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

VIZUALIZACE HYPERLINEK

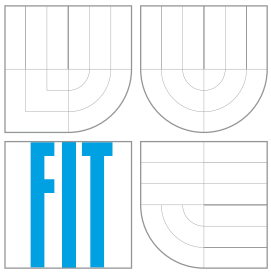
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

MARTIN SKOPAL

BRNO 2010

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

VIZUALIZACE HYPERLINEK

VISUALIZATION OF HYPERLINKS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — MARTIN SKOPAL
AUTHOR

VEDOUCÍ PRÁCE — Ing. RUDOLF KAJAN
SUPERVISOR

BRNO 2010

Abstrakt

Tato bakalářská práce se zabývá vytvořením náhledu hypertextového odkazu, za pomocí využití renderovacích jader. K řešení tohoto problému využívá renderovací jádra Trident a Webkit. Taktéž vysvětluje jednotlivé technologie, které se běžně vyskytují na moderních webových stránkách, a poukazuje na problémy, které mohou nastat při implementaci. Ve výsledcích je poukázano na odlišnosti, které mohou nastat při renderování, zejména podpora technologií HTML5 nebo vektorových obrázků SVG.

Abstract

This Bachelor's thesis deals with creating thumbnail from hypertext link, through the use of layout engines. To solve this problem it uses Trident and Webkit layout engine. Also explains the different technologies that are commonly used on modern web pages and highlights the problems that may arise during the implementation. In results are explained problems that can occur when rendering, especially support of HTML5 and SVG vector images.

Klíčová slova

ASP.NET, Renderovací jádro, Webkit, Trident, Webová služba

Keywords

ASP.NET, Layout engine, Webkit, Trident, Web service

Citace

Martin Skopal: Vizualizace hyperlinek, bakalářská práce, Brno, FIT VUT v Brně, 2010

Vizualizace hyperlinek

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Rudolfa Kajana

.....
Martin Skopal
17. mája 2010

Poděkování

Rád bych poděkoval vedoucímu Ing. Rudolfa Kajanovi, který měl počas vývoje vhodné připomínky, které pomohli k úspěšnému vypracování.

Obsah

Kapitola 1

Úvod

V tejto bakalárskej práci sa budem zaoberať rôznymi metódami, ktoré sú vhodné na vizualizáciu hypertextových odkazov. Taktiež predstavím ich implementáciu a porovnám ich možnosti.

Pre korektné vytvorenie obrázku z URL adresy je potrebné v pamäti počítača pristúpiť k obsahu webovej stránky, ten následne spracovať do grafickej podoby a až potom môžeme odoslať vytvorený obrázok. Práve toto spracovanie za nás obstaráva renderovacie jadro.

Prehľad jednotlivých technológií, ktoré sa na webových stránkach bežne vyskytujú sú popísané v kapitole 2. V tejto kapitole sú taktiež uvedené programové nástroje, ktoré boli využité počas implementácie služby. Zahrnuté je aj popis niekoľkých komplexných testových systémov (Acid a Javascript testy), ktoré majú za úlohu overiť výkon a správnosť renderovania webových stránok.

Ďalšia kapitola je venovaná kompletnej analýze problematiky renderovania. Sú v nej vysvetlené jednotlivé časti aplikácie. Taktiež je zdôvodnené, prečo neboli niektoré časti zahrnuté do systému a ktoré sú implementované nad rámec tejto problematiky.

V kapitole 4 je podrobne vysvetlený implementačný postup jednotlivých komponent. Sú spomenuté aj problémy, ktoré boli objavené počas celkovej implementácie.

V predposlednej kapitole sú zhodnotené jednotlivé výsledky, ktoré služba dosahuje a je poukázané na jednotlivé problémy, ktoré sa môžu vyskytnúť počas renderovania stránky. Výsledky a možnosti implementovaný jadier sú podrobne porovnané a rozdelené do niekoľkých kategórií. Na koniec sů popísane možné rozšírenia, ktoré by mohli byť neskôr zahrnuté.

Záver práce je venovaný kompletnému zhrnutiu a zhodnoteniu práce ako celok.

Kapitola 2

Použité technológie a nástroje

Táto kapitola je venovaná jednotlivým technológiam a nástrojom, ktoré boli využité pri tvorbe tejto práci. Jednotlivé technológie sú predovšetkým popísané z ich teoretického hľadiska.

2.1 HTML

HyperText Markup Language je prevládajúci značkovací jazyk určený na tvorbu web stránok. Dovoľuje vkladať obrázky a rôzne objekty do tela dokumentu a taktiež vytvárať interaktívne formuláre pomocou využitia špeciálnych značiek – *tagov*. Aktuálna verzia HTML je 4.01, v príprave je verzia 5. Dokumenty v jazyku HTML majú predpísanú štruktúru:

1. Deklarácia typu dokumentu
2. koreňový element `<html>`
3. Hlavička dokumentu `<head>`, ktorá obsahuje metadáta pre celý dokument (napr. typ kódovania, použitý jazyk)
4. Telo dokumentu, ktoré obsahuje samotný text dokumentu.

HTML tagy môžeme rozdeliť do viacerých typov. Štrukturálne – `<p>, <h1>`..., ktoré navrhujú štruktúru dokumentu. Prezentačné – `<strong>, <em>`..., ktoré popisujú výzor dokumentu a *hypertextové odkazy*, ktoré umožňujú vytvárať odkazy na ďalšie dokumenty [11].

Medzi značkovacie jazyky zaraďujeme aj jazyk XHTML(Extensible Hypertext Markup Language). Na rozdiel od jazyka HTML, ktorý vychádza zo špecifikácie SGML, jazyk XHTML je založený na XML špecifikácii. Tento fakt spôsobuje, že XHTML má prísnejšiu syntax ako HTML. Jeden z hlavých rozdielov je, že v jazyku XHTML musia byť všetky tagy uzatvorené (vrátane nepárových tagov).

HTML5

Je nový koncept jazyka HTML. Predstavuje nové možnosti pre vývoj webových aplikácií. Jednou z noviniek je tag `<video>`, určený na vkladanie video súborov do dokumentu HTML. V súčasnosti sa pre vkladanie videa vo väčšine prípadov používa Adobe Flash player. Medzi ďalšie novinky patrí napríklad tag `<canvas>`, ktorý slúži na dynamické vykresľovanie 2D objektov a bitmapových obrázkov. Taktiež predstavuje nové sémantické tagy (`<nav>, <article>`...).

2.2 Renderovanie HTML

Renderovanie HTML značí analýzu tagov ako sú napr. `<strong>`, `<div>`, `<em>`... následné ich spojenie s textom a CSS štýlmi, interpretovanie vloženého javascriptového a vyprodukovanie formátovaného výstupu. Túto analýzu za nás obstaráva *renderovacie jadro* (layout engine) [12]. Tieto renderovacie jadrá sú typicky využité vo webových prehliadačoch, ako napr. Mozilla Firefox, Internet Explorer, Safari alebo Opera. Veľké využitie majú taktiež v emailových klientoch [9].

Termín renderovacie jadro začal byť populárny až v dobe, keď ho bolo možné jednoducho oddeliť od prehliadača. Napr. jadro Trident - vyvíjané firmou Microsoft, je využívané v rôznych produktoch. Napríklad doplnok IE-Tab pre Mozillu Firefox či Google Chrome. Medzi najznámejšie jadrá patria:

- Mozilla Gecko
- Microsoft Trident
- Webkit
- Presto

Prvé tri jadrá môžeme voľne využívať. Jadro Presto je pod proprietárnou licenciou a nie je možné ho voľne využívať.

Microsoft Trident

Taktiež známy ako MSHTML je renderovacie jadro využívané v prehliadači Internet Explorer. Prvýkrát bolo predstavené v októbri roku 1997 spolu s uvedením Internet Explorera 4.0. Jadro Trident bolo navrhnuté ako softwarová komponenta s cieľom umožniť softwarovým vývojárom jednoducho pridať funkciu renderovania HTML [1].

Taktiež predstavuje rozhranie COM, ktoré umožňuje prístup a editáciu web stránok v akomkoľvek prostredí podporujúce COM (napr. .NET).

Jadro Trident je využívané vo veľkom množstve prehliadačov, ale taktiež v programových komponentách operačného systému Microsoft Windows.

Mozilla Gecko

Je renderovacie jadro, ktoré je známe vďaka prehliadaču Firefox. Medzi hlavné výhody tohto jadra patrí jeho multiplatformnosť, dodržiavanie webových štandardov a taktiež otvorená licencia. Gecko vďaka svojmu bohatému API dovoluje nielen renderovanie webových stránok, ale taktiež, vykreslenie grafického rozhrania (XUL)[10]. Gecko implementuje niektoré nové prvky z prichádzajúceho HTML5 (canvas, video) a CSS3 (napr. zaoblenie rohov). Tento fakt taktiež pridáva na jeho popularite.

Na rozdiel od jadra Trident nepodporuje rozhranie COM. Preto je implementácia jadra Gecko do vlastnej aplikácie veľmi zložitá.

Webkit

Bol pôvodne vytvorený z jadra KHTML a bol primárne určený pre prehliadač Safari. Jadro webkit je rozdelené do viacerých komponent. Komponenta *WebCore* má na starosti

renderovanie, prácu s HTML DOM (Document Object Model) a SVG. Spolu s komponentou *JavaScriptCore* je uvoľnená pod licenciou LGPL. Zvyšok Webkitu je uvoľnený pod BSD licenciou [14]. Tieto komponenty je možné využívať samostatne, napríklad prehliadač Google Chrome využíva jadro Webkit, ale implementuje svoj vlastný modul pre prácu s javascriptom – V8.

Webkit bol portovaný pre niekoľko programových frameworkov ako napríklad GTK+, QT alebo Adobe AIR. Práve toto portovanie umožňuje vývojárom jednoducho začleniť webkit do ich vlastných aplikácií.

Medzi najznámejšie programy využívajúce Webkit patrí prehliadač Safari, Google Chrome a Webbrowser for S60, ktorý je určený pre mobilnú platformu Symbian.

Presto

Je renderovacie jadro využívané v prehliadači Opera. Medzi hlavné výhody tohto jadra patrí rýchlosť, dodržiavanie webových štandardov a schopnosť čiastočne renderovať CSS3 a HTML5. Jadro presto je ale prístupné len ako časť prehliadača Opera a jej príbuzných produktov. Zdrojový kód alebo binárna forma (DLL knižnica) nie je verejne prístupná.

2.3 Acid testy

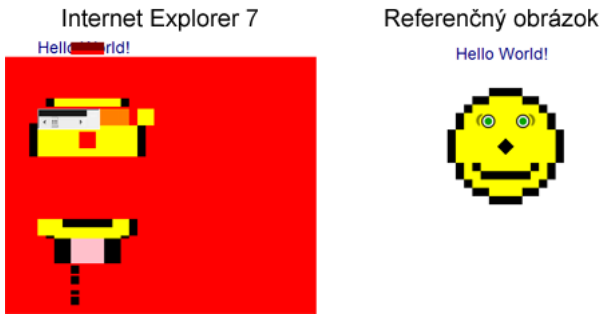
Je séria testov z projektu „Web Standards“, ktoré majú za úlohu posúdiť podporu webových štandardov v internetových prehliadačoch. Rozlišujeme 3 základné testy [7].

Acid 1

Bol vytvorený v roku 1998. V dobe svojho vydanie ukázal problémy vo webových prehliadačoch a zohral tak úlohu pri vytváraní podpory CSS špecifikácie 1.0. V súčasnosti všetky 4 vyššie spomenuté jadrá prejdú testom na 100%.

Acid 2

Bol zverejnený v roku 2005 za účelom uľahčenia opravy chýb vo webových prehliadačoch a zobrazovania web stránok. Test bol hlavne zameraný na korektné renderovanie CSS 2.1 a PNG obrázkov. Microsoft Internet Explorer až do verzie 8 nebol schopný prejsť testom.



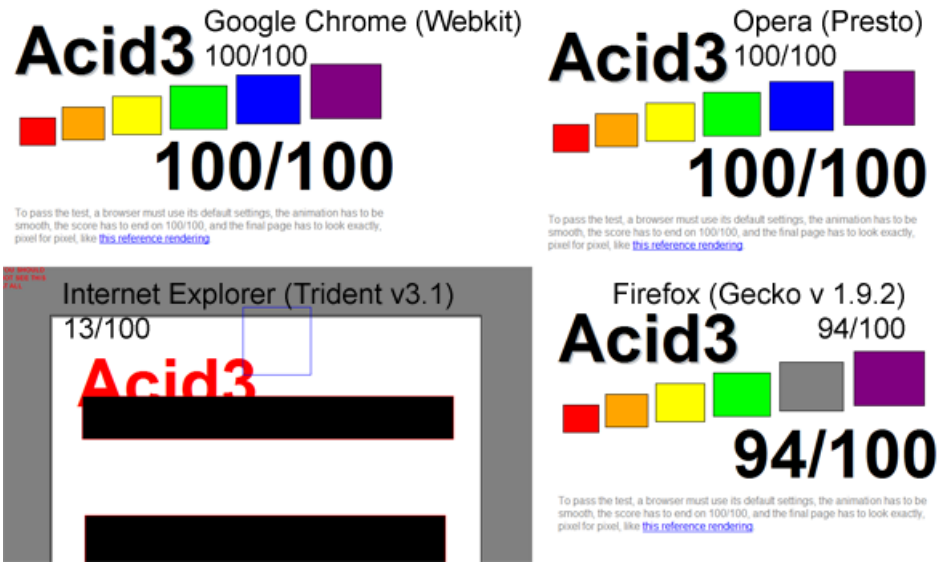
Obrázok 2.1: Problémy prehliadača Internet Explorer 7 oproti referenčnému obrázku

Acid 3

Je primárne zameraný na renderovanie CSS ale taktiež na použitie technológií, ktoré sú používané v moderných a vysoko interaktívnych webstránkach napr. javascript a DOM level 2. Niekoľko podtestov je tiež zameraných na prácu s SVG grafikou, XML a dátovým URI. Celý test je napísaný v Javascripte a pozostáva celkovo zo 100 podtestov, ktoré postupne testujú vyššie uvedené technológie.

Pre úspešné splnenie testu musí prehliadač zobraziť stránku úplne rovnako ako je referenčný obrázok a taktiež musí byť animácia počas renderovania plynulá. Po ukončení testu je možné kliknúť na písmeno „A“ v slove Acid, pre zobrazenie javascriptového okna, obsahujúceho podrobnejšie informácie. Napriek tomu, že prehliadače využívajúce renderovacie jadrá Webkit a Presto získali v tomto teste hodnotenie 100/100, nie všetky dokážu splniť výkonnostný aspekt tohto testu.

Prehľad jednotlivých hodnotení môžeme sledovať na obrázku 2.2



Obrázok 2.2: Prehľad jednotlivých hodnotení v ACID 3 teste

2.4 Javascript

Je objektovo orientovaný skriptovací jazyk, ktorý sa primárne využíva v HTML stránkach. Slúži predovšetkým na manipuláciu s HTML DOM. Interpretom javascriptu je často webový prehliadač, konkrétne jeho renderovacie jadro. Pre zjednodušenie programovania v javascripte existujú rôzne frameworky. Jedným z najpopulárnejším frameworkom je v súčasnosti jQuery alebo Prototype. Oba frameworky dovoľujú jednoduchú manipuláciu s DOM, vytváranie užívateľsky prívetivých efektov a taktiež jednoduchšie vytváranie AJAXu.

Webové stránky sú často javascriptom preplnené a preto je dôležitá rýchlosť jeho interpretácie. Pre meranie výkonnosti interpreta javascriptu, existuje niekoľko špeciálnych testov, napr. *Dromaeo, SunSpider* a *V8 Benchmark Suite*.

SunSpider

Je sada testov, ktorá pochádza od vývojárov Webkitu. Prvýkrát bola predstavená koncom roka 2007. Celkovo obsahuje 26 testov, ktoré sa zameriavajú rôzne konštrukcie javascriptu. Nevýhoda tohto testu je, že vôbec netestuje rýchlosť manipulácie s DOM, ktorá reálne ovplyvňuje celkovú rýchlosť javascriptových aplikácií [5].

Sadu testov je možné spustiť priamo on-line zo stránok projektu, ale taktiež aj z príkazového riadku. Jednotlivé testy sú spúšťané viackrát a výsledná hodnota je nakoniec priemer z jednotlivých meraní.

Dromaeo

Je rozsiahla sada testov pomenovaná podľa Dromaeosaurus – rýchly jašter. Autorom je John Resig, ktorý je známy vďaka vývoju frameworku jQuery. Testovacia sada je veľmi pestrá a na rozdiel od SunSpideru obsahuje taktiež testy pre prácu s DOM. Taktiež testuje rýchlosti frameworkov Prototype, jQuery a prácu s CSS selektormi. Okrem vlastných testov Dromaeo obsahuje i testy zo SunSpideru a V8 Benchmark Suite [3].

Jednotlivé testy bežia vopred stanovený čas a meria sa počet priebehov. Podobne ako u SunSpideru Dromaeo umožňuje spustenie z príkazového riadku alebo priamo on-line zo stránok projektu.

V8 Benchmark Suite

Je testovacia sada uverejnená spoločnosťou Google. Hlavným rozdielom od vyššie spomenutých testovacích sád je, že je veľmi jednoduchá. Obsahuje len 7 testov. Testuje sa predovšetkým práca s reťazcami, numerické výpočty a rekurzívne volania funkcií [6]. Testy podobne ako u Dromaea bežia pevne stanovený čas. Počet priebehov testov, potom určuje výsledné skóre.

2.5 Kaskádové štýly

Sú jednoduchý mechanizmus, ktorým je možné výsledný vzhľad HTML dokumentu. Špecifikácia kaskádových štýlov (CSS) je spravovaná Webovým Konzorciom W3C. Existujú 3 špecifikácie CSS, *CSS 1*, *CSS 2* a *CSS 3*. Špecifikácia CSS 3 je v súčasnosti ešte vyvíjaná. Väčšina renderovacích jadier si dokáže poradiť so špecifikáciou CSS 2.

Použitie CSS v porovnaní s čistým HTML prináša niekoľko výhod.

- jednoduchšia údržba HTML dokumentov
- je možne definovať rôzne štýly pre rôzne výstupné zariadenia ako napr. mobil, tlač, bežný prehliadač ...
- rozsiahlejšie možnosti formátovania HTML dokumentov.
- CSS štýly su zvyčajne uchované vo vyrovnávacej pamäti prehliadača (cache), čo šetrí výsledný prenos dát
- oddelenie štruktúry dokumentu od štýlov.

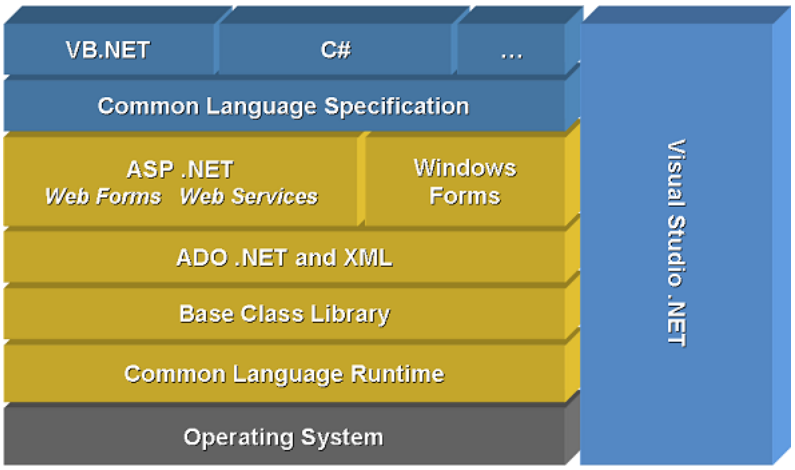
2.6 Použité technológie

.NET Framework

je framework, určený pre počítače s operačným systémom Microsoft Windows (existuje, ale aj jeho open source implementácia Mono, ktorá je taktiež prístupná pre operačné systémy založené na platforme Unix).

Programy napísané v .NET frameworku sú spúšťané pod *Common Language Runtime* (CLR), ktorý spracuje ich behové požiadavky. CLR taktiež poskytuje ďalšie služby ako sú bezpečnosť, správa pamäte a výnimiek [13]. Framework pokrýva širokú škálu potrieb programovania ako napr. programovanie webových aplikácii (ASP.NET), tvorba užívateľského rozhrania (WinForms, WPF) alebo prístup k databázovým systémom. Schéma .NET môžeme pozorovať na obrázku 2.3

Medzi najpoužívanejšie jazyky pre vývoj .NET aplikácií patrí C#, VB.NET a J#. Pri kompilácii je zdrojový kód preložený do CIL kódu, ktorý je následne zostavený do byte kódu.



Obrázok 2.3: Schéma .NET Frameworku

ASP.NET

ASP.NET je súčasť .NET frameworku, ktorá dovoľuje programátorom vyvíjať webové aplikácie a služby. Samozrejme je možné využiť akýkoľvek programovací jazyk, ktorý je podporovaný v .NET. Taktiež umožňuje jednoducho oddeliť programový kód (napísaný v C#, VB.NET ...) od grafického dizajnu – *code-behind model*.

V súčasnosti je v ASP.NET možné využiť dva spôsoby programovania. Jedným z nich sú tradičné *WebForms* a druhým je využitie *ASP.NET MVC*.

Medzi hlavné výhody ASP.NET patrí jeho výkonnosť, pretože sú predkompilované do jedného alebo viacerých DLL súborov. Táto kompilácia sa deje automaticky pri prvom požiadavku na stránku. Nevýhoda tohto spôsobu je, že pri prvom požiadavku môže spôsobiť určité oneskorenie.

LINQ

LINQ (*Language Integrated Query*) je komponenta .NET frameworku, ktorá sa stará o dotazovanie sa nad dátami a uľahčuje typické operácie nad dátami ako je napr. vyhľadávanie alebo ich zoradenie.

Pomocou LINQu je veľmi jednoduché sa dotazovať na dáta uložené v MS SQL databázi. Výhodou použitia LINQu oproti tradičnému prístupu je hlavne jeho objektový pohľad na dáta

QT Framework

QT je multiplatformný vývojový framework vyvíjaný spoločnosťou Nokia. Využíva sa hlavne pre tvorbu aplikácií s užívateľským rozhraním, ale taktiež aplikácií konzolového typu. Medzi najznámejšie aplikácie využívajúce QT patrí napr. Google Earth, KDE alebo prehliadač Opera.

QT Využíva štandardný jazyk C++, ale taktiež bohato využíva špeciálneho pre-procesoru (tzv. Meta objekt kompilátor – MOC) pre obohatenie jazyka.

Framework existuje pod dvomi typmi licencií. Jedna z nich je GNU LGPL a druhá proprietárna, ktorá je určená pre vývoj aplikácií komerčného typu.

Ajax

Asynchronous JavaScript and XML (AJAX) Je technika, ktorá umožňuje vytváranie rýchlych a dynamických webstránok. Pomocou tejto technológie je možné získať dáta zo serveru bez toho aby sme akokoľvek zasahovali do stávajúcej web stránky. Dáta sa často získavajú pomocou objektu *XMLHttpRequest*.

AJAX je veľmi obľúbená technika pri vývoji web aplikácií a je bohato využitá napríklad na stránkach Google Maps, alebo u služby Gmail.

Medzi jej hlavné výhody patrí, že je možne aktualizovať časť web stránky bez znovunačítania. Taktiež pri správnom využití znižuje záťaž na webové servery a objem prenášaných dát.

Samozrejme existuje aj niekoľko nevýhod. Medzi hlavné nevýhody patrí nemožnosť uloženia aktuálnej stránky medzi obľúbené položky, pretože sa nemení URI adresa v adresnom riadku prehliadača. Vyhľadávače a roboty často neinterpretujú javascriptový kód, preto môžu nastať problémy so zaradením stránky do vyhľadávacích služieb. Taktiež je nutnosť využiť moderné prehliadače, špeciálne prehliadače určené pre mobilné telefóny, môžu mať problémy s načítaním stránky.

REST Architektúra

Representational State Transfer (REST) je architektúra navrhnutá pre distribuované prostredia. Prvýkrát bola predstavená v roku 2000 v dizertačnej práci Roya Fieldinga. REST určuje spôsob ako pristupujeme k dátam. Zdroje dát majú svoj vlastný identifikátor URI a architektúra REST definuje 4 základné metódy prístupu, známe pod označením CRUD (Create, Read, Update, Delete) [8]. Práve touto architektúrou je inšpirované API rozhranie, ktoré je implementované v rámci webovej služby.

Získanie dát – GET

Pretože každý zdroj dát ma svoj vlastný identifikátor URI, tak pomocou HTTP GET požiadavku môžeme jednoducho tieto dáta získať.

Vytvorenie dát – POST

Pre vytvorenie dát sa používa HTTP metóda POST, ktorá je známa predovšetky z HTML formulárov. U metódy POST nie je vo chvíli volania ešte presne známy identifikátor zdroja (zdroj ešte neexistuje). Preto sa pri metóde POST využíva dohodnutý spoločný identifikátor (koncový bod – endpoint).

Upravenie dát – PUT

Operácia zmeny dát je veľmi podobná operácii vytvorenia. Zásadný rozdiel, je že sa odkazujeme na konkrétny a nie na spoločný identifikátor zdroja. Pre tieto účely sa využíva metóda PUT. Pretože vytvorenie PUT požiadavku môže byť na rozdiel od požiadavkou GET a POST náročne, tak sa často pre tento účel využívajú aj požiadavky typu GET.

Zmazanie dát – DELETE

Ak chceme zmazať zdroj dát, tak postupujeme podobne ako pri jeho získaní, ale namiesto metódy GET použijeme metódu DELETE. Podobne ako u operácii zmeny dát, tak metóda DELETE nemusí byť u zdroja dostupná. V tom prípade môžeme využiť požiadavku typu POST alebo GET (záleží na dohodnutých pravidlách).

Kapitola 3

Analýza, popis a návrh aplikácie

Táto kapitola popisuje jednotlivé požiadavky, ktoré boli vyžadované od výsledného systému. Podrobne je vysvetlený proces vytvárania obrázkov a celkové ovládanie aplikácie.

3.1 Špecifikácia požiadavkov

Hlavným cieľom tejto webovej služby bolo vytvorenie malého náhľadu (bitmapového obrázku) zo zadanej URI. Podobné vytváranie náhľadov webových odkazov využíva napr. portál seznam.cz pri výsledkoch vyhľadávania. Práve toto využitie je pre koncového užívateľa prívetivé, pretože má možnosť okamžite vidieť ako vyzerá odkazovaná stránka.

Pre vytvorenie náhľadu sa využívajú rôzne renderovacie jadrá. V mojej práci je implementované Jadro Trident od spoločnosti Microsoft a open source jadro Webkit. Jadrá renderujú webové stránky rôzne. preto sa výsledné obrázky môžu líšiť.

3.2 Analýza problému

Vytvorenie obrázku zo zadanej webovej stránky, nie je úplne triviálna záležitosť, ako by mohlo zdať na prvý pohľad. Je na to potreba využiť parser HTML a CSS kódu a následne výstup z tohto parseru vykresliť do bitmapového obrázku. Programovanie vlastného intrepretu HTML stránok by bolo veľmi náročné, pretože existuje mnoho odlišných foriem HTML dokumentov a taktiež niekoľko verzií kaskádových štýlov. Preto som sa rozhodol použiť voľne prístupné renderovacie jadrá. Pretože pre implementáciu služby bola použitá technológia ASP.NET, bolo relatívne jednoduché implementovať renderovacie jadro Trident. Jadro Trident je implementované v komponente *WebBrowser*, ktorá je súčasťou .NET frameworku. Schopnosti tohto jadra sú často kritizované, pretože nedodržiava niektoré dohodnuté štandardy, konkrétne zle počíta veľkosti *margin* a *padding* v CSS.

Pretože by bolo veľmi zaujímavé porovnanie s ostatnými renderovacími jadrami, tak sa nám naskytá možnosť implementovať ďalšie voľne prístupné jadrá a to Gecko a Webkit. Implementovať Webkit do aplikácie postavenej na ASP.NET nie je až tak jednoduchá záležitosť ako implementácia vyššie spomenutého jadra. Vývojari Webkitu oficiálne neposkytujú žiadne rozhranie, pre začlenenie jadra do aplikácií postavených na .NET. Existuje ale open source implementácia – WebkitDotNet, ktorá sa snaží portovať Webkit na .NET platformu. Tento projekt taktiež implementuje vlastnú webbrowser komponentu, využívajucu jadro Webkit. Projekt je ale ešte vo veľmi nestabilnej fáze a pri niektorých bežných operáciach skončí chybovou hláškou. Navyše pre korektné fungovanie vyžaduje vytvorenie

inštancie triedy *System.Windows.Forms.Form*. Vzhľadom na to, že naša aplikácia beží ako webová služba, tak vytvorenie tejto inštancie nie je možné.

Ďalšia možnosť ako implementovať jadro webkit je siahnuť po frameworku GTK+ alebo QT. Pretože väčšina informácií ako vytvoriť aplikáciu využívajúcu webkit pod GTK+ je určené pre operačný systém GNU Linux, tak som sa rozhodol implementovať jadro webkit v QT frameworku. Výhoda implementácie v QT je taktiež v tom, že firma Nokia prispieva do zdrojových kódov webkitu a taktiež je vo frameworku implementovaná posledná verzia tohto jadra. Výhodou je, že táto implementácia jadra nevyžaduje spustenie grafického prostredia, alebo X-Serveru na systémoch založených na UNIXe. Podrobnejšie informácie sú dostupné v kapitole 4.

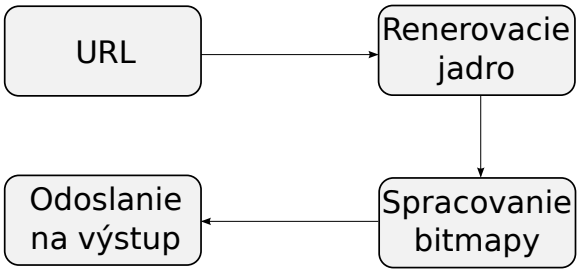
Vzhľadom k tomu, že som jadro Webkit implementoval s využitím QT, tak sa do popredia dostáva nový problém a to jeho napojenie na webovú službu. Existuje niekoľko implementácií QT určených pre rôzne programovacie jazyky. Pre využitie QT v jazyku C# a .NET je určený projekt *Qyoto*, tento projekt je ale primárne určený pre programové prostredie KDE (K Desktop Environment) v operačnom systéme GNU Linux. Kompilácia zdrojových kódov pod prostredím Windows je samozrejme možná. ale výsledný projekt je čiastočne nestabilný. Z tohto dôvodu som sa rozhodol pre jazyk C++, ktorý je pre QT framework priamo určený. Existuje niekoľko možností ako využívať C++ kód v jazyku C#. Jednou z možností je kompilácia programov do *dynamicky linkovanej knižnice* (*DLL*) následne vytvorenie triedy v jazyku C#, ktorá pomocou atribútu *DLLImport* dokáže využiť funkcie, ktoré nám naša DLL knižnica poskytuje. Takáto trieda prakticky obaľuje DLL knižnicu a poskytuje jej funkcie ďalším triedam v jazyku C#, v angličtine sa pre tento typ funkcie často využíva pomenovanie *wrapper*. Druhou z možností nie je kompilácia do knižnice DLL ale do klasického spustiteľného súboru (EXE). Následne si môžeme tento súbor zavolať pomocou objektu *Process*, ktorý súčasť .NET frameworku. Tento spôsob je výrazne jednoduchší a taktiež bezproblémovo funguje, preto som sa rozhodol pre jeho implementáciu. Prvú z uvedených možností by som síce považoval ako „čistejšie riešenie“, ale použitie QT frameworku, to výrazne komplikuje.

Zostáva nám ešte implementácia jadra Gecko. Podobne ako u jadra webkit, vývojári neposkytujú žiadne rozhranie pre využitie v .NET aplikáciách. Taktiež ako u Webkitu, existuje open source projekt GeckoFX, ktorý sa snaží sprístupniť jadro pre .NET. Projekt je stabilnejší, ale podobne ako u projektu WebkitDotNet vyžaduje pre korektné fungovanie vytvorenie inštancie triedy *System.Windows.Forms.Form*. Pokiaľ by som vytváral aplikáciu určenú pre stolný počítač a nie službu, ktorá beží vzdialene na serveri, tak by to nebol problém. Ďalšia možnosť ako implementovať toto jadro v .NET je využitie *Mozilla ActiveX Control*, ktorý výrazne zjednodušuje implementáciu. Bohužiaľ tento projekt sa prestal ďalej vyvíjať a posledná verzia podporuje len verziu Gecko 1.7 (Firefox 1.07). Táto verzia je už ale veľmi zastaraná, tak by výsledky renderovania dnešných webových stránok boli značne skreslené a nedokonalé. Podobne ako u jadra Webkit existuje jadro implementované v programovom frameworku GTK+ (GTKMoz). Problémom tohto riešenia je, že pre svoj beh vyžaduje operačný systém GNU Linux so zapnutým X-Serverom.

Poslednou možnosťou je napísanie aplikácie, bez vyžitia programových frameworkov [4]. Táto možnosť je ale veľmi zložitá a časovo náročná. Navyše existuje len málo informácii o tom, ako je možné napísať podobnú službu pre operačný systém Microsoft Windows. Pre operačný systém Linux je situácia lepšia, pretože Gecko je na linuxe často využívané ako primárna komponenta pre renderovanie stránok, zatiaľ čo pod Windows sa vo väčšine prípadov využije jadro Trident.

3.3 Princíp vytvárania obrázkov

Na vstupe je adresa URI, ktorá sa ďalej pošle do objektu, ktorý implementuje požadované jadro. Následne sa overí, či sa daný odkaz nesnaží presmerovať na inú adresu (HTTP redirect). Výsledná URI adresa sa nakoniec odošle do renderovacieho jadra, ktoré ju spracuje – interpretuje HTML, CSS a javascript. Výsledné dáta sa ďalej spracujú do obrázka typu png. Celý proces je znázornený na obrázku 3.1



Obrázok 3.1: princíp vytvárania náhľadov

3.4 Možnosti zadania vstupu

Vstupnú URI adresu je možné zadať viacerými spôsobmi (obrázok 3.2).

1. používateľ zadá priamo na stránke služby
2. používateľ zadá potrebné údaje ako parametre URI adresy
3. využitie XML api.

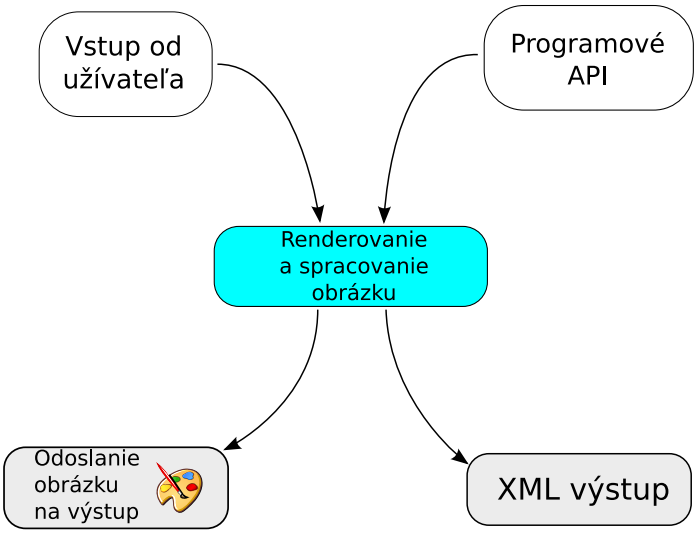
Priame zadanie

Pokiaľ užívateľ zadáva dáta priamo, tak si na hlavnej stránke môže vybrať veľkosť výsledného obrázku, veľkosť virtuálneho okna renderovacieho jadra a typ renderovcieho jadra. Samozrejme je možné zadať aj niekoľko URI adries naraz. Pre túto možnosť je určená špeciálna stránka. Pri jadre Webkit má navyše možnosť zadať čas, ako dlho sa má maximálne jadro snažiť renderovať web stránku. Ďalej je možnosť povoliť alebo zakázať interpretáciu javascriptu.

Na základe tohto užívateľského vstupu sa spustí proces vytvárania obrázku. Celý tento proces je vyriešený ako ajaxový požiadavok. Počas renderovanie je užívateľovi zobrazená „ajaxová“ animácia. po ukončení procesu, je obrázok odoslaný užívateľovi.

Zadanie pomocou URI

Dotaz na vytvorenie náhľadu je taktiež možné zadať podľa špeciálnej adresy (*Snap.aspx*), na ktorej sa dajú špecifikovať parametre definované v nasledujúcej tabuľke:



Obrázok 3.2: Zadanie vstupu a zobrazenie výstupu

Parameter	**Popis**	**Dovolené hodnoty**
url	Url webstránky, z ktorej sa bude vytvárať obrázok	platná url adresa
size	Nastavenie veľkosti výsledného obrázku	100x80, 160x120, 200x150, 320x240, 640x480 (v px)
browser	Nastavenie veľkosti virtuálneho okna prehliača	640x480, 800x600, 1024x768, 1280x1024 (v px)
engine	Nastavenie renderovacieho jadra, ktoré spracuje obrázok	Webkit / Trident

Pričom povinný parameter je len url, pokiaľ nie sú špecifikované ďalšie parametre, tak sa použijú programom prednastavené hodnoty. Výsledná url môže vyzerať asi takto:

```
http://<doména>/Snap.aspx?url=http://www.fit.vutbr.cz&size=320x240
```

Pokiaľ sú parametre zadané chybne, tak na to služba upozorní chybovou hláškou. Takto zadaný obrázok je uchovaný na serveri po dobu 14 dní. Po uplynutí tejto doby nastane jeho automatická aktualizácia.

Popis API

Pre zadávanie vstupu bolo taktiež vytvorené jednoduché API založené na technológii XML. Výsledné API bolo inšpirované architektúrou REST. Vzhľadom na to, že nám postačuje len vytváranie a následné získanie zdroja (v našom prípade obrázku), tak operácie DELETE a PUT neboli implementované.

Pre vytvorenie nového požiadavku odošleme špeciálne vytvorené xml na adresu:

```
http://<doména>/api/new
```

XML dokument môže obsahovať nasledujúce elementy

- *url*, obsahujúci adresu webovej stránky, tento element je povinný.
- *browser*, obsahujúci nastavenie veľkosti virtuálneho okna prehliadača, formát vstupných dát musí byť: *číslo*x*číslo*
- *size*, obsahujúci nastavenie veľkosti výsledného obrázku vo formáte *číslo*x*číslo*
- *engine*, obsahujúci jadro, ktoré sa použije na vyrenderovanie (Webkit alebo Trident).
- *javascript*, obsahujúci nepovinný element pri jadre Webkit, určujúci povolenie alebo zakázanie javascriptu pri renderovaní
- *silverlight*, nepovinný parameter pri jadre Webkit, ktorý povolí/zakáže spúšťanie silverlightu
- *delay*, Ďalší nepovinný parameter pri jadre Webkit, určujúci počet sekúnd, ktoré je potreba počkať po renderovaní HTML. Táto možnosť je chodná hlavne pre stránky obsahujúce silverlight.

Výsledný XML dokument môže vyzerať asi takto:

```
<root>
  <url>http://bp.martinskopal.net</url>
  <size>230x230</size>
  <browser>1156x800</browser>
  <engine>Webkit</engine>
  <javascript>1</javascript>
  <silverlight>1</silverlight>
  <delay>3</delay>
</root>
```

V prípade, že nastane chyba pri renderovaní, alebo pre nekorektné zaslanie požiadavku, tak nám webový server vráti XML kód obsahujúci chybové hlásenie. V prípade korektného spracovanie obdržíme návratový kód 201 (Created) a XML kód obsahujúci odpoveď.
Ukážka úspešnej odpovede:

```
<root>
  <url>http://www.fit.vutbr.cz</url>
  <browser>1156x600</browser>
  <size>230x230</size>
  <engine>Webkit</engine>
  <javascript>1</javascript>
  <imageurl>http://<doména>/shots/ht1LjINT.png</imageurl>
  <key>OC41LjIwMTAgMjE6MzU6NTg</key>
</root>
```

Odpoveď okrem elemetov, ktoré sú špecifikované v pri vytváraní požiadavku, navyše obsahuje aj element `<imageurl>`, ktorý obsahuje adresu vyrenderovaného obrázku. Taktiež je prítomný element `<key>`, ktorý jednoznačne označuje získanú odpoveď. Tento kľuč použijeme neskôr pri získavaní zdroja (Požiadavka GET).

Pokiaľ sa chceme odkázať na už nami vytvorené dáta, tak použijem kľúč, ktorý sme obdržali po ich vytvorení. Vytvorím HTTP GET požiadavok, kde do URI zdroja pridáme nami požadovaný kľúč, napr.:

```
http://<doména>/api/OC41LjIwMTAgMjE6MzU6NTg
```

Na základe tohto požiadavku server odpovie zaslaním vyššie spomenutého XML kódu.

3.5 Galéria

Zo zadaných webových stránok sa taktiež generuje jednoduchá galéria stránok. Galéria umožňuje užívateľom hodnotiť jednotlivé webové stránky. Stránkovanie a hodnotenie je implementované s využitím technológie ajax, ktorá zvyšuje interaktivitu celej stránky. Jednotlivé stránky sú zoradené v galérii podľa počtu hodnotení a následne podľa výšky hodnotenia. Každý používateľ má možnosť hodnotiť každý obrázok len raz. Systém si uchováva jednotlivé ip adresy, z ktorých sa hlasovalo.

Hodnotenie je implementované pomocou piatich hviezdičiek, pričom čím viac hviezdičiek, tým je lepšie hodnotenie. Celkové hodnotenie sa počíta ako priemer jednotlivých hodnotení.

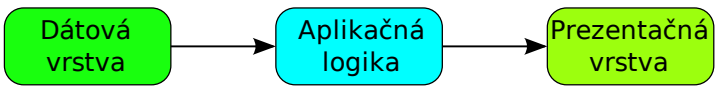
3.6 Rozdelenie aplikácie

Celá webová služba bola implementovaná s využitím vývojového prostredia Visual Studio 2008. Obsahuje dva projekty, 1. projekt je typu webstránka – ASP.NET implementuje samotnú webovú službu. Druhý projekt je typu QT aplikácia a implementuje Jadro Webkit. Podpora QT nie je štandardne zahrnutá, a pre úspešnú kompiláciu je potreba stiahnuť a nainštalovať doplnok zo stránok QT frameworku.

Celá webová služba je rozdelená do troch vrstiev:

- dátová vrstva
- logika aplikácie
- prezentačná vrstva

Dátová vrstva (*DatabaseModel.cs*)je má na starosti databázové operácie, využíva technológiu LINQ To SQL, ktorá zjednodušila jednotlivé databázové dotazy. Ďalej nasleduje logika celej aplikácie, ktorá využíva vyššie spomenutú dátovú vrstvu. Aplikačná logika spracúvava požiadavky od prezentačnej vrstvy a po ich spracovaní odosiela výsledky späť do prezentačnej vrstvy. Celá logika je umiestnená v adresári *APP_CODE*. Prezentačnú vrstvu predstavujú *aspx* dokumenty, umiestnené v koreňovom adresári. Predstavuje tzv. rozhranie medzi užívateľom a programom[2]. Celkové schéma použitej 3-vrstvovej architektúry znázorňuje obrázok 3.3.



Obrázok 3.3: Schéma 3-vrstvovej architektúry

Výhody trojvrstvovej architektúry spočívajú predovšetkým v znovu použiteľnosti kódu a prehľadnosti celej aplikácie. Taktiež rozdeľuje aplikáciu na samostatné logické celky.

3.7 Užívateľské rozhranie

Dovoľuje koncovému užívateľovi pohodlnú manipuláciu s programom. Vďaka tomu, že je program implementovaný ako webová služba, tak sú použité technológie pre užívateľa úplne transparentné. Jediným požiadavkom je funkčné pripojenie k internetu a použitie moderného webového prehliadača.

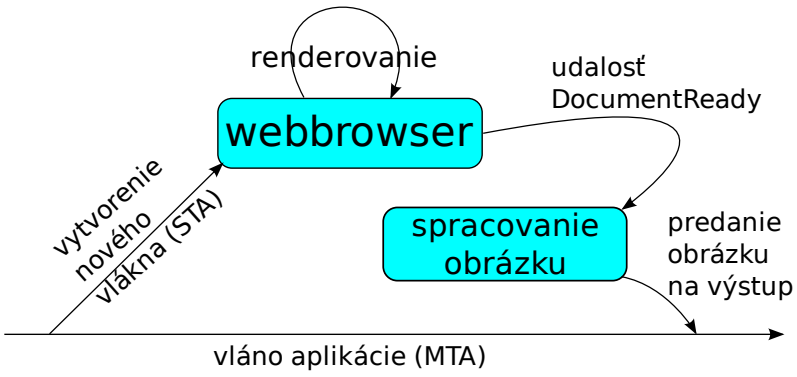
Kapitola 4

Implementácia

V tejto kapitole postupne rozoberiem implementáciu vybraných častí aplikácie. Konkrétne sa budem zaoberať implementáciou renderovacích jadier a databázoveho modelu. Všetky jadrá ktoré sú vo webovej službe implementované musia implementovať rozhranie *iWebRenderer*. Tento fakt mi neskoršie umožňuje pohodlnú manipuláciu s renderovacími jadrami.

4.1 Implementácia jadra Trident

Jadro Trident je implementované v triede *TridentRenderer*. Je využitá webbrowser komponenta, ktorá vyžaduje pre svoj beh tzv. STA (Single Threaded Apartment) vlákno. Bežné stránky v ASP.NET, ale maju prostredie MTA (Multi Threaded Apartment). Dosiahnuť vlákno STA sa dá dvomi spôsobmi. Prvý je v nastavením atribútu ASPCOMPAT a druhá možnosť je spustenie v samostatnom vlákne, pre ktoré nastavíme STA. Zaujímavosťou je že samotné jadro nepracuje len v jednom vlákne. Platí ale pravidlo, že len vlákno ktoré vytvorilo webbrowser, má možnosť ďalej k nemu pristupovať. Na začiatku renderovania odošleme požadovanú adresu do jadra. Jadro pristúpi na nami zvolenú adresu a začne proces renderovania. Po úplnom načítaní webovej stránky nastane špeciálna udalosť, ktorá nás o tom informuje. Práva táto udalosť je ideálne miesto pre spracovanie výsledného obrázku. Následne sa obrázok a ďalšie potrebné informácie predajú do prezentačnej vrstvy a zobrazia sa užívateľovi. na obrázku 4.1 je znázornený zjednodušený model ako dané jadro pracuje.



Obrázok 4.1: Model znázorňujúci funkčnosť jadra Trident

4.2 Implementácia jadra Webkit

Ako som už spomínal jadro je implementované v C++ s využitím QT frameworku. Jadro je skompilované do spustiteľného súboru qtWebkitRenderer.exe, ktorý je umiestnený v adresári Bin. Tento spustiteľný súbor je volaný z pomocou objektu process v triede *WebkitRenderer*. Implementácia v C++ pozostáva z troch hlavných súborov *MyQWebpage.cpp*, *MyWebkitRenderer.cpp* a *main.cpp*. Na obrázku 4.2 je podrobnejšie znázornená komunikácia jednotlivých komponent.

MyQWebpage

QWebpage reprezentuje samotné renderovacie jadro. Je to vlastne „virtuálne plátno“, do ktorého sa vykresľuje webová stránka. Pri renderovaní, ale môžu nastať problémy s javacriptom. Tieto problémy nastávajú pokiaľ sa na webovej stránke vyskytuje javascriptové upozornenie (alert). Je potrebné zabrániť tomuto upozorneniu, aby bolo možné stránku ďalej vyrenderovať. Preto je vytváraná nová trieda MyQWebpage, ktorá je priamym potomkom QWebpage a preťažím v nej všetky metódy, ktoré sa starajú o udalosti podobné javascriptovému upozorneniu.

MyWebkitRenderer

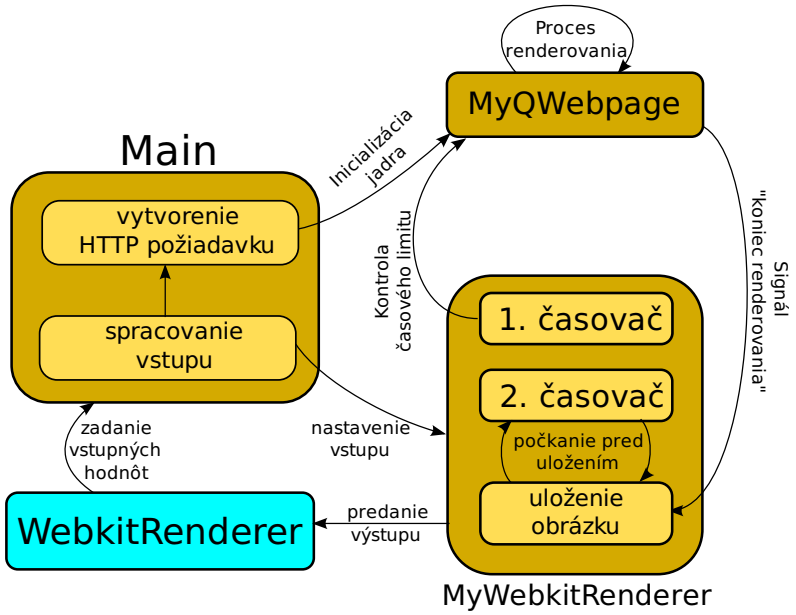
Táto trieda sa stará o samotné uloženie obrázku a správu obslužných časovačov. Hlavnou časťou triedy, je metóda *saveSnapshot*, ktorá spracuje vyrenderovanú stránku a uloží obrázok na disk. V triede sú využité 2 časovače. Prvý má za ulohu ukončiť renderovanie po dosiahnutí časového limitu a druhý časovač je určený na počkanie niekoľkých sekúnd po renderovaní HTML. Toto počkanie je napríklad dôležité pri načítavaní silverlightu, práve vďaka počkaniu je silverlight vo väčšine prípadov korektne vyrenderovaný.

Main

Tento súbor využíva vyššie spomenuté triedy, inicializuje obslužné parametre a využíva samotné renderovacie jadro. Taktiež vytvára webový požiadavok, ktorý v prípade HTTP presmerovania obstará príslušnú réžiu. Následne je tento webový požiadavok pomocou metódy *load* predaný renderovaciemu jadru. po ukončení renderovanie je spustený signál, ktorý singalizuje ukončenie a následne je možné obrázok uložiť.

WebkitRenderer

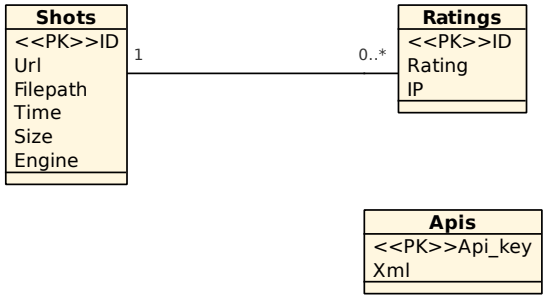
Jedná sa o triedu napísánu v jazyku C#. Je účelom je presmerovať volania z webovej služby do implementovaného jadra a taktiež, presmerovať spätne výstup z jadra do webovej služby.



Obrázok 4.2: Model znázorňujúci funkčnosť implementované Webkit jadra

4.3 Implementácia dátovej vrstvy

ER diagram



Pre správne fungovanie webovej služby bolo potrebné uchovávať si jednotlivé obrázky v databázi. Hlavnou tabuľkou je tabuľka *Shots*. ktorá uchováva informácie o vygenerovaných obrázkoch. V tabuľke *Ratings* sú uchované jednotlivé hodnotenia ku obrázkom. Každý užívateľ možnosť hlasovať za každý obrázok len raz, pretože sa zaznamenávajú jednotlivé hodnotenia. V poslednej tabuľke *Apis* sa nachádzajú jednotlivé xml súbory, ktoré boli vygenerované pomocou API. Pre potreby služby je využitá voľne prístupná MSSQL Databáza v edícii Express.

Shots

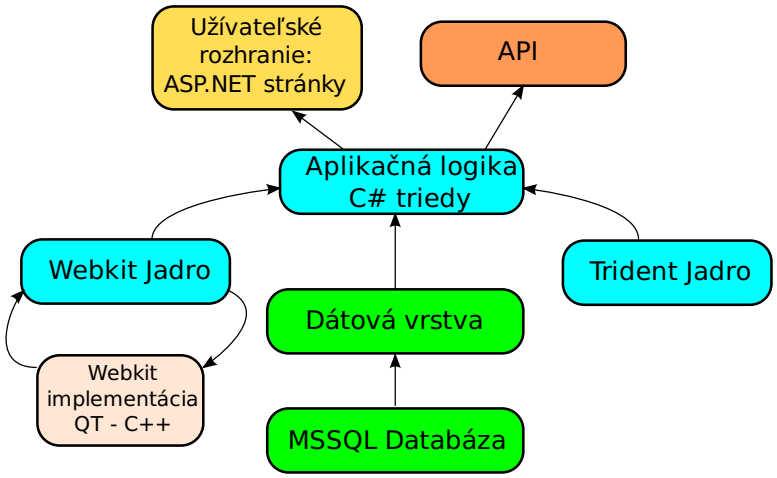
- URL – adresa z ktorej sa generoval obrázok
- FilePath – relatívna cesta ku vyrenderovanému obrázku
- Time – čas, kedy bola požiadavka zadaná do systému
- Size – veľkoť obrázku
- Engine – použité jadro

Ratings

- Rating – hodnotenie obrázku *1–5*
- IP – adresa z ktorej užívateľ hlasoval

4.4 Zhrnutie implementácie

Vo vyššie popísaných kapitolách som podrobnejšie popisoval jednotlivé časti webovej služby. Na obrázku 4.3 je zobrazený zjednodušený návrh aplikácie. Jednotlivé logické časti sú odlišené rôznymi farbami.



Obrázok 4.3: Kompletné schéma aplikácie

Kapitola 5

Výsledky

V tejto kapitole sa budem venovať porovnaniu implementovaných jadier, o ich možnostiach renderovania a taktiež ich rýchlostiach. Samozrejme budú spomenuté aj problémy a načrtnem možné varianty ich riešenia.

5.1 Porovnanie možností implementovaných jadier

Obe jadrá som testoval na veľkom počte webových stránok. Rýchlosť renderovanie je takmer u oboch jadrách zhodná. Jadro Trident je ale trochu pomalšie. V jadre Webkit je experimentálne zahrnutá podpora pre Silverlight, podpora technológie Flash je zahrnutá v jadre Trident. Ani jedno z jadier nemá problémy pri spracovaní priehľadných obrázkov typu png. Renderovanie použitých stránok by som rozdelil do niekoľkých kategórií.

- bezproblémové renderovanie
- renderovanie stránok so *Silverlightom*
- spracovanie obrázkov *SVG*
- možnosti spracovania HTML5
- Vypnutie alebo zapnutie podpory javascriptu
- Problémové stránky

Bezproblémové renderovanie

Väčšina stránok má zhodné výsledky renderovania v oboch jadrách. Je to spôsobené práve tým, že veľké množstvo stránok na internete je optimalizované pre všetky bežne používané internetové prehliadače.

(a) Jadro Trident (b) Jadro Webkit

Obrázok 5.1: Webová stránka *http://www.fit.vutbr.cz* vyrenderovaná v oboch prehliačoch

Stránky so Silverlightom

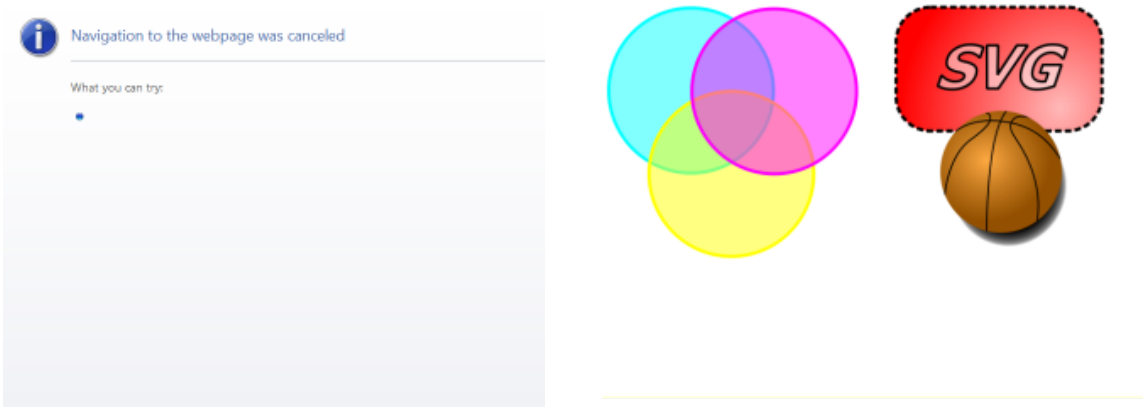
Technológia Silverlight sa pomaly začína na dnešných stránkach presadzovať. Jej hlavnou nevýhodou je, že je oficiálne podporovaná pre prehliadače Internet Explorer, Chrome 4 a Firefox 3. Pokiaľ použijeme iný prehliadač, tak sa nemusí silverlight zobraziť vôbec. Vo väčšine prípadov síce sa síce renderovanie nepodarí, ale jednoduchšie stránky sa vyrenderujú korektne. Pre renderovanie silverlightu je dôležité počkať niekoľko sekúnd po kompletnom načítaní. Je to potrebné aby sa stihol načítať jeho obsah. Pre jadro Trident nie je táto podpora implementovaná a namiesto silverlightového obsahu sa zobrazí len biely obdĺžnik.

(a) Jadro Trident (b) Jadro Webkit

Obrázok 5.2: Web *http://www.silverlight.net/showcase* vyrenderovaný v oboch prehliačoch

Spracovanie obrázkov SVG

Podpora obrázkov SVG nie je v jadre Trident vôbec zahrnutá. Pravdepodobne bude, až s príchodom v Internet Explorera 9. Jadro Webkit podobne ako aj ine jadrá nemá s touto technológiou problémy. Pri pokuse renderovať svg obrázok v jadre Trident, nastane chyba a vráti sa štandardná chybová stránka.

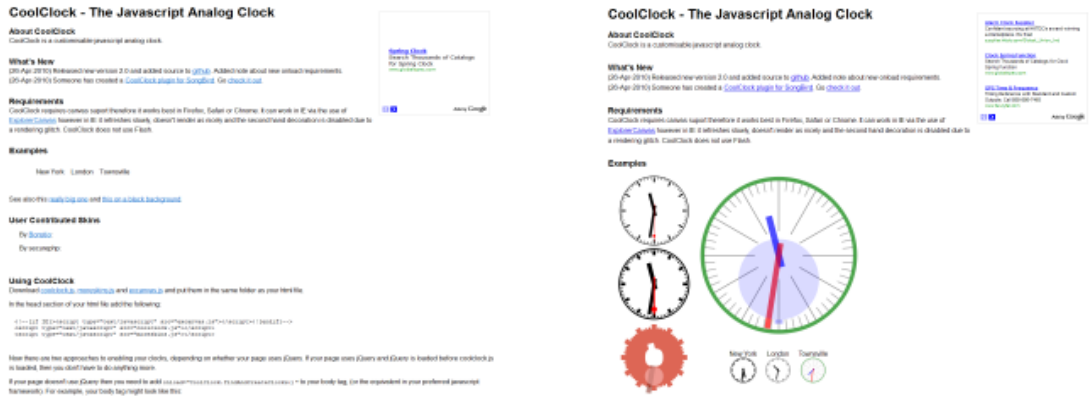


(a) Jadro Trident (b) Jadro Webkit

Obrázok 5.3: Renderovanie SVG obrázkov

Renderovanie HTML5 elementov

Podobne ako u SVG obrázkov nie je podpora jazyka HTML5 v jadre Trident zahrnutá. V niektorých prípadoch skonči renderovanie chybou, alebo skončí neúspechom na prekročenie časového limitu. V jadre Webkit, je integrovaná podpora pre tento prichádzajúci štandard. V implementovanom jadre, ale nie možné renderovať element `<video>`, renderovanie prvku `<canvas>` nie je problém.

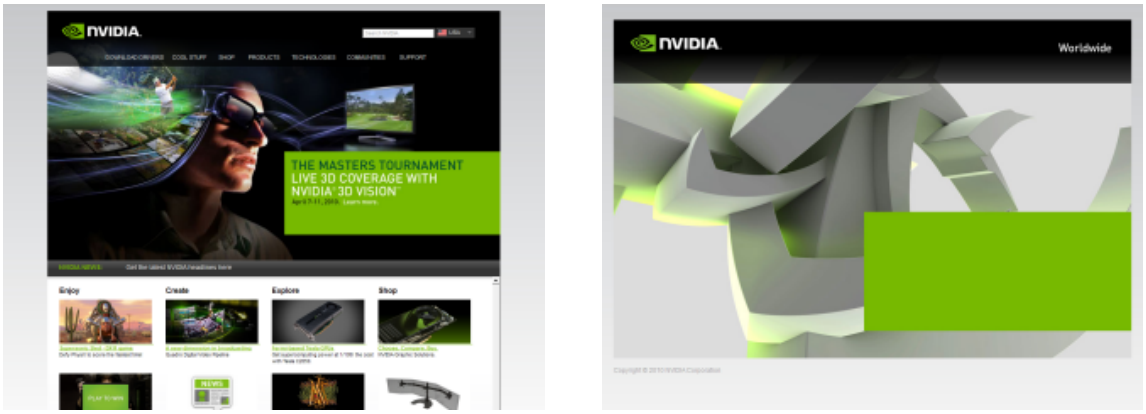


(a) Jadro Trident (b) Jadro Webkit

Obrázok 5.4: Anaógové hodiny v HTML5 – *http://randomibis.com/coolclock*

Vypnutie alebo zapnutie podpory javascriptu

V jadre webkit je implementovaná podpora pre vypnutie interpretovania jazyka javascript. V jadre Trident je podpora pre javascript zapnutá vždy. Na stránke nvidia.com môžeme pozorovať práve pozorovať zmeny pri zapnutom a vypnutom javascripte.

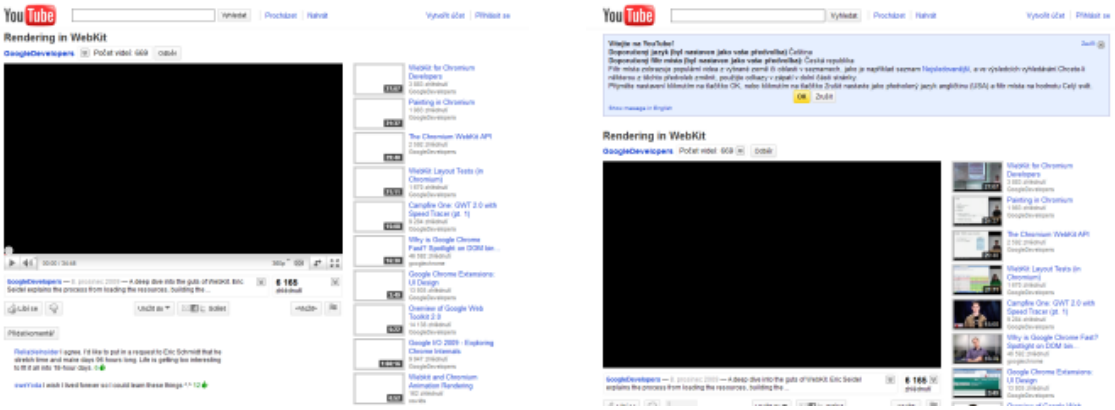


(a) Vypnutý javscript (b) Zapnutý javascript

Obrázok 5.5: Rozdiely na stránke nvidia.com

Problémové stránky

Medzi problémové stránky by som hlavne zaradil tie, ktoré obsahujú Flash alebo Silverlight. Často nie je na webových stránkach uvedený alternatívny obsah, ktorý by sa mal zobraziť v prípade nepodporovania daného zásuvného modulu. Flash objekty sú síce podporované v jadre Trident, ale zásuvné moduly sa začínaju inicializovať až po ukončení procesu renderovania, preto sa často Flash nezobrazí. Práve preto sa často pri generovaní obrázku zobrazí biele miesto namiesto vloženého objektu. Ako som už spomínal vyššie, stránky obsahujúce HTML5 alebo obrázky SVG je možné renderovať len za pomoci Webkit jadra. Celkovo sú dosiahnuté lepšie výsledky s Webkit jadrom, ktoré navyše obsahuje bohatšie možnosti nastavenia jednotlivých parametrov. Ďalšou zaujímavosťou je napríklad stránka *youtube.com*. Preto, že všetky obrázky na stránke načítavajú pomocou technológie ajax až vtedy, keď sú skutočne zobrazené na stránke (*lazy loading*). Z technického hľadiska je to síce veľmi zaujímavé, pretože to odľahčuje webové servery a obrázky a taktiež sa výrazne urýchli proces renderovanie stránky. Na druhú stranu ani jedno z implementovaných jadier nedokáže poznať, kedy je potrebné obrázky zobraziť, tak vo výslednom obrázku sa zobrazia „biele miesta“. Vo Webkit jadre sa dá tento nepríjemný fakt obísť počkaním niekoľkých sekúnd po skončení renderovania.



(a) nezobrazenie obrázkov (b) Jadro Webkit s počkaním 3 sekúnd

Obrázok 5.6: chybné renderovanie stránky *youtube.com*,

Problém taktiež predstavujú stránky ktoreé vyžadujú pre svoje zobrazenie HTTPS protokol. Ani jedno z jadier nedokáže pristúpiť ku stránke z toho dôvodu renderovanie skončí neúspechom pre presiahnutie časového limitu a následne je vrátený prázdny obrázok.

5.2 Výhody a nevýhody implementovanej služby

Hlavnou nevýhodou tejto aplikácie je hlavne jej náročnosť na programové prostriedky a hardware. Služba vyžaduje pre svoj beh operačný systém Microsoft Windows s nainštalovaným serverom IIS s minimálnou verziou .NET 3.5. Pre správne fungovanie je vyžadovaná možnosť spúšťať spustiteľné súbory exe (Webkit jadro). Práve toto obmedzenie je často na webhostingu z bezpečnostného hľadiska zakázané. Samozrejmosť je korektná inštalácia modulu pre Silverlight a Flash. Preto je pre umiestnenie najvhodnejší vlastný server, na ktorom by sa dali tieto parametre bezproblémov nastaviť.

Medzi výhody by som zaradil hlavne rýchlosť celej aplikácie. Na rozdiel od podobných služieb je rýchlosť vytváranie jednotlivých obrázkov vyššia a taktiež je možnosť si vybrať renderovacie jadro, pre ktoré je možne špecifikovať podrobnejšie nastavanie. Užívateľské rozhranie aplikácie je naprogramované tak, aby bolo prehľadné, ale zas po dôkladnejšom preskúmaní je možné zadať podrobnejšie nastavania. Hlavnú výhodu tiež vidím v naprogramovaní jednoduchého API, ktoré umožnuje využívať službu vzdialene a tak ju začleniť do vlastných programov.

5.3 Možné rozšírenia aplikácie

Na záver by som rád ešte uviedol možné rozšírenia, ktoré ma napadli počas implementácie. Najprínosnejším rozšírením by bolo pridanie ďalších renderovacích jadier do projektu. Taktiež plná podpora zásuvných modulov ako Flash, Silverlight a Javy. Zaujímavé by bolo taktiež rozšírenie API, pre pokročilejšie možnosti zadávania. Napríklad by bolo možné nastaviť maximálnu dobu, po ktorej uplynutí sa obrázok sám aktualizuje, možnosť zadať renderovanie stránok v pravidelnom časovom intervale, alebo renderovanie len určitej časti stránky (súčasný systém renderuje vždy celú stránku). Pokročilí užívatelia by určite ocenili vytváranie užívateľských účtov, kde by mohli spravovať svoje vyrenderované obrázky. Ob-

rázky by bolo vhodné taktiež ďalej spracovávať napríklad pridaním rôznych efektov alebo možnosť vložiť vlastný popisok priamo do výsledného obrázku. Určite by bolo zaujímavé prepracovať niektoré časti užívateľského rozhrania. Ako napríklad postupné zobrazovanie vygenerovaných obrázkov počas viacnásobného zadania. Súčasné rozhranie je síce veľmi interaktívne, ale vďaka skúsenostiam z praxe viem, že na užívateľské rozhranie sa kladie najväčší dôraz, pretože práve túto časť aplikácie vidí koncový užívateľ. Galéria by si tiež zaslúžila niekoľko zmien, ako napríklad, možnosť komentovať jednotlivé obrázky a taktiež pridať možnosť zdieľania na sociálnych sieťach. Pre pohodlné využívanie služby by bolo vhodné vytvoriť doplnky pre známe CMS systémy ako sú Drupal, Wordpress alebo Joomla.

Kapitola 6

Záver

Zadanie tejto práce bolo veľmi zaujímavé a pri jej implementácii som sa oboznámil s množstvom technológií, ktoré som doposiaľ nepoznal. Hlavným prínosom pre mňa bolo hlavne podrobnejšie preštudovanie platoformy .NET a zoznámenie sa s vývojový prostredím Visual Studio, ktoré umožnuje pohodlný vývoj. Všetky využité technológie som popísal v teoretickej časti.

Zo začiatku vyzerala celková implementácia veľmi jednoducho. Ako prvé som implementoval jadro Trident, ktoré je zahrnuté priamo v .NET. Následne som sa snažil implementovať aj ďalšie jadrá. Táto časť už bola výrazne náročnejšia. Celkovo ma prekvapila nepoužiteľnosť open-source projektov, ktoré sa snažia o pridanie renderovacích jadier pre .NET aplikácie. Následne som sa zoznámil aj s inými programovým technológiami, ktoré síce nie sú určené pre .NET, ale implementujú požadované jadrá. Z tohto dôvodu som sa taktiež začal zaujímať o prepojenie takejto aplikácie s ASP.NET stránkou. Z nájdených projektov som sa začal viac zaujímať o QT framework. Tento framework je kvalitne zdokumentovaný a navyše dodáva doplnok, ktorý umožňuje pohodlný vývoj QT aplikácií priamo vo Visual Studiu. Vďaka nemu sa mi podarilo implementovať jadro Webkit s rôznymi vylepšeniami, ktoré nie sú v Trident jadre zahrnuté. Prekvapila ma kvalita a rýchlosť tohto jadra, tak som ho určil ako predvolené jadro pre renderovanie stránok. Narozdiel od Trident jadra zvláda všetky moderné technológie s ktorými sa môžeme na internete stretnúť. Hlavne by som vystihol podporu obrázkov SVG a HTML5.

Pôvodná myšlienka bola implementovať tri jadrá. Zostávalo ešte relatívne známe jadro Gecko. Toto jadro je vďaka svojej komplexnosti najzložitejšie na implementáciu. Navyše pre operačný systém Microsoft Windows som našiel menej informácií o možnom zahrnutí tohto jadra. Pre GNU Linux je informácii a existujúcich implementácii podstatne viac. Po viacerých neúspechoch o pridanie podpory tohto jadra som sa rozhodol od neho upustiť a ostať len pri dvoch jadrách.

Literatúra

[1] Trident Layout Engine. [online] 2006 [cit. 2010-05-10].
URL http://www.guideto.com/web-browsers/trident-layout-engine

[2] Three Tier Architecture in ASP.NET. [online] 2009 [cit. 2010-05-10].
URL http:
//www.beansoftware.com/ASP.NET-Tutorials/Three-Tier-Architecture.aspx

[3] Dromaeo. [online] 2010 [cit. 2010-05-10].
URL https://wiki.mozilla.org/Dromaeo

[4] Embedding Mozilla. [online] 2010 [cit. 2010-05-10].
URL http://www.mozilla.org/projects/embedding

[5] SunSpider JavaScript Benchmark. [online] 2010 [cit. 2010-05-10].
URL http://www2.webkit.org/perf/sunspider-0.9/sunspider.html

[6] V8 Benchmark Suite - version 5. [online] 2010 [cit. 2010-05-10].
URL http://v8.googlecode.com/svn/data/benchmarks/v5/run.html

[7] Hickson, I.: The Web Standards Project Acid Tests. [online] 2010 [cit. 2010-05-10].
URL http://www.acidtests.org

[8] Malý, M.: REST: architektura pro webové API. [online] 2009 [cit. 2010-05-10].
URL http://zdrojak.root.cz/clanky/rest-architektura-pro-webove-api

[9] Stanclift, M.: Under the hood, a quick look at browser engines. [online] 2008 [cit. 2010-05-10].
URL http://www.neowin.net/news/main/08/09/07/
under-the-hood-a-quick-look-at-browser-engines

[10] Wikipedia: Gecko (layout engine). [online] 2010 [cit. 2010-05-10].
URL http://en.wikipedia.org/wiki/Gecko_(layout_engine)

[11] Wikipedia: HTML. [online] 2010 [cit. 2010-05-10].
URL http://en.wikipedia.org/wiki/HTML

[12] Wikipedia: Layout engine. [online] 2010 [cit. 2010-05-10].
URL http://en.wikipedia.org/wiki/Layout_engine

[13] Wikipedia: .NET Framework. [online] 2010 [cit. 2010-05-10].
URL http://en.wikipedia.org/wiki/.NET_Framework

[14] Wikipedia: WebKit. [online] 2010 [cit. 2010-05-10].
URL http://en.wikipedia.org/wiki/WebKit

Dodatok A

Spustenie aplikácie

Pretože sa nepodarilo zohnať vhodný hosting pre beh aplikácie, je nutné si ju spustiť na vlastnom lokálnom počítači. Najjednoduchšia možnosť je otvorenie projektu vo Visual Studiu 2008 (Verzia 2010 nebola otestovaná). Ak by sme chceli aplikáciu skompilovať tak, je potrebné zo stránok QT frameworku stiahnuť verziu QT pre Visual Studio a taktiež doplnok(*Visual Studio Add-in*), ktorý umožní kompiláciu a editáciu Webkit jadra. Pokiaľ nechceme inštalovať QT framework, tak samozrejme aplikácia obsahuje už predkompilovanú verziu pre Microsoft Windows 32bit. Webovú službu spustíme z menu Visual Studia – *Debug - Start Without Debugging*. Následne sa nám spustí vývojový server so stránkou našej služby.

Takéto spustenie je ale nedoporučené. Oveľa vhodnejšia je možnosť umiestnenia služby pod IIS serverom. Spustíme si IIS konzolu, kde si založíme novú stránku. Pre názornosť ju pomenujeme `Webcapture`. fyzické umiestnenie zvolíme `C:\inetpub\wwwroot\webcapture` doménové meno zvolíme webcapture (predpokladá sa existencia dns záznamu `webcapture IN A 127.0.0.1`) a potvrdíme. Na záložke Application Pools vyberieme vytvorený webcapture a zvolíme *Advanced Settings*, kde zmeníme položku *Identity* na *LocalSystem*. Ako posledný krok, je spustenie Visual Studia, otvoríme projekt obsahujúci zdrojové kódy a z vyberieme možnosť `Build - Publish Website`. Umiestnenie zvolíme, tak ako sme zadali v pri vytváraní webstránky z IIS konzoly. Pre správne fungovanie služby je potrebné nastaviť adresár `shots` na zapisovanie. Následne už môžeme do ľubovoľného web prehliadača zadat adresu `http://webcapture` a môžeme službu začať využívať. Pre podrobnejšie popísanie postupu je na CD umiestnený obrázkový návod.