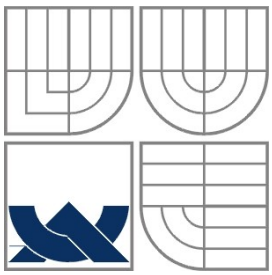
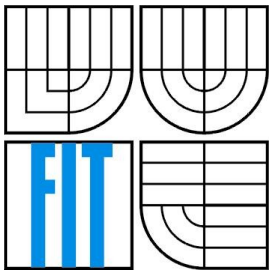


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

POROVNÁNÍ JAZYKŮ PRO TVORBU WEBOVÉHO REDAKČNÍHO SYSTÉMU

LANGUAGE COMPARISION FOR CONTENT MANAGEMENT SYSTEM

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE
AUTHOR

MARTIN KLEBAN

VEDOUCÍ PRÁCE
SUPERVISOR

ING. PETR CHMELAŘ

BRNO 2008

Abstrakt

Táto práca sa zaoberá porovnaním technológií Java (JSP/Servlet) a .NET (ASP.NET a C#) pre tvorbu webového redakčného systému. Teoreticky popisuje používané technológie, definuje požiadavky kladené na vytváraný systém a rozoberá jeho návrh. Popis implementácie systému je písaný formou porovnávania spôsobu implementácie jednotlivých častí v konkrétnom jazyku. V závere sú zhrnuté získané poznatky z implementácie.

Klíčová slova

Java, JSP, Servlet, .NET, ASP.NET, C#, porovnanie jazykov, webový redakčný systém

Abstract

This bachelor thesis deals with comparison of Java technologies (JSP/Servlet) and .NET (ASP.NET and C#) for the purpose of the formation of the content management systerm. It describes technologies used in theory, defines demands for the formation of the system and analyses its concept. The depiction of implementation of the system is realised in the form of comparison of implementation of individual parts of particular language. Gained pieces of knowledge are to be found in the enclosure.

Keywords

Java, JSP, Servlet, .NET, ASP.NET, C#, language comparision, web content management system

Citace

Martin Kleban: Porovnání jazyků pro tvorbu webového redakčního systému, bakalářská práce, Brno, FIT VUT v Brně, 2008

Porovnání jazyků pro tvorbu webového redakčního systému

Prohlášení

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

.....
Jméno Příjmení
Datum

Poděkování

Ďakujem svojmu vedúcemu Ing. Petrovi Chmelařovi za trpezlivosť pri vedení tejto práce, za jeho konštuktívne návrhy a podnetné pripomienky.

Obsah

1 Úvod

V súčasnosti sa internet stáva úložiskom nespočetného množstva informácií, každý deň pribúda kvantum nových správ, článkov, diskusných príspevkov, multimédií a pod., dostupných vďaka tejto službe širokému spektru užívateľov. Preto je takmer nemysliteľné, aby sa obsah internetových stránok publikoval editovaním statických stránok. Na tvorbu obsahu sa využívajú tzv. webové redakčné systémy, ktoré umožňujú kolektívu autorov efektívne spolupracovať pri budovaní obsahovej časti internetových portálov.

Webový redakčný systém predstavuje rozhranie umožňujúce užívateľovi jednoducho manipulovať s dátami, ktoré väčšinou bývajú fyzicky uložené v databáze, popr. v súborovom systéme ako súbory, napríklad vo formáte XML.

Na výstavbu redakčných systémov sa väčšinou používajú technológie bežiace na strane serveru, ako napríklad PHP, Coldfusion, Ruby on Rails, Java, .NET a pod. Táto práca sa zameriava na porovnanie niektorých z týchto technológií, konkrétne na technológie Java (JSP/Servlet) a .NET (ASP.NET a C#) pri vytváraní webového redakčného systému.

Redakčný systém vytvorený pomocou technológie Java využíva open-source databázu PostgreSQL, systém bežiaci na platforme .NET používa MSSQL.

Kapitola 2 v skratke popisuje základné princípy fungovania World Wide Web, protokolu HTTP, značkovacieho jazyka HTML, kaskádových štýlov CSS a jazyka JavaScript na strane klienta. Podrobnejšie sa zameriava na porovnávané technológie Java (JSP/Servlet) a .NET (ASP.NET a C#). Kapitola 3 popisuje redakčný systém z teoretického hľadiska, definuje základné požiadavky kladené na vytváraný systém, zaoberá sa jeho návrhom, popisuje priložený ER diagram a diagram prípadov užitia. Taktiež je rozobraná implementácia systému použitím technológie Java (JSP/Servlet) a implementácia rovnakého systému na platforme .NET.

Porovanie použitých technológií rozoberá kapitola 4, v závere sú zhrnuté poznatky z implementácie.

2 Technológie

Nasledujúca kapitola v skratke rozoberá základné princípy fungovania služby World Wide Web, popisuje technológie využívané pri tvorbe internetových stránok a aplikácií, ako napríklad HTML, CSS a JavaScript. Hlavnú pozornosť však venuje porovnávaným technológiám, a teda Jave a platforme .NET pre vývoj webových informačných systémov.

2.1 World Wide Web

World Wide Web (WWW) je množina internetových protokolov a ďalších, s nimi súvisiacich technológií, ktorá slúži k prezentácií hypertextových informácií [1].

Technológiu vyvinul v roku 1989 Timothy Bernes-Lee, ktorý pôsobil v Európskom centre pre jadrový výskum CERN. Prvým webovým prehliadačom bol Mosaic vytvorený v národnom počítačovom centre NCSA. V súčasnosti riadi vývoj štandardov súvisiacich s webovými technológiami konzorcium World Wide Web Consortium [1]. Podľa tohto konzorcia predstavuje World Wide Web akési univerzum na sieti dostupných informácií, úložisko ľudských poznatkov [2].

Na zostavovanie a prenos dát sa využívajú existujúce protokoly MIME a TCP/IP. Do jednotného prostredia sa integrujú aj služby dalších protokolov, ako napríklad FTP a Telnet [1].

Špeciálne pre WWW bol vyvinutý protokol HTTP, jazyk HTML, rozhranie CGI a adresy URL [1].

2.1.1 Hypertext

Hypertext je text, ktorý obsahuje odkazy na dalšie texty [3]. Rodina protokolov Web bola jedna z prvých, ktorá umožňovala rýchle a efektívne prezeranie veľkého množstva nesekvenčne uložených informácií. Informácie na Webe sú uložené vo forme orientovaného grafu [1].

2.1.2 Zdroje

Jednotlivé miesta v prostredí internetu nazývame zdroje. Zdroje poskytujú nejaké dáta, ktoré sa následne spracúvajú, jedná sa teda o zdroje dát [1].

Zdroje sú označované identifikátormi. Podľa [1] existujú dva typy identifikátorov:

- identifikátory špecifikujúce miesto v sieti, a teda sú závislé na umiestnení v sieti (anglicky Uniform Resource Locator URL)
- identifikátory pomenovavajúce zdroj, nezávislé na umiestnení v sieti (anglicky Uniform Resource Name URN)

Spoločným pojmom pre obe tieto skupiny je URI (Universal Resource Identifier) [1].

2.1.3 HTTP

Najpoužívanejším protokolom pre prenos dát medzi klientom a serverom je v súčasnosti HTTP, ktorý v značnej miere nahradil protokol FTP [1]. Pracuje s adresami URL a bol vytvorený pre hypertextové prostredia. Podľa W3C [4] sú posledné špecifikácie HTTP extensions a HTTP/1.1 stabilné a konzorcium pozastavilo svoje aktivity v ďalšom vývoji. Posledná verzia splnila všetky požiadavky na vytvorenie úspešného štandardu, ktorý odstránil nedostatky predchádzajúcich verzií HTTP protokolu.

2.1.4 CGI

CGI (Common Gateway Interface) je štandardom na prepájanie externých aplikácií s informačnými servermi. CGI program je vykonávaný v reálnom čase, a teda sú jeho výstupom dynamické informácie [5]. Na vytvorení CGI sa podieľalo viacero ľudí, známym je Robert M. McCool, ktorý okrem iného vytvoril aj referenčnú implementáciu CGI 1.0 web servera NCSA HTTPd [6].

2.2 HTML

HTML je typom dokumentu SGML, jeho značkám je priradená sémantika hypertextového dokumentu v prostredí WWW [7]. Pripravovaná verzia HTML5 už ale závislosť na SGML obsahovať nebude [8].

HTML je charakterizovaný množinou značiek a ich atribútov definovaných pre danú verziu. Medzi značky sa uzatvárajú časti textu dokumentu a tým sa určuje sémantika obsiahnutého textu. Názvy jednotlivých značiek sa uzatvárajú medzi uhlové zátvorky. Časť dokumentu tvorená otváracou značkou, nejakým obsahom a odpovedajúcou uzatváracou značkou tvorí tzv. element dokumentu. Napríklad <*strong*> je otváracia značka pre zvýraznenie textu a <*strong*>*nejaký text*</*strong*> je element obsahujúci zvýraznený text. Súčasťou obsahu elementu môžu byť ďalšie vnorené elementy. Atribúty sú doplňujúce informácie, ktoré upresňujú vlastnosti elementu [8].

Pre každú verziu existuje definícia pravidiel DTD (Document Type Definition). Od verzie 4.01 musí byť odkaz na deklaráciu DTD v dokumente uvedený pomocou kľúčového slova DOCTYPE. DTD definuje pre určitú verziu elementy a atribúty, ktoré je možno používať [8].

Dokument môže obsahovat aj ďalšie prvky [8]:

- direktívy, ktoré sú určené pre spracovateľa dokumentu
- komentáre, určené pre programátora, nie sú súčasťou obsahu dokumentu a nezobrazujú sa
- kód skriptovacích jazykov

- definície udalostí a kód pre ich obsluhu

2.2.1 Štruktúra dokumentu

Dokument v jazyku HTML má predpísanú štruktúru [8]:

1. Deklarácia DTD, povinná až vo verzii 4.01.
2. Koreňový element html, ktorý reprezentuje celý dokument.
3. Hlavička, ktorá obsahuje metadáta vzťahujúce sa k celému dokumentu. Definujú napríklad názov dokumentu, jazky, kódovanie, kľúčové slová, popis, a pod.
4. Telo dokumentu, ktoré obsahuje vlastný text dokumentu.

2.3 CSS

CSS je jazyk pre popis spôsobu zobrazenia dokumentov napísaných v jazykoch HTML, XHTML alebo XML. Jazyk bol navrhnutý organizáciou W3C. Doposiaľ boli vydané dve úrovne špecifikácie, CSS1 a CSS2, pracuje sa však aj na verzii CSS3 [9].

Hlavným zmyslom je umožniť návrhárom oddeliť vzhľad dokumentu od jeho štruktúry a obsahu. Pôvodne to mal umožniť už jazyk HTML, ale v dôsledku nedostatočných štandardov a konkurenčného boja výrobcov prehliadačov sa vyvinul inak [9].

Štýl definuje pravidla ako prvok reprodukovať. Slovo reprodukovať v sebe zahrňuje ako zobraziť, tak i iné spôsoby interpretácie prvku. Takýchto pravidiel môže byť mnoho, najviac používaným je napríklad farba. Kaskádovosť štýlov je daná hierarchickým usporiadaním pravidiel pre štýly. Každý podriadený štýl predefinuje rovnomenné nadriadené pravidlá pre istý prvok [7].

Pravidlá sú interpretované v tejto kaskáde [7]:

1. extérne šablony štýlov
2. lokálne štýly dokumentu
3. lokálny štýl konkrétneho prvku

Používanie kaskádových štýlov v zrovnaní so samotným HTML v praxi prináša výhodu rozsiahlejších možností formátovania. Ďalším prínosom je jednoduchá možnosť zachovania konzistentného štýlu. Napríklad, ak by sme chceli na všetkých stránkach našej webovej prezentácie zachovať rovnaké štýly pre nádpisy, zoznamy, zdôraznené časti textu a pod., bez použitia CSS by sme museli u každého objektu v každom dokumente nastavovať jeho vzhľad vždy odznova. Vykonať potom zmenu štýlu našej prezentácie by vyžadovalo nájsť a nahradiť atribúty všetkých značiek. V prípade použitia CSS by na to postačovala zámena externého súboru popisujucého štýly [9].

Taktiež môžu existovať rôzne štýly pre rôzne výstupné zariadenia. Existuje tak možnosť prostredníctvom CSS štýlov dokumentu určiť, ako bude vyzerať na papiery, pri projekcii či na PDA a

pod. Špecifikácie CSS nezabúdajú ani na zrakovo postihnutých. Je možné napísať štýly pre hlasový syntetizátor alebo hmatovú čítačku Braillovho písma. Taktiež je možnosť upraviť formátovanie podľa prehliadača, ktorým si užívateľ daný dokument zobrazuje [9].

2.4 JavaScript

JavaScript je interpretovaný programovací jazyk so základnou objektovo orientovanou koncepciou. Klientská verzia tohto jazyka je súčasťou väčšiny všeobecne rozšírených prehliadačov ako sú napríklad Internet Explorer, Mozilla Firefox a Opera.

Jádro jazyka je syntakticky veľmi podobné C, C++ alebo Jave. Podobnosť však končí na úrovni syntaxe. JavaScript je jazykom, ktorý má potlačenú typovú kontrolu. Ide o interpretovaný jazyk, ktorý mnohé myšlienky preberá z jazyka Perl. Nejedná sa však o zjednodušenú verziu Javy.

JavaScript pôvodne vznikal pod názvom LiveScript a zmena názvu nastala tesne pred jeho uvedením na trh z marketingových dôvodov. V produktoch Microsoft sa tento produkt vyskytuje pod názvom JScript [10].

JavaScript je možné použiť aj na strane serveru. Prvou implementáciou JavaScriptu na strane serveru bol LiveWire firmy Netscape uvedený v roku 1996. Dnes existuje niekoľko možností vrátane opensource implementácie Rhinola založenej na Rhino, gcj a Apache [11].

Pojem klientského JavaScriptu vznikol integráciou JavaScriptu do internetového prehliadača. Klientský JavaScript používa okrem univerzálnych rysov jazyka aj objektový model dokumentov DOM. Ich synergiou je možné dosiahnuť dynamické chovanie webových stránok, pre ktoré sa ustálil názov DHTML. Okrem klientského využitia je možné jadro jazyka obohatené o ďalšie knižnice používať aj mimo prehliadače, napríklad formou tzv. Script Hostingu [10].

2.5 Java

Od roku 1991 vyvíjala firma Sun Microsystem programovací jazyk na princípoch C a C++ pre vstavané systémy. Jazyk mal pôvodne názov Oak, podľa dubu, ktorý stál pred oknom pána Goslinga, vedúceho týmu. Projektu sa príliš nedarilo, ale v roku 1993 si firma Sun Microsystems uvedomila vzrastajúcu dôležitosť WWW a možnosť využiť Javu aj pre vývoj webových aplikácií. V máji roku 1995 bola Java oficiálne predstavená na konferencii. Už behom tejto prednášky bolo mnohým účastníkom jasné, že ide o nástroj, ktorý zohrá významnú rolu pri vytváraní webových aplikácií [12].

2.5.1 Základné charakteristiky jazyka Java

Pri štandardnom postupe spracovania prechádza program v Jave piatimi fázami [12]:

1. editovaním
2. prekladom
3. zavedením
4. overovaním
5. vykonaním

Štyri z týchto fáz sú bežné aj pre ostatné programovacie jazyky. Fáza overovania je niečím novým, pre Javu (najmä pri programovaní webových aplikácií) veľmi dôležitým aspektom umožňujúcim dosiahnutie vysokej bezpečnosti spusteného programu, čím je myslená hlavne ochrana toho, kto program spúšťa [12].

Jazyk Java môžeme ďalej charakterizovať ako [13]:

- **jednoduchý** - jeho syntax je zjednodušenou verziou syntaxe jazyka C a C++. Odpadla väčšina konštrukcií, ktoré spôsobovali programátorom problémy a na druhú stranu pribudla rada užitočných rozšírení.
- **objektovo orientovaný** - s výnimkou ôsmich primitívnych dátových typov sú všetky ostatné dátové typy objektové.
- **distribuovaný** - je navrhnutý pre podporu aplikácií v sieti (podporuje rôzne úrovne sieťového spojenia, prácu so vzdialenými súbormi, umožňuje vytvárať distribuované klientské aplikácie a servery).
- **interpretovaný** - namiesto skutočného strojového kódu sa vytvára iba tzv. medzikód. Tento formát je nezávislý na architektúre počítača alebo zariadenia. Program tak môže pracovať na ľubovoľnom počítači alebo zariadení, ktorý ma k dispozícii interpret Javy, tzv. virtuálny stroj Javy JVM. V neskorších verziách nebol medzikód priamo interpretovaný, ale pred svojím vykonaním dynamicky skompilovaný do zdrojového kódu daného počítača, tzv. just in time compilation - JIT. Táto vlastnosť zásadným spôsobom zrýchlila vykonávanie programov v Jave, ale výrazne spomalila štart programu. V súčasnosti sa prevažne používajú technológie zvané HotSpot compiler, ktoré medzikód spočiatku interpretujú a na základe štatistík získaných z tejto interpretácie neskôr vykonajú preklad často používaných častí do strojového kódu vrátane ďalších dynamických optimalizácií, ako je napríklad inlining krátkych metód a pod.
- **robustný** - je určený pre vytváranie vysoko spoľahlivého softwaru a z toho dôvodu neumožňuje niektoré programátorské konštrukcie, ktoré bývajú častou príčinou chýb, napríklad správa pamäti, príkaz goto, používanie ukázateľov a pod. Používa tzv. silnú typovú kontrolu, teda všetky premenné musia mať definovaný svoj dátový typ. Správa pamäti je realizovaná pomocou automatického Garbage collectoru, ktorý automaticky vyhľadáva už nepoužívané časti pamäte a uvoľňuje ich na ďalšie použitie. To bolo v prvých verziách opäť

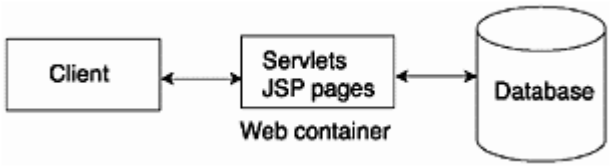
príčinou pomalejšieho behu programov. V posledných verziách behových prostredí je vďaka novým algoritmom a tzv. generačnej správe pamäte tento problém značne eliminovaný. Pri generačnej správe pamäte je pamäť rozdelená na viacero častí, v každej sa používa iný algoritmus pre garbage collection a objekty sú medzi týmito časťami presúvane podľa dĺžky svojho života.

- **bezpečný** - ma vlastnosti, ktoré chránia počítač v sieťovom prostredí, na ktorom je program spracovávaný, pred nebezpečnými operáciami alebo napadnutím vlastného operačného systému kódom útočníka.
- **nezávislý na architektúre** - vytvorená aplikácia beží na ľubovoľnom operačnom systéme alebo ľubovoľnej architektúre. Ku spusteniu programu postačuje len to, aby bol na danej platforme nainštalovaný správny virtuálny stroj. Podľa konkrétnej platformy sa môže prispôsobiť vzhľad a chovanie aplikácie.
- **prenositeľný** - vedľa zmienenej nezávislosti na architektúre je jazyk nezávislý aj čo sa týka vlastností základných dátových typov. Je napríklad explicitne určená vlastnosť a veľkosť každého z primitívnych dátových typov. Pod prenositeľnosťou sa však myslí iba prenášanie v rámci jednej platformy Javy. Pri prenášaní medzi platformami Javy je potrebné dať pozor na to, že platforma určená pre jednoduchšie zariadenia nemusí podporovať všetky funkcie dostupné na platforme pre zložitejšie zariadenia a okrem toho môže definovať niektoré vlastné triedy doplňujúce nejakú špeciálnu funkcionalitu alebo nahradzujúce triedy vyššej platformy, ktoré sú pre nižšú platformu príliš komplikované.
- **viacúlohový** - podporuje spracovanie viacvláknových aplikácií.
- **výkonný** - aj keď sa jedná o jazyk interpretovaný, nie je stráta výkonu významná, pretože prekladače pracujú v režime HotSpot.
- **dynamický** - bol navrhnutý pre nasadzovanie vo vyvíjajúcom sa prostredí. Knižnica môže byť dynamicky za chodu rozširovaná o nové triedy a funkcie, a to ako z extérnych zdrojov, tak aj vlastným programom.
- **elegantný** - je jednoducho čitateľný, priamo vyžaduje ošetrenie výnimiek a typovú kontrolu.

Ako nevýhodu Javy môžme uviesť pomalší štart programov oproti jazykom so statickou kompiláciou. U jednoduchších programov tiež väčšia pamäťová náročnosť pri behu spôsobená nutnosťou mať v pamäti celé behové prostredie [13].

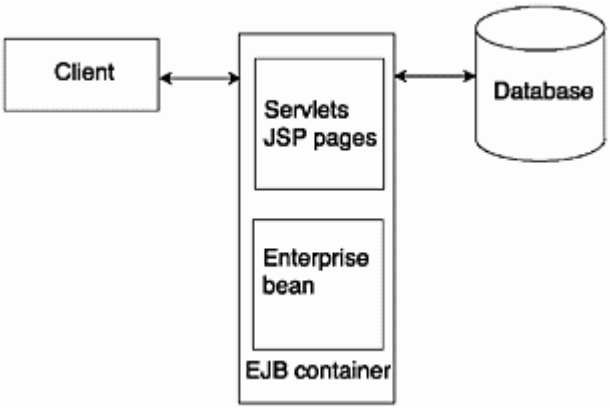
2.5.2 Webové aplikácie v Jave

Pri vytváraní webových aplikácií sa bežne používajú dve základné architektúry [14]. Prvá z nich využíva servlety a JSP v strednej vrstve za účelom obsluhy klientov a spracovania programovej logiky. Zobrazená je na obrázku 1.



Obr. 1: Architektúra JSP/servlet (zdroj: [14])

Druhá architektúra zahŕňa použitie servera J2EE a technológie EJB. Táto architektúra nachádza svoje využitie hlavne pri rozsiahlých aplikáciach, ktoré vyžadujú rozšíriteľnosť a preto presahuje rozsah tejto práce. Zobrazená je na obrázku 2.



Obr. 2: Architektúra J2EE (zdroj: [14])

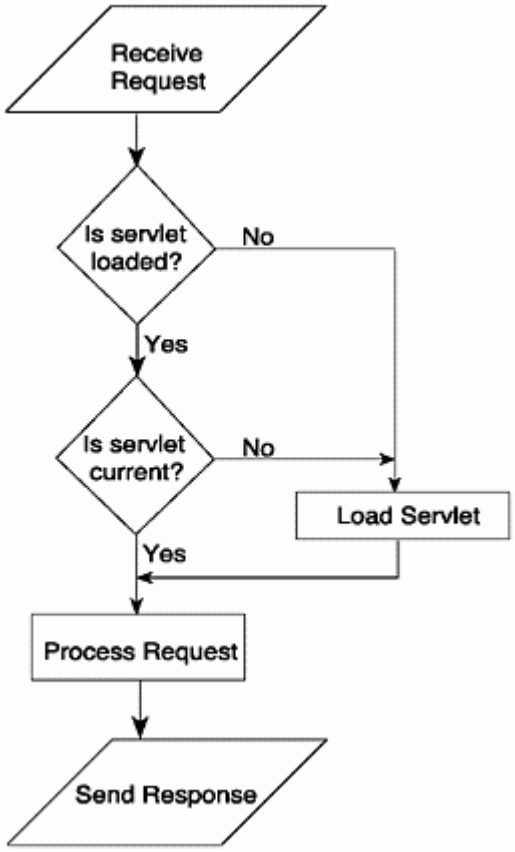
2.5.2.1 Servlety

Servlety sú odpoveďou technológie Java na programovanie pomocou CGI. Jedná sa o program, ktorý beží na webovom servery a pôsobí ako stredná vrstva medzi požiadavkom prichádzajúcim z webového prehliadača alebo od iného HTTP klienta a databázou či aplikáciou na servery HTTP [15].

Ich úlohou je [15]:

1. Prečitať všetky dáta odosielané užívateľom. Tieto údaje sa obvykle ukládajú do formulára na webovej stránke, ale mohli by rovnako prichádzať aj z javovského apletu alebo zo zákaznického programu klienta HTTP.
2. Vyhľadať všetky ďalšie informácie o požiadavku, ktoré sú uložené v požiadavku HTTP. Tieto informácie zahrňujú podrobnosti o schopnostiach prehliadača, cookies, hostiteľskom mene klienta a pod.
3. Vytvoriť výsledky. Tento proces môže vyžadovať komunikáciu s databázou.
4. Sformátovať výsledky vo vnútri dokumentu. Vo väčšine prípadov to zahňuje uloženie informácie do stránky HTML.
5. Nastavenie vhodných parametrov odozvy HTTP. To znamená zdelenie prehliadaču, aký typ dokumentu sa vracia, nastavenie paramterov cookies, odkládacej pamäte a pod.
6. Odosielanie dokumentu späť ku klientovi.

Servlet je triedou jazyka Java, ktorá môže byť dynamicky načítaná a spustená špeciálnym webovým serverom. Tento server sa zvykne nazývať kontajner.



Obr. 3 Princíp servletu (zdroj: [14])

Obrázok 3 zobrazuje princíp fungovania servletov. Ak príde na servlet prvý požiadavok, kontajner daný servlet načíta. Prepošle sa mu užívateľský požiadavok, ten následne spracuje a odpoveď vráti kontajneru. Ten odošle odpoveď späť užívateľovi. Servlet ostane uložený v pamäti pre prípadnú obsluhu ďalších požiadavkov. Kontajner ho odstráni až pri prípadnom nedostatku pamäte.

Pri každom požiadavku na servlet, kontajner porovná uložený servlet s jeho súborom class. Ak je súbor class novší, servlet sa znovu načíta do pamäte. Nie je teda potrebné reštartovať kontajner pri každej zmene servletu.

Technológia servletov je základom tvorby webových aplikácií pomocou jazyka Java. Je to jedna z najdôležitejších technológií a tvorí základ pre ďalšiu techológiu zvanú JavaServer Pages - JSP [14].

2.5.2.2 JSP

Technológia JSP umožňuje zmiešať bežné statické HTML stránky s dynamicky generovaným obsahom zo servletov. Aj ked nezaisťuje žiadne schopnosti, ktoré by v princípe nemohli byť vykonané servletom, ich výhodou je možnosť oddelenia výzoru aplikácie od programovej logiky. Na pozadí sa dokumenty JSP automaticky prekladajú do servletov [15].

JSP je rozšírením technológie servletov pre vývoj webových aplikácií a teda je bežné používať servlety a JSP súčasne v rovnakej aplikácii [14].

2.5.2.3 JSP Custom Tags

Custom Tag je užívateľom definovaný element používaný na stránkach JSP. Pri konverzii JSP stránky na servlet sa tento element prekonvertuje na operácie na objekte zvanom tag handler. Pri vykonávaní servletu stránky JSP sú tieto operácie následne vyvolané [16].

Užívateľom definované elementy môžu [16]:

- byť upravované atribútmi predanými volajúcou stránkou.
- pristupovať ku všetkým dostupným objektom stránky JSP.
- modifikovať odpoveď generovanú volajúcou stránkou
- komunikovať medzi sebou navzájom. Napríklad je možné vytvoriť a inicializovať komponentu JavaBeans, vytvoriť premennú odkazujucú sa na túto komponentu v jednom elemente a potom túto komponentu použiť v nejakom ďalšom elemente.
- byť v sebe vnorené a umožňovať tak komplexnú interakciu vo vnútri stránky JSP.

2.5.2.4 JavaBeans

JavaBeans sú softwarové komponenty, v praxi, triedy napísane v jazyku Java, ktoré dodržujú určité konvencie. Používajú sa na zapúzdrenie viacerých objektov do jedného. Nie je teda potrebné

predávať všetky objekty individuálne. Dodržiavané konvencie sa týkajú názvoslovia metód, konštrukcie a správania [17].

Konkrétne sa jedná o tieto konvencie [17]:

- Triedy musia mať verejný konštruktor bez argumentov. To umožňuje jednoducho vytvárať ich inštancie.
- Vlastnosti musia byť dostupné pomocou get, set a ďalších metód dodržujúcich konvencie názvoslovia. To umožní rôznym frameworkom jednoduchú prácu s týmito komponentami.
- Trieda by mala byť serializovaná. To umožní aplikáciam a frameworkom spoľahlivo ukladať a obnovovať stav komponenty.

2.5.2.5 Filtre

Špecifikácia servletov verzie 2.3 predstavila nový typ komponenty, zvaný filter. Filtre dokážu dynamicky zachytiť prichádzajúce požiadavky alebo odpovede pre klienta ešte pred ich prijatím adresovaným prvkom, a pozmeniť alebo nejako použiť informácie v nich obsiahnuté [18].

Filtre sú dôležité hneď z niekoľkých dôvodov [18]. Jednak umožňujú zapúzdriť periodické úlohy do znovupoužiteľných jednotiek. Taktiež môžu byť použité na transformovanie odpovede zo servletu alebo stránky JSP. Častou úlohou pre webové aplikácie je formátovanie dát odoslaných späť ku klientovi. So vzrastajúcou mierou klienti požadujú iné formáty ako len HTML (napríklad WML).

Mnoho kontajnerov pre servlety a stránky JSP predstavili vlastný mechanizmus filtrov. Na ich implementácii pracovala pre každý kontajner iná skupina vývojárov, čoho výsledkom bola nízka znovupoužiteľnosť kódu filtrov pri prechode na iný kontajner. Až po zahrnutí filtrov do špecifikácie Java servletov majú vývojári možnosť vývoja filtrov prenositeľných medzi kontajnermi.

Častým využitím filtrov je napríklad [18]:

- Autentifikácia - blokovanie požiadavkov založené na identite užívateľa.
- Logovanie a kontrola účtov užívateľov webovej aplikácie
- Kompresia dát.
- Lokalizácia - smerovanie požiadavku a odpovede do konkrétneho jazyka.
- Konverzia obrázkov
- XSL/T transformácie XML obsahu

To sú len niektoré z veľkého množstva využitia filtrov. Ďalšími oblasťami sú napríklad kryptovanie, spuštanie akcií pri prístupe ku zdroju, ukladanie do vyrovnávacej pamäte a pod.

Samotné naprogramovanie filtrov je len polovicou práce pri používaní filtrov. Je taktiež potrebné určiť, ako sa tieto filtre namapujú na servlety keď sa aplikácia umiestní do webového kontajnera [18].

Oddelenie programovania a konfigurácie je hlavnou výhodou mechanizmu filtrov [18].

- Pri zmene vstupov alebo výstupov nie je potrebná prekompilácia. Stačí len zmeniť textový konfiguračný súbor. Napríklad, pridanie kompresie pri sťahovaní PDF súborov je len záležitosťou namapovania kompresného filtra na servlet, ktorý spravuje sťahovanie súborov.
- Experimentovanie s filtrami je vďaka jednoduchej konfigurácii zjednodušené.

2.6 .NET

Microsoft .NET Framework je softwarová komponenta, ktorá je súčasťou operačných systémov Microsoft Windows. Obsahuje obrovskú knižnicu predprogramovaných riešení na bežné programátorské problémy a spravuje programy napísané špecificky pre tento framework. Predprogramované riešenia, ktoré tvoria základ Base Class Library, pokrývajú širokú škálu oblastí vrátane užívateľského rozhrania, prístupu k dátam, databáze, kryptografie, numerických algoritmov, sieťovej komunikácie a vývoja webových aplikácií [19].

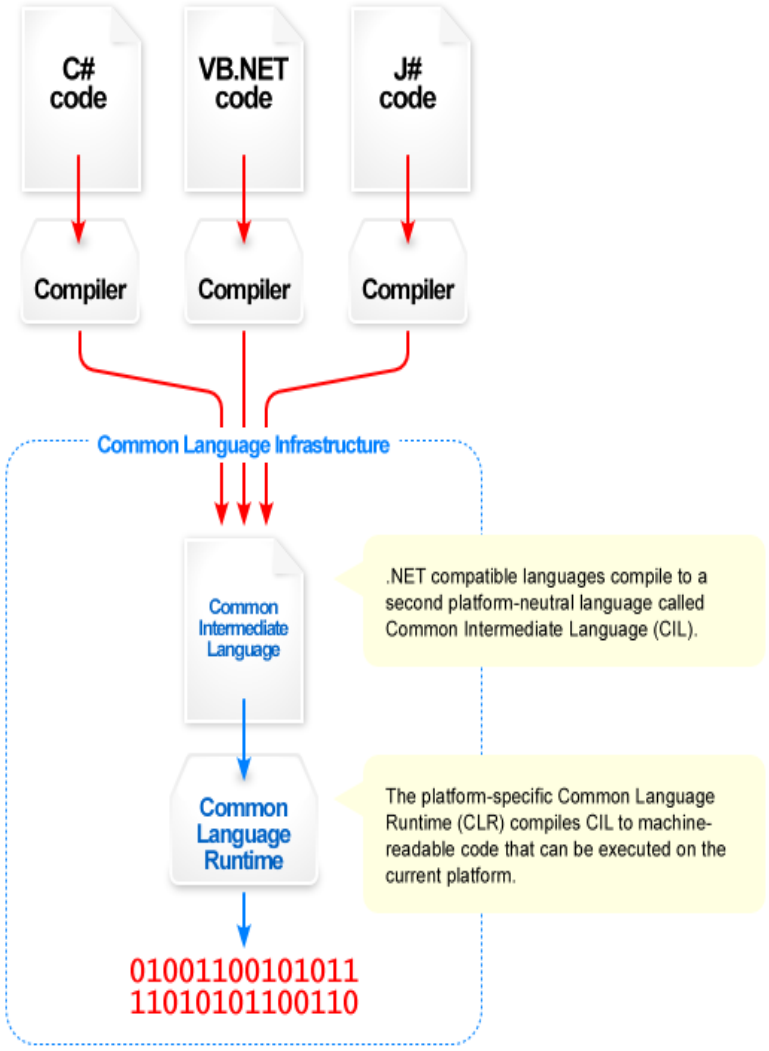
Programy napísané pre .NET Framework vyžadujú softwarové prostredie, ktoré spravuje požiadavky na ich beh. Behové prostredie, ktoré je taktiež súčasťou .NET Frameworku, je známe ako Common Language Runtime (CLR). CLR zastáva úlohu aplikačného virtuálneho stroja, takže programátori nemusia zvažovať parametre špecifického CPU, na ktorom program pobeží. CLR sa taktiež stará o ďalšie služby ako bezpečnosť, správa pamäte a pod. Základná knižnica tried spolu s CLR tvoria .NET Framework [19].

2.6.1 Základné charakteristiky .NET Frameworku

Medzi hlavné charakteristiky platformy .NET patria [19]:

- **súčinnosť** - pretože interakcia medzi novšími a staršími aplikáciami je bežne požadovaná, .NET Framework v sebe zahŕňa prostriedky na prístup ku funkcionalite, ktorá je implementovaná v progamoch, ktoré nebežia v prostredí .NET.
- **Common Runtime Engine** - programovacie jazyky .NET Frameworku sa kompilujú do medzijazyka, tzv. Common Intermediate Language (CIL). V implementácii firmy Microsoft sa tento medzijazyk neinterpretuje, ale s využitím JIT sa kompiluje do strojového kódu. Kombinácia týchto konceptov sa nazýva Common Language Infrastructure (CLI).

Implementácia CLI firmou Microsoft je známa ako Common Language Runtime (CLR). Obrázok Obr.4 znázorňuje infraštruktúru CLI.



Obr.4: CLI

- **nezávislosť na jazyku** - .NET Framework predstavil Common Type System (CTS). Špecifikácia CTS definuje všetky možné typy dát a programovové konštrukcie podporované CLR a určuje spôsob ich vzájomnej interakcie. Vďaka tomu .NET podporuje výmenu inštancií typov medzi aplikáciami napísanými v ktoromkoľvek z jazykov podporovaných platformou .NET. Existuje mnoho podporovaných jazykov, zaujímavým je J#, čo je vlastne implementácia Javy podľa špecifikácie Common Language Specification (CLS).
- **Base Class Library (BCL)** - základná knižnica tried, je súčasťou Framework Class Library (FCL). Obsahuje funkcionalitu dostupnú všetkým jazykom .NET Frameworku. BCL

obsahuje triedy, ktoré v sebe zahŕňajú veľké množstvo bežných funkcií, napríklad pre prácu so súbormi, vykresľovaním grafiky, prácu s databázou a XML dokumentmi.

- **bezpečnosť** - .NET poskytuje všetkým aplikáciam bezpečnostný model a tým zamedzuje vzniku bezpečnostným problémom ako napríklad pretečenie vyrovnávacej pamäte a pod.
- **prenositeľnosť** - návrh .NET frameworku v sebe zahŕňa nezávislosť na cieľovej platforme. To znamená, že programy napísané pre .NET pobežia bez akejkoľvek zmeny na každom type systému, pre ktorý je tento framework implementovaný. Komerčné implementácie firmy Microsoft pokrývajú Windows, Windows CE a Xbox360. Naviac firma Microsoft predložila špecifikácie pre CLI, C# a C++/CLI pre ECMA a ISO, čo z nich robí dostupné ako otvorené štandardy. Takto umožnili tretím stranám vytvoriť kompatibilné implementácie frameworku a jeho jazykov na iných platformách.

2.6.2 ASP.NET

ASP.NET je framework na tvorbu dynamických webových stránok, aplikácií a služieb. Prvá verzia vyšla v roku 2002 spolu s .NET Frameworkom. ASP.NET je taktiež postavený na Common Language Runtime, čo programátorom umožňuje písať kód ASP.NET použitím ľubovoľného .NET jazyka [20, 21].

ASP.NET nie je len novou verziou klasického ASP. Podľa [21] predstavuje najpokročilejšiu platformu pre tvorbu webových aplikácií. ASP.NET taktiež označuje ISAPI rozšírenie webového servera IIS.

2.6.2.1 Životný cyklus ASP.NET aplikácie

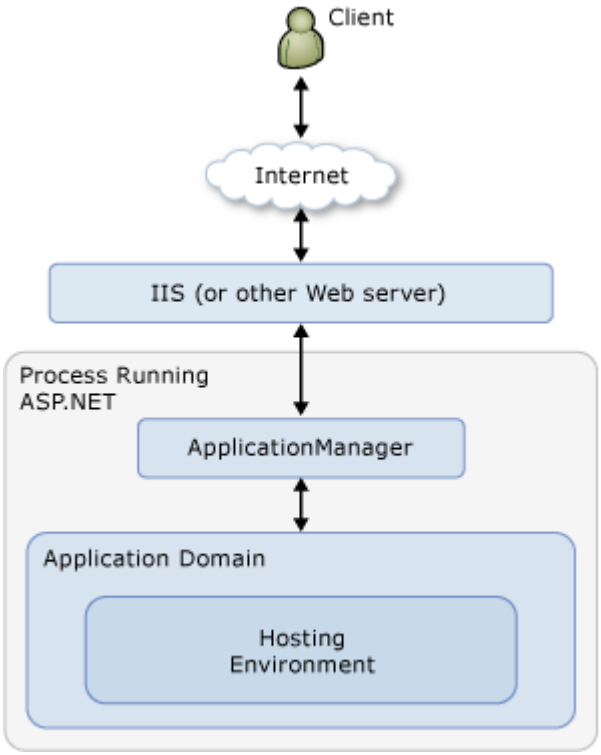
Životný cyklus aplikácie ASP.NET začína odoslaním požiadavku na webový server (pre aplikácie ASP.NET je to väčšinou IIS). ASP.NET je ISAPI rozšírenie webového servera. Ak server príjme požiadavok, preverí príponu požadovaného súboru, určí ktoré ISAPI rozšírenie by malo tento požiadavok obslúžiť a odošle mu ho. ASP.NET obsluhuje súbory s príponami, ktoré sú pre neho namapované, ako napríklad .aspx, .ascx, .ashx a .asmx [23].

Keď ASP.NET dostane prvý požiadavok, trieda ApplicationManager vytvorí aplikačnú doménu, ktorá zabezpečuje izolovanie aplikácií. Je taktiež vytvorená aj inštancia triedy HostingEnviroment, ktorá ponúka prístup k informáciam o aplikácii, ako napríklad názov adresára kde je aplikácia uložená [23]. Vytvorenie aplikačnej domény popisuje obrázok 5.

Následne sú vytvorené základné objekty ako HttpContext, HttpRequest a HttpResponse. Po nich nastáva štart aplikácie vytvorením inštancie triedy HttpApplication. Ak má ale aplikácia súbor

Global.asax, ASP.NET vytvorí namiesto toho inštanciu tejto tiredy, ktorá je odvodená od HttpApplication [23].

Štart aplikácie zobrazuje obrázok 6.

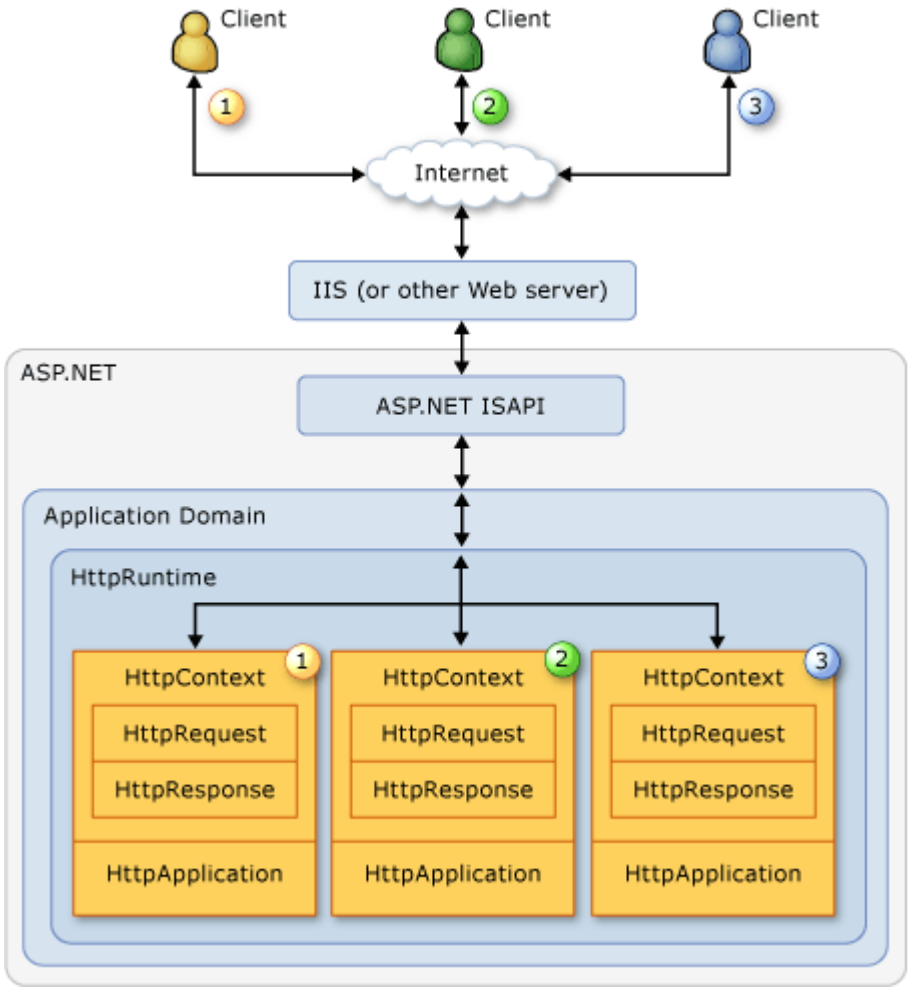


Obr. 5: Vytvorenie aplikačnej domény [23].

Po spustení aplikácie sa pri spracovaní požiadavku vyvolajú nasledujúce události [23]:

1. Validácia požiadavku, ktorá preveruje informácie odoslané prehliadačom a zisťuje, či náhodou neobsahujú známky potenciunálneho útoku.
2. Vykonanie mapovania URL.
3. Spustenie události BeginRequest.
4. Spustenie události AuthenticateRequest.
5. Spustenie události PostAuthenticateRequest.
6. Spustenie události AuthorizeRequest.
7. Spustenie události PostAuthorizeRequest.
8. Spustenie události ResolveRequestCache.
9. Spustenie události PostResolveRequestCache.

10. Podľa prípony súboru požadovaného zdroja sa vyberie trieda ktorá implementuje rozhranie IHttpHandler, aby tento požiadavok spracovala.
11. Spustenie události PostMapRequestHandler.
12. Spustenie události AcquireRequestState.
13. Spustenie události PostAcquireRequestState.



Obr. 6: Štart aplikácie [23].

14. Spustenie události PreRequestHandlerExecute.
15. Zavolanie metódy ProcessRequest (alebo jej asynchronnej verzie) odpovedajucej triedy na spracovanie požiadavku.
16. Spustenie události PostRequestHandlerExecute.
17. Spustenie události ReleaseRequestState.
18. Spustenie události PostReleaseRequestState.
19. Vykonanie prípadného filtrovania odpovede.
20. Spustenie události UpdateRequestCache.

21. Spustenie události PostUpdateRequestCache.
22. Spustenie události EndRequest.
23. Spustenie události PreSendRequestHeaders.
24. Spustenie události PreSendRequestContent.

Životný cyklus ASP.NET aplikácie je možné rozšíriť pomocou tried implementujúcich rozhranie IHttpModule.

2.6.2.2 Webové formuláre

Stránky ASP.NET, oficiálne známe ako webové formuláre (Web Forms), sú dôležitou časťou aplikácie ASP.NET. Poskytujú skutočný výstup webovej aplikácie, ktorý požadujú klienti. Webové formuláre sú obsiahnuté v súboroch s príponou ASPX. Tieto súbory typicky obsahujú statický obsah HTML alebo XHTML spolu so značkami definujúcimi serverové ovládacie prvky. Dynamický kód, ktorý sa vykonáva na serverovej strane môže byť umiestnený medzi značky <% a %>, čo je podobné aj v iných webových technológiach ako PHP, JSP a ASP. V praxi sa však tento spôsob veľmi nepraktizuje až na výnimku pripájania dát, pretože vyžaduje viacero volaní pri vykresľovaní stránky a tým vlastne spomaľuje celý tento proces [20, 22].

Po spustení ASP.NET stránka prechádza niekoľkými fázami. V nich dochádza napríklad ku inicializácii, ku vytvoreniu inštancií ovládacich prvkov, vykresleniu a pod. Presný zoznam stavov je podľa [24]:

1. **Požadovanie stránky**. Požadovanie stránky nastáva ešte pred začiatkom životného cyklu stránky. Keď je stránka požadovaná užívateľom, ASP.NET určí, či sa stránka musí prekompilovať (začiatok životného cyklu) alebo sa použije verzia uložená vo vyrovnávacej pamäti.
2. **Štart**. Pri štarte stránky sa nastavia jej vlastnosti ako Request, Response, UICulture a IsPostBack.
3. **Inicializácia stránky**. Počas inicializácie sa sprístupnia všetky ovládacie prvky, na stránku sa aplikujú šablóny a pod.
4. **Načítanie**.
5. **Validácia**.
6. **Ošetrenie události odoslania požiadavku metódou POST**.
7. **Vykreslenie**. Počas fázy vykresľovania zavolá stránka metódu Render všetkých ovládacích prvkov. Táto metóda vypíše výstup daného ovládacieho prvku.
8. **Uvolnenie**. Táto fáza nastáva po tom, ako je stránka vykreslená, odoslaná klientovi a pripravená na vyradenie.

2.6.2.3 Model Code-behind

Pri práci s dynamickým kódom sa odporúča používať tzv. Code Behind Model, ktorý umiestňuje programový kód do osobitného súboru. Tento súbor zvykne niesť názov Súbor.aspx.cs, kde Súbor.aspx je webový formulár ku ktorému sa viaže. Pri používaní tohto štýlu programovania sa kód píše tak, aby reagoval na určité udalosti, ako napríklad načítanie stránky, kliknutie myšou na ovládací prvok a pod. Jedná sa teda o technológiu podobnú oddeleniu kontroléra od pohľadu pri architektúre model-view-controller [20, 22].

2.6.2.4 Serverové ovládacie prvky

ASP.NET rieši problém neprehľadného kódu pomocou serverových ovládacich prvkov. Serverové ovládacie prvky sú elementy vykonávané na strane serveru. Existujú tri základné typy [25]:

1. HTML serverové ovládacie prvky.
2. Webové ovládacie prvky
3. Validačné ovládacie prvky

3 Redakčný systém

Redakčný systém (anglicky Content Management System) je program, ktorý sa používa na tvorbu a správu dokumentov, najčastejšie webového obsahu (webový redakčný systém). V praxi sa môžme stretnúť aj s ďalšími pomenovaniami, ako napríklad publikačný systém alebo systém na správu obsahu.

Na výstavbu redakčných systémov sa používajú prevažne technológie bežiace na strane serveru, ako napríklad PHP, Coldfusion, Ruby on Rails, Java, ASP.NET a pod. V mnohých prípadoch sa na uloženie dát využívajú databázy, no zvyknú sa ukládať aj priamo na súborový systém vo forme súborov.

Existuje veľké množstvo redakčných systémov, jednak komerčných ako aj otvorených. Len na [26] môžme nájsť okolo 2500 otvorených systémov. Tieto systémy tvoria akési jadro, ktoré zabezpečuje hlavne správu obsahu a oprávnení vykonávať špecifické akcie konkrétnymi užívateľmi. Systémy sa potom pomocou modulov rozšírujú o funkcionalitu potrebnú pre konkrétny projekt.
Pokiaľ by sme kategorizovali systémy na správu obsahu do troch skupín, na malé, stredné a veľké, môžme tak utvoriť približnú charakteristiku, ktorá popisuje tieto systémy v danej kategorii.

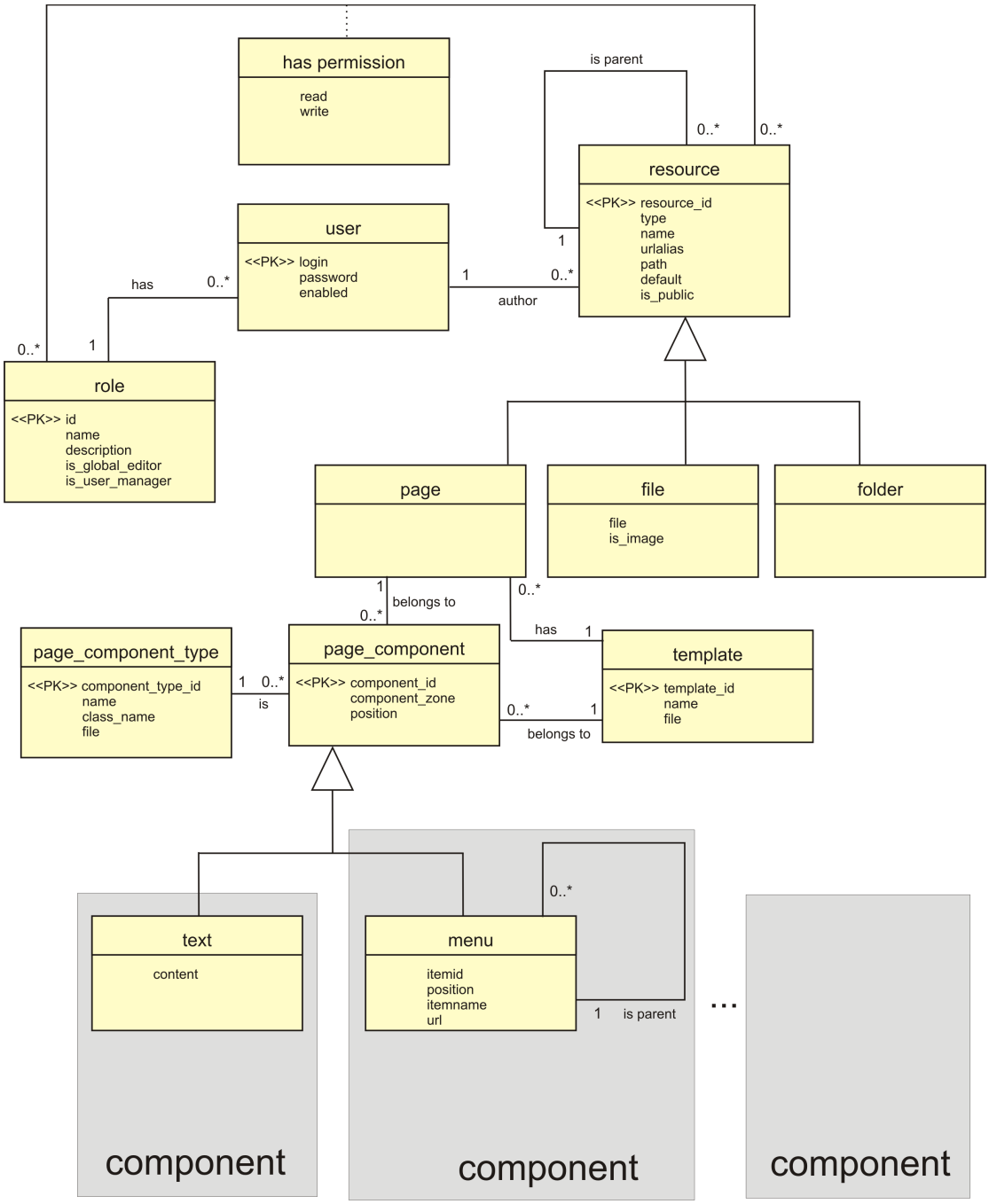
1. Malé systémy sú určené na správu jednoduchých webových stránok, kde počet spravovaných dokumentov nedosahuje vysokých čísel. Väčšinou sú určené len pre jedného užívateľa a neumožňujú vytváranie užívateľských účtov.
2. Pri stredných redakčných systémoch sa funkcionalita rozširuje tak, aby umožnila efektívnu prácu s väčším množstvom spravovaných dokumentov viacerým užívateľom súčasne.
3. Veľké redakčné systémy sú určené sa správu obrovského množstva dokumentov a ďalších typov dát. Jedná sa o robustné systémy, ktoré majú často vytvorený a integrovaný svoj vlastný skriptovací jazyk aby tak umožnili integrovať akékoľvek požiadavky užívateľa. Majú vybudovanú podporu pre archiváciu, automatické plánovanie úloh, a pod. Bežnou súčasťou je aj podpora workflow, čo je systém na vytváranie dokumentov spoluprácou viacerých užívateľov. Vytváraný dokument sa môže postupne presúvať napríklad od autora článku, cez editora až po korektora, ktorý môže mať zároveň aj oprávnenie daný dokument publikovať.

3.1 Požiadavky na systém

Podľa predchadzajúceho rozdelenia redakčných systémov by sme mohli vytváraný systém zaradiť medzi stredné redakčné systémy. A mal by teda umožňovať jednoduchú správu webového obsahu,

vytváranie užívateľov, rolí a oprávnení pre konkrétnu rolu. Na systém sú kladené bežné požiadavky na bezpečnosť, údržbu a rozšíriteľnosť.

3.2 Návrh redakčného systému



Obr.7: ER diagram

Obrázok 7 zobrazuje konceptuálny návrh redakčného systému vytvorený na základe požiadavkov kladených na vytváraný systém. Základ systému tvorí entita s názvom resource. Jedná sa o tzv. zdroj, ktorý zlúčuje spoločné vlastnosti všetkých dostupných typov dokumentov (zdrojov), ktoré vytváraný systém môže obsahovať. Ako výplýva z diagramu, exitujú tri typy zdrojov, stránka (page), súbor (file) a zložka (folder). Význam stránky a súboru je zrejmý podľa názvu, zložka slúži len kvoli možnosti hierarchického usporiadania predchadzajúcich dvoch typov zdrojov. V prípade rozšírenia systému by zdroje patriace do zložky mohli dediť určité vlastnosti priradené danej zložke, ako napríklad prístupové práva a podobne. Zložka, rovnako ako aj stránka a súbor, obsahuje atribút urlalias, teda existuje nejaký alias, ktorý označuje práve tento typ zdroja. Význam toho vysvetľuje následujúci príklad:

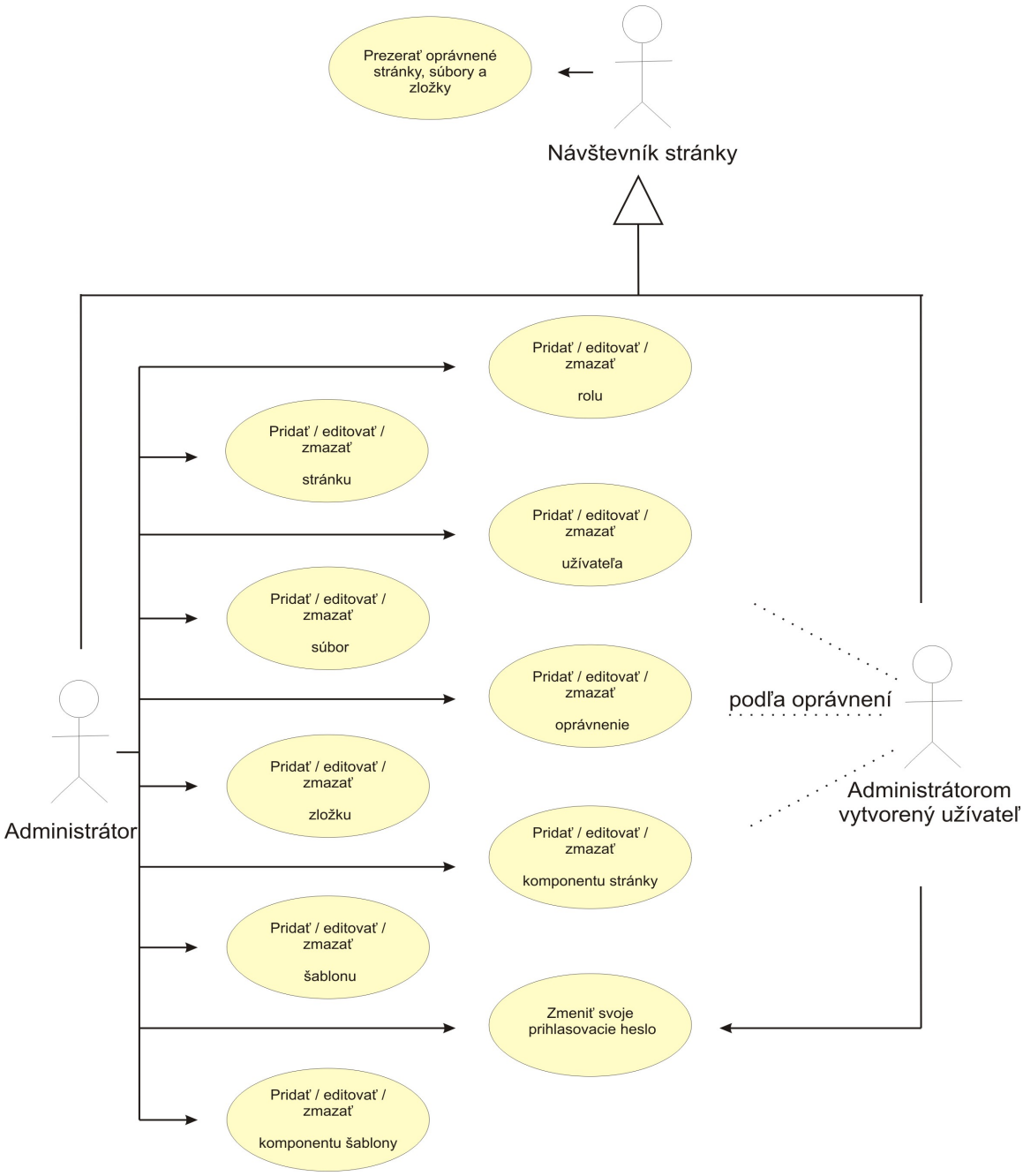
Nech znak '/' predstavuje koreň štruktúry zdrojov. Nech A je zložka, ktorej alias je 'a' a B je stránka s aliasom 'b'. V prípade, že stránka B je v hierarchickej štruktúre umiestňená priamo pod koreňom, cesta ku tejto stránke bude '/b'. Ak by sme ju umiestnili do zložky A, cesta k stránke by sa zmenila na '/a/b'.

Entita page_component predstavuje komponentu, ktorá môže byť súčasťou stránky alebo šablóny. Týchto komponent môžu byť rôzne typy a vďaka zabudovanej podpore je možné vytvárať a inštalovať nové komponenty bez nutnosti meniť zdrojové kódy jadra systému. Týmto je splnená požiadavka na jeho rozšíriteľnosť. Ako príklad týchto komponent môžeme uviesť napríklad textový obsah, navigáciu alebo anketu. Komponenty sa viažu buď na konkrétnu stránku alebo na šablónu stránky. Pokiaľ sa teda komponenta viaže na šablónu stránky, zobrazí sa na každej stránke, ktorá ma priradenú túto šablónu.

Ďalšiou entitou je user, ktorá predstavuje užívateľa systému. Implicitným užívateľom je administrátor, ktorý má právomoc vykonávať všetky akcie, ktoré tento systém umožňuje. Môže teda vytvárať aj ďalších užívateľov a priradzovať im určitú rolu v systéme.

Entita role predstavuje typ roly, ktorú možno priradiť užívateľovi. Systém umožní vytváranie nových rolí a priradzovanie oprávanení pre každú rolu. Oprávnenia by sme mohli rozdeliť do dvoch skupín, na oprávnenia systémové a oprávnenia zdrojové. Systémové oprávnenia určujú, či užívatelia priradení ku konkrétnej role môžu v systéme vytvárať systémové operácie, ako napríklad vytvorenie novej roly alebo užívateľa. Na druhej strane, zdrojové oprávnenia určujú , aké akcie môžu užívatelia danej roly vykonávať s konkrétnými zdrojmi. Napríklad či môžu vytvárať nové stránky, editovať alebo mazať už existujúce, a pod.

Táto správa užívateľov a oprávnení splňuje ďalšiu podmienku kladenú na vytváraný systém, a to na jeho bezpečnosť. S bezpečnosťou súvisí aj atribút is_public obsiahnutý v entite resource, ktorý určuje, či k danému zdroju môže pristupovať aj bežný návštevník stránky, a teda neregistrovaný užívateľ systému.



Obr.8: Diagram prípadov užitia

Obrázok 8 zobrazuje diagram prípadov užitia. Ako je možno vyčítať z tohto obrázka, užívateľ Administrátor ma oprávnenia vykonávať všetky dostupné akcie, ktoré tento systém umožňuje, vrátane vytvárania ďalších užívateľov. Vytvoreným užívateľom môže priradzovať oprávnenia, a takto definuje všetky akcie, ktoré môžu v tomto systéme vykonávať. Na tomto diagrame to znázorňuje užívateľ s názvom Administrátorom vytvorený užívateľ.

Ďalším typom užívateľa je Návštevník stránky, ktorý môže prezerať všetky zdroje označené ako verejné.

Takto navrhnutý systém je implementovaný pomocou technológií Java a ASP.NET. Popis implementácie rozoberá nasledujúca podkapitola.

3.3 Implementácia

Táto podkapitola popisuje implementáciu navrhnutého systému. Rozoberá jednak všeobecný princíp spoločný pre obe implementácie a taktiež porovnáva konkrétne riešenia v jednotlivých implementáciach.

Prvým článkom implementácie je konfiguračný súbor, ktorý definuje konštanty ako napríklad prístupové údaje k databáze, koreňový adresár webovej aplkácie, cestu k verejným súborom, šablonám stránok a umiestnenie súborov nahrávaných cez formulár na server. V Jave sa na tieto, ale aj mnohé dalšie účely používa súbor web.xml, pre ASP.NET je to web.config.

Ďalším dôležitým bodom implementácie je odchytávanie všetkých prichádzajúcich požiadavkov na server a ich následné presmerovanie do tzv. filtra na ich ďalšie spracovanie. V tomto filtry sa zisťuje či sa požadovaná adresa odkazuje na adresár obsahujúci verejné súbory. Ak áno, prepošle požiadavok na daný zdroj a nechá webový server aby sa postaral o jeho odoslanie ku klientovi. V opačnom prípade smeruje všetky požiadavky do jadra systému na ich vyhodnotenie a odoslanie adekvátnej odpovede žiadajúcemu klientovi.

Technológia Java umožňuje mapovanie URL prichádzajúcich požiadavkov na konkrétny zdroj v konfiguračnom súbore web.xml. Takže použitím vzoru <url-pattern>/*</url-pattern> sú presmerované úplne všetky požiadavky na UrlFilter (UrlFilter.java), čo je vlastne trieda implementujúca rozhranie Filter. Technológia Java umožňuje vytváranie reťaze filtrov, ktoré môžu upravovať požiadavok a vykonávať rôzne ďalšie akcie ešte pred spracovaním požiadavku adresovaným zdrojom. Tieto filtre je potrebné zaregistrovať v súbore web.xml a to v požadovanom poradí.

V .NETe sa tento problém vyriešil zaregistrovaním HTTP modulu CustomFilter (CustomFilter.cs) v konfiguračnom súbore web.config. V ňom, ako reakcia na udalosť context_BeginRequest je riešené spomínané odchytávanie a presmerovanie prichádzajúcich požiadavkov.

Základným kameňom systému je trieda (v Jave MainServlet.java, v .NETe MainUrlHandler.cs), ktorá vyhodnocuje prichádzajúce požiadavky na zdroje a odosiela klientovi adekvátnu odpoveď. Každý zdroj je identifikovaný cestou, ktorá musí byť unikátna pre každý jeden

zdroj. Táto cesta sa vytvára zložením aliasu konkrétneho zdroja, všetkých jeho rodičovských zdrojov a koreňa štruktúry.

Príklad: Nech A a B sú zdroje, nech zdroj A má alias 'a' a je hierarchický umiestnený pod koreňom štruktúry označeným ako '/', nech zdroj B má alias 'b' a je hierarchický umiestnený pod zdrojom A (a teda zdroj A je rodičom zdroja B). Potom cesta ku zdroju A je '/a' a cesta ku zdroju B je '/a/b'.

Aby sme takto dokázali rozlíšiť jednotlivé zdroje, musí byť cesta pre každý zdroj unikátna. A teda ak by sme mali zdroje A, B a C, kde A je rodičom B a C, potom musí platiť, že alias zdroja B je rôzny od aliasu zdroja C.

Jadro systému teda podľa požadovanej cesty dokáže v databáze vyhľadat daný zdroj, určiť o aký typ zdroja sa jedná a odoslať klientovi odpoveď. Ak sa jedná o súbor, vyhľadá ho na súborovom systéme podľa jeho názvu a umiestnenia. Pri stránke je tento proces o niečo zložitejší.

Každá stránka má priradenú práve jednu šablónu. Táto šablóna určuje vzhľad stránky pomocou HTML a CSS. Môže však obsahovať aj špeciálne vytvorené značky s názvom componentZone alebo zóna komponent. Zóny komponent určujú umiestnenie v šablóne, na ktoré sa zobrazí obsah komponent. Zóna komponent musí mať unikátne ID vrámci celej šablóny. Ďalším povinným atribútom je zdroj komponent, ktorý určuje, či sa v danej zóne zobrazia komponenty priradené konkrétnej stránke alebo komponenty priradené tejto šablóne.

Komponenty sú objekty tvoriace obsahovú časť stránok. Ako príklad môžeme uviesť textový obsah s obrázkami, navigáciu, anketu, fotogalériu a pod. Komponenta sa viaže buď na konkrétnu stránku alebo na šablónu stránok. Taktiež je potrebné špecifikovať aj ID zóny komponent, v ktorej sa má zobraziť.

V Jave reprezentujú šabóny stránok stránky JSP. Môžu obsahovať ako statický obsah, tak aj kód vracajúci obsah dynamický. Java umožňuje aj definovanie vlastných značiek. V našom prípade sme definovali značku componentZone v súbore taglib.tld. Jedná sa o súbor so štruktúrou XML, kde vytváraným značkám definujeme okrem iného názvy ich atribútov a tiež triedu, ktorá sa volá pri interpretovaní tejto značky (ComponentZoneTag.java). Nami vytvorenú knižnicu značiek musíme taktiež zaregistrovať v konfiguračnom súbore web.xml.

Pri implementácii v prostredí .NET sa ako šablóny stránok používajú formuláre ASPX. Pri vytváraní novej značky je potrebné pridať záznam do súboru web.config, kde sa páruje vytváraný ovládaci prvok s nejakým menným priestorom a triedou. Táto trieda dedí z triedy WebControl, povinné atribúty vytváraného ovládacieho prvku sa definujú vytvorením verejných vlastností (public properties). Pred interpretovaním ovládacieho prvku sa volá metóda Init(), na vykreslenie obsahu metóda Render().

Šablóny majú presne definovanú štruktúru, teda pri vytvaraní nového súboru so šablónou je potrebné zahrnúť všetky súčasti, ktoré sa majú vykonať na strane serveru. Šablóna teda bude obsahovať kód HTML a CSS doplnený o výkonný kód, popr. špeciálne vytvorený tag componentZone.

Prihlasovanie užívateľov zabezpečuje v Jave LoginServlet.java, v .NETe Login.cs. Overuje existenciu užívateľa v databáze a zhodu zadaného hesla. Na udržiavanie kontextu sa využíva sedenie (v Jave HttpSession, v .NETe HttpSessionState), ktoré je v obidvoch prípadoch uložené na serveri. Využíva tiež aj cookies na strane klienta, ale len na uloženie unikátneho identifikátora sedenia.

Po prihlásení sa užívateľovi zobrazí zdroj označený ako implicitný. Je však doplnený o administrátorské menu a ďalšie ovládacie prvky. Užívateľ tak môže v systéme vykonávať akcie podľa svojich oprávnení.

Implementácia ostatných (nekľúčových) častí systému bola v princípe rovnaká pre obe platformy, pretože vychádzala z rovnakého objektového návrhu systému. Pre podrobnejšie pochopenie princípu fungovania vytvoreného systému odporúčam prečítať komentáre zdrojového kódu, popr. programovú dokumentáciu.

4 Zhrnutie

Na základe skúseností získaných pri realizácií tejto práce, jednak z jej teoretickej časti ako aj praktickej by som použité technológie charakterizoval následovne:

- .NET
  - Visual Studio je mocný nástroj, ktorý značne urýchľuje vývoj aplikácií pre platformu .NET.
  - Aj keď návrh .NET frameworku v sebe zahŕňa nezávislosť na cieľovej platforme, prenesenie vytvoreného projektu na iný operačný systém nie je v súčasnosti až také jednoduché. Síce existuje projekt s názovom Mono [27], ktorý poskytuje potrebný software na spúštanie .NET aplikácií pod operačnými systémami Linux, Solaris, Mac OS X, Windows a Unix, v dobe písania tejto práce bola však podporovanou len verzia .NET Frameworku niekde medzi 1.1 a 2.0. Čo je oproti najnovšej verzii 3.5 od firmy Microsoft značný sklz.
  - Výhodou je nezávislosť na programovacom jazyku.
  - Pre vývoj redakčného systému ponúkol dostatočné prostriedky na realizáciu.

- Java
  - Existuje veľké množstvo vývojových prostredí, ako napríklad Eclipse, NetBeans a pod., čo dává vyvojárovi možnosť vybrať si vhodné riešenie.
  - Pre vývoj redakčného systému ponúkol dostatočné prostriedky na realizáciu.

Na základe skúseností pri realizácii tohto projektu môžme vyvodiť nasledujúci záver :

1. Obe technológie sú si v princípe podobné, a preto prepodkladám, že technológia .NET vznikla do istej miery ako reakcia na technológiu Java.
2. Obe technológie umožnili implementovať všetky požiadavky na vytvaraný systém. Princíp implementácie bol vo väčšine prípadov podobný, výnimočne sa použili špecifické rysy konkrétnej technológie, popísané v kapitole o implementácii.

Je teda nemožné jednoznačne určiť, ktorá technológia je lepšia. Kedže sa jedná o vzájomných konkurentov na trhu, každá významná novinka sa časom premietne aj v konkurenčnej technológií.

5 Záver

Táto práca porovnala technológie Java (JSP/Servlet) a .NET (ASP.NET a C#) pri implementácii prototypu webového redakčného systému.

Výsledný systém v sebe zahŕňa:

- správu obsahu (stránok a súborov)
- prepracovaný systém šablónovania a tvorby obsahu stránok
- správu užívateľov, rolí a oprávnení
- WYSIWYG editor (FCK Editor) na formátovanie textových komponent stránky
- užívateľsky priateľské URL
- možnosť rozšírenia pomocou komponént stránky
- a pod.

Redakčný systém by sa v budúcnosti mohol rozšíriť o podporu archivácie vytváraných dokumentov, zamedzenie prístupu ostatným užívateľom k práve editovanému zdroju, popr. o podporu WebDAV. Ďalšími súčasťami by mohli byť napríklad plánovač, ktorý by umožnil spušťanie úloh v zadanom čase, a taktiež podpora workflow.

Ako komponenty stránok by sa v praxi často používali ankety, diskusné fóra, štrukturované dáta s podporou RSS, a mnohé ďalšie.

Cieľom tejto práce bolo vytvorenie prototypu redakčného systému so základnými požiadavkami na funkcionalitu za pomoci technológií Java a .NET, popísať jeho implementáciu a zhrnúť získané poznatky.

Pri vytváraní systému podobného rozsahu ako má tento redakčný systém teda neexistuje jednoznačná odpoveď, ktorú platformu by bolo výhodnejšie použiť. Voľbu technológie by mali ovplyvniť požiadavky na vytvaraný systém, ako napríklad náklady na nasadenie systému, dostupnosť a cena kvalifikovaných odborníkov a pod.

Literatúra

[1] Hruška, T. *Internetové aplikace (WAP) I. část Internet a WWW (Studijní opora)*. Brno, 2007.

[2] W3C. *About the World Wide Web [online]*. Posledná modifikácia: 24. januára 2001. [cit. 2008-05-06]. URL: <http://www.w3.org/WWW/>.

[3] W3C. *What is Hypertext [online]*. [cit. 2008-05-06]. *URL:* <http://www.w3.org/WhatIs.html>.

[4] W3C. *HTTP - Hypertext Transfer Protocol [online]*. Posledná modifikácia: 27. februára 2008. [cit. 2008-05-06]. *URL:* <http://www.w3.org/Protocols/>.

[5] W3C. *CGI : Common Gateway Interface [online]*. Posledná modifikácia: 13. októbra 1999. [cit. 2008-05-06]. *URL:* <http://www.w3.org/CGI/>.

[6] Komunita Wikipedie. *Robert McCool [online]*. Posledná modifikácia: 6. februára 2008. [cit. 2008-05-10]. URL: <http://en.wikipedia.org/wiki/Rob_McCool>.

[7] Hruška, T., Burget, R. *Internetové aplikace (WAP) II. část SGML, HTML, CSS, DOM (Studijní opora)*. Brno, 2007.

[8] Komunita Wikipedie. *HyperText Markup Language [online]*. Posledná modifikácia: 3. mája 2008. [cit. 2008-05-07]. URL: <http://cs.wikipedia.org/wiki/HyperText_Markup_Language>.

[9] Komunita Wikipedie. *Cascading Style Sheets [online]*. Posledná modifikácia: 30. apríla 2008. [cit. 2008-05-07]. URL: <http://cs.wikipedia.org/wiki/Cascading_Style_Sheets>.

[10] Hruška, T. *Internetové aplikace (WAP) VI. Programování klienta (JavaScript) (Studijní opora)*. Brno, 2007.

[11] Komunita Wikipedie. *JavaScript [online]*. Posledná modifikácia: 30. apríla 2008. [cit. 2008-05-07]. URL: <http://cs.wikipedia.org/wiki/Javascript>.

[12] Herout, P. *Učebnice jazyka Java*. České Budejovice, 2001.

[13] Komunita Wikipedie. *Java [online]*. Posledná modifikácia: 15. apríla 2008. [cit. 2008-05-08]. URL: <http://cs.wikipedia.org/wiki/Java>.

[14] Kurniawan, B. *Java for the Web with Servlets, JSP, and EJB- A Developer's Guide to J2EE Solutions*. Indianapolis, 2002.

[15] Hall, M. *Java servlety a stránky JSP*. Praha, 2001.

[16] Sun Microsystems. *What Is a Custom Tag? [online]*. [cit. 2008-05-10]. URL: <http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/JSPTags2.html#67760>.

[17] Komunita Wikipedie. *JavaBean [online]*. Posledná modifikácia: 6. mája 2008. [cit. 2008-05-10]. URL: <http://en.wikipedia.org/wiki/JavaBeans>.

[18] Sun Microsystems. *The Essentials of Filters [online]*. [cit. 2008-05-10].

URL: <http://java.sun.com/products/servlet/Filters.html>.

[19] Komunita Wikipedie. *.NET Framework [online]*. Posledná modifikácia: 10. mája 2008. [cit. 2008-05-09]. URL: <http://en.wikipedia.org/wiki/.NET_Framework>.

[20] Komunita Wikipedie. *ASP.NET [online]*. Posledná modifikácia: 9. mája 2008. [cit. 2008-05-08]. URL: <http://en.wikipedia.org/wiki/ASP.NET>.

[21] Andrew Duthie, G. *Microsoft ASP.NET krok za krokem.* Brno, 2003.

[22] MacDonald, M., Szpuszta, M. *ASP.NET 2.0 a C#. Tvorba dynamických stránok profesionálne*. Brno, 2006.

[23] Microsoft. *ASP.NET Application Life Cycle Overview for IIS 5.0 and 6.0 [online]*. [cit. 2008-05-10]. URL: <http://msdn.microsoft.com/en-us/library/ms178473.aspx>.

[24] Microsoft. Microsoft. *ASP.NET Page Life Cycle Overview [online]*. [cit. 2008-05-10]. URL: <http://msdn.microsoft.com/en-us/library/ms178472.aspx>.

[25] W3C. A*SP.NET - Server Controls [online]*. [cit. 2008-05-06]. *URL:* <http://www.w3schools.com/ASPNET/aspnet_controls.asp>.

[26] SourceForge.net. [cit. 2008-05-06]. *URL:* <http://sourceforge.net/>.

[27] Mono. [cit. 2008-05-06]. *URL:* <http://www.mono-project.com/>.

Zoznam príloh

Príloha 1. Obsah priloženého CD

Priložené CD obsahuje:

- Zdrojový súbor textovej časti tejto bakalárskej práce.
- Redakčný systém so zdrojovými kódmi v Jave.
- Redakčný systém so zdrojovými kódmi v ASP.NET / C#.
- Programové dokumentácie.
- Inštalačný a užívateľský manuál.
- Textový súbor readme.txt, ktorý bližšie popisuje štruktúru priloženého CD.

Príloha 2. Inštalačný manuál (Java)

Systémové požiadavky :

- databáza PostgreSql (v.8.0)
- webový server Tomcat (v.6.x)
- webový prehliadač so zapnutou podporou cookies a JavaScript (doporučeným prehliadačom je Firefox, na ostatných nebola aplikácia testovaná).

Postup inštalácie:

1. Nainštalujte webový server Tomcat.
2. Nainštalujte databázový server PostgreSql.
3. Vytvorte novú databázu.
4. Vykonajte na nej všetky dotazy obsiahnuté v súbore cms.sql (nachádza sa na priloženom CD).
5. Nakopírujte súbor cms.war do adresára webapps webového servera Tomcat.
6. Reštartujte Tomcat.
7. Po zavedení systému upravte v konfiguračnom súbore web.xml prístupové údaje k vytvorenej databáze.
8. Reštartujte Tomcat.
9. Vo webovom prehliadači zadajte ardesu smerujúcu na koreňový adresár nainštalovaného systému.
10. Pre prihlásenie do systému použite heslo uvedené v užívateľskom manuále.

Príloha 3. Inštalačný manuál (ASP.NET)

Systémové požiadavky :

- databáza MSSQL 2005
- webový server IIS (v.6.x)
- webový prehliadač so zapnutou podporou cookies a JavaScript (doporučeným prehliadačom je Firefox, na ostatných nebola aplikácia testovaná).

Postup inštalácie:

1. Nainštalujte webový server IIS.
2. Nainštalujte databázový server MSSQL.
3. Vytvorte novú databázu.
4. Vykonajte na nej všetky dotazy obsiahnuté v súbore cms.sql (nachádza sa na priloženom CD).
5. Nakopírujte obsah archívu cms.zip do koreňového adresára.
6. Nastavte práva pre zápis procesu ASP.NET potrebné pre zápis uploadovaných súborov do adresára webovej aplikácie.
7. Po zavedení systému upravte v konfiguračnom súbore web.config prístupové údaje k vytvorenej databáze.
8. Vo webovom prehliadači zadajte ardesu smerujúcu na koreňový adresár nainštalovaného systému.
11. Pre prihlásenie do systému použite heslo uvedené v užívateľskom manuále.

Príloha 4. Užívateľský manuál

Tento manuál popisuje možnosti vytvoreného systému na konkrétnych príkladoch. Predpokladá, že koreňovým adresárom nainštalovaného systému je *http://localhost:8080/CMS/*.

- **Konfigurácia systému**. Systém sa môže konfigurovať pomocou súborov web.xml popr. web.config. Medzi kľúčové konštanty týchto súborov patria prístupové údaje k databáze a konštanta ROOT-PATH, ktorá predstavuje relatívnu cestu vrámci webového servera ku nainštalovanej aplikácii. Ak by teda adresa ku koreňu aplikácie bola napríklad *http://localhost:8080/moje/cms/*, je potrebné túto konštantu nastaviť presne na *moje/cms* (nie /moje/cms, ani /moje/cms/ a pod...). V našom prípade by sme ju nastavili na *CMS*. Význam ostatných konštánt je zrejmý podľa ich názvu.
- **Prihlásenie do systému.** Na prihlásenie je potrebné pridať relatívnu cestu cms/login ku adrese koreňa. V našom prípade teda *http://localhost:8080/CMS/cms/login*. Ako prihlásovacie meno a heslo použijeme *admin* / *admin*.
- **Správa zdrojov**. Medzi zdroje v tomto systéme patria stránky, súbory a adresáre. Spravovať ich môžme cez "*Obsah - Zobraziť obsah"* v hlavnom menu redakčného systému.
  - *Príklad : Vytvorenie novej stránky*. Novú stránku vytvoríme cez "*Obsah - Pridať - Novú stránku"*. Zadáme jej názov a alias, v prípade ak by sme chceli, aby táto stránka bola hlavnou (default) stránkou, zaškrtneme voľbu *"Východzia stránka"*. Stránku môžme umiestniť pod iný zdroj, čo ovplivní cestu pri prístupe k nej. Stránke potom priradíme šablónu *Základná šablóna* a nastavíme jej prístupnosť. Takto vyplnený formulár odošleme a uzavrieme okno s nastaveniami. Obnovíme obsah pôvodnej stránky aby sa v nej prejavili zmeny. Cez položku menu *"Prejsť na"* sa presunieme na našu stránku. Pomocou tlačítka "*Pridať"* umiestneného na každej hlavičke označujúcej zónu komponent môžme stránku naplniť požadovaným obsahom. Ako si však môžme všímnúť, naša stránka už obsahuje hlavné menu. Toto menu je tiež iba obsahovou komponentou, je však priradené ku šablóne *Základná šablóna,* a preto sa zobrarí na každej stránke s touto šablónou. Ak chceme spravovať komponenty priradené tejto šablóne, presunieme sa cez hlavné menu redakčného systému na "*Aktuálna stránka- Zobraziť šablónu"*. Späť sa vrátime cez "*Aktuálna stránka- Zobraziť stránku"*. Náhľad takto vytvorenej stránky zobrazíme pomocou "*Aktuálna stránka- Zobraziť náhľad"*.

Podobným spôsobom vytvárame aj ďalšie zdroje, konkrétne adresáre a súbory.

- **Vytváranie šablón.** Novú šablónu môžme najjednoduchšie vytvoriť duplikovaním vzorovej šablóny a nahradením jej HTML a CSS kódu vlastným. Túto šablonu nakoniec musíme zaregistrovať do systému a to cez "*Obsah - Pridať - Novú šablónu"*.
- **Správa rolí, oprávnení a užívateľov.** Spravovať užívateľov, role a ich oprávnenia môžme pomocou "*Správa rolí a užívateľov - Editovať"*. Intuitívne takto môžme vytvárať nových užívateľov a ich role. Za zmienku stoja atribúty roly, konkrétne *Je globálnym editorom* a *Je správcom užívateľov*. Prvý z nich určuje, že daná rola ma prístup ku všetkým vytvoreným zdrojom, aj v prípade, že neexistujú oprávnenia, ktoré by jej tento prístup oficiálne umožnili. Druhý atribút určuje, či daná rola môže spravovať účty ostatných užívateľov.
    - Príklad: Vytvorenie užívateľa.. Nového užívateľa pridáme cez "*Správa rolí a užívateľov - Pridať - Nového užívateľa"*. Zadáme jeho prihlasovacie meno, zo zoznamu mu priradíme rolu a označíme, či je povolený. Odoslaním formulára sa v databáze vytvorí nový záznam o užívateľovi, kde ako heslo je implicitne nastavený prázdny reťazec. Zmenu svojho hesla vykoná užívateľ po prihlásení do systému pomocou "*Môj profil - Zmeniť heslo"*.

- **Tvorba nových komponent stránky.** Vytvorenie novej komponenty požaduje jednak znalosť danej platformy, ako aj pochopenie princípu fungovania redakčného systému. Štúdium zdrojových kódov je teda jednou z prerekvizít. Všetky komponenty musia dediť z triedy PageComponent a vzniknutý typ komponenty je potrebné zaregistrovať v databáze ku ostatným.