovskému prvku včetně podrobnějšího určení místa, kam data vložit a možnost přepsat otcovský element, tak jak popisuje kapitola *3.3.4 Začlenění do dokumentu, typy akcí*.

Po začlenění do stránky kontrolér uloží stav do fragmentu URL.

4.5.6 Ukládání stavu aplikace

Kontrolér ukládá každou vykonanou akci do fragmentu URL tak, aby bylo možno obnovit aktuální stav zpětným zpracováním tohoto fragmentu.

Fragment URL je oddělen od adresy znakem „#“ a jeho změny nezpůsobí nové načítání stránky prohlížečem. Kontrolér používá speciální formát zápisu, tj. fragment začíná znaky „`#!`“, což je standard pro vyhledávače, se kterým přišla společnost Google [10]. Princip spočívá v tom, že robot prohledávající odkazy, které obsahují fragment začínající na uvedené znaky, jej nahradí proměnnou `_escaped_fragment_` s hodnotou, která odpovídá textu za úvodními znaky. Tzn. fragment

`www.example.com#!state=homepage`

robot převede na

`www.example.com?_escaped_fragment_=state=homepage.`

Je tak možné na serveru vytvořit alternativní zobrazení pro roboty, které obsahem odpovídá stavu aplikace při použití AJAX a dalších akcí frameworku.

Po úvodních znacích následuje zápis akce. Jednotlivé akce jsou od sebe odděleny dalším znakem „`/`“. Zápis akce se skládá ze tří částí

a) část identifikace objektu – v případě operací s datovými objekty je to id příslušného objektu, v případě uložení komunikace se serverem je to relativní adresa požadavku určená podle kořenového URL nastaveného v kontroléru
b) část představující akci
   - `url` pro komunikaci se serverem
   - `search` pro vyhledávání
   - `searchIn` pro nastavení položky, ve které se provádí vyhledávání
   - `sortBy` pro řazení
   - `sortDesc` pro nastavení sestupného pořadí řazení
   - `page` pro přechod na určitou stranu
   - `listing` pro změnu počtu záznamů na jedné straně
c) část představující hodnotu akce (případně více hodnot oddělených separátorem)

Jednotlivé části jsou spojeny znakem „=“. Zápis jedné akce tedy vypadá následovně: `identifikátor=typ_akce=hodnota`. Pokud se v části identifikátoru nebo v části hodnoty vyskytuje znak „`/`“, který by způsoboval problém při zpětném pracování, je nahrazen zástupnou skupinou symbolů „`_|_`“, obdobně se nahrazuje znak „=“ za skupinu „`_||_`“.

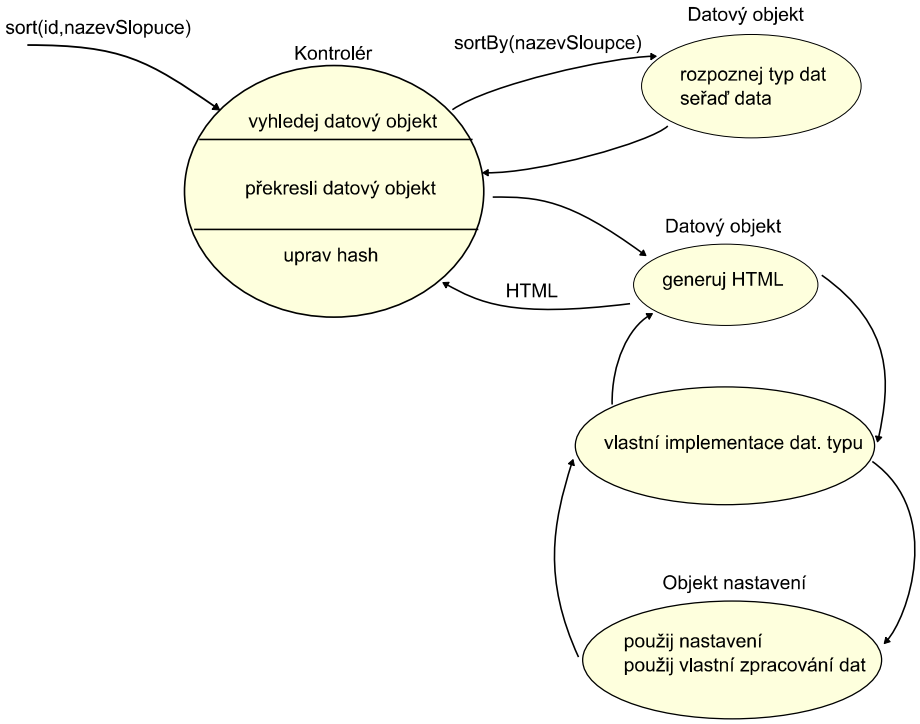
Příklad celého segmentu zaznamenaných akcí vypadá následovně:
`http://www.example.com#!/page1.php=url=/main_table=search=abc/main_table=sortBy=col1`

Tento zápis by byl interpretován tak, že by se vytvořil požadavek na `http://www.example.com/page1.php`, následně by byl vyhledán ve všech datech objektu `main_table` výraz „abc“ a poté by se výsledek seřadil podle sloupce `col1` vzestupně.

4.5.7 Operace stránkování, řazení, vyhledávání

Kontrolér implementuje operace stránkování, řazení, vyhledávání s přijatými daty tak, jak jsou popsány v kapitole *3.4 Operace nad daty*.

Činnost kontroléru při těchto operacích budu popisovat na operaci řazení (viz Obrázek 4.3).



Obrázek 4.3 Schéma činnosti kontroléru při operaci řazení

Je spuštěna akce na seřazení objektu `id` podle položky `nazevSloupce`. Kontrolér vyhledá v kolekci datový objekt s příslušným identifikátorem a předá mu řízení. Podle implementace datového typu pak proběhne seřazení dat.

Řízení přebírá opět kontrolér a znovu vygeneruje HTML kód (stejným způsobem jako je uvedeno v kapitole *4.5.5 Získávání dat a generování HTML kódu*). Kontrolér po získání vygenerovaného kódu nahradí element v dokumentu, který odpovídá danému datovému objektu.

V případě operace změny stránkování, tj. počtu záznamů zobrazených na jedné straně, a změny aktuálně zobrazené stránky je činnost kontroléru jednodušší. Po vyhledání objektu kontrolér pouze upraví příslušnou hodnotu v objektu nastavení, který přísluší pro zadaný datový objekt, a spustí překreslení objektu.

V případě vyhledávání je naopak činnost kontroléru mnohem složitější. Proces vyhledávání není řízen samotným datovým objektem, ale řídí ho sám kontrolér.

Metoda vyhledávání nejprve kontroluje samotný vyhledávaný výraz. Pokud je výraz prázdná hodnota, znamená to, že není třeba nic vyhledávat, ale naopak zobrazit všechny záznamy, kontrolér přesune všechny záznamy z pole skrytých dat (viz kapitola *3.4.3)* a překreslí objekt. Pokud je hodnota neprázdná, kontrolér rovněž přesune tyto záznamy (aby vyhledávání probíhalo opět nad celou množinou dat a ne jen nad podmnožinou z předchozího vyhledávání) a pokračuje v činnosti. Hledaný výraz je vyhledáván ve všech řádcích (v případě omezení jen v příslušných sloupcích) a indexy těchto řádků jsou uloženy v pomocném poli. Současně se ukládání do druhého pomocného pole indexy řádků, které podmínce nevyhověly. Výsledkem tohoto průchodu jsou dvě komplementární množiny indexů, jejichž sjednocením dostaneme kompletní data.

V dalším kroku kontrolér porovnává, které z pomocných polí je menší, aby se zvolil co nejefektivnější způsob přesunu řádků z viditelných dat. V případě, že množina nalezených záznamů je menší než množina nenalezených, znamená to, že do skrytých dat se přesunou všechna data a následně se přesunou řádky s indexy z pomocného pole nalezených hodnot zpět do pole viditelných dat.

V případě, že množina nalezených záznamů je naopak větší, pak se prochází viditelná data a řádky s indexy z pomocného pole nenalezených hodnot jsou přesunuty do pole skrytých dat. Poté je hledaný výraz uložen do nastavení objektu a objekt je překreslen.

Vždy po překreslení objektu se akce zapíše do fragmentu URL.

4.5.8 Systém událostí

Kontrolér implementuje systém událostí (viz kapitola *3.3.3.2*). Implementován je pomocí pole událostí, které je uloženo jako atribut kontroléru. Jedna událost je pak reprezentována opět polem, které má předepsanou strukturu:

1. Identifikátor objektu
2. Název události
3. Odkaz na obslužnou funkci
4. Pole argumentů, které budou předány obslužné funkci

Kontrolér obsahuje metodu `attachEvent`, která slouží k zaregistrování obsluhy události. Přijímá 3 povinné parametry, které odpovídají prvním třem bodům ve struktuře události. Všechny další parametry jsou spojeny do pole, které je uloženo do poslední položky v této struktuře.

Kontrolér rovněž obsahuje metodu `detachEvent`, která umožňuje zrušení nastavené obsluhy a také metodu `isAttachedEvent`, která slouží k zjištění, zda na objektu je již nastavena obsluha dané události.

Pro spuštění události slouží metoda `triggerEvent`, která jako parametr dostává identifikátor objektu, kde událost nastala, a její typ. Metoda projde pole uložených událostí a hledá, jestli je pro

tento objekt a tento typ zaregistrována obsluha. Pokud je obsluha nalezena, spustí se s uloženými parametry a její návratový kód je uložen do vnitřní proměnné. Metoda může spustit více obslužných funkcí na jednu událost, v takovém případě se návratové hodnoty všech funkcí spojí do jednoho řetězce, který metoda vrací.

4.5.9 Pomocné ladící metody

Pro usnadnění vývoje kontrolér nabízí tři pomocné ladící metody, metoda `dump` slouží ke strukturovanému výpisu objektů. Pomocí této metody lze vypsat vývojářem definovaný objekt, ovšem v případě použití bez parametrů, metoda vypisuje objekt samotného kontroléru.

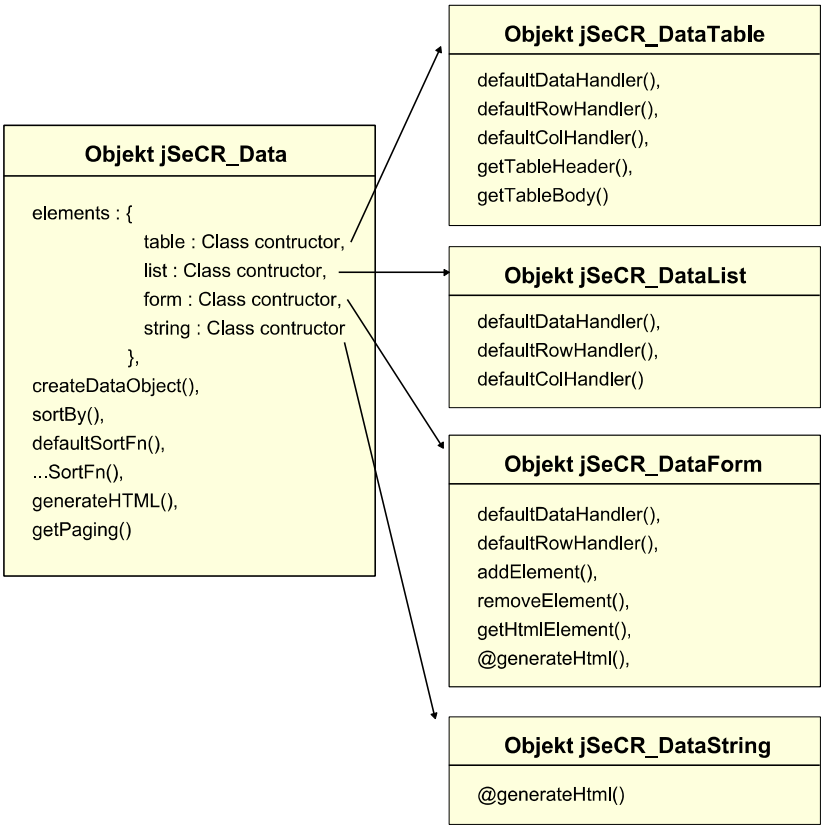
Další dvě ladící metody slouží ke správě odchycených událostí. V případě, že je kontrolér inicializován v ladícím módu, metody odchycené výjimky vypisují do „konzole“ – nového HTML elementu, s identifikátorem `jSeCR_debug_div`, který je připojen do dokumentu. Součástí práce je i externí soubor s kaskádovými styly, který je možno použít pro výchozí vzhled konzole.

4.6 Třídy datových typů

Reprezentaci přijatých dat v frameworku tvoří instance datových tříd. Datové třídy, které reprezentují konkrétní typy dat (tabulka, seznam) jsou potomky třídy `jSeCR_Data` (dále *hlavní datová třída*), která obsahuje metody, které se v různých implementacích nemění (jako např. řazení dat) a také metody, které definují rozhraní, s kterým může komunikovat kontrolér. Návrh je koncipován tak, aby bylo možné vytvořením nové třídy jednoduše vytvořit podporu pro nový datový typ – pokud jsou požadavky příliš odlišné tak, že nejde použít žádný již implementovaný typ ani se všemi možnostmi jak upravit proces generování HTML kódu (viz kapitola *3.3.3 Možnosti přizpůsobení generování kódu*).

Přidání nového typu se provádí jednak vytvořením nové třídy, která reprezentuje typ a která je potomkem hlavní datové třídy a jednak zaregistrováním podporovaného typu. Model struktury datových tříd (Obrázek 4.4) ukazuje, jakým způsobem jsou jednotlivé třídy zaregistrovány a také jakým způsobem se liší jejich implementace. Hlavní datová třída obsahuje atribut `elements`, kterým je objekt sdružující podporované typy. Název atributu představuje název datového typu, jeho hodnota pak odkaz na konstruktor třídy, která jej implementuje.

Při vytváření nového datového objektu kontrolérem je volána metoda hlavní datové třídy `createDataObject`, která kontroluje atribut `elements` a hledá příslušný typ. Pokud je typ zaregistrován, vrací novou instanci dané třídy, pokud není, generuje výjimku.



Obrázek 4.4 Struktura datových tříd

4.6.1 Hlavní datová třída

Hlavní datová třída obsahuje metody pro ukládání a nastavování svých položek (tzv. „getter/setter“ metody) a dále metody používané pro řazení dat. Řazení se provádí na úrovni dat uložených v dvojrozměrném poli (tedy „tabulkový model“), které používají všechny typy, u kterých řazení dává smysl (např. u prostého textu jsou data uložena pouze v řetězci a ne v poli, ale taková data nemá smysl řadit). Neovlivňuje je tedy různá implementace výstupu HTML kódu, který se liší u různých datových typů, a je tedy možné, aby tyto metody byly v hlavní datové třídě. Výhodou je, že jednotlivé typy již nemusí takovou operaci implementovat, pouze ji zdědí.

Hlavní datová třída implementuje řazení dat, jako je popsáno v kapitole *3.4.2 Řazení*. Obsahuje několik různých porovnávacích funkcí:

a) funkci pro porovnání řetězců (výchozí)

b) funkci pro porovnání čísel

c) funkci pro porovnání dat (ve formátu americkém 2011-12-31 a českém 31.12.2011)

Určení porovnávací funkce se děje pomocí regulárních výrazů, které se porovnávají s první nalezenou neprázdnou hodnotou.

Dále třída obsahuje metodu `generateHtml`, kterou volá kontrolér při generování HTML kódu. Metoda implementuje výchozí podobu této funkce, kterou si v případě potřeby děděná třída

může přetížit (např. třída formuláře a třída prostého textu). Tato metoda vyhledá v nastavení objektu, zda uživatel definoval vlastní manipulátor dat. Pokud ano, tak ho volá, pokud ne, spouští metodu výchozího manipulátoru, který je definován v podřízené třídě.

4.6.2 Třída pro tabulku

Třída pro tabulku je tvořená (mimo metody a atributy zděděné z hlavní třídy) metodami výchozích manipulátorů (dat, řádku, sloupce) a navíc metodami pro tvorbu hlavičky tabulky a těla tabulky. Výchozí manipulátor dat vytvoří obalovou značku tabulky a volá metodu pro hlavičku. Ta vytvoří řádek s hlavičkou, kde se nachází názvy sloupců uložené ve struktuře dat. (viz Tabulka 4.1). Dále je volána metoda pro tělo tabulky, ve které je logika manipulátoru dat popsána v kapitole *3.3.3.1 Systém datových manipulátorů*. V tomto místě jsou aplikovány i hodnoty uložené v nastavení, např. aktuální strana a počet záznamů na straně, které řídí cyklus volání řádkových manipulátorů.

Řádkový manipulátor tvoří HTML značky pro jednotlivé řádky a volá sloupcové manipulátory.

Při tvorbě kódu jsou elementům nastavovány atributy třídy tak, aby bylo co nejsnazší přizpůsobit vzhled tabulky pomocí kaskádových stylů. Celá tabulka dostává třídu, každý řádek dostává třídu s číslem řádku (možnost odlišení vzhledu prvního řádku, každého desátého řádku, apod.) a třídu označující sudý resp. lichý řádek, každá buňka dostává třídu s identifikátorem sloupce.

4.6.3 Třída pro seznam

Třída pro seznam implementuje pouze metody výchozích manipulátorů, které tvoří příslušné značky pro seznam a jeho položky. Proces řízení cyklu volání řádkových manipulátorů je stejný jako v případě tabulky. Také jako v případě tabulky dostává třídu celý seznam, každá položka seznamu dostává třídu s číslem řádku (položky) a třídu označující lichý resp. sudý řádek.

4.6.4 Třída pro formulář

Třída pro formulář používá jiný model dat, než předchozí dvě třídy. Nepoužívá strukturu dat a data tvoří pole elementů. Element tvoří objekt s jedním povinným atributem `type`, který určuje, o jaký prvek jde.

Typ elementu může být jednoduchý, tj. takový typ, který neobsahuje další zanořené elementy. Mezi jednoduché typy, které třída implementuje, patří: `textarea`, `button`, `option`, `text`, `password`, `radio`, `checkbox`, `submit`, `number`, `email`, `url`, `range`, `date`, `month`, `week`, `time`, `datetime`, `datetime-local`, `color`,`search`. Jednotlivé typy jsou navrženy podle specifikace HTML5 [4]. Tyto typy elementů obsahují takové atributy, které odpovídají jejich HTML reprezentaci, resp. atributy jsou pouze přetransformovány na HTML značky. Výjimku tvoří atribut `label`, jehož použití netvoří atribut elementu samotného, ale vytvoří před ním nový HTML element <`label`>.

Dále implementuje složité typy, což jsou elementy, které mohou obsahovat libovolný počet dalších elementů. Mezi tyto typy patří: `fieldset`, `optgroup` a `select`. Typ `fieldset` obsahuje atribut `elements`, který je tvořen opět polem elementů, které tento `fieldset` obaluje. Typy `select` a `optgroup` mají atribut `options`, který obsahuje pole elementů typu `option`.

Třída přetěžuje metodu `generateHtml`, kvůli odlišnému přístupu k datům. Obsahuje metodu výchozího datového i řádkového manipulátoru. Jako řádek se vnímá pořadí elementu v první úrovni zanoření. Třída dále obsahuje metodu pro přidání nového elementu do své kolekce a naopak pro odebrání elementu (který je identifikován atributem `id`). Tyto metody nejsou přímo používány kontrolérem, ale jejich funkčnost je možno využít při tvorbě dalších skriptů, které běží v aplikaci.

4.6.5 Třída prostého textu

Tato třída slouží pro použití v případech popsaných v kapitole 3.2 Segmentace dokumentu. Je implementována prostým přetížením metody `generateHtml`, která jen vrací hodnotu uloženou v atributu `data`.

4.7 Třída nastavení

Třída nastavení (`jSeCR_Settings`) slouží k uchovávání informací datového objektu. Obsahuje jedinou metodu, a to metodu sloužící k uložení hodnot, které metoda dostává v objektu. Argument je porovnáván se strukturou třídy a odpovídající položky jsou nastaveny.

Struktura dat je podobná struktuře položky `settings` ve formátu zasílané zprávy (viz Tabulka 4.1), ale přidává položku `dataHandlers` (viz Tabulka 4.3), která umožňuje definovat vlastní manipulátory dat.

id	řetězec
className	řetězec
parentId	řetězec
actionType	řetězec
userRights	objekt **move** boolean **resize** boolean **sort** boolean

filters	objekt **page** číslo **listing** číslo **search** řetězec **searchIn** pole řetězců **sort** řetězec **sortDesc** boolean
dataHandlers	objekt **dataHandler** funkce **rowHandler** funkce **colHandler** objekt **col_id_1** funkce **col_id_2** funkce **…** funkce

Tabulka 4.3 Schéma atributů třídy nastavení

Popis atributů

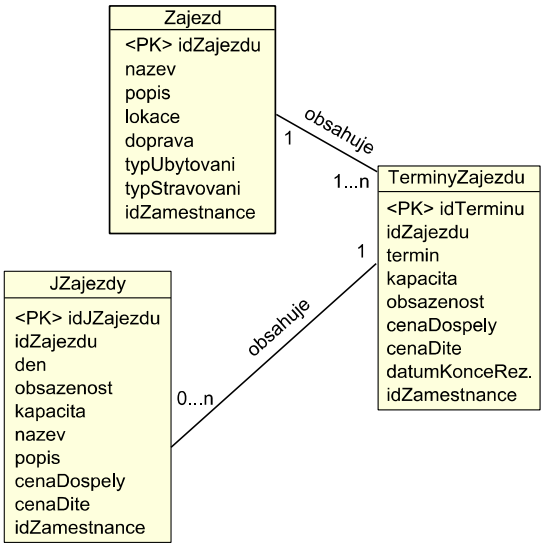
- *id* představuje identifikátor datového objektu
- *className* značí třídu objektu (v případě dynamicky generovaných prvků, u kterých není předem známé id, se použije nastavení s odpovídajícím atributem className)
- *parentId* představuje identifikátor otcovského objektu (elementu v HTML dokumentu)
- *actionType* definuje akci, která má být s prvkem provedena
- *userRights* nastavuje práva uživatele
    - *move* – umožnit uživateli přesunovat element
    - *resize* – umožnit uživateli měnit velikost elementu
    - *sort* – umožnit uživateli řadit data prvku
- *filters*
    - *page* – aktuálně zobrazená strana
    - *listing* – počet záznamů na jedné straně
    - *search* – aktuálně vyhledaný řetězec
    - *searchIn* – pole identifikátorů sloupců, ve kterých vyhledávat data
    - *sort* – identifikátor sloupce, podle kterého je aktuálně seřazeno
    - *sortDesc* – přepínač označující, zda je seřazeno sestupně
- *dataHandlers*
    - *dataHandler* – manipulátor dat
    - *rowHandler* – manipulátor řádku

- *colHandler* – manipulátory sloupce (Tvoří je objekt, který obsahuje metody - obslužné funkce. Název metody označuje identifikátor sloupce, na který má být aplikována)

5 Implementace testovacího informačního systému

Jako demonstrační aplikaci jsem zvolil kostru staršího školního projektu. Jedná se o velice zjednodušený systém cestovní kanceláře. Pro účely testování frameworku jsem jej zjednodušil na pouhé zobrazení výpisu zájezdů, které jsou uloženy v databázi.

Informace se skládají z několika tabulek. (viz Obrázek 5.1) Jedna tabulka obsahuje údaje o zájezdu (název zájezdu, místo, typ ubytování, apod.), další tabulka obsahuje konkrétní termíny zájezdu a informace spojené s nimi (data odjezdu, příjezdu, cena, atd.). V další tabulce jsou uloženy jednodenní výlety. Mezi tabulkami platí vztah, že jeden zájezd může mít více termínů a jeden termín více jednodenních výletů.



Obrázek 5.1 ER diagram testovacího systému

Pro vygenerování reprezentativního vzorku dat jsem analyzoval stránky cestovních kanceláří (www.alexandria.cz, www.victoria-ck.cz) a jako testovací vzorek dat jsem zvolil cca 2000 termínů, které jsem pomocí php skriptu náhodně vygeneroval v rámci určitých parametrů.

Úvodní strana systému je klasický HTML dokument. Slouží k načtení kontroléru a inicializační fázi – načtení jazykového slovníku, načtení souboru s nastaveními datových typů.

Nabídka systému obsahuje 4 odkazy. Dva slouží pro načtení datových objektů, jeden typu tabulka a druhý typu seznam, další dva slouží jako přepínače jazykových mutací a reprezentují je ikony vlaječek.

Zobrazení dat ve formě seznamu načte všechny záznamy termínů a zobrazí první stranu. V tomto zobrazení je použito definování vlastního manipulátoru řádku, nastavení počtu záznamů na stránce. Před samotný seznam je pak vložen objekt typu prostý text, který obsahuje ovládací prvky a

pomocí události zaregistrované v souboru `settings.js` se po vložení prvku naváže ovládání stránkování (pomocí události `elementAdded`). Zobrazení umožňuje řazení dat podle položek Název, Země, Cena, dále umožňuje vyhledávání v datech, přechody mezi stranami a změnu počtu záznamů na jedné straně.

Druhým zobrazením je forma tabulky. Zobrazuje obsahově stejný vzorek dat, jako v případě seznamu, jen s úpravou některých sloupců (např. neobsahuje položku popis). Tabulka je vytvořena pouze s úpravou sloupce `id`, na který je použit manipulátor sloupce, který mění všechna data na odkazy, které by vedly na podrobnosti o zájezdu. Jedná se o demonstraci použití sloupcového manipulátoru. Jinak je tabulka ponechána ve výchozí podobě. Výchozí podoba tabulky obsahuje na posledním řádku ovládací prvky pro přechody mezi stránkami a pro změnu počtu záznamů na jedné straně. Dále výchozí tabulka nabízí možnost seřadit data podle libovolného sloupce kliknutím na jeho název v záhlaví tabulky.

Další odkazy, které reprezentují ikony vlaječek, slouží k načtení jazykových slovníků – českého a anglického.

Tento jednoduchý systém ukazuje možnost praktického využití navrženého frameworku v reálných webových aplikacích. Na tomto systému jsem také provedl testování, jakým způsobem framework ovlivňuje zatížení serveru.

6 Testování

6.1 Metodologie testování

Testování bylo prováděno na výše uvedeném testovacím systému. Databáze obsahovala 2094 záznamů jednotlivých termínů zájezdů, které byly předmětem zobrazení.

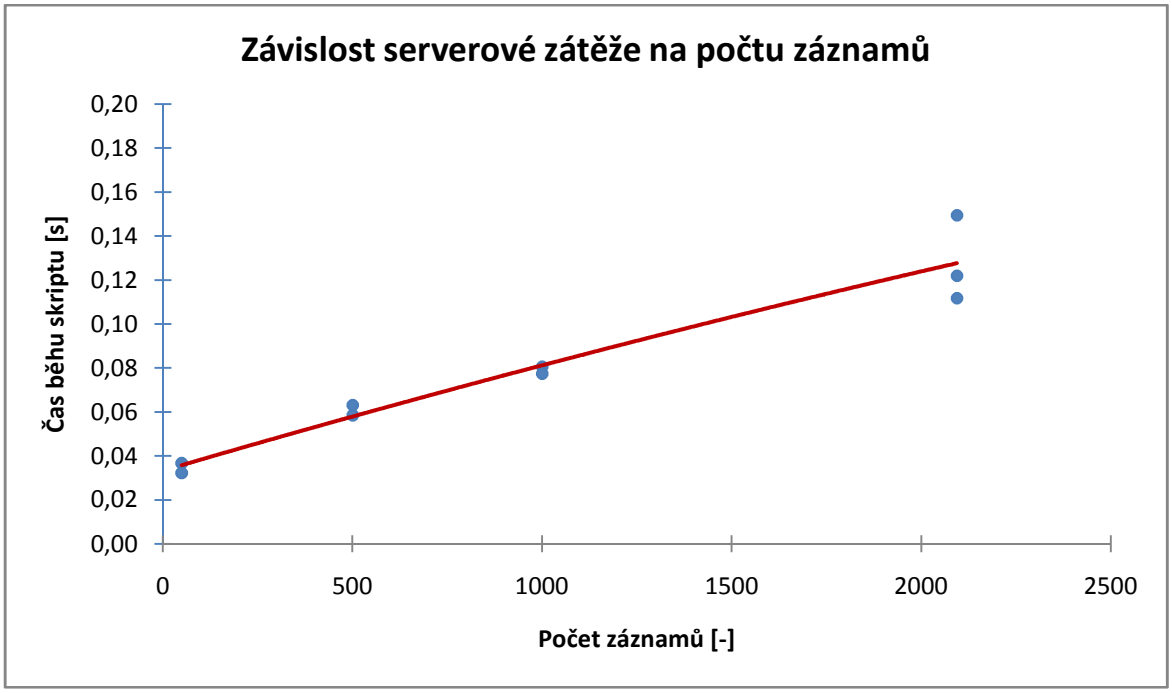
Test byl prováděn měřením reálného času serveru při zahájení skriptu a při jeho ukončení a výsledný rozdíl představoval čas nutný k obsloužení požadavku. Tato položka představuje zátěž serveru potřebnou ke generování jednoho požadavku. Do tohoto času je započítán i čas obsluhy databázového serveru a komunikace mezi webovým a databázovým serverem.

Test byl prováděn na notebooku HP6735s (CPU RM-70, 4 GB RAM, HDD 5400 ot./min, OS WIN 7 x64), tedy na úrovni notebooků určených pro běžné kancelářské využití. Oba servery byly spuštěny na lokálním počítači.

Objem přenášených dat nebyl současní měření. V případě seznamového zobrazení data zabírala 551,7 KB, v případě tabulkového zobrazení 297,4 KB. Nejedná se tedy o extrémní hodnoty a jejich objem by neměl tedy představovat větší problém.

6.2 Výsledky měření

První měření jsem provedl na určení závislosti zatížení serveru na množství dat, které má zpracovat. Zkoumal jsem dobu běhu skriptu při zpracování 50, 500, 1000 a 2094 záznamů. Zpracováním se rozumí získání daného počtu záznamů z databáze a jejich odeslání ve zprávě ve formě popsané v části *4.2 Formát zpráv*.Výsledky jsem zanesl do grafu (Obrázek 6.1) a proložil polygonální spojnicí trendu 2. řádu. Z výsledků lze usoudit, že závislost doby běhu skriptu je lineárně závislá na počtu záznamů.



Obrázek 6.1 Graf závislosti serverové zátěže na počtu záznamů

Další měření jsem prováděl na určení vztahu mezi zátěží serveru při použití frameworku a při jeho nepoužití. Průměrná doba běhu skriptu při použití klasické metody (odesílání vzorku vždy 50 záznamů na straně) byla v průměru $t_1 \cong 72{,}83\, ms$. Při použití frameworku (odesílání kompletního vzorku 2084 záznamů) byla průměrná doba běhu skriptu $t_2 \cong 127{,}77\, ms$.

Z výsledku tohoto měření tedy vyplynulo, že použití frameworku snižuje zátěž serverů (webového i databázového) v případě, že uživatel provede jakoukoliv akci, kterou je schopen obsloužit kontrolér. Míra snížení zátěže pak závisí na aktivitě uživatele a počtu operací (seřazení, vyhledání, listování stránek). Přesné procento snížení zátěže tedy není možno stanovit bez statistických podkladů, které by monitorovaly uživatelskou činnost na takovém systému, ale myslím, že je možné předpokládat, že uživatel alespoň jednu akci provede.

7 Závěr

Cílem práce bylo prozkoumat možnosti přesunu zátěže ze serverů na klienty ve webových aplikacích a pokusit se tyto poznatky uvést do praxe vytvořením JavaScriptového frameworku, který nabízí některé ze zmíněných návrhů. Výsledný framework se ukázal jako prakticky použitelný nástroj a měření ukázalo, že při reálném datovém rozsahu aplikace je schopen snížit zátěž serveru, i když konkrétní čísla by vyžadovala hlubší analýzu činnosti uživatelů.

Framework neimplementuje všechny návrhy, které byly popsány v teoretickém rozboru problému, zejm. filtrování dat, kontrolu formulářových prvků a odstranění části datových objektů. Také neřeší problém aktuálnosti dat. Toto jsou vhodná rozšíření, která by přispěla k větší praktičnosti a použitelnosti frameworku.

Jako další možná rozšíření bych jmenoval zjednodušení systému přidávání datových tříd, aby vývojář mohl snadno vytvořit typy přesně odpovídající potřebám jeho aplikace, dále rozšíření použití systému událostí pro zvýšení flexibility běhu aplikace či možnost jednoduše přidávat vlastní typy událostí.

Velmi zajímavým rozšířením mohou být i modifikace využívající novinky specifikace HTML5, které jsou popsány v analýze.

8 Literatura

[1] GARRETT, Jesse James. *Adaptive path* [online]. 2005-02-18 [cit. 2011-05-02]. Ajax: A New Approach to Web Applications. Dostupné z WWW: http://www.adaptivepath.com/ideas/e000385

[2] *World Wide Web Consortium* [online]. 1999 [cit. 2011-05-02]. Web Content Accessibility Guidelines 1.0. Dostupné z WWW: http://www.w3.org/TR/WCAG10/

[3] *World Wide Web Consortium* [online]. 2008 [cit. 2011-05-02]. Web Content Accessibility Guidelines (WCAG) 2.0. Dostupné z WWW: http://www.w3.org/TR/WCAG20/

[4] *World Wide Web Consortium* [online]. 2011 [cit. 2011-05-04]. HTML5. Dostupné z WWW: http://www.w3.org/TR/html5/

[5] *World Wide Web Consortium* [online]. 2011 [cit. 2011-05-04]. Web Storage. Dostupné z WWW: http://dev.w3.org/html5/webstorage/ .

[6] *World Wide Web Consortium* [online]. 2011 [cit. 2011-05-04]. Indexed Database API. Dostupné z WWW: http://www.w3.org/TR/IndexedDB/

[7] DUFFY, Scott. *How to do everything with JavaScript*. USA : Osborne / McGraw Hill, 2003. 349 s. ISBN 0-07-222887-3

[8] *JQuery Documentation* [online]. c2010 [cit. 2011-05-09]. Dostupné z WWW: http://docs.jquery.com/Main_Page

[9] HAUSER, Marianne; HAUSER, Tobias; WENZ, Christian. *HTML a CSS : Velká kniha řešení*. Brno : Computer Press, a.s., 2006. 912 s. ISBN 80-251-1117-2

[10] SANTAMARIA, Jose Maria Arranz. *ItsNat: Natural AJAX. Component based Java Web Application Framework* [online]. 2011-01-05 [cit. 2011-05-12]. The Single Page Interface Manifesto . Dostupné z WWW: http://itsnat.sourceforge.net/php/spim/spi_manifesto_en.php