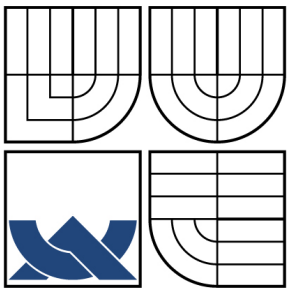
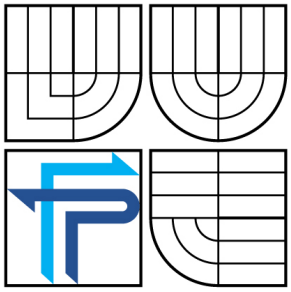


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA PODNIKATELSKÁ

ÚSTAV INFORMATIKY

FACULTY OF BUSINESS AND MANAGEMENT

INSTITUTE OF INFORMATICS

VYTVOŘENÍ SYSTÉMU PRO E-SHOP

CREATION OF E-SHOP SYSTEM

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE — JAKUB ZAPLETAL

AUTHOR

VEDOUCÍ PRÁCE — Ing. JIŘÍ KŘÍŽ, Ph.D.

SUPERVISOR

BRNO 2009

Anotace

Tato práce se věnuje vytvoření aplikace e-shopu pro účely komerčního šíření. Je zde probrán průzkum prostředí a technologiemi, kterými se může daná aplikace vyvíjet. Nakonec je popsána vlastní práce a spokojenost s volbou daného řešení.

Klíčová slova: ASP, ASP.NET, PHP, MS SQL, Oracle, DB2, Postre SQL, MySQL, ZenCart, OpenCart, OsCommerce, VirtueMart, Joomla, Magento, Zend Framework, Prado, Symfony, CakePHP, Nette, MVC, e-shop, internetový obchod.

Annotation

This thesis is dedicated to the creation of e-shop for the purposes of commercial distribution. It is discussed exploration of platforms and technologies, which application can be developed. Finally, it is described my own labor and satisfaction with the choice of the solution.

Keywords: ASP, ASP.NET, PHP, MS SQL, Oracle, DB2, Postre SQL, MySQL, ZenCart, OpenCart, OsCommerce, VirtueMart, Joomla, Magento, Zend Framework, Prado, Symfony, CakePHP, Nette, MVC, e-shop.

OBSAH

Úvod

Tato práce je zaměřena na tvorbu aplikace pro tvorbu internetové obchodu, a to ne pro individuální obchod, ale aplikaci, kterou by šlo komerčně nabízet zájemcům o internetové obchody. Jednotlivé kapitoly jsou postupně zaměřeny na výběr programovacího jazyka, databázového systému, zjištění aktuální stavu na poli open-source řešení, frameworku programovacího jazyka a následně vývoj aplikace. Při práci na aplikaci pro individuální internetový obchod by nebylo tolik nutné se zaměřovat na volbu, která je v dnešní době k dispozici, ale u využití pro komerční účely nabízet tento produkt, je nutné důkladně prozkoumat prostředí a technologie, v kterých se má vyvíjet. Často je to boj mezi dvěma platformami, a to Linux a Windows. Platforma Windows často nemá dobré mínění, to je důsledkem nestability operačních systémů pro stolní počítače, které byly v posledních patnácti letech k dispozici a právě naopak dobré mínění si získává operační systém Linux. Samozřejmě to není dáno pouze stabilitou, ale také cenou samotných systémů a softwaru s daným systémem spojeným. Dalším velkým rozhodovacím kritériem jsou finanční náklady. Objevují se zde prostředí, které jsou zcela zdarma, ale také prostředí, které si žádají desítky, či stovky tisíc. Je tedy nutné si uvědomit, jaký má daný projekt potenciál a zda je finančně přijatelné se dát danou cestou.

1 Vymezení problémů a cíle práce

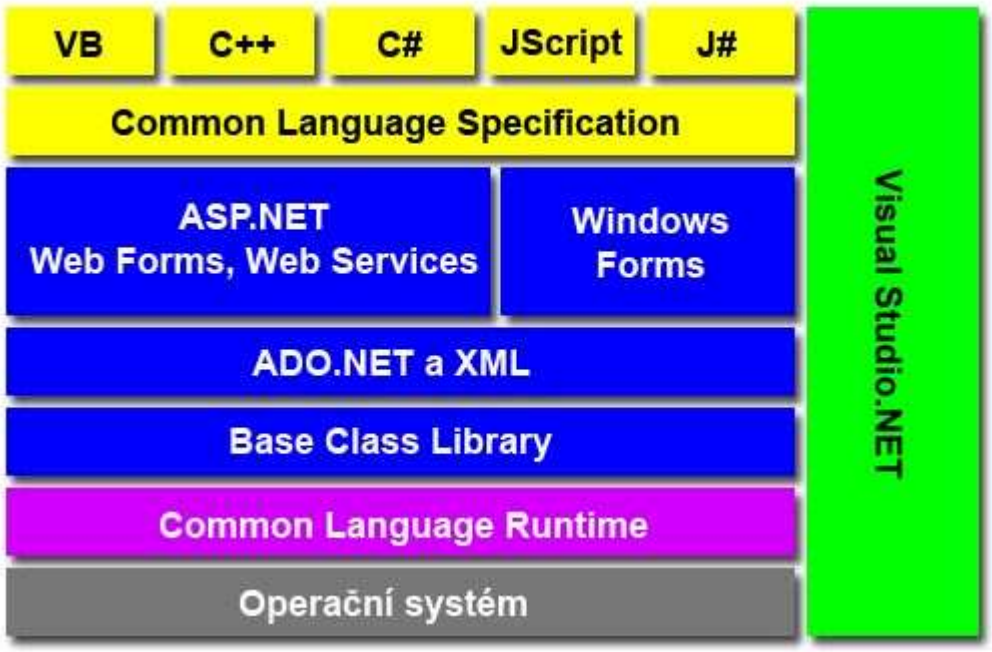
Chystáme se vytvořit aplikaci, která budou sloužit jako kompletní systém pro provoz e-shopu, a to jak back-endové části, tak front-endová části. Na začátku je třeba si stanovit, v jakém programovacím jazyku budeme tuto aplikace vytvářet, a dále je zapotřebí si uvědomit, že veškerá data obsažená v dané aplikaci musejí být někde uložena. Ukládání do textových souborů není určitě tou správnou cestou, je tedy i nutné určit, jakého databázového systému využijeme. Právě těmito dvěma problémy se budu dále zabývat a stanovím nejvhodnější řešení.

1.1 Programovací jazyky

Volba vhodného programovacího jazyka je základním kamenem, od kterého se bude odvíjet další rozvoj vytvoření systému. Mezi nejpoužívanější technologie v tomto odvětví se dají zařadit ASP a PHP. Samozřejmě jsou zde i další světově známé, mezi které se řadí Perl, Python, nebo Ruby. Já jsem se zaměřil pouze na ASP a PHP, jelikož mají na celém světě největší komunity příznivců a zařadil bych je i v Česku jako nejoblíbenější technologie, kde převažuje komunita kolem PHP.

1.1.1 ASP

ASP (Active Server Pages) je skriptovací platforma společnosti Microsoft, primárně určená pro dynamické zpracování webových stránek na straně serveru. Její první vydání bylo na konci roku 1996. Její nástupce ASP.NET, lze chápat jako širší a komplexnější technologii, která se od ASP v mnoha ohledech fundamentálně liší. ASP.NET je součástí .NET Frameworku pro tvorbu webových aplikací a služeb od začátku roku 2002. Struktura .NET Frameworku je znázorněn na Obrázku 1.



Obrázek 1 - Architektura .NET Frameworku

ASP.NET není novou verzí ASP, ale je to zcela nová technologie, která je integrální součástí platformy .NET Framework. Koncept ASP.NET WebForms ulehčuje programátorům přechod od programování klasických aplikací pro Windows do prostředí webu: stránky jsou poskládány z objektů, ovládacích prvků (*Controls*), které jsou protějškem ovládacích prvků ve Windows. Při tvorbě webových stránek je tedy možné používat ovládací prvky jako tlačítko (*Button*), nápis (*Label*) a další. Těmto prvkům lze přiřazovat určité vlastnosti, zachytávat na nich události, atd. Tak, jako se ovládací prvky pro Windows samy kreslí do formulářů na obrazovku, webové ovládací prvky produkují HTML kód, který tvoří část výsledné stránky poslané do klientova prohlížeče.

Klady

- Programátoři mohou vytvářet aplikace v jakémkoliv programovacím jazyce podporujícím CLR, např. Visual Basic.NET, JScript.NET, C#, Managed C++, ale i mutace Perlu nebo Pythonu a další.
- Firma, která se specializuje jak na software, tak na webové aplikace, nemusí mít více týmů programátorů. Stejným nástrojem s velmi podobnými komponentami lze napsat i konzolovou aplikaci nad .NET Frameworkem.

- Objektový design od začátku.
- Výborný vývojový nástroj VisualStudio.
- Generované HTML nebo XHTML je zcela validní za předpokladu, že programátor osobně něco nenaruší.
- Možnost přímo provázat s SharePoint Services.
- Dobrá podpora cachování. Díky tomu se dají velmi efektivně zrychlit některé části aplikace, předchází se tím neustálým zpracováním částí, které se pravidelně opakují a jsou náročné na výpočet.
- Celý kód je kompilován. Tím se předchází chybám, které vzniknou v průběhu programování. Také to umožňuje rychlejší zpracování, protože převod na programové instrukce proběhne v průběhu kompilace.

Zápory

- Vázáno na platformu Win/IIS. S tím také souvisí velké náklady při pořizovaní vlastního serveru. Na tuto platformu je nejlepší využívat software přímo od výrobce Microsoft.
- Drahá školení pohybující se řádově v desítkách tisíc.
- Náročnější na výkon hardwaru oproti Unix platformě.
- Je nutné platit vyšší částky za kvalitní nástroje na tvorbu aplikací.

1.1.2 PHP

PHP je rekurzivní zkratkou pro PHP: Hypertext Preprocessor (kde původní význam zkratky PHP je Per-sonal Home Page). Zakladatelem PHP je Rasmus Lerdofr, který v roce 1995 vytvořil jednoduchou sadu skriptů jazyka Perl pro vlastní účely. Když byla potřeba větší funkčnost, napsal rozsáhlejší implementaci jazyka C, která byla schopna komunikovat s databází, a umožnila uživatelům vytvářet jednoduché dynamické webové aplikace. Zpočátku byl tento jazyk znám jako PHP/FI.

V roce 1997, kdy bylo PHP/FI 2.0, měla komunita několik tisíc příznivců na celém světě, kteří tento jazyk aplikovali na 50 000 domén, to v té době znamenalo 1% z celkového počtu domén. Krátce poté vznikla první alfa verze PHP 3.0.

PHP 3.0 byla první verze, která velmi připomínala dnešní podobu jazyka, jak ji nyní známe. Vytvořili ho Andi Gutmans a Zeev Suraski jako kompletní předělání jádra PHP/FI 2.0, jelikož jim připadlo málo schopné pro vytváření komerčních projektů, na kterých spolu pracovali. Úsilně spolupracovali a začali pracovat pod již vytvořeným uživatelským základem PHP/FI, kdy se Andi, Zeev a Rasmus rozhodli spolupracovat a vzniklo PHP 3.0 jako nástupce PHP/FI 2.0, jehož vývoj byl zastaven.

Jedna z největších výhod PHP 3.0 byla jeho schopnost rozšířitelnosti. Přínosem byla dobrá infrastruktura pro hodně databází, protokolů a různých API. Jeho rozšířitelnost přitáhla spousty vývojářů a vznikaly nové rozšiřující moduly. Další klíčová vylepšení byla objektově orientovaná syntaxe, výkon a lepší konzistence syntaxe. Koncem roku 1998 využívalo PHP desítky tisíc uživatelů a na stovkách tisíc webových stránek bylo použito, což byl celkový nárůst na 10% z celého internetu.

Krátce poté, co oficiálně vyšlo PHP 3.0, Andi Gutmans a Zeev Suraski začali pracovat na předělání jádra. Hlavními výhodami bylo zlepšit výkon komplexních aplikací a zlepšit modularitu základního kódu. Nové jádro, nazvané Zend Engine (vzniklo spojením slov Zeev a Andi) bylo prvně představeno v roce 1999. PHP 4.0 bylo založeno na tomto jádře a spolu s nepřeberným množství nových vlastností bylo vydáno v roce 2000. Kromě zvýšené výkonnosti přineslo PHP 4.0 několik dalších podstatných změn, jako např. HTTP sessions, podporu pro mnoho web serverů, lepší ochrana vstupních dat a několik nových jazykových konstruktorů. Tato verze se postupně rozšířila až na 20% internetu, což skýtalo několik miliónů webových stránek.

Nejnovějším přírůstkem je PHP 5.0, které bylo vydáno 13. července 2004. Opět přineslo několik vylepšení, ale jedno je mimořádného významu: PHP 5 výrazně zlepšuje objektově orientovanou syntaxi. Přibyly výjimky, konstruktory, destruktory a mnoho dalších jazykových rozšíření. Mezi další podstatná rozšíření patří např. přepracovaná podpora XML, podpora COMu a technologie .NET, lepší podpora MySQL a další. Nyní je PHP ve verzi 5.2.9 a pomalu se již blíží verze PHP 6.0.

PHP se stalo velmi oblíbeným především díky jednoduchosti použití a tomu, že kombinuje vlastnosti více programovacích jazyků a nechává tak vývojáři částečnou svobodu v syntaxi. Softwarovými programátory bývá často zatracováno kvůli jeho jednoduché základní syntaxi a dříve malé podpory objektově orientovaného programování. Přispívá k tomu také fakt, že základy PHP se naučí studenti již na střední škole a pyšní se svými programátorskými schopnostmi.

Klady

- Multiplatformní řešení. Můžeme jej provozovat jak Linuxu, tak na Windows. Pro Windows ale není příliš doporučován kvůli rychlosti, stabilitě a podpoře.
- Jednoduché nastavení, které se provádí v jednom souboru PHP.INI, který případně jen stačí přehrát jinam a nemusí se vše nastavovat znovu.
- Nízké nároky na výkon hardware. Pro základní testovací provoz stačí téměř jakýkoliv hardware.
- Mnoho open-source řešení přímo ke stažení z internetu.
- Velká zahraniční, tak i česká komunita.
- Od verze 5.0, kdy dostalo dobrou podporu objektově orientovaného programování, se přibližuje plnohodnotným programovacím jazykům. Od verze 6 se má podpora objektového programování ještě více zlepšit.
- Velké množství webhostingů nebo free webhostingů.

Zápory

- Není třeba alokovat/deklarovat proměnné. Sice to vypadá jako výhoda, ale u rozsáhlejších projektů je to nevýhoda. V případě překlepu v názvu proměnné vznikne nová proměnná a programátor nezjistí, že udělal chybu. Samozřejmě těmto chybám se můžeme několika způsoby vyhnout. Zaprvé můžeme nastavit, aby nám PHP vypisovalo veškeré chyby v kódu, kde je i zaznamenáno, že daná proměnná nebyla deklarována. Toto varování dostaneme ale pouze v případě, kdy chceme danou proměnnou někde použít, pokud bychom chtěli do dané špatné proměnné přiřadit nějakou hodnotu tak nám to žádnou chybu nezobrazí. Zadruhé je možnost využívat kvalitní vývojové prostředí určené pro PHP, které

při vytváření kódu a psaní proměnných nabízí již vytvořené proměnné a tak nám to urychlí práci a předejdeme překlepům.

- Implicitně není odděleno PHP od HTML kódu. Tam kde vyžadujeme jejich oddělení, musíme si sami určit programovou strukturu, jakou to budeme psát, například MVC. Zároveň to může sloužit jako výhoda, že si člověk může používat zápisy, jak sám uzná za vhodné. Problém může nastat v případě, kdy se předělává již existující kód anebo při spolupráci více programátorů.
- Není jednotné pravidlo pro názvy funkcí. Některé víceslovné funkce jsou odděleny podtržítky, někdy jsou spojené.
- Při změně verze mohou nastat problémy s některými funkcemi, které částečně změnily svoji funkci. Je proto někdy potřeba změnit části kódu v celém projektu. Pokud si tyto změny nezjistí programátor před použitím na nové verzi, mohou nemile nastat nečekané problémy a pak dlouhé dohledávání jejich příčiny.
- PHP není kompilováno, ale interpretováno. Jeho spuštění je tedy náročnější na rychlost. Pokud se někde v kódu vyskytuje nějaká chyba, tak se na ni přijde až v momentě, kdy se prochází tou konkrétní částí kódu. Při neúplném testování aplikace mohou později nastat problémy, o kterých se dříve nevědělo. Je zde tedy nutné provádět pečlivé testování, i když to by se mělo provádět v každém případě.

1.1.3 Shrnutí

I když srovnávat ASP.NET a PHP moc nejde, jelikož ASP.NET je platforma a PHP je skriptovací jazyk, je potřeba se rozhodnout , kterou cestou se dát.

ASP.NET nabízí velkou možnost vývoje aplikací, a to nejen webových aplikací, ale také softwaru. Tato možnost je ale úzce vázána na platformu Windows. Nechci se zde zabývat srovnáním kvality Linuxu nebo Windows, tím by se mohla zabývat kompletně samostatná práce. Bohužel pohodlí ve formě Windows je kompenzováno vyššími náklady. Nejenže by provoz serveru vyšel mnohem dráž, ale bylo by potřeba zakoupit kvalitní vývojové prostředí, například VisualStudio, a také se zúčastnit několika školeních. Celkové náklady by určitě přesáhly sto tisíc. Z důvodu neznalosti žádného programovacího jazyka, ve kterém bych mohl tyto aplikace vyvíjet, bych se musel od

začátku nějaký naučit a časově by to bylo náročnější, než by se funkční aplikace povedla vytvořit.

PHP je zato vázáno na platformu Linux, i když ne striktně, dá se použít i na platformě Windows. Nízké náklady související s tímto jazykem jsou mnohem přívětivější, převážně pokud jedinec/společnost nemá velké finanční možnosti. V PHP jsem začínal již v roce 2001 a tak vyvíjet v něm rozsáhlejší aplikaci je pro mě přívětivější. Při řešení problému se mohu obrátit na některou komunitu, diskusní fórum, nebo i některé hotové řešení.

V konečné fázi se tedy přikláním k řešení pomocí PHP. Očekávám, že to nebude krok špatným směrem a celá aplikace bude vyhovovat všem požadavkům.

1.2 Databázové systémy

Volba vhodného databázového systému je stejně důležitá, jako výběr vhodného programovacího prostředí. Databáze nám umožňuje, aby data nebyla izolována v samostatných soubor, ale zajišťuje nám organizování dat v komplexně centrálně zpracované struktuře. Centrální správa databáze, tzn. všechny implementační programy, jsou realizovány prostřednictvím speciálního programového vybavení, které se nazývá systém řízení báze dat SŘBD (database management system, DBMS). Ten spolu s databází tvoří databázový systém DBS (database system).

Komunikace mezi aplikací a databázovým systémem je zajišťována pomocí jazyka SQL. Historie jazyka SQL spadá do 70. a 80. let. První standard byl přijat v roce 1986, označován jako SQL86. Časem se však projevily některé nedostatky. Opravená verze je z roku 1992 a je označována jako SQL92. Ten je v oblasti relačních databází standardem dodnes. Zkratka SQL značí Structured Query Language. Jazyk v sobě zahrnuje nástroje pro tvorbu databází a dále nástroje na manipulaci s daty. SQL patří mezi tzv. deklarativní programovací jazyky, což v praxi znamená, že kód jazyka SQL nepíšeme v žádném samostatném programu, jako by tomu bylo např. u jazyka C nebo Pascal, ale vkládáme jej do jiného programovacího jazyka, který je již procedurální. Se samotným jazykem SQL můžeme pracovat pouze v případě, že se terminálem připojíme

na SQL server a na příkazový řádek bychom zadávali přímo příkazy jazyka SQL. Jak už jsem se zmínil, SQL se skládá z několika částí. Některé části jsou určeny pro administrátory a návrháře databázových systémů, jiné pak pro koncové uživatele a programátory. První částí jazyka SQL je jazyk DDL (data definition language). Jedná se o jazyk pro vytváření databázových schémat a katalogů. Způsob ukládání tabulek definuje jazyk SDL (storage definition language). Třetí částí pro návrháře a správce je jazyk VDL (view definition language), určující vytváření pohledů. Poslední částí je jazyk DML (data manipulation language), který obsahuje základní příkazy. S jazykem DML pracují nejvíce koncoví uživatelé a programátoři databázových aplikací.

Dále se pokusím vypsat nejpoužívanější databázové systémy, stručně je popsat a vybrat z nich ten nejvhodnější pro tento projekt.

1.2.1 MS SQL

V roce 1988 MS SQL začalo jako součást aplikace Sybase pro použití v OS/2. To byl počáteční vstup firmy Microsoft na databázový trh. První verze, která byla použita pro Windows, byla vydána v roce 1993 a byla to první verze SQL Serveru naprogramována pro NT. V roce 1994 následovalo rozdělení od Sybase a MS SQL si šlo vlastní cestou.

Databázová tabulka zná přímo typ XML, do kterého lze XML dokument uložit. Je možné kontrolovat validitu těchto XML dat a také je možné přímo XML indexovat. Díky tomu se lze nad XML daty dobře vyhledávat.
Umožňuje velmi dobrou integraci do operačního systému. Například ve sledování výkonu lze velmi pěkně sledovat nebo zobrazovat velké množství informací o provozu MS SQL. Je možné u MS SQL sledovat mnoho údajů o výkonu a efektivitě tohoto serveru a například správně nastavených indexů.
Nové možnosti nabízí typ sloupce uniqueidentifier. Jde o 16 bajtový GUID (globally unique identifier), který by měl být jedinečný v celé databázi a možná, že i dokonce na celém širokém světě. GUID je ve formátu 936DA01F-9ABD-4d9d-80C7-02AF85C822A8. Na GUID je příjemné to, že ve skutečnosti jde o číslo a ne o řetězec, jak vypadá na první pohled. Práce s čísly je u databází mnohem levnější než s řetězci. Navíc je zde možnost u takového typu sloupce nastavit automatické generování

hodnoty. To je jako ekvivalent timestamp, který MS SQL samozřejmě také umí. Je to velmi užitečné, pokud v databázi potřebujeme mít jedinečnou hodnotu přes více tabulek, nebo potřebujeme mít dokonce jedinečnou hodnotu přes více databází. Například při pobočkovém zpracování dat.

MS SQL nezná příkaz INSERT INTO (sloupec1, sloupec2) VALUES (hodnota1, hodnota2), (hodnota3, hodnota4). Je tedy nutné pro každý řádek použít nový příkaz INSET. To je zbytečné jak programátorsky, tak to musí mít i negativní dopad na výkon.

1.2.2 Oracle

V roce 1977 společnost Relational Software Incorporated vyvíjí databázový systém nazvaný Oracle. V roce 1982 vyšla verze 3, která byla pro sálové počítače, minipočítače i stolní počítače, kde již byla implementována podpora transakčního zpracování. Ve verzi 5 z roku 1985 byla již architektura klient - server a podpora distribuovaného zpracování. V letech 1992 – 1994 byla vytvořena verze 7 pro platformy Unix a PC. Již v roce 1998 ve verzi 8 byla podpora rozsáhlých databází, nových datových typů pro ukládání obrazových a multimediálních dat a podpora objektově orientovaných technologií.

Oracle patří mezi nejznámější databázové systémy a dokonce se dá tvrdit, že i mezi nejrozšířenější databázové systémy používaná pro velké nasazení jako například v bankovním sektoru nebo velké nadnárodní společnosti. Tomu napomáhá zpřístupnění pro celou řadu provozních platforem s tím, že z jednotlivých hardwarových produktů je vytěženo maximum, například v oblasti paralelního zpracování a nepřetržité dostupnosti. Největším konkurentem je samotný Oracle, kdy s každou verzí se snaží překonat tu stávající a držet se tímto stále na vrcholu.
Kladen je také důraz na problematické zákoutí národních prostředí a zvyklostí. Nejedná se pouze o znakové sady, ale například také o správnou práci s časem v různých lokalitách.
Výhodou Oracle je komplexní sada produktů spojená s tímto databázovým systémem. Není zde tedy nutné využívat různých řešení třetích stran. Zde je jedinou výjimkou operační systém, ale díky multiplatformní podpoře to není problém.

Velkým kladem je jeho bezpečnost, v jeho prospěch mluví splnění podmínek mezinárodně uznávaných bezpečnostních certifikátů. Procesy související s uznáním certifikátu jsou dosti obtížně a je tedy možné toto považovat za bezpečnostní kvalitu tohoto systému.

Další neméně důležitou výhodou je vysoký výkon. Ten je dosažen díky konceptu gridu. Grid spočívá v rozložení výpočetní kapacity mezi vzájemně propojenými počítači. Vývojář se nemusí zabývat rozdělením tohoto výkonu, ale celé to zajišťuje grid. Je jedno, zda jsou počítače propojení stále nebo jen dočasně. Na systém je kladen dotaz a ten je okamžitě zpracován, a to přímo ve chvíli, kdy jej potřebujeme provést a ne až v momentě, kdy má dostupný počítač volný výpočetní výkon. Rozložením výpočetního systému a maximálním využití dostupných výpočetních zdrojů umožňuje snižovat celkové náklady na celý výpočetní systém a snižuje se tímto potřeba budovat stále větších systému.

Oracle také nabízí nástroje pro audit celého systému. To umožňuje sledování a analýzu systému bezpečnostních hrozeb. Jeho doplňkem je také Oracle Database Vault. Ten umožňuje administrátorovi databáze odebrat práva nad provozními daty, má tedy možnost pracovat jen s daty pro běžný provoz. Tímto se můžeme vyvarovat vnitřním bezpečnostním chybám.

Dále komprese dat nám pomůže při velkém množství dat. Tato komprese je tak optimalizována, že nemá negativní dopad na vyhledávání. To nám celé umožní snižovat náklady na datové sklady.

Dalším doplňkem je systém pro obnovu dat. Ten dokáže v případě ztráty dat doporučit nejlepší postup, jak minimalizovat dopady nedostupnosti databázového prostředí.

1.2.3 DB2

DB2 má velmi dlouhou historii a je považováno za nejstarší databázový produkt. Jméno DB2 dostalo v roce 1983, kdy ho IBM vydalo na své MVS platformě. Dokonce i dalo za vznik nového jazyka SQL, které vzniklo z původního SEQUEL. Po několik let bylo DB2, jako plnohodnotný databázový systém, jako jediné dostupné na sálových počítačích IBM. Později IBM přineslo DB2 to dalších platforem, včetně IS/2, Unix, Windows, Linux a PDA.

Databázový systém DB2 se vyznačuje vysokým výkonem a vysokou dostupností. Nechybí podpora paralelního zpracování, k dispozici je vícedimenzionálního clusteringu nebo automatické prostředky pro správu a výkonnostní ladění.
Součástí systému je technologie pro zotavení systému v případě havárie, možnost dělení databáze do několika oddílů v případě větší databáze.
V případě velkých tabulek je považována za výhodu podpora on-line reorganizace databáze s mechanismem přesunu reorganizovaných bloků dat bez nutnosti alokace dodatečného prostoru a časových nároků zpomalující běžný provoz.
Nechybí zde ani možnosti komprese dat. Přináší to jak finanční úsporu v nákladech na uchovávání dat, ale také zvýšení výkonu díky menšímu množství přenášených dat. Výhodou je i možnost efektivního využití paměti v některých typech zpracování, když není třeba dekomprimovat všechna zpracovávaná data.
V případě rozsáhlého systému, kdy je infrastruktura rozmístěna ve více lokalitách, přichází vhod replikační technologie.
Tak jako u předešlých databázových systémů, ani zde nechybí podpora XML dat. Umožňuje nativní uložení včetně rychlého zpracování nebo změn jednotlivých částí XML dokumentu a ne jen jako celku.
K dispozici je podpora nestrukturovaných dat, jako například obrázky, multimediální data nebo videa. Součástí jsou také rozšíření potřebná pro práci s prostorovými daty. K dispozici je propojení libovolných dat s prostorovými, lze využít rychlé a prostorové indexy nebo mapová projekce. Je tedy vhodné pro analytická řešení, a tedy i často nasazováno pro datové sklady.
Má také podporu integrace do VisualStudia nebo podporu distribuovaného ladění.
V rámci bezpečnosti jde udělovat práva až na jednotlivé záznamy nebo sloupce.

1.2.4 PostgreSQL

PostgreSQL je relační databázový systém s otevřeným zdrojovým kódem. Je šířen pod licencí BSD, která umožňuje volné spojování otevřeného kódu s uzavřeným. Často je srovnáván s další rozšířenou otevřenou databází MySQL. Předchůdcem byl systém Ingres vyvinutý v letech 1977 – 1985 na kalifornské univerzitě v Berkley. Jeho nástupce byl vyvíjen jako objektově-relační databázový server pod názvem Postgres. Později byl doplněn o podporu jazyka SQL a byl označen Postgres95. V létě 1996 byl sestaven tým

lidí, kteří pokračovali na vývoji jako open-source a nezávisle na univerzitě. Na konci roku 1996 byl projekt přejmenován na PostgreSQL.

Je to multiplatformní systém, tedy lze jej provozovat na operačních systémech Linux, Unix, Windows, Mac OS X a další. I přes to, že má podporu pro Windows, tak na tomto systému nefunguje na 100%. Mnoho vývojářů, kteří se snažili nasadit PostgreSQL na Windows a velmi brzy museli buď přejít na jiný databázový systém, nebo použít PostgreSQL na jiném operačním systému, kde je nejvíce doporučován Linux/Unix.
PostgreSQL umožňuje běh uložených procedur napsaných v několika programovacích jazycích, v Perlu, v Pythonu, v jazyku C nebo v speciálním PL/pgSQL, který vychází z PL/SQL vytvořený firmou Oracle.
Předností systému PostgreSQL je rozšířitelnost. Systém může být bezproblémově rozšiřován o nové datové typy, funkce, operátory, agregační funkce, procedurální jazyky. Díky tomu mohla vzniknout spousta rozšíření, některá jsou i volně ke stažení na internetu.
I přesto, že je konkurenční MySQL rychlejší než PostgreSQL, tak to platí pouze u menší zátěže serveru a základních příkazů. Při vyšší náročnosti se vyrovná v rychlosti s MysSQL, někdy dokonce je i rychlejší.
Replikace databáze je zde stejně možná, jako u komerčních databázových systémů. Není přímo jeho součástí, ale je třeba ji doplnit pomocí rozšíření. Rozšíření pro replikaci je několik a nejpoužívanější je asi Slony. Stejným způsobem je zde podporován i clustering.

1.2.5 MySQL

MySQL je databázový systém, který původně vznikl v roce 1996 vytvořen firmou TcX, nyní vyvíjen švédskou firmou MySQL AB, kterou nedávno koupila firma Sun Microsystems. Je považován za úspěšného průkopníka dvojího licencování. Je k dispozici jak pod bezplatnou licencí GPL, tak pod komerční placenou licencí.

MySQL je plnohodnotný multiplatformní systém. Oproti PostgreSQL se dá bezproblémově použít jak na Linuxu, tak na Windows.

Je řazen mezi nejrychlejší databázové systémy. Určitě se nedá jeho výkon srovnávat s Oracle a jeho grid technologií, ale firma Google by určitě mohla potvrdit, že je reálné ho využít na projektu s obrovským množství dat, jelikož sami jej používají pro mezinárodní internetový vyhledavač Google a jeho výsledky během vyhledávání jsou zpracovány opravdu rychle.
V Česku je to nejrozšířenější databázový systém, převážně mezi začínajícími vývojáři, nízkonákladové firmy a studenty. Přispívá k tomu také fakt, že převážná většina českých free webhostingů a placených webhostingů používá právě databázový systém MySQL. MySQL není tolik náročné na instalaci a konfiguraci, tudíž mnohem jednodušší pro provozovatele webhostingů.
Díky takové oblíbenosti je v Česku komunita kolem MySQL mnohem větší než kolem ostatních databázových systémů. Také hodně tutoriálů, návodů nebo open-source aplikací týkající se převážně webových aplikací nebo programovacím jazykům, používají právě MySQL. Tedy spousta začínajících vývojářů začínají na MySQL a mnohdy u něj také zůstanou. Je to také zapříčiněno bezplatnou licencí.
Dříve bohužel nepodporovalo pohledy, triggery a uložené procedury, které naštěstí jsou již v posledních letech podporovány. Jelikož tato podpora se objevila mnohem později než u ostatních nejrozšířenějších databázových systémů, přišlo tak MySQL o spoustu svých příznivců, kteří takto přešli na jiný databázový systém.

1.2.6 Shrnutí

Databázový systém MS SQL je určitě kvalitně provedený systém s dobrými funkcemi. Jeho výhoda oproti konkurenci jsou převážně dobré nástroje na sledování a analýzu chodu celého databázového systému. Bohužel je vázáno na operační systém Windows. Na první pohled to může být také jeho výhoda, jelikož s propojením s dalšími aplikacemi od firmy Microsoft může vzniknout spolehlivý a výkonný webhosting, bohužel tato možnost připadá v úvahu pouze při použití ASP.NET.

Shledávám Oracle jako nejkvalitnější produkt na poli databázových systémů. Jeho výhody a síla oproti ostatním systémům je zřejmá. Bohužel jeho velké finanční nároky jak na jeho zakoupení, tak i provoz, jsou obrovské. Pro využití tohoto databázového

systému bych se rozhodl v případě, kdybych ho chtěl použít na rozsáhlý projekt provozovaný na větším množství serverových stanic a velkým finančním kapitálem.

DB2 se mi nezdá jako nejlepší řešení pro nasazování na běžné projekty. Patří mezi pouze komerční produkty a pro běžné projekty nepřinese nic závratného. Opakem by bylo, kdyby bylo potřeba mít datový sklad. V tomto případě bych volil právě DB2, kde je podpora prostorových dat a indexů, mapové projekce a další možnosti, které jsou pro datové sklady přínosem.

Mezi nekomerčními databázovými systémy bezpochyby vládne PostgreSQL. I když samotný systém nemá funkce, kterými by vynikal, jeho rozšíření z něj mohou udělat silný nástroj, mnohdy je uváděno, že se blíží k systému Oracle. Databázový systém Oracle se mi jeví jako nejlepší řešení pro středně velké projekty, kdy se už využívá i více databázových serveru napojených na sebe. Velkou výhodou je samozřejmě také jeho bezplatná licence.

MySQL je největším rivalem PostgreSQL. MySQL se naštěstí před pár lety vzpamatovalo a doplnilo si funkce, za které bylo kritizováno, že mu scházejí. Tím se může znovu srovnávat s ostatními kvalitními databázovými systémy. Volba mezi MySQL a PostgreSQL závisí na konkrétním projektu, který se má realizovat. Pro tento projekt je ale MySQL po stránce funkcionality zcela dostačující a navíc oproti PostgreSQL vykazje rychlejší výkonnostní časy, kdy tuto vlastnost uvítám. Nepředpokládám u tohoto projektu složité dotazy na databázový systém a tak MySQL zůstane pro nás stále rychlejším řešením.

1.3 Cíl práce

V předchozích kapitolách jsme si představily možné programovací jazyky pro vývoj webových aplikací a databázové systémy. V dalších částech práce se budu zabývat řešením aplikace, naprogramovaným pomocí PHP, využívající databázový systém MySQL a to celé poběží na operačním systému Linux, kde mají oba dva stoprocentní podporu.

2 Analýza problému a současná situace

První možnost, která připadá v úvahu je využití open-source systému. Tato možnost se zdá být výhodnější než vytváření celého systému úplně od začátku.

Všechny systémy, kterými se budu zabývat, budou šířeny pod licencí GNU GPL. GNU GPL licence dává každému právo instalace softwaru za úplatu a jde spolu s administrací asi o nejčastější placený úkon spojený s „GPL softwarem“. Je zde také právo na poplatek za distribuci tohoto software, což je jedním ze základů „free“ softwaru (např. u freewaru nesmí být poplatek ani za distribuci). Proto můžeme za distribuci vyžadovat peníze, nesmí to být ovšem cena za program. Kdokoliv může tyto zdrojové programy z licence GNU GPL přeložit na příslušnou platformu, lokalizovat, přidat manuály, instalační program, zabalit a prodat. V ceně nesmí být zahrnuty zdrojové programy použité z licence GNU GPL. Dále platí, že uživatel, který daný software jakýmkoliv způsobem obdrží, byť za úhradu, ho může dále modifikovat, kopírovat a dále nabízet ke stažení zdarma či za poplatek popsaný výše. Licence GNU GPL se vztahuje pouze na software pod ní šířený, autorská práva na grafický design, texty a třeba fotografie, tedy obsah stránek je stále v držení jeho autora. Licenci softwaru nelze měnit. Pokud je jakýkoliv software šířen pod licencí GNU GPL, musí takto být šířen nadále.

Dalším, samozřejmě také důležitým kritériem bylo, aby všechny systémy byly vytvořeny v jazyce PHP a využívaly databázový systém MySQL.

Open-source systém pro e-shop je mohutné nástroje. Obsahuje stovky funkcí a nastavení. Kategorie zboží a jejich obsah umožňuje neomezený počet jazykových mutací, grafické návrhy a možnost vlastních úprav vzhledu a funkčnosti, nastavení způsobu prodeje, objednávek, doručení a plateb. U některých systémů má každý zákazník má vlastní administrační prostředí, kde přehledně vidí svou historii objednávek, jejich stav zpracování, veškerou komunikaci mezi ním a vámi.

Systém samozřejmě umožňuje evidenci skladových zásob, upozornění na podlimitní stav, zboží na cestě, výprodej, slevy a další nutné funkce pro každý e-shop.

Přehledná správa objednávek a jejich stav vyřízení. Informace o zákaznících, rozesílání novinek a hromadnou korespondenci. Umožňuje také vést e-shop v několika měnách a automaticky převádí ceny podle zadaných kurzů. Umožňuje hromadnou aktualizaci zboží například importem excelovských tabulek a podobně. Pro potřebu dalších funkcí a nastavení existují tisíce zásuvných modulů, které e-shop rozšiřují a umožňují tak další funkce, například reklamní bannery, zálohování, export XML souborů do seznamů katalogů, SEO nástroje a podobně.

Software zvládá od jednotlivých produktů až po webové servery s desítkami tisíc položek (zde již bývají velké hardwarové požadavky a doba vykonávání skriptů). Pro všechny tyto funkce a možnosti je to oblíbený nástroj pro miliony spokojených uživatelů po celém světě další miliony jejich spokojených zákazníků. Rozsáhlá komunita uživatelů již vyřešila mnohé možnosti a možné i nemožné problémy spojené s internetovým prodejem a na specializovaných fórech věnovaných danému systému se dají nalézt informace o jakémkoliv nastavení a možnostech toto software.

2.1 ZenCart

ZenCart je kompletní e-shop se vším všudy – kategorie zboží, detaily zboží s řadou fotek, možnosti různých variant zboží (např. barva, velikosti), samozřejmě košík, různé způsoby zaplacení, nastavení poštovného podle hodnoty objednávek a mnoho dalších možností.

ZenCart je jeden s nejvíce rozšířených e-shop systémů. Popularity se více dočkal v zahraničních zemí než v Česku, a to převážně z důvodu dříve nepodporované češtiny. Nyní má tento systém u nás českou komunitu včetně diskusního fóra, kde mohou registrovaní uživatelé řešit své problémy s instalací a úpravami. O oblíbenosti ZenCartu svědčí jeho vysoká používanost, což lze považovat za jednu z nevýhod.

Klady

- Česká komunita, včetně diskusního fóra.
- Podpora českého prostředí a česká lokalizace.
- Spousta grafických šablon, a to jak zdarma, tak i placené.

Zápory

- Velká náročnost na databázový server. Při jednom znovunačtení stránky se provede přibližně 500 – 800 SQL dotazů na databázi. Počet dotazů se dále zvyšuje s narůstajícím množství záznamů.
- Složité zápisy zdrojového kódu. To zapříčiňuje velmi staré jádro systému. Toto je velká slabina v rámci využívání tohoto systému pro komerční nasazování tohoto systému. Je to tedy vhodné nasadit pouze pro zájemce, kterým vyhovují současné možnosti systému, anebo jeho zásuvných modulů.
- Pro správce systému je zpočátku složité spravovat samotný e-shop, a to z důvodu složitého administračního prostředí, které obsahuje spoustu mnohdy nepotřebných nastavení a možností.
- S novou verzí můžete přijít o vzhled a vlastní doprogramované věci

2.2 OpenCart

OpenCart je zajímavým e-shop systémem, který je dobrou alternativou k oblíbeným systémům jako je například ZenCart. Oproti nim zatím neobsahuje tolik funkcí a zásuvných modulů, avšak působí mnohem elegantnějším a přehlednějším dojmem. Česká lokalizace není standardní součástí systému.

Klady

- OpenCart vypadá jednoduše a přitom má asi všechno, co menší internetový obchod potřebuje. Jednoduchý je také v administraci.
- Grafické šablony, a to jak zdarma, tak i placené.
- Aplikace je napsána objektově a snaží se dodržovat oddělení jednotlivých vrstev architektury.

Zápory

- Česká lokalizace není k dispozici. Naštěstí na českém internetu se dají najít balíčky s českou lokalizací, kterou je možné dodatečně dohrát, a zároveň je nutné provést menší úpravy v kódu týkající se komunikace PHP s databází (kódování znakové sady).

- Česká lokalizace je sice ke stačení na oficiálních stránkách aplikace, ale má několik problému. Největší asi je, že není kompletní, jedná se hlavně o administrátorskou část. Kdyby texty byly alespoň v angličtině, tak lze zjistit chybějící texty a dodělat je, problém je, že pokud není nalezen odpovídající text, nezobrazí se nic.
- Na problém lokalizace navazuje problém s doplňky, ty jsou vázaný na jazyk, takže pokud se podaří češtinu sehnat, tak zmizí všechny doplňky.

2.3 OsCommerce

OsCommerce je e-shop určený zejména pro neprofesionály a začátečníky. Má jednoduchý kód a jeho instalace je velice rychlá a intuitivní.

Klady

- Obsahuje přehledný systém podpory reklamy.
- Grafické šablony, a to jak zdarma, tak i placené.
- Obrovská komunita, ale převážně na zahraničních serverech.

Zápory

- Staré jádro systému. Vhodné pro začátečníky v programování v PHP, ale nevhodné pro komerční využití a jeho vylepšování.
- Slabší funkcionalita oproti ZenCartu.
- Vyžaduje podobnou náročnost serveru jako ZenCart.

2.4 VirtueMart (Joomla)

VirtueMart je open-source internetové komerční řešení, internetový obchod a rozšíření pro redakční systém Joomla a Mambo. Joomla je jeden z nejvíce používaných redakčních open-source systémů. Historicky se vyvíjí ze svého dnes už konkurenta Mambo. Může se provozovat jako internetový obchod, případně poue jako katalog zboží pro web.

Klady

- Podpora českého prostředí a česká lokalizace.
- Velké množství šablon zdarma.
- Velká česká komunita.
- Objektově orientované programování
- Softwarová architektura MVC

Zápory

- Nemá všechny potřebné funkce jako plnohodnotný e-shop systém.
- Náročný na databázový server. Oproti systému ZenCart vykazuje databázové zatížení na počet dotazů pouhých 10%, ale stále je to 10krát více, než by bylo vhodné.

2.5 Magento

Magento je nový profesionální systém pro tvorbu e-shopu, který se rychle rozrůstá a vyniká především svou moderností a uživatelskou příjemností. Funkčně se Magento plně vyrovná nejpopulárnějšímu e-shop systému ZenCart. Jádro systému je postaveno na Zend Frameworku.

Klady

- Podpora českého prostředí a česká lokalizace.
- Česká komunita, která se velmi dynamicky rozrůstá.
- Jádro systému je postaveno na Zend Frameworku.
- Intuitivní administrační rozhraní.

Zápory

- Spousta částí nedodržuje aplikační logiku Zend Frameworku.
- Pomalejší vykonávání dotazu, větší hardwarové nároky.
- Na některém webhostingu se nedaří tento systém úspěšně funkčně provozovat.

2.6 Shrnutí

Výše uvedené systému se budu snažit charakterizovat do společných rysů.

Velkou výhodou těchto systémů jsou komunity jak v Česku, tak i v zahraničí. Jelikož jsou všechny systému zahraničních vývojářů, největší komunity se právě objevují na zahraničních mezinárodních serverech. To také vyžaduje, aby správce systému měl znalost anglického jazyka.
Další výhodou jsou rozsáhle galerie a seznamy grafických šablon. Bohužel tohle není výhoda, která by se týkala mého případu, kde bych nevyužíval předvytvořených šablon, ale vytvářely by se zcela unikátní grafické návrhy stránek.

Hlavní nevýhodou všech systémů je jejich hardwarová náročnost. U starších systémů se to projevuje ve velkém zatížení databázového serveru, u nejmladšího systému Magenta je tento problém v náročnosti vykonání skriptů. V obou případech vznikají problémy na dvou stranách. Zaprvé je potřeba zajistit výkonný webhosting, a i na výkoném webhostingu to bude zpomalovat celkové zpracování serveru. Zadruhé se problém objevuje na straně klienta, kdy klient musí čekat mnohdy i několik sekund, než se mu stránka načte. Samozřejmě několik sekund se zdá být jako zanedbatelná doba, ale v dnešní době, kdy většina domácností má vysokorychlostní internet a stránky se jim načítají průměrně za jednu sekundu, tak poznají zpomalení načítání, které bude několikanásobně pomalejší než i jiných stránek a raději přejdou na jiný internetový obchod, který bude nabízet stejný sortiment.

Závěrem bych uvedl, že žádné řešení již hotového open-source systému mi nepřišlo natolik vyhovující, aby na něm šly stavět e-shopy dle mého očekávání. Volím tedy další možnost, kdy celou aplikaci vytvořím celou sám a nebudu přebírat již hotové řešení.

3 Teoretická východiska práce

Databázovým systémem se již v této kapitole zabývat nebudu. Je ale potřeba se ještě blíže věnovat programovacímu jazyku.

Dříve při vzniku programovacích jazyků se počítalo, že se bude s programovacím jazykem pracovat tak, jak jsou jeho možnosti navrhnuty, a tento způsob je označován jako „procedurální“ programování. Později se ukázalo, že je vhodné pří programování využívat tzv. objekty a z toho vzniklo i objektově orientované programování. Tyto objekty daly programování úplně jiný směr a umožnilo vznik frameworkům.

3.1 Co je to framework?

Framework je balík naprogramovaných objektů, kdy je definováno, jakým způsobem se mají objekty používat, popis jejich funkcí, jak je modifikovat, případně jak vytvářet zcela nové.
Některá pravidla je nutné dodržet kvůli základní funkčnosti frameworku, některá pravidla jsou jen jakýmsi „doporučením“, kdy to napomáhá při orientování se ve dříve vytvořených aplikací nebo ve zdrojovém kódu napsaném někým jiným. Samozřejmě žádný framework nemůže měnit základní pravidla syntaxe daného programovacího jazyka, ty vždy zůstávají.
Framework nám také může urychlit práci. Při dobře definovaných pravidlech a využití vhodné aplikační architektury nám velmi zkrátí čas na vývoj aplikací.
Jsou vytvářeny 3 typy frameworků. Zaprvé soukromé frameworky, kdy si programátor sám vytvoří framework pouze pro vlastní využití. Zde je nevýhoda strávení spoustu času nad jeho vytvářením, testováním, opravováním, modifikováním nebo rozšiřováním. Zadruhé jsou tou frameworky vytvoření v týmu programátorů určité firmy. Zde se již mohou role na vývoji frameworku rozdělit mezi jednotlivé programátory. Tyto frameworky bývají striktně uchovány v know-how dané firmy a mimo firmu se nešíří. Třetí a nejrozšířenějším řešením jsou frameworky vytvářeny týmem nezávislých vývojářů, kteří ani nemusejí spolu pracovat na stejných projektech, ale spolupracují na vývoji daného frameworku. Tyto frameworky jsou dále volně nabízeny společnosti, což nabízí velký potenciál hlavně pro testování. Týmy se

postupně rozrůstají o nové programátory a kolem frameworku vznikají komunity vývojářů využívající tento framework pro vlastní účely.

Právě tou třetí možností volně šiřitelného frameworku se budu chtít zabývat. Vytvořím přehled několika nejpoužívanějších PHP frameworků a vyberu nejvhodnější. Výběr frameworku bude převážně můj subjektivní názor a vlastní předpoklad, která framework se v budoucnu bude nejlépe vyvíjet. Frameworky v PHP jsou celkem mladou záležitostí několika let, a tak řada jich je stále ještě ve vývoji a ladí se nalezené problémy a chyby.

3.2 Volně šiřitelné PHP frameworky

Zend Framework

Mezi nejznámější PHP framework dozajista patří Zend Framawork, podporovaný přímo firmou Zend, která vyvíjí jádro PHP. Jeho začátky sahají do roku 2005 a nyní je dostupné ve finální verzi 1.8. Celý framework je objektově orientovaný a implementovaný v PHP 5. Většina frameworku vznikla za určitým účelem, kdy postupně jak se framework vyvíjel, dospíval k obecnějšímu použití. Zend Framework byl od počátku stavěn pomocí modulárních balíčků, kdy mají třídy jasně strukturované pojmenování a je možné různé balíčky používat také samostatně. Částečné závislosti mezi komponentami avšak existují. Návrhy tříd jsou striktně vytvářeny podle návrhových vzorů. Je kladen důraz na to, aby si programátor mohl třídy sám rozšiřovat. Zend Framework jako celek využívá návrhový vzor MVC, což je aplikační architektura, oddělující od sebe model-view-controller. Zend Frameworku je převážně vyčítána jeho rychlost, kdy zaostává za svou konkurencí.

Prado

Prado je komponentový framework, který byl jako jeden z prvních PHP frameworků napsán pro kdysi nové PHP 5, ale původně byl napsán pro PHP 4 v roce 2004 a jeho zakladatelem je Qiang Yue. Aktuálně je ve verzi 3.1.4. Obsahuje více než sto tisíc řádků kódu a okolo pěti set tříd. Qiang Yue si nevzal za vzor Runy on Rails jako většina PHP frameworků, ale inspiroval se spíše ASP.NET a Borland Delphi. nicméně tato inspirace je více vidět ve starších verzí, nyní již přibývá mnoho nových komponentů a mění se funkčnost. Prado je komponentový framework využívající programování událostí. Komponentový framework se liší oproti ostatním tím, že se nezobrazí uživateli taková

stránka, na kterou kladl požadavek, ale jednotlivé komponenty jsou rozmístěny na stránce a existují zcela nezávisle. Toho je samozřejmě dosaženo za pomocí JavaScriptu a technologii AJAX, kdy mezi klientským webovým prohlížečem a webovým serverem probíhá komunikace, aniž by muselo dojít k znovu načtení celé stránky. Uživateli se tedy zobrazí na daném místě pouze ten nový obsah, o který žádal. Prado dále nabízí oddělení prezentace od aplikační logiky, konfiguraci pomocí XML souborů, modulární architekturu a další, v dnešní době snad už standardní, možnosti jako například validace vstupních, autorizace nebo autentizace. Celkově je Prado průlomovým frameworkem a vytváří zcela nový přístup k vytváření PHP aplikací.

Symfony

Symfony je PHP framework od firmy Sensio Labs. Je zaměřen na urychlení vytváření a údržby webových aplikací tím, že odstraňuje opakované psaní kódů. Přístup Symfony je zas odlišný od ostatních frameworků. Na server se nenahrává pro každou webovou aplikaci vlastní Symfony, ale nahraje se na server mimo dosah webového přístupu a tak, aby na něj měly všechny aplikace přístup, nejlépe nad rámec webových aplikací. Navíc ke většině funkcí se dá přistupovat přes příkazovou řádku. V adresáři webové aplikace je obsah, který má být přístupny z webu, ale veškerá aplikační logika, včetně konfigurace, je v Symfony. Konfigurace Symfony využívá kaskádový přístup, tedy nejvyšší je konfigurace frameworku, ta může být přenastavena konfigurací projektu a dále pokračuje aplikace a modul. Celá konfigurace j prováděna pomocí jazyka YAML. Dalším přínosem od Prado je, že se nemusíte zabývat tvorbou administračního prostředí pro danou webovou aplikaci, ale Prado ho automaticky vytvoří za nás.

CakePHP

CakePHP se nechalo také inspirovat po vzoru Ruby on Rails. Vzniklo v roce 2005, kdy Ruby on Rails získalo svoji popularitu. Nyní je vyvíjeno firmou Cake Software Foundation, Inc. a jeho aktuální verze je 1.2.2.8120. CakePHP není přímo bránou Ruby on Rails do světa PHP, ale přivlastnilo si některé jeho postupy. Jako snad jediný PHP framework patřící mezi nejrozšířenější není použitelný pouze na PHP 5, ale je také zpětně kompatibilní s PHP 4. Aby se mohlo CakePHP dál vyvíjet, určitě si nebude moct již dlouho podporu PHP 4 udržet, kvůli nedostatečné podpoře objektového

programování v PHP 4, a navíc zabývat se v dnešní době verzí PHP 4 je bezpředmětné a neefektivní. CakePHP využívá, tak jako Zend Framework, návrhový vzor aplikační architektury MVC (model-view-controller). Nabízí také CLI (command-line interface), neboli možnost práce přes příkazovou řádku, kde požadavky jsou prováděny přes tzv. „cake“ příkazy. CakePHP není modulární, je tedy nutné vždy využít celkového konceptu CakePHP.

Nette

Nette je PHP framework od českého autora Davida. Je napsaný v PHP 5 s plným využitím objektově orientovaného programování. Ačkoliv vznikl už v roce 2004, teprve nedávno byl uvolněn jako open-source a zpřístupněn veřejnosti. Je to řádný případ situace, kdy framework vytvořit jeden programátor pro vlastní použití, ale postupem času vyšlo najevo, že jedině jako open-source řešení má možnost se rychleji a lépe rozšiřovat o nové funkce. Vyrostla kolem něj aktivní skupina českých PHP vývojářů a už jej využívají i některé významné české společnosti. Nette je koncipováno jako „otevřený“, je tedy možné ho používat i v primitivních webových aplikacích nebo společně s jiným otevřeným frameworkem, jako je například Zend Framework. Zajímavou výhodou je jeho debugovací nástroj zvaný „Laděnka“. V klasickém případě, kdy uděláte při programování PHP chybu, zobrazí se chybová hláška, kde neznalý programátor v této problematice těžce dohledává chybu. Při zapnutí debugovací nástroje je chování mnohem přívětivější k vyhledání dané chyby, vypíše to jednoduše, v kterém objektu nastala chyba a také zobrazí část PHP souboru s jeho kódem a zvýrazní kód, kde se chyba vyskytla.

3.3 Výběr frameworku

Z výše uvedených PHP frameworků bych vybral pouze 3, o kterých se vyplatí přemýšlet jako o možném řešení. Jedná se tedy o Zend Framework, Prado a Symfony. Ostatní 2 jsou sice také zajímavé, ale nepřinášejí žádné nové funkce, ale jsou spíše menším bratříčkem jiného frameworku.

CakePHP má blízko k Zend Frameworku, kdy má sice možnost práce přes příkazový řádek, ale kompatibilita s PHP 4 ho posouvá na žebříčku směrem dolů, nehledě na to, že má i mnohem méně funkcí.

Nette se svým komponentním zpracováním podobá Prado. V budoucnu může mít potenciál stát se silnějším nástrojem, ale jeho konkurence je zatím v pokroku dále a to hlavně díky delší době, kdy jsou známy pro širokou veřejnost. Dále je Nette zatím šířeno jen v Česku, což mu zatím dává také omezení v růstu. I přes to, musím pochválit jeho debugovací nástroj.

Všechny tři hlavní frameworky nabízejí rozdílný přístup k programováním a určitě mají tímto i své výhody, záleží tedy jen na použití, pro které ho chceme využívat. Zde bych vyzvednul u Zend Frameworku jeho obrovský potenciál v komunitě, Prado má zajímavý přístup pomocí komponentů a AJAXu a Symfony šetří místo a práci díky uložení Symfony na jedno místo pro všechny aplikace a dále kvalitním generováním administračního rozhraní.

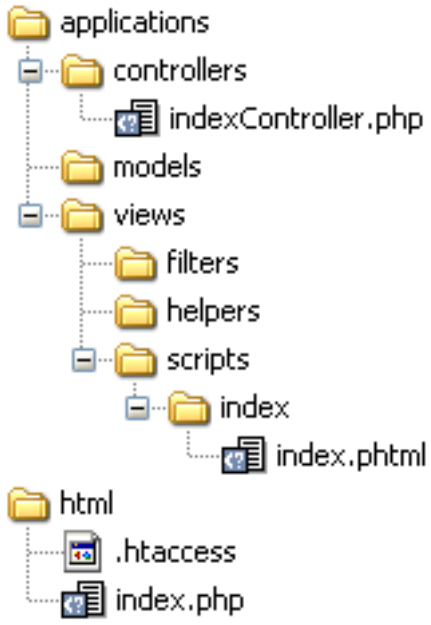
Vítězem je pro mě Zend Framework. Velká komunita je pro mě velkým aspektem, kdy se i jeho potenciál ukazuje na rychlém vývoji, kterým doslova „probíhá“. Má sice slabost, a to v pomalejším výkonu, ale věřím, že tento hendikep bude brzy odstraněn. Někde se můžeme také dočíst o jeho složitosti. To bych ale nebral jako nevýhodu a už vůbec ne jako aspekt pro výběr frameworku. Je to jako bychom vybírali programovací jazyk podle složitosti, ale přitom nám jde o to, aby zvládal vše, co potřebujeme, a v momentě, kdy se naučíme daný framework správně používat, bude velmi mocným nástrojem pro vývoj.

3.4 Bližší seznámení se Zend Frameworkem

Dále se budu snažit blíže představit Zend Framework. Nebudu se již zabývat teoretickým přehledem, co umí, ale budu již pojednávat do hloubky jeho funkci.

Zend_Controller

Jádrem Zend Frameworku je Zend_Controller. Je to balíček, který se stará o obsloužení požadavku, vytvoření modelu a vyrenderování view. Je postaven na návrhovém vzoru aplikační architektury MVC (model-view-controller) a Front_Controller, který zajišťuje aby všechny požadavky šly přes jeden objekt, který rozhodne, která akce kontrolerů požadavek obslouží.



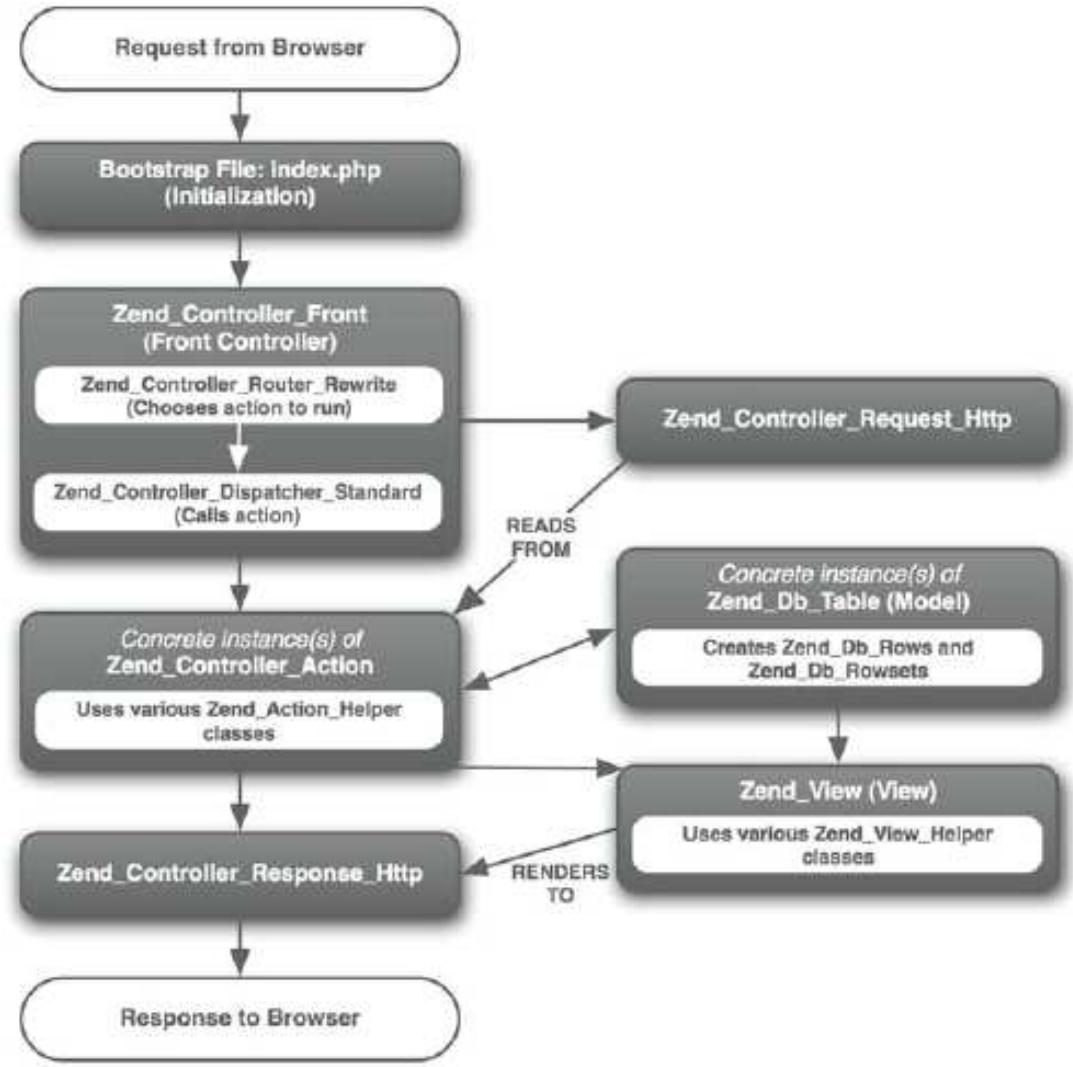
Obrázek 2 - Základní adresářová struktura v Zend Framework

Cyklus obsluhy požadavku probíhá následovně:

1. V bootstrapu (index.php) se zavolá metoda Front_Controlleru run nebo dispatch, která začne obsluhovat požadavek. Pokud nebyly předány instance objektů Zend_Controller_Request a Zend_Controller_Response, tak jsou vytvořeny.
2. Dalším krokem je routování. Routování probíhá na základě URL adresy. Defaultně se počítá, že tvar URL bude ve tvaru *domena/:modul/:kontroler/:akce*, pokud chceme, aby routování probíhalo jiným způsobem, stačí pro Zend_Controller_Front nadefinovat nová pravidla routování.
3. Nyní vykonává svoji práci Zend_Controller_Dispatcher, který na základě názvu kontroleru a jeho akce vytvoří jeho instanci a zavolá příslušnou metodu.

Dispatcher pracuje s danou konvencí v názvech kontrolerů a akcí. Název kontroleru bude přeložen na *názevkontroleruController* a název akce na *názevakceAction*.

4. V příslušné metodě kontroleru dojde na naplněné modelu dat. Tento bod může probíhat i ve smyčce. Je tedy možné z určité metody volat další metody, a to i v jiných kontrolerech.
5. Dojde k vyrenderování view a jeho nastavení do objektu Zend_Controller_Response.



Obrázek 3 - Cyklus požadavku v Zend_Controlleru

To byl ve zkratce výčet základních bodů, kterými prochází obsloužení každého požadavku. Zend Framework dále obsahuje tři typy objektů, které můžeme použít při obsluhování požadavku:

- Pluginy – můžeme je napojit na cyklus obsluhy požadavku a mají přístup k Zend_Controller_Request a Zend_Controller_Response. Cyklem jsou myšleny následující události: začátek routování, konec routování, začátek dispatchování, začátek dispatchovací smyčky, konec dispatchovací smyčky, konec dispatchování. Na kteroukoliv z těchto událostí se v pluginu můžeme napojit a modifikovat ji.
- Action Helpery – jsou podobné pluginům v tom směru, že mají přístup k událostem. V tomto případě se ale jedná o jiné události, respektivě: inicializace kontroleru, před zavoláním metody v kontroleru, po zavoláním metody v kontroleru. Můžeme k nim přistupovat staticky nebo přímo v kontroleru.
- View Helpery – tyto pomocné skripty slouží pouze pro views. Zde není jejich význam pro změnu při zpracování požadavku, ale pouze nám pomáhají při zpracování výstupu. Například se dají použít pro generování formulářů, opakovaně zapisované kódy, stránkování, formátování a další.

Zend_DB

Balíček Zend_Db je brán také jako jádro Zend Frameworku, i když někteří uživatelé s ním nejsou spokojeni a místo něj používají buď vlastní třídy na práci s databází nebo převzatou z jiného frameworku. Zend_Db nabízí abstrakci nad přístupem k databázi podobně jako PDO.

Třída Zend_Db_Select umožňuje skládat SQL příkazy a generovat SQL příkaz podle dialektu použitého databázového systému.

Zend_Db_Table je třída pracující s tabulkami a řádky tabulky jako se skutečnými objekty. Jedná se o implementaci návrhového vzoru Table Data Gateway a Row Data Gateway u objektu Zend_Db_Table_Row. Bez explicitního mapování lze přistupovat

k sloupcům tabulky jako kdyby to byly public vlastnosti objektů. Třída také umožňuje namapování tabulek mezi sebou pro spojování tabulek.

Zend Framework zahrnuj ještě spoustu dalších balíčků. Ale ty už nejsou vždy potřeba a záleží jen na konkrétních požadavcích aplikace.

4 Vlastní návrhy řešení, přínos návrhů řešení

Tak jsme v předchozích kapitolách vybrali programovací jazyk, k němu také vhodný framework a nesmíme zapomenout na databázový systém. Ani žádné open-source řešení nepřipadá v úvahu jako vhodné. Je tedy na řadě se vrhnout na vlastní řešení, které bude programované jazykem PHP pomocí Zend Framework a využívající databázový systém MySQL.

4.1 Specifikace aplikace

Je třeba si na začátku definovat, jaké má mít aplikace funkce. Nejdříve chci ujasnit, že se nejedná o individuální obchod. Nebudu se tedy zabývat tvorbou front-endu, což je rozhraní pro nakupující zákazníky, ale budu se zabývat back-endem, tedy administračním rozhraním, který bude sloužit pro správce stránky. Aplikace nemá byt vytvořena pro jedno použití, ale má se využívat pro tvorbu komerčních internetových obchodů. Je tedy nejdůležitější, aby právě administrační část a systémová logika vyhovovali obecně pro tvorbu internetových obchodů, pokud možno s co nejmenšími modifikacemi na individuální potřeby klienta.

Navigace

Na začátek by měla každá webová aplikace obsahovat alespoň základní navigaci. U internetových obchodů se jedná o základní informace o obchodu, případně firmě, která ho provozuje, obchodní podmínky a kontaktní informace. Potřebujeme tedy, abychom mohli vytvářet nové položky v navigaci. Každá položka by měla mít svůj název, pořadí pro zobrazení, pro víceúrovňovou navigaci uvádět i nadřazenou položku a v neposlední řadě obsah stránky.

Kategorizace zboží

Určitě nemůžeme mít zboží v obchodě jen tak rozházené a netříděné. Je tedy nutné definovat kategorie, do kterých může libovolně zboží řadit a také podle nich je nabízet návštěvníkům stránek. Pro kategorii nám bude stačit definovat název, nadřazenou položku a pořadí pro zobrazení.

Produkty

Tato část je základním kamenem každého internetového obchodu. Dokonce i bez objednávkového systému bychom se mohli obejít, a to třeba v případě pouhé prezentace produktů nebo by objednání mohlo probíhat pomocí e-mailů. U produktu budeme definovat jak vlastní specifika, tak u něj budeme volit již vytvořené záznamy, jako například zařazení do dané kategorie. U produktů budeme tedy počítat s následujícími možnostmi: název, obrázek, zařazení do kategorie, kód, výrobce, dodavatel, typ DPH, cena, specifické parametry, přidružené produkty, expedice, odkaz na produkt na stránkách výrobce, skladové zásoby, popis a v neposlední řadě nějaký status.

Dodavatelé

Tato část systému slouží pro vlastní potřeby správce internetového obchodu. Je tedy pohodlné, když se předdefinují vlastní dodavatelé a u produktu se určí, od kterého je. To se hodí v případě, kdy dohled nad internetovými objednávkami má i jiná osoba než osoba řešící nákup zboží. U objednávky a přehledu objednaných produktů si zjistí, od kterých dodavatelů má dané zboží objednat. Věřím, že tuto informaci by občas využil i majitel firmy provozující internetový obchod. Zde stačí uvádět základní informace o firmě a rabat, který nám firma poskytuje. Patří mezi ně: název, jméno a příjmení kontaktní osoby, IČ, DIČ, e-mailová adresa, telefonní číslo, faxová číslo a adresa.

Výrobci

Tato část není příliš podstatná, ale jen doplňuje informace o produktech. Mohli bychom tuto informaci uvádět u názvu produktu, ale takto nám to umožní řadit produkty podle jeho výrobce, případně dát možnost návštěvníkovi podívat se na stránky výrobce daného produktu. Vystačíme si tedy pouze s názvem a odkazem na internetové stránky.

Zákazníci

Můžeme chtít, aby se nám objednávky vytvářely samotné s údaji od návštěvníka stránek, tedy umožnit mu nákup bez registrace, anebo můžeme podporovat či vyžadovat registraci návštěvníka před nakoupením, a tím také získáme informace o zákaznících pro pozdější použití. U zákazníků potřebujeme podrobné informace: název firmy,

jméno, příjmení, IČ, DIČ, e-mailová adresa, telefonní číslo, faxové číslo, heslo, cenová kategorie a adresa.

Objednávky

Objednávky nemusím nějak extra vysvětlovat. Shromažďují informace jak o objednaném zboží, tak o zákaznících, kteří objednávku provedli. Přesněji potřebujeme tyto informace: identifikační číslo objednávky, identifikační číslo zákazníka, telefonní číslo, e-mailová adresa, způsob platby a dopravy, datum vytvoření objednávky, status, celková cena objednávky, fakturační údaje zákazníka, adresa doručení a přehled objednaného zboží.

Uživatelé

Zde se nejedná o zákazníky, ale o uživatele, kteří se přihlašují do administračního rozhraní a spravují internetový obchod. Nastavují se zde přihlašovací údaje, hesla, jména a příjmení uživatelů a případně i jejich práva.

Rozšiřující možnosti

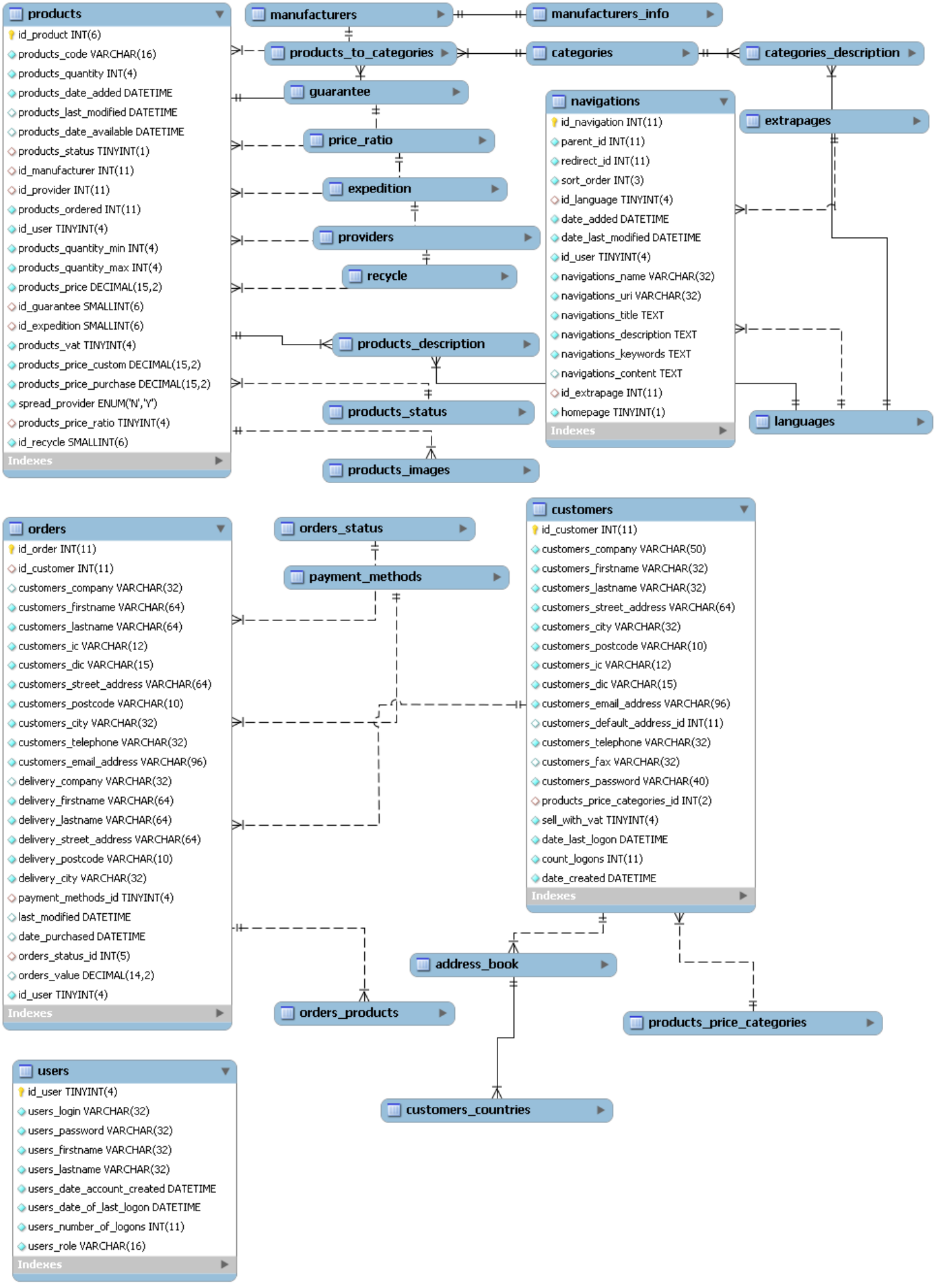
Sem patří doplňující moduly, jako jsou novinky, extra stránky (například formuláře), jazyky a různé nastavení týkající se front-endu (prostředí pro návštěvníky stránek).

Nastavení

Tady bychom mohli zařadit definování e-mailů, které se budou zasílat při objednání zboží, obchodní údaje o firmě provozující internetový obchod, způsoby plateb a dopravy nebo číslování faktur a objednávek.

4.2 Návrh databáze

Nyní provedu kompletní návrh celé databáze. V návrhu budu klást důraz na normalizaci, která nám odstraní redundanci dat, omezí složitosti, zabrání aktualizačním anomáliím, což by mělo vést k přehlednější, rozšiřitelnější a výkonnější databázi.



Obrázek 4 - Schéma databáze

4.3 Vývoj aplikace

Databázi máme hotovou, tak se teď můžeme pustit na programování samotného systému. Níže se budu snažit popsat různé nastavení, použité paginy nebo jiné funkce, které jsem při programování systému použil.

Bootstrap

Bootstrap neboli *index.php* je hlavním souborem, na který jsou směrovány veškeré požadavky a to pomocí souboru *.htacces*, který vše na něj přesměrovává.

Na začátku jsem si hned nadefinoval *Zend_Loader::registerAutoload()*, což mi umožňuje vyvolávat třídy, aniž bych je nejdříve přidal například pomocí *require_once*, nyní se to vše provádí automaticky.

Vytvořil jsem si vlastní třídu pro vypisování chybových hlášek a nastavil, aby všechny chyby se zobrazovaly pouze tam. Tím jsem zamezil vyskakování chyb mezi text zobrazované stránky. Blíže použitou třídu popíši v kapitole o vlastních knihovnách.

Konfigurace potřebné pro připojení k databázi ukládám do externího souboru *config.ini*, tento soubor načtu pomocí *Zend_Config_Ini*, a uložím do paměti, abych s tím mohl později pracovat.

Nastavil jsem používání layoutu přes knihovnu *Zend_Layout*, a dále upřesnil, v kterém adresáři se nacházejí soubory se šablonami. Využívání layoutu je velmi pohodlné. Umožňuje to pracovat s určitými metodami v kontroleru jako se samostatnými komponentami, které se navzájem nemusejí ovlivňovat.

Tak jako u konfigurace databáze, tak jsem uložil pravidla routování do samostatného souboru *routes.ini* a zpracoval jej stejným způsobem. Místo uložení do paměti jsem ho předal *Zend_Controller_Front*.

Dále jsem zaregistroval 5 pluginů:

- TimerPlugin – je to jen pomocný pagin, umožňuji mi sledovat rychlost vykonání celého požadavku klienta.
- DbConnectPlugin – v tomto paginu se připojím k databázi díky konfiguraci, kterou jsem si dříve uložil do paměti. Kromě klasického přihlášení zde definuji také typ databáze a znakovou sadu, v které bude PHP komunikovat s databází. V případě použití jiného databázového systému by mi stačilo změnit pouze typ databáze, a celá aplikace by bez problémů nebo jiných nutných zásahů dokázala existovat.
- ViewHelpersPlugin – zde pouze definuji cesty k adresářům pro view helpery.
- AclPlugin – v tomto pluginu načtu soubor *acl.xml*, ve kterém jsou uložena autorizační práva pro uživatele.
- Zend_Wildfire_Channel_HttpHeaders::getInstance – aktivuji mi možnost posílat v hlavičce data pro prohlížeč. Význam vysvětlím dále.

Aktivuji *Zend_Log_Writer_Firebug*. Tato knihovna mi za pomoci posledního pluginu umožní zasílat do prohlížeče data. V prohlížeči je nutní mít nainstalované rozšíření FireBug. Nyní si mohu poslat data jako je nějaký řetězec, pole hodnot nebo dokonce i tabulku. Díky tomuto rozšíření si také posílám informace o rychlosti zpracování požadavku. Tento požadavek je poslán pokaždé, takže běžného uživatele neobtěžuje, ale já si ho mohu zobrazit kdykoliv a mít představu o tom, zda se někde nevyskytuje úzké hrdlo v rámci výkonu aplikace.

Nakonec vyvolám metodu dispatch na *Zend_Controller_Front*.

Controller

Po vykonání pluginů je na řadě kontroler. Původní abstraktní *Zend_Controller_Action* jsem nahradil vlastním (i když dědí jeho vlastnosti), kde jsem nadefinoval několik proměnných pro view, v kterých jsou uvedeny cesty k adresářům s obrazkáma, kaskádovýma stylama, javascriptama a ostatníma souborama. Tímto je nemusím definovat pokaždé v každém kontroleru znova a ušetřím zápis kódu.

Před každou načtenou akcí proběhne kontrola, zda je uživatel přihlášený a také se prověří, jestli má na danou akci práva. Pokud není přihlášen, bude přesměrován na přihlašovací stránku a v případě nepovolení práv mu bude zobrazeno zamítnutí přístupu.

Uvnitř akce se řídím striktně dle pravidel, které definuje Zend Framework. Mohl bych sem vypsat různé akce, ale nebyly by to žádné jiné skripty, než které se objevují v každém jiném zpracování objektově orientovaného PHP kódu. Případné specifikace knihoven Zend Frameworku a jejich metod je možné si dočíst v oficiálním manuálu na stránkách firmy Zend.

4.4 Vlastní knihovny

Níže představím pár knihoven, které jsem vytvořil, a ulehčují mi další práci.

Kvik_Image

Tato knihovna nám pomáhá pracovat s obrázky, přesněji řečeno zajišťuje uložení obrázku do požadovaného adresáře.

Při použití této třídy na začátku nadefinujeme potřebné parametry. Nejprve zvolíme maximální možnou velikost, kterou může soubor mít, abychom omezili používání s velkými obrázky. Dále nadefinujeme cestu k adresáři kam, se má obrázek uložit a název obrázku. Pokud potřebujeme z obrázku udělat více obrázku, ale třeba jen s rozdílnými rozměry, nemusíme tuto třídu používat několikrát, ale stačí při zadávání parametrů určit, kolik má být verzí obrázků a nadefinovat jejich parametry. Každý obrázek může byt uložen v jiném adresáři, mít jiné rozměry nebo mít jiné rozměry.

Kvik_Db_Table

Rodičem této třídy je *Zend_Db_Table*, takže dědí od ní veškerý vlastnosti. Já její funkce rozšířil o další dvě metody, a to *fetchAllWithJoins* a *fetchRowWithJoins*. Obě tyto metody slouží pro výběr z databáze, a to buď několik řádků anebo jednoho řádku. Rozdíl oproti klasickému výběru je v tom, že výsledek nebude pouze řádek(y) z dané tabulky, ale rozšířený dotaz o všechny spojené tabulky pomocí *JOIN*. Definování

spojení tabulek nemusíme dělat pokaždé, co využíváme tuto funkci, ale musí to být nadefinováno v třídě pro danou tabulku, kde přesně uvedeme jaké tabulky jsou s ní spojené spolu s klíči, kterými se na sebe vážou. *Zend_Db_Table* bohužel nenabízí takovou pohodlnou práci se spojenými tabulkami, tak bylo nejlepší, si takovou metodu sám vytvořit.

Kvik_ErrorHandler

O této třídě jsem se zmínil již v bootstrapu. Zajišťuje vypisování chybových hlášek.

Původní výpisy chybových hlášek mohou být někdy matoucí (především pro začátečníky) a těžko se dohledává, kde je chyba. Já jsem celou chybovou hlášku počeštil a uvedl v takovém pořadí, aby to bylo co nejlépe srozumitelné. Hláška je tedy ve formátu *TYP ZPRÝVY + POPIS CHYBY + „na řádku“ + ČÍSLO ŘÁDKY + „v souboru“ + NÁZEV SOUBORU*. Dále to mám ještě nastavené tak, aby se hlášky neobjevovaly na stránce, ale posílá se mi to přes hlavičky do prohlížeče, kde si to přečtu pomocí rozšíření FireBug. Typ zprávy vyhodnocuji dle čísla chybové hlášky, kterou mi předá PHP funkce *set_error_handler*.

Kvik_Search

V českém jazyce je problém s vyhledáváním, především kvůli diakritice. Na vyhledávání se dá standardně využít fulltextové vyhledávání v MySQL, bohužel nastává zde problém, pokud uživatel zapíše vyhledávané slovo bez kritiky, nebo je dané slovo uložené v databázi s jinou velikostí počátečního písmene.

Při použití mé funkce je třeba nejprve zadat 3 parametry: seznam sloupců pro vyhledávání, výběr pro vyhledávání, minimální úroveň relevance. Seznam sloupců je klasické pole se seznam názvů sloupců, v kterých se má daná vyhledávaná fráze hledat. U každého sloupce můžeme také uvést bodovou hodnotu, která se připočte, pokud zde bude dané slovo nalezeno a počítá se to za každý výskyt slova zvlášť. Můžeme tímto dát větší relevanci na výskyt v nadpisu článku než v jeho obsahu. Výběr pro vyhledávání je výpis s databáze z dané tabulky nebo spojením tabulek, kterých se má vyhledávání týkat. Minimální úroveň relevance nám zajistí, aby se zobrazily pouze ty výsledky,

které mají bodové hodnocení relevance větší než dané minimum. Problém v podobě velikosti písmen nebo diakritiky jsem vyřešil tím, že vyhledávané slovo i každý text, v kterém se vyhledává, převedu na malá písmena a odstraním veškerou diakritiku. Výsledkem tohoto vyhledávání je pole záznamů, které odpovídají danému minimu relevance a jsou seřazeny podle velikosti relevance. Tento způsob vyhledávání se v praxi velmi osvědčil. Bohužel předpokládám, že při obsáhlejší databázi, kdy by tabulky skýtaly tisíce záznamů, mohly nastat problémy s rychlostí výpočtu tohoto vyhledávání.

4.5 Přínos vlastního řešení

Hlavní výhodou bych uvedl přesnou znalost celého řešení. U cizích řešení bych mohl narazit na problémy, kdy bych nevěděl, kde to programátor řešil. Zde, když jsem celý systém vytvářel od začátku sám, znám každou drobnost aplikace. Je to také velká výhoda při vytváření rozšíření.

Celou aplikaci jsem vytvořil na vlastním návrhu layoutu a grafického řešení. Snažil jsem se, aby to bylo pro klienty co nejvíce přívětivé a intuitivní ovládání.

Při vlastním vytváření jsem mohl po celou dobu vývoje sledovat výkonovou náročnost aplikace. Umožnilo mi to vyhnout se chybám, které by aplikaci zbytečně zpomalovaly.

Se Zend Frameworkem jsem velmi spokojený. Velmi dobře se s ním pracuje také díky dobré dokumentaci. Nemusel jsem se starat o aplikační logiku aplikace, stačilo si osvojit postupy Zend Frameworku a veškerý vývoj šel velmi hladce. Pravda je, že pro začátečníky mlže být dosti složitý, ale jakmile si na něj zvyknout, zjistí, že práce jim jde mnohem rychleji a následné modifikace jde provádět mnohem lépe.

Závěr

Tato práce pojednává o rozhodování při vývoji aplikace pro internetové obchody.
Nejdříve je probrána problematika výběru programovacího jazyka, kde se rozhodovalo mezi ASP.NET a PHP. Vybráno bylo PHP nejen kvůli velké komunitě, ale hlavně díky nízkým nákladům potřebné na vývoj a provoz.
Dalším základním kamenem byly databázové systémy. Až na výjimku MS SQL patřily všechny databázové systémy mezi multiplatformní. V případě změny operačního systému by nám nevznikly problémy a mohli bychom zůstat u stávajícího databázového systému. Vynikající databázový systém Oracle jsme bohužel museli zamítnout, jelikož pořizovací a provozní náklady by byly tak velké, že by si je žádná menší firma nemohla dovolit.
Po výběru PHP a MySQL jsme probrali nejrozšířenější open-source řešení pro e-shopy. Bohužel žádné řešení nevyhovovalo, zejména díky velkým nárokům na výkon serveru.
Předposlední kapitola pojednávala o možnostech frameworku pro PHP. Rozhodování v této části je čistě subjektivní, a určitě v každém uvedném frameworku by šla aplikace pro e-shopy vytvořit. Přiklonil jsem se na stranu Zend Framework, kdy jeho vývoj je oproti ostatním velmi rychlý a nabízí snad největší komunitu. Myslím, že Zend Framework má do budoucna největší potenciál.
Samotný vývoj aplikace přinesl návrh databázového schématu, který jsme vytvořili na základě specifikování požadavků systému. Napojení na databázi díky Zend Framework bylo již jednoduché a zabýval jsem se pouze specifickou konfigurací a použití pluginů. Bylo potřeba vytvořit několik vlastních tříd, které mi v balíčku Zend Framework chyběly. Vlastní třídy mi velmi pomohly při práci s databází nebo obrázky, ušetřilo mi to spoustu času.
Myslím, že se mi finálně podařila vytvořit úspěšná aplikace, která vyhovuje vše požadavkům a navíc je na výkon dost rychlá, tedy nenáročná na zátěž serveru. V případě vývoje dalších aplikací se určitě zase obrátím na kombinaci Zend Framework s MySQL.

Seznam obrázků

Literatura

[1]. KOSEK, Jiří. *PHP – Tvorba interaktivních internetových aplikací*. Praha: Grada Publishing, 1998. 492 s. ISBN 80-7169-373-1

[2]. SCHLOSSNAGLE, George. *Pokročilé programování v PHP 5*. Brno: Toner Press, 2004. 640 s. ISBN 80-86815-14-5

[3]. MASLAKOWSKI, Mark. *Naučte se MySQL za 21 dní*. Brno: Computer Press, 2001. 478 s. ISBN 80-7226-448-6

[4]. ALLEN, Rob, LO, Nick, BROWN Steven. *Zend Framework in Action*. Manning Publications, 2008. 432s. ISBN 1933988320

[5]. Databázový svět. [online] Dostupné z: http:// www.dbsvet.cz

[6]. Wikipedia. [online] Dostupné z: http:// www.wikipedia.org

[7]. Czech PHP User Group. [online] Dostupné z: http:// blog.php-group.cz

Seznam přílohy

Příloha 1

index.php

```php
<?

ini_set('display_errors', true);
date_default_timezone_set('Europe/Prague');

define('ROOT_DIR', dirname(dirname(__FILE__)));

set_include_path('.' . PATH_SEPARATOR . ROOT_DIR
                    . PATH_SEPARATOR . './library/'
                    . PATH_SEPARATOR . './application/plugins/'
                    . PATH_SEPARATOR . './application/models/'
                    . PATH_SEPARATOR . get_include_path());

require_once('Zend/Loader.php');
Zend_Loader::registerAutoload();

// nastaveni ErrorHandleru
set_error_handler(array(new Kvik_ErrorHandler(), 'add'));

// nacteni starych knihoven
require ("constants.php");

// load configuration
$config = new Zend_Config_Ini('./application/config.ini', 'general');
$registry = Zend_Registry::getInstance();
$registry->set('config', $config);

$locale = new Zend_Locale('cs_CZ');
Zend_Registry::set("Zend_Locale", $locale);
Zend_Locale::setDefault('cs_CZ');

// setup layout
Zend_Layout::startMvc(array('layoutPath' => './application/layouts'));

// setup controller
$frontController = Zend_Controller_Front::getInstance();
$frontController->throwExceptions(true);
$frontController->setControllerDirectory(
    array(
        'default'   => './application/default/controllers',
        'admin'     => './application/admin/controllers'
    )
);

// setup router
$router = $frontController->getRouter();
$config_routes = new Zend_Config_Ini('./application/routes.ini', 'general');
$router->addConfig($config_routes, 'routes');

// setup database
$frontController->registerPlugin(new TimerPlugin());
$frontController->registerPlugin(new DbConnectPlugin($config->db->toArray()));
$frontController->registerPlugin(new ViewHelpersPlugin());
$frontController->registerPlugin(new AclPlugin());
$frontController->registerPlugin(Zend_Wildfire_Channel_HttpHeaders::getInstance());

// run!
$response = new Zend_Controller_Response_Http();
$response->setRawHeader("Expires: " . gmdate("D, d M Y H:i:s") . " GMT");
$response->setRawHeader("Last-Modified: " . gmdate("D, d M Y H:i:s") . " GMT");
$response->setRawHeader("Cache-Control: no-store, no-cache, must-revalidate");
$response->setRawHeader("Pragma: no-cache");

// firebug init
$log = new Zend_Log();
$writer = new Zend_Log_Writer_Firebug();
//$writer = new Zend_Log_Writer_Null();
$log->addWriter($writer);
Zend_Registry::set('log',$log);

$frontController->dispatch();
```

Image.php

```php
<?

class Kvik_Image extends Kvik_Image_Abstract
{
    protected $_path;
    protected $_maxWidth;
    protected $_maxHeight;
    protected $_prefix;

    protected $_id;
    protected $_name;

    function __construct($id = 0)
    {
        $this->_id = $id;
    }

    public function create($image, array $parameters = array(), $size = false)
    {
        $result = array();

        if ($size) $this->_maxSize = $size;

        if(filesize($image) < $this->_maxSize)
        {
            $velikost = getimagesize($image);

            $fileExtension = $this->_getFileExtension($velikost["mime"]);
            if ($fileExtension == false) return false;

            if ($fileExtension != "swf")
            {
                $img = $this->_imageCreateFrom($image);
                if ($img == false) return false;

                $width = imagesx($img);
                $height = imagesy($img);
            }
            else
            {
                $width = $velikost[0];
                $height = $velikost[1];
            }

        }
        else return false;

        $i = 0;
        foreach ($parameters as $par)
        {
            if (isset($par["id"])) $this->_id = $par["id"];

            if (isset($par["size"]))
            {
                $this->_maxWidth = $par["size"]["width"];
                $this->_maxHeight = $par["size"]["height"];
            }

            if (isset($par["path"])) $this->_path = $par["path"];

            if (isset($par["prefix"])) $this->_prefix = $par["prefix"];

            if (!$this->_name)
            {
                $dir = dir("." . $this->_path);
                for($x = "a"; $x <= "z"; $x++)
                {
                    $dir->rewind();
                    while($file = $dir->read())
                    {
                        $break = false;
```

```php
                        if ($file == "." OR $file == "..") continue;
                        if (!eregi("^" . $this->_id . "-", $file)) continue;

                        if ($file == $this->_id . "-" . $x . "." . $fileExtension)
                        {
                            $break = true;
                            break;
                        }
                    }

                    if ($break == false)
                    {
                        $this->_name = $this->_id . "-" . $x . "." . $fileExtension;
                        break;
                    }
                }
                $dir->close();
            }

            $result[$i]["name"] = $this->_prefix . $this->_name;

            $path = "." . $this->_path . $this->_prefix . $this->_name;

            if ($width > $this->_maxWidth OR $height > $this->_maxHeight)
            {
                if ($width >= $height AND ($width / $height) >= ($this->_maxWidth / $this->_maxHeight))
                {
                    $newWidth   = $this->_maxWidth;
                    $percent    = $newWidth / $width;
                    $newHeight  = round($height * $percent);
                }
                elseif ($height > $width OR ($width / $height) < ($this->_maxWidth / $this->_maxHeight))
                {
                    $newHeight  = $this->_maxHeight;
                    $percent    = $newHeight / $height;
                    $newWidth   = round($width * $percent);
                }

                if ($fileExtension != "swf")
                {
                    $img2 = imagecreatetruecolor($newWidth, $newHeight);
                    imagecopyresampled($img2, $img, 0, 0, 0, 0, $newWidth, $newHeight, $width, $height);

                    if($velikost["mime"] == "image/jpeg")
                    {
                        $tmp_result = imagejpeg($img2, $path);
                    }
                    elseif ($velikost["mime"] == "image/gif")
                    {
                        $tmp_result = imagegif($img2, $path);
                    }
                    elseif ($velikost["mime"] == "image/png")
                    {
                        $tmp_result = imagepng($img2, $path);
                    }
                    else
                    {
                        return false;
                    }

                    imagedestroy($img2);
                }
                else
                {
                    $newWidth = $width;
                    $newHeight = $height;

```

```php
                    $tmp_result = copy($image, $path);
                }
            }
            else
            {
                $newWidth = $width;
                $newHeight = $height;

                $tmp_result = copy($image, $path);
            }

            $result[$i]["width"] = $newWidth;
            $result[$i]["height"] = $newHeight;
            $result[$i]["size"] = filesize($path);

            chmod ($path, 0644);

            $i++;
        }

        if ($img) imagedestroy($img);

        return $result;
    }

    public function delete($image, array $parameters = array())
    {
        foreach ($parameters as $par)
        {
            if (isset($par["path"])) $this->_path = $par["path"];

            if (isset($par["prefix"])) $this->_prefix = $par["prefix"];

            unlink("." . $this->_path . $this->_prefix . $image);
        }

        return true;
    }
}
```

Table.php

```php
<?

class Kvik_Db_Table extends Zend_Db_Table_Abstract
{

    const TABLEALIAS         = 'tableAlias';

    protected $_tableAlias = null;


    public function info()
    {
        $info = parent::info();

        $info[self::TABLEALIAS] = $this->_tableAlias;

        return $info;
    }

    public function fetchAllWithJoins($where = null, $order = null, $count = null, $offset =
    {
        $select = $this->_db->select();

        // prejmenovani sloupcu rodicovske tabulky
        $colsAliases = array();
        $cols = $this->info();
        foreach($cols[self::METADATA] as $col)
        {
            $colsAliases[$this->_tableAlias . "_" . $col["COLUMN_NAME"]] = $col["COLUMN_NAME
        }

        // nastaveni vyberu z databaze
        $select ->from(array($this->_tableAlias => $this->_name), $colsAliases)
                ->order($order)
                ->limit($count, $offset);

        if (!is_null($where))
        {
            $select->where($where);
        }

        if (!is_null($group))
        {
            $select->group($group);
        }

        // pripojeni dcerinych tabulek

        if(count($this->_referenceMap) > 0)
        {
            foreach($this->_referenceMap as $referenceMap)
            {
                $dependentTable = new $referenceMap["refTableClass"]();

                $dependentTableInfo = $dependentTable->info();

                // urceni aliasu dcerine tabulky
                $table = array($dependentTableInfo[self::TABLEALIAS] => $dependentTableInfo[

                // urceni relacnich sloupcu
                $columns = null;
                for($i = 0; $i < count($referenceMap[self::REF_COLUMNS]); $i++)
                {
                    $columns .= $this->_tableAlias . "." . $referenceMap[self::REF_COLUMNS][
                    if ($i + 1 < count($referenceMap[self::REF_COLUMNS]))
                    {
                        $columns .= " AND ";
                    }
                }

                // prejmenovani sloupcu dcerine tabulky
```

```
67                      $columns .= " AND ";
68                  }
69              }
70
71              // prejmenovani sloupcu dcerine tabulky
72              $colsAliases = array();
73              foreach($dependentTableInfo[self::METADATA] as $col)
74              {
75                  $colsAliases[$dependentTableInfo[self::TABLEALIAS] . "_" .
    $col["COLUMN_NAME"]] = $col["COLUMN_NAME"];
76              }
77
78              // pripojeni relacni tabulky k vyberu
79              $select->joinLeft($table, $columns, $colsAliases);
80          }
81      }
82
83      $stmt = $select->query();
84
85      $data = array(
86
87          'table'     => $this,
88          'data'      => $stmt->fetchAll(),
89          'rowClass'  => $this->_rowClass,
90          'stored'    => true
91
92      );
93
94      return new $this->_rowsetClass($data);
95  }
96
97  public function fetchRowWithJoins($where = null, $order = null, $count = null, $offset
    = null)
98  {
99
100     $rows = $this->fetchAllWithJoins($where, $order, 1);
101
102     foreach($rows as $row)
103     {
104         return $row;
105         break;
106     }
107 }
108
109 public function getMaxPrimaryKey ($where = null)
110 {
111     $select = $this->_db->select();
112
113     $select->from($this->_name, array('max' => 'MAX(' . $this->_primary[1] . ')'));
114
115     if (!is_null($where))
116     {
117         $select->where($where);
118     }
119
120     return $select->query()->fetchObject()->max;
121 }
122
123 public function CountAllWithJoins ($group = null, $where = null)
124 {
125     $select = $this->_db->select();
126
127     // nastaveni vyberu z databaze
128     $select->from(array($this->_tableAlias => $this->_name), array('count' =>
    'COUNT(*)'));
129
130     if (!is_null($group))
131     {
132         $select->reset();
133         $select->from(array($this->_tableAlias => $this->_name), array('count' =>
    'COUNT(' . $group . ')'));
```

```php
            $select->group($group);
        }

        if (!is_null($where))
        {
            $select->where($where);
        }

        // pripojeni dcerinych tabulek
        if(count($this->_referenceMap) > 0)
        {
            foreach($this->_referenceMap as $referenceMap)
            {
                $dependentTable = new $referenceMap["refTableClass"]();

                $dependentTableInfo = $dependentTable->info();

                // urceni aliasu dcerine tabulky
                $table = array($dependentTableInfo[self::TABLEALIAS] =>
$dependentTableInfo[self::NAME]);

                // urceni relacnich sloupcu
                $columns = null;
                for($i = 0; $i < count($referenceMap[self::REF_COLUMNS]); $i++)
                {
                    $columns .= $this->_tableAlias . "." .
$referenceMap[self::REF_COLUMNS][$i] . " = " . $dependentTableInfo[self::TABLEALIAS] . "."
. $referenceMap[self::COLUMNS][$i];
                    if ($i + 1 < count($referenceMap[self::REF_COLUMNS]))
                    {
                        $columns .= " AND ";
                    }
                }

                // pripojeni relacni tabulky k vyberu
                $select->joinLeft($table, $columns, array());
            }
        }

        if (!is_null($group))
        {
            return count($select->query()->fetchAll());
        }
        else
        {
            return $select->query()->fetchObject()->count;
        }
    }

    public function getReferenceMap()
    {
        return $this->_referenceMap;
    }

}
```

ErrorHandler.php

```php
<?

class Kvik_ErrorHandler
{
    protected $_log;

    function __construct()
    {
        $log = new Zend_Log();
        $writer = new Zend_Log_Writer_Firebug();
        //$writer = new Zend_Log_Writer_Null();
        $log->addWriter($writer);
        $this->_log = $log;
    }

    function add($cislo, $zprava, $soubor, $radka)
    {
        if (!eregi("Loader.php$", $soubor))
        {
            $message = $zprava . " na řádku " . $radka . " v souboru " . $soubor;

            switch ($cislo)
            {
                case 1:
                    $this->_log->err("Error: " . $message);
                    break;
                case 2:
                    $this->_log->warn("Warning: " . $message);
                    break;
                case 4:
                    $this->_log->warn("Parsing Error: " . $message);
                    break;
                case 8:
                    $this->_log->notice("Notice: " . $message);
                    break;
                case 16:
                    $this->_log->err("Core Error: " . $message);
                    break;
                case 32:
                    $this->_log->warn("Core Warning: " . $message);
                    break;
                case 64:
                    $this->_log->err("Compile Error: " . $message);
                    break;
                case 128:
                    $this->_log->warn("Compile Warning: " . $message);
                    break;
                case 256:
                    $this->_log->err("User Error: " . $message);
                    break;
                case 512:
                    $this->_log->warn("User Warning: " . $message);
                    break;
                case 1024:
                    $this->_log->notice("User Notice: " . $message);
                    break;
                case 2048:
                    $this->_log->notice("Runtime Notice: " . $message);
                    break;
                case 4096:
                    $this->_log->err("Catchable Fatal Error: " . $message);
                    break;

                default:
                    $this->_log->info("Other: " . $message);
                    break;
            }
        }
    }
}

```

Search.php

```php
<?

class Kvik_Search
{
    protected $_columns;
    protected $_rowset;
    protected $_rankLimit = 0;
    protected $_result;

    function __construct($params = array())
    {
        if (isset($params["columns"])) $this->_columns = $params["columns"];
        if (isset($params["rowset"])) $this->_rowset = $params["rowset"];
        if (isset($params["rankLimit"])) $this->_rankLimit = $params["rankLimit"];
    }

    function search($search)
    {
        $searchPartial = explode(" ", $this->_convert($search));

        foreach ($this->_rowset as $row)
        {
            $relevance = 0;
            foreach ($this->_columns as $col => $rel)
            {
                foreach ($searchPartial as $searchPart)
                {
                    $relevance = $relevance + ($this->_compare($searchPart, $this->_convert(
                }
            }
            if ($relevance > $this->_rankLimit)
            {
                $subresult = $row->toArray();
                $subresult['_relevance'] = $relevance;
                $this->result[] = $subresult;
            }
        }
    }

    function setRowset($rowset)
    {
        if (isset($rowset)) $this->_rowset = $rowset;
    }

    function setColumns($columns)
    {
        if (isset($columns)) $this->_columns = $columns;
    }

    public function getResult()
    {
        $this->result = array_reverse($this->_columnSort($this->result, '_relevance'));

        $tmp_result = array();
        foreach ($this->result as $row)
        {
            $tmp_result[] = (object)$row;
        }
        return $tmp_result;
    }

    private function _columnSort($unsorted, $column) {
        $sorted = $unsorted;
        for ($i = 0; $i < count($sorted) - 1; $i++)
        {
            for ($j = 0; $j < count($sorted) - 1 - $i; $j++)
            {
                if ($sorted[$j][$column] > $sorted[$j + 1][$column])
                {
                    $tmp = $sorted[$j];
                    $sorted[$j] = $sorted[$j + 1];
```

```
                    $sorted[$j] = $sorted[$j + 1];
                    $sorted[$j + 1] = $tmp;
                }
            }
        }
        return $sorted;
    }

    private function _convert($string)
    {
        $trans = array(
            "á"=>"a",
            "ä"=>"a",
            "č"=>"c",
            "ď"=>"d",
            "é"=>"e",
            "ě"=>"e",
            "ë"=>"e",
            "í"=>"i",
            "&#239;"=>"i",
            "ň"=>"n",
            "ó"=>"o",
            "ö"=>"o",
            "ř"=>"r",
            "š"=>"s",
            "ť"=>"t",
            "ú"=>"u",
            "ů"=>"u",
            "ü"=>"u",
            "ý"=>"y",
            "&#255;"=>"y",
            "ž"=>"z",
            "Á"=>"A",
            "Ä"=>"A",
            "Č"=>"C",
            "Ď"=>"D",
            "É"=>"E",
            "Ě"=>"E",
            "Ë"=>"E",
            "Í"=>"I",
            "&#207;"=>"I",
            "Ň"=>"N",
            "Ó"=>"O",
            "Ö"=>"O",
            "Ř"=>"R",
            "Š"=>"S",
            "Ť"=>"T",
            "Ú"=>"U",
            "Ů"=>"U",
            "Ü"=>"U",
            "Ý"=>"Y",
            "&#376;"=>"Y",
            "Ž"=>"Z"
        );

        return strtolower(strtr($string, $trans));
    }

    private function _compare($searchPart, $value)
    {
        if (strLen($searchPart) >= 2) {
            return subStr_count($this->_convert($value), $searchPart);
        }
        else {
            return 0;
        }
    }
}
```