Michael Peacock

Programujeme vlastní e-shop v PHP 5

Průvodce tvorbou flexibilního MVC frameworku

Studie, návrh a implementace e-shopu a jeho funkcí

Snadné přizpůsobení na jakýkoliv e-shop

WWW

Ke stažení zdrojové kódy příkladů z knihy

CPRESS

Michael Peacock

Programujeme vlastní e-shop v PHP 5

Computer Press, a. s.
Brno
2011

Programujeme vlastní e-shop v PHP 5

Michael Peacock

Computer Press, a.s., 2011. Vydání první.

Překlad: Ondřej Gibl
Jazyková korektura: Zdeněk Dan
Vnitřní úprava: Ctibor Foltýn
Sazba: Ctibor Foltýn
Rejstřík: Daniel Štreit
Obálka: Martin Sodomka

Komentář na zadní straně obálky: Lukáš Krejčí
Technická spolupráce: Jiří Matoušek, Zuzana Šindlerová, Dagmar Hajdajová
Odpovědný redaktor: Lukáš Krejčí
Technický redaktor: Jiří Matoušek
Produkce: Petr Baláš

Computer Press, a.s.,
Holandská 3, 639 00 Brno

Objednávky knih:
http://knihy.cpress.cz
distribuce@cpress.cz
tel.: 800 555 513

ISBN 978-80-251-3181-7
Prodejní kód: K1845
Vydalo nakladatelství Computer Press, a.s., jako svou 3723. publikaci.

Stručný obsah

Obsah

Úvod

Nakupování na internetu získalo v posledních letech výrazně na popularitě. Nemáte-li v plánu vytvářet si vlastní elektronický obchod, můžete vybírat z mnoha již existujících řešení. Někdy se však vyplatí sáhnout po vlastním řešení. Může být snadné najít systém pro elektronické obchodování, když ale přijde řeč na jeho rozšiřitelnost a použitelnost, může se objevit řada problémů.

Tato kniha vám ukáže, jak si vytvořit vlastní framework v jazyce PHP, který je možné snadno rozšířit a použít zejména pro elektronické obchody. S použitím tohoto frameworku budete moci zobrazovat a spravovat produkty, upravovat je, vytvářet seznamy přání (tzv. wish list), doporučovat zákazníkům produkty na základě jejich předchozích objednávek, zasílat upozornění e-mailem na dostupnost určitých produktů, hodnotit produkty online a mnoho dalšího.

Tato kniha vám pomůže vytvořit framework vycházející z architektury MVC (Model-View-Controller), který se následně použije jako základ pro elektronický obchod. Framework umožňuje správu šablon, správu databáze a správu uživatelů. Po dokončení základní funkcionality se budou postupně přidávat funkce zaměřené na elektronické obchodování, jako jsou produkty, kategorie, přizpůsobitelné produktu s různými obměnami, uživatelský vstup, seznamy přání, doporučení, nákupní košík a kompletní proces objednávky.

Na konci této knihy budete mít v rukou architekturu pro elektronické obchodování, která zajistí vše od prohlížení a vyhledávání produktů a jejich přidávání do nákupního košíku, přes proces vyřízení objednávky a její zaplacení až po samotné odeslání zboží. Mimořádná pozornost se věnuje flexibilitě, tak aby bylo možné framework rozšířit podle aktuálních potřeb obchodu, jak také ilustruje jedna z příloh, která popisuje postup úpravy obchodu za účelem prodeje nejenom fyzických produktů, ale také těch, které je možné přímo stahovat z internetu.

Nechybí ani další informace, jako například jak online obchod prosadit a propagovat a jak provádět jeho pravidelné zálohování a údržbu tak, aby šance na úspěch vašeho obchodu, založeného na vašem vlastním frameworku, byla maximální.

Struktura knihy

Kapitola 1, *Elektronické obchodování a PHP*, se zabývá vzrůstající potřebou a využitím elektronického obchodování, včetně několika populárních online obchodů, a uvádí, co budeme v průběhu knihy dělat a proč.

Kapitola 2, *Plánování frameworku*, vás v průběhu vývoje struktury a klíčové funkcionality našeho frameworku, včetně správy šablon, správy databáze a autentizace uživatelů, seznámí s několika klíčovými návrhovými vzory, včetně vzorů MVC, registr a singleton.

Kapitola 3, *Produkty a kategorie*, jde o krok dál a ukazuje, jak s použitím našeho frameworku zobrazit a rozdělit pro zákazníky produkty podle kategorií.

Kapitola 4, *Varianty produktů a nahrávání uživatelských souborů na server*, zdokonaluje standardní výpisy produktů přidáním přizpůsobitelných produktů, variant produktů a možnosti nahrávat uživatelské soubory s objednávkami na server.

Kapitola 5, *Větší uživatelská přívětivost*, uvádí tipy a triky pro dosažení vyšší uživatelské přívětivosti a zaměřuje se na vyhledávání, filtrování produktů, seznamy přání, zasílání upozornění e-mailem a další užitečná vylepšení pro zákazníky.

Kapitola 6, *Nákupní košík*, ukazuje, jak strukturovat, vytvořit a spravovat nákupní košík s podporou jak standardních tak přizpůsobitelných produktů.

Kapitola 7, *Objednávka a její zaplacení*, se zaměřuje na proces objednávky a její platby používaný některými populárními elektronickými obchody a popisuje jejich pro a proti ve snaze nastínit postup použitý v našem frameworku.

Kapitola 8, *Doprava a daňové zatížení*, se zaměřuje na výpočet nákladů na dopravu v závislosti na zvolené formě přepravy, za tímto účelem také integruje rozhraní API třetí strany, a dále přidává možnost zasílání informací o stavu objednávek a jejich sledování a do systému integruje výpočet daňového zatížení.

Kapitola 9, *Slevy, poukazy a doporučení*, se zaměřuje na rozšíření frameworku za účelem oslovení nových zákazníků a dosažení vyššího počtu objednávek pomocí slevových kupónů, prodejních poukázek a slev na základě doporučení.

Kapitola 10, *Zaplacení*, spojuje vše dohromady (vzhledem k tomu, že je většina funkcionality pro realizaci plateb už na svém místě) a rozšiřuje proces objednávky o potvrzení objednávky připravené k zaplacení.

Kapitola 11, *Příjem plateb za objednávky*, představuje zpracování plateb a popisuje různé typy plateb a s nimi spojené kroky.

Kapitola 12, *Funkce uživatelského účtu*, krok za krokem popisuje vývoj oblasti pro zákazníky, kde mohou vidět a také upravovat své objednávky a osobní informace.

Kapitola 13, *Administrace*, prochází vývojem administrační oblasti, kde mohou administrátoři sledovat objednávky, produkty a nastavení a všechny výše uvedené přidávat, editovat a odstraňovat.

Kapitola 14, *Nasazení, zabezpečení a údržba*, se zabývá nasazením webu, založeného na našem frameworku, do produkčního prostředí a zkoumá aspekty zabezpečení a údržby. Popisuje několik různých způsobů, jak zvýšit bezpečnost našeho frameworku a jak obnovit web ze zálohy.

Kapitola 15, *Marketing, SEO a udržení zákazníků*, uvádí tipy a triky pro efektivní marketing a prosazení webů a elektronických obchodů založené na online marketingových technikách, optimalizaci pro vyhledávače a strategiích pro udržení zákazníků.

Příloha A, *Interakce s webovými službami*, vysvětluje, jak lze komunikovat s jinými webovými službami pro elektronické obchodování, jako jsou např. Google Products, Google Analytics, webové služby serverů Amazon a eBay, za účelem oslovení nových trhů a také usnadnění našich úkolů.

Příloha B, *Stahovatelné produkty*, ilustruje, jak rozšířit váš obchod o produkty, které se přímo stahují z internetu.

Příloha C, *Kuchařka*, uvádí několik užitečných ukázek kódu zdokonalujícího náš framework i výsledný elektronický obchod.

Co budete potřebovat

V průběhu této knihy budete potřebovat následující software, abyste mohli vyzkoušet nejrůznější ukázky kódu:

- Webový server Apache 1.3 nebo novější (doporučuje se verze 2)
- Modul `mod_rewrite` na serveru Apache
- MySQL 5.0
- PHP 5.0 (doporučuje se verze 5.2+)

Výše uvedený software je možné nainstalovat pomocí balíku, jako je např. WampServer 2.0 pro Windows.

Pro vývoj postačí pouze textový editor, pokud možno schopný zvýrazňování syntaxe.

Komu je kniha určena

Ať už jste webový vývojář nebo se jen snažíte prohloubit své znalosti v oblasti vývoje systémů pro elektronický obchod, je tato kniha právě pro vás. Primárně je cílena na vývojáře v PHP, je však vhodná pro každého webového vývojáře zajímajícího se o elektronické obchodování anebo pro vývojáře zvažujícího přechod k jazyku PHP.

Předpokládá se středně pokročilá znalost jazyka PHP a objektově orientovaného programování, výhodou jsou základní znalosti z oblasti elektronického obchodu.

Konvence

V této knize najdete řadu stylů textu, které rozlišují různé typy informací. Zde je několik příkladů s objasněním jejich významu.

Kód se v textu uvádí následovně: „Přípona `ecomframe` slouží pro rozlišení jednotlivých spojení s databází uložených ve stejném poli."

Blok kódu má následující podobu:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
  <title>{title}</title>
  <meta http-equiv="content-type"
    content="text/html;
    charset=iso-8859-1" />
  <meta name="description" content="{metadescription}" />
  <meta name="keywords" content="{metakeywords}" />
</head>
<body>
```

Na místech, kde chceme upoutat vaši pozornost k určité části kódu, jsou odpovídající řádky kódu zvýrazněné tučně:

```
SELECT v.name AS product_name, c.ID AS product_id,
  (SELECT GROUP_CONCAT( a.name,'--AV--', av.ID, '--AV--',
    av.name SEPARATOR '---ATTR---' )
  FROM product_attribute_values av,
    product_attribute_value_association ava,
    product_attributes a
  WHERE a.ID = av.attribute_id AND av.ID=ava.attribute_id
    AND ava.product_id=c.ID ORDER BY ava.order ) AS attributes,
    p.image AS product_image, p.stock AS product_stock,
    p.weight AS product_weight, p.price AS product_price,
    p.SKU AS product_sku, p.featured AS product_featured,
    v.heading AS product_heading,
    v.content AS product_description,
    v.metakeywords AS metakeywords,
    v.metarobots AS metarobots,
    v.metadescription AS metadescription
  FROM content_versions v, content c, content_types t,
    content_types_products p
  WHERE c.active=1 AND c.secure=0 AND c.type=t.ID
    AND t.reference='product' AND p.content_version=v.ID
    AND v.ID=c.current_revision AND c.path='{$productPath}'
```

Veškeré příkazy pro příkazový řádek jsou vyznačené takto:

```
Mysql -u username -p databasename < /home/junipert/backup.sql
```

Nové termíny a **důležitá slova** jsou zvýrazněná tučně. Texty, které můžete vidět na obrazovce (např. v nabídkách nebo dialogových oknech), jsou vyznačené takto: „Poté, co zadáte uživatelské jméno a heslo, stiskněte tlačítko **Další krok**.“

Upozornění

Varování a důležité poznámky jsou vyznačené takto.

Tip

Tipy a triky jsou vyznačené takto.

Zpětná vazba od čtenářů

Nakladatelství a vydavatelství Computer Press, které pro vás tuto knihu přeložilo, stojí o zpětnou vazbu a bude na vaše podněty a dotazy reagovat. Můžete se obrátit na následující adresy:

redakce PC literatury
Computer Press
Spielberk Office Centre
Holandská 8
639 00 Brno

nebo

sefredaktor.pc@cpress.cz

Další informace a případné opravy českého vydání knihy najdete na internetové adrese *http://knihy.cpress.cz/k1845*. Prostřednictvím uvedené adresy můžete též naší redakci zaslat komentář nebo dotaz týkající se knihy. Na vaše reakce se srdečně těšíme.

Zdrojové kódy ke knize

Z adresy *http://knihy.cpress.cz/k1845* si po klepnutí na odkaz Soubory ke stažení můžete přímo stáhnout archiv s ukázkovými kódy.

Errata

Přestože jsme udělali maximum pro to, abychom zajistili přesnost a správnost obsahu, chybám se úplně vyhnout nedá. Pokud v některé z našich knih najdete chybu, ať už chybu v textu nebo v kódu, budeme rádi, pokud nám ji nahlásíte. Ostatní uživatele tak můžete ušetřit frustrace a pomoci nám zlepšit následující vydání této knihy. Pokud si přejete zadat errata, učiňte tak na adrese *http://knihy.cpress.cz/k1845*, kde klepněte na odkaz Poslat komentář.

Veškerá existující errata zobrazíte na adrese *http://knihy.cpress.cz/k1845* po klepnutí na odkaz Errata.

Dotazy

Máte-li s knihou jakýkoli problém, kontaktujte nás pomocí formuláře na adrese *http://knihy.cpress.cz/k1845*, kde klepněte na odkaz Poslat komentář. Pokusíme se udělat vše, abychom vám ho pomohli vyřešit.

KAPITOLA 1

Elektronické obchodování a PHP

Vítejte na začátku cesty vedoucí k vytvoření frameworku v jazyce PHP pro elektronické obchodování. V průběhu této knihy vytvoříme s použitím jazyka PHP flexibilní framework pro elektronické obchodování, který je možné rozšířit a upravit pro potřeby jakéhokoli elektronického obchodu.

V této kapitole se seznámíme:

- S obchodním modelem stojícím za elektronickým obchodem
- Proč (a kdy) byste měli použít svůj vlastní systém namísto existujícího produktu
- S přednostmi frameworku
- S existujícími weby a produkty pro elektronické obchodování

Elektronické obchodování

Elektronické obchodování, či zkráceně e-obchod, představuje formu nákupu a prodeje zboží a služeb elektronickou cestou. V našem případě je onou elektronickou cestu internet. Na internetu existuje celá řada různých aplikací pro elektronické obchodování, včetně:

- Online obchodů prodávajících produkty, jako např. Amazon, nebo online protipólů ke kamenným obchodům.
- Online aukce, jako je eBay.
- Online služby/webové služby, jako je BaseCamp, nebo weby založené na předplatném.

Přehled elektronického obchodování

Elektronický obchod dnes představuje velmi populární způsob podnikání. Pojďme se tedy podívat, kdo elektronické obchodování používá a k čemu.

eBay

Web eBay uvádí, že má přibližně 84 milionů aktivních uživatelů, kteří každou sekundu obchodují se zbožím v hodnotě 1900 dolarů. To znamená, že 84 milionů z nás používá eBay pro nákup a prodej zboží, ať už jako pravidelný zdroj příjmu anebo ve snaze něco málo si přivydělat prodejem nechtěných nebo nepotřebných věcí povalujících se po domě.

eBay je elektronický obchod se sociálním aspektem, pracující na bázi online aukčního domu, který sám o sobě nic neprodává, nýbrž umožňuje komunitě uživatelům skrze web nejenom prodávat, ale také na něm nakupovat. To nejenom ilustruje popularitu elektronického obchodování, ale také, že se dají vydělat peníze zprostředkováním malého (i velkého) objemu online prodejů.

Amazon

Se ziskem více jak 19 miliard dolarů v roce 2008 se Amazon řadí mezi nejpopulárnější elektronické obchody na internetu. Výzkumy ze začátku roku 2009 ukázaly, že Amazon byl nejúspěšnějším prodejcem hudby a videa v Anglii.

Kamenné obchody

Velké a zaběhlé kamenné obchody, jako je Wal-Mart, Tesco a Borders, prodávají prostřednictvím internetu produkty, které zpravidla mají na prodejnách. V případě obchodů Wal-Mart a Tesco si zákazníci často určí dobu, kdy se mají jejich potraviny dodat. Tyto obchody nabízejí kvůli pohodlí svých zákazníků také zboží, které na prodejně nemají a mohou ho snadno obstarat. V případě online prodejů prodejce neomezuje to, kolik toho mohou vystavit v prodejně, ale kapacita jejich distribučních skladů.

Menší kamenné obchody používají online prodej jako způsob, jak oslovit svými produkty širší publikum, bez omezení v podobě fyzické přítomnosti produktů.

Společnosti poskytující služby

Společnosti, jako je 37signals, vytváří online aplikace (jako jsou nástroje Project Management a BaseCamp) na bázi měsíčního předplatného. Mezi další příklady takovýchto webů patří rozsáhlé weby pro distribuci souborů (dovolující „odesílat" velké soubory e-mailem s využitím webu třetí strany) a příplatkové funkce některých webů jako např. Get Satisfaction.

Proč elektronický obchod?

Online obchodování v posledních letech výrazně získalo na popularitě. Nejenom že umožňuje zákazníkům nakupovat v pohodlí svého domova, ale také společnostem prosadit se na globálním trhu a oslovit tak ještě více potencionálních zákazníků. Protože se vše odehrává elektronickou formou, mohou elektronické obchody pomoci vykazovat trvalý zisk vycházející z možnosti doporučovat zákazníkům nové produkty v závislosti na jejich předchozích objednávkách a také díky neustále dostupnému aktuálnímu katalogu produktů pro zákazníky.

Použití vlastního frameworku

V průběhu této knihy vytvoříme v jazyce PHP náš vlastní framework namísto použití již existujícího produktu. Někdy je vhodnější použít existující řešení, někdy je tomu právě naopak. Když tuto knihu čtete, pak nejspíše víte, proč chcete vytvořit svůj vlastní framework. Přesto se pojďme podívat na důvody, proč si vytvořit vlastní framework.

Proč PHP

PHP je velmi populárním jazykem, a protože sám o sobě není frameworkem, můžeme s jeho pomocí snadno vytvořit náš vlastní framework, tak jak potřebujeme. Volba programovacího jazyka je zpravidla otázkou osobních preferencí.

Většina moderních webových hostingů podporuje PHP a MySQL, a přestože i další jazyky, jako je např. Ruby on Rails získávají na popularitě, jejich podpora ze strany hostingů není ani zdaleka tak běžná. Tato kniha předpokládá, že už máte dostatečné základy jazyka PHP, což nejspíše bude také jeden z důvodů, proč chcete tento jazyk použít. Možná chcete jen něco rychle vyvinout anebo nechcete použít programovací jazyk či platformu, které vám nepřirostly k srdci.

Proč framework

Místo vývoje elektronického obchodu, navrženého tak, aby zvládal všechny typy úkolů spojených s elektronickým obchodováním, vytvoříme framework. To umožní s minimálním úsilím rozšířit systém pro potřeby jakéhokoli projektu elektronického obchodu. Protože vytváříme náš vlastní framework, bude to něco, co velmi dobře známe a rozumíme tomu. Bude tedy snadné ho používat a případně rozšiřovat.

Zatímco typický elektronický obchod zobrazuje produkty v režimu prohlížení a vyhledávání, framework může umožnit integraci produktů do dalších oblastí webu – např. určité kategorie produktů zobrazit na relevantních stránkách, což je obzvláště důležité pro web, který má plnit i jiné úlohy, než je online prodej. Pokud např. prodáváte knihy, můžete určité stránky věnovat konkrétním autorům, s informacemi, recenzemi a dalšími podklady o autorovi, a do těchto stránek zakomponovat některé z jejich populárních děl, které máte v nabídce.

Kdy použít existující řešení

Existují mnohé systémy pro elektronický obchod naprogramované v nejrůznějších jazycích, jejichž použití může být vhodnější:

- Je-li termín projektu napjatý a nemáte žádný framework.
- Pokud se projektu účastní mnoho vývojářů. Kvalitně zdokumentované řešení bude pravděpodobně užitečnější, přinejmenším zpočátku (pokud tedy za vývojem frameworku nestojí většina vývojářů).
- Když klient naznačil, kterému systému dává přednost.
- Má-li odpovídající funkčnost – pokud má jiný systém všechny funkce, které chcete a potřebujete, a pracuje způsobem, který vám vyhovuje, pak je jistě vhodnější použít tento stávající systém.

Současná nabídka

Z dostupných aplikací pro elektronické obchodování patří ty následující mezi nejpopulárnější:

- **Magento** – velmi modulární a flexibilní platforma pro elektronický obchod, získávající na popularitě.
- **Drupal** – populární, snadno rozšiřitelný a upravitelný systém CMS (Content Management System). K dispozici jsou dva balíky s názvy Ubercart a e-Commerce, které k tomuto známému systému CMS přidávají bezpočet funkcí pro elektronické obchodování.
- **CubeCart** – velmi snadno použitelné řešení pro elektronické obchodování, dostupné zdarma i v placené verzi.

Pohled na elektronické obchody

V rychlosti jsme se už podívali, kdo používá elektronické obchodování. Nyní se trochu detailněji podívejme na některé z populárních webových obchodů a na to, jak pracují a jaké funkce poskytují svým uživatelům.

iStockphoto

iStockphoto je populární web umožňující nákup a prodej fotografií. Fotografové se mohou na tomto webu registrovat a posílat své fotografie ke schválení. Schválené fotografie mohou zákaz-

níci nakupovat za kredity, jejichž množství se odvíjí od velikosti fotografie a licence, kterou zákazník požaduje.

Charakteristické rysy

- **Proces schválení pro prodejce** – než je možné fotografie zakoupit, musí je schválit web iStockphoto.
- **Flexibilita pro prodejce** – prodejci si mohou zvolit ceny, několik velikostí obrázku a licence, pod kterými chtějí obrázky prodávat.
- **Kredity** – protože je většinu fotografií možné zakoupit jen za pár dolarů, má web iStockphoto kreditní systém. Zákazníci si mohou zakoupit kredity v nominální hodnotě nejméně 10 dolarů, které se jim připíší na účet. Kredity se z účtu následně odečítají při každém nákupu.
- **Sociální aspekt** – fotografie je možné hodnotit a komentovat, což činí web velmi interaktivním a společenským.

WooThemes

WooThemes je online obchod s tématy rychle získávající na popularitě, který pracuje jinak, než většina webů zabývajících se prodejem témat. Je možné zakoupit buďto tematický balík (téma a odpovídající barevné schéma), nebo zaplatit jednorázový poplatek za měsíční členství dovolující během daného měsíce stahovat jakákoli požadovaná témata, s garancí minimálního počtu témat každý měsíc.

WooThemes také vybízí zavedené členy komunity návrhářů webů, aby pro web vytvořili téma a tím pomohli pozvednout web a mohli dále nabízet svá kvalitní témata.

Charakteristické rysy

- Detailní znalost průmyslu a uznávaní návrháři pomáhají zvyšovat tržní hodnotu nabízených produktů.
- Dva způsoby nákupu, každý se svými přednostmi a poskytující přístup k jinému obsahu:
  - Platby za konkrétní obsah
  - Nabídky pro členy

eBay

Jak už víte, eBay je online aukční dům. Co specifického však nabízí svým zákazníkům?

Charakteristické rysy

- Propracované vyhledávání produktů
- Přímý nákup produktů s použitím odkazu **Koupit teď**
- Přihazovat/projevit zájem o nákup produktu
- Realizovat platby a spravovat objednávky

Amazon

Amazon, na rozdíl od webu eBay, nefunguje jako online aukční dům. Přesto oplývá řadou sociálních služeb, jako je hodnocení, recenze a doporučení. Dovoluje uživatelům také prodávat již použité produkty nabízené na webu, a to pomocí Amazon Marketplace. Toto tržiště je integrováno s výpisem produktů hlavní části webu a informuje zákazníky o možnosti zakoupit použité či úplně nové produkty nikoli přímo od webu Amazon, ale na elektronické tržnici.

Charakteristické rysy

Na základní úrovni nabízí web Amazon svým zákazníkům následující služby:

- Procházení a vyhledávání produktů
- Hodnocení a recenze produktů
- Nákup produktů
- Realizace plateb a správa objednávek
- Prodej produktů skrze tržiště Amazon Marketplace

Play.com

Web Play.com pracuje na podobném principu jako web Amazon. Nejenom že prodává produkty, ale umožňuje také uživatelům prodej vlastního zboží (v části PlayTrade). Web Play.com se výrazně odlišuje především způsobem rozdělení produktů do kategorií, které mohou být více dynamické, jako např. produkty s určitou maximální cenou nebo sezónní zboží (např. nápady na vánoční dárky).

Charakteristické rysy

- Procházení a vyhledávání produktů
- Hodnocení a recenze produktů
- Nákup produktů
- Realizace plateb a správa objednávek
- Prodej produktů skrze PlayTrade

Požadavky na elektronický obchod

Na základě seznámení s několika populárními elektronickými obchody je zřejmé, že našemu obchodu nesmí chybět určité klíčové funkce. Uživatelé musí mít možnost vyhledávat a procházet produkty, které jsou rozdělené do nejrůznějších kategorií. Návštěvníci webu musí mít přirozeně možnost tyto produkty zakoupit, což vede k potřebě nákupního košíku, kam se budou ukládat produkty, které chce návštěvník zakoupit, propracovaného procesu realizace objednávky, během kterého se určí detaily dodávky, vypočítá výše daně a náklady na dopravu, zpracuje platba, a samozřejmě také správy objednávek pro administrátory. Z této nezbytné funkcionality budeme vycházet při budování našeho frameworku.

Výjimku s ohledem na tyto funkce tvoří eBay, který se obejde bez nákupního košíku. Navíc však umožňuje sledovat zboží, automaticky přihazovat a pomocí tlačítka **Koupit teď** také zboží ihned zakoupit.

Produkty

V souvislosti s produkty potřebujeme následující funkce:

- **Vyhledávání produktů** – potřebujeme funkcionalitu pro vyhledávání i procházení stávajících produktů, aby mohli zákazníci snadno nalézt požadovaný produkt.
- **Zobrazení produktů** – poté, co zákazník najde produkt, který ho zajímá, musí mít přirozeně možnost zobrazit detailní informace o produktu. To také znamená, že musíme vzít do úvahy, jaké typy informací jsou spojené s produkty (např. název, cena, hmotnost atd.). Také informace přicházející od uživatelské komunity, jako např. hodnocení a recenze, jsou vítané.
- **Projevení zájmu o produkt** – to se může projevit přidáním produktu do nákupního košíku anebo do seznamu přání pro pozdější koupi.

Proces objednávky

Proces objednávky má v zásadě tři kroky a nezbytné součásti:

- Seskupení produktů do objednávek (nejedná-li se o aukční web)
- Přijetí a zpracování plateb za objednávky, popř. přijetí informací o způsobu, jakým se platba provede
- Získání informací o dodání k zákazníkovi

Obecné

Existují i další podpůrné funkce, které bude muset framework implementovat:

- Umožnit administraci obchodu
- Umožnit zákazníkům spravovat jejich objednávky a měnit nastavení uživatelského účtu (uživatelské jméno, heslo, dodací adresa)

Požadavky na náš framework

Vytvoříme framework disponující všemi funkcemi, které potřebujeme. Přesné požadavky projektu se samozřejmě projekt od projektu liší. Proto zajistíme základní funkčnost, kterou je následně možné podle potřeb rozšiřovat. Následující funkce představují minimum v oblasti toho, co bude náš framework nabízet:

- zobrazení a správa produktů,
- zobrazení a správa kategorií produktů,

- vkládání produktů, výpisů a kategorií do dalších částí webu nebo webové aplikace (konec konců vytváříme framework),
- úprava produktů na míru,
- vyhledávání produktů,
- filtrování produktů na základě preferencí uživatele, jako je značka a další vlastnosti,
- seznam přání, tj. seznam produktů, které by si uživatel jednou rád koupil nebo by si je přál od někoho dostat (včetně rádce pro výběr dárků),
- generování doporučení na základě předchozích nákupů,
- odesílání upozornění na dostupnost určitých produktů e-mailem,
- zveřejnění hodnocení a recenzí produktů,
- nákupní košík pro uložení produktů umožňující zákazníkovi nastavit požadované množství produktů,
- výpočet nákladů na dopravu,
  - na základě produktů a jejich váhy,
  - na základě dodací adresy,
  - na základě speciálních pravidel (např. doprava zdarma pro objednávky nad určitou částku),
- výpočet daně,
- správa slevových kuponů,
- správa dárkových certifikátů,
- poskytování slev na základě doporučení,
- zpracování plateb,
- zpřístupnění správy účtu uživatelům,
- zpřístupnění správy obchodu administrátorům.

Kromě této funkčnosti se dále zaměříme ještě na následující funkce:

- nasazení frameworku do produkčního prostředí,
- zálohování a obnova elektronického obchodu ze zálohy,
- zpřístupnění elektronického obchodu pomocí zabezpečeného připojení protokolem SSL.

V rámci názorné ukázky rozšíření frameworku pro potřeby konkrétního elektronického obchodu obsahuje kniha také tři přílohy, ve kterých se podíváme na tři různé způsoby rozšíření frameworku:

- integrace webových služeb, jako je např. Google Product Search,
- rozšíření obchodu o stahovatelné produkty,
- nejrůznější ukázky kódu (ve formě kuchařky) ukazující, jak rychle tento i jiné frameworky rozšířit o některá specifická vylepšení.

Ukázkové nasazení našeho frameworku

Většina online obchodů existuje pro dobrý důvod, ať už je to prodej konkrétních produktů anebo coby online odnož kamenného obchodu. Smyslem frameworku je přirozeně přizpůsobit se jakémukoli účelu. Pro potřeby této knihy však budeme potřebovat vytvořit ukázkový web.

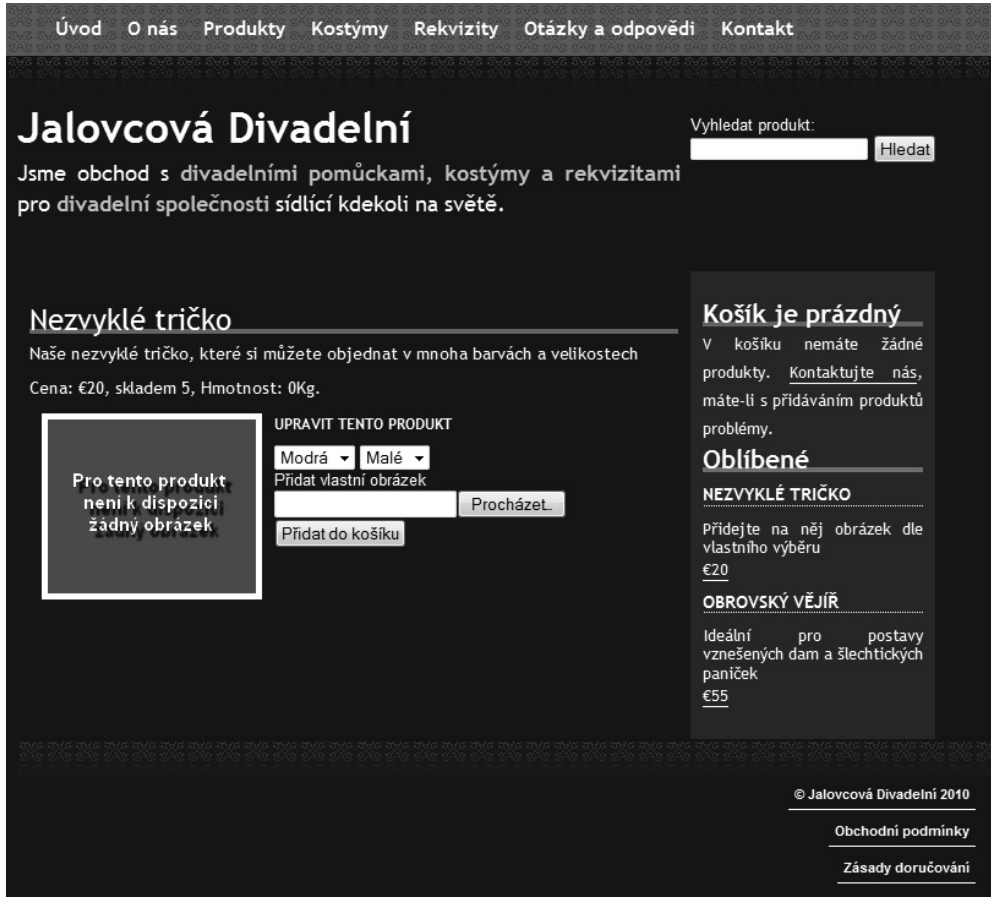
Jalovcová Divadelní

Jalovcová Divadelní je fiktivní kamenný obchod dodávající divadelní rekvizity a zboží. Někteří jeho zákazníci realizují objednávky telefonicky, jiní faxem a místní přímo osobně. Framework použijeme pro vybudování webové prezentace tohoto fiktivního obchodu. Kromě dříve uvedených funkcí má obchod ještě několik specifických požadavků:

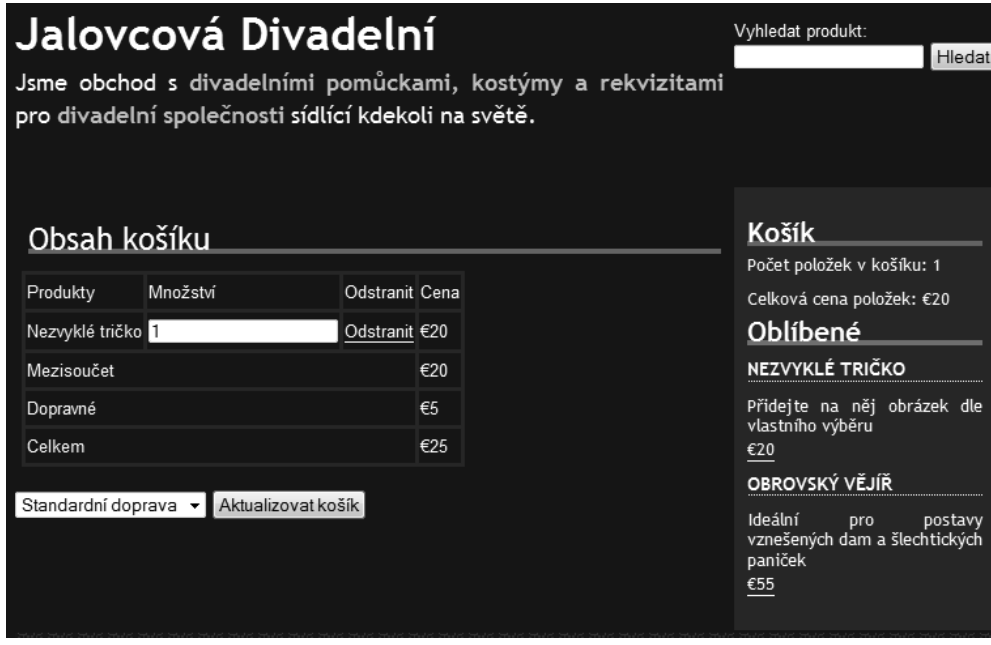
- **Přizpůsobitelné produkty** – oděvy.
- **Vlastní produkty** – uživatelsky definované obrázky a texty na oděvech, s případnou možností je následně nabízet k prodeji.
- **Virtuální prodeje** – elektronické lístky na akce.

Protože obchod nemá žádný web, musí se framework postarat o celý web a integrovat funkčnost elektronického obchodu tam, kde je to na místě. Přestože má společnost sídlo v ČR, má web za úkol oslovit zejména nové zákazníky ze Slovenska, a proto bude jako primární měnu používat euro.

Pojďme se podívat na to, jak bude web společnosti Jalovcová Divadelní, založený na našem frameworku, vypadat. Následující obrázek ukazuje podobu stránky s detaily o produktu.


Úvod
O nás
Produkty
Kostýmy
Rekvizity
Otázky a odpovědi
Kontakt
Jalovcová Divadelní
Jsme obchod s divadelními pomůckami, kostýmy a rekvizitami pro divadelní společnosti sídlící kdekoli na světě.
Vyhledat produkt:
Hledat
Nezvyklé tričko
Naše nezvyklé tričko, které si můžete objednat v mnoha barvách a velikostech
Cena: €20, skladem 5, Hmotnost: 0Kg.
Pro tento produkt není k dispozici žádný obrázek
UPRAVIT TENTO PRODUKT
Modrá
Malé
Přidat vlastní obrázek
Procházet...
Přidat do košíku
Košík je prázdný
V košíku nemáte žádné produkty. Kontaktujte nás, máte-li s přidáváním produktů problémy.
Oblíbené
NEZVYKLÉ TRIČKO
Přidejte na něj obrázek dle vlastního výběru
€20
OBROVSKÝ VĚJÍŘ
Ideální pro postavy vznešených dam a šlechtických paniček
€55
© Jalovcová Divadelní 2010
Obchodní podmínky
Zásady doručování

A takto vypadá nákupní košík:



Shrnutí

V této kapitole jsme se podívali na elektronické obchodování a probrali důvody pro vytvoření vlastního frameworku pro elektronický obchod v jazyce PHP, včetně funkcí, které bude nabízet. Podívali jsme se také na některé existující elektronické obchody a seznámili se s různými typy elektronických obchodů dostupných na internetu. Nyní, když víme, co budeme vytvářet a proč, jsme připraveni začít budovat základní strukturu a prvky našeho frameworku a následně přidat prvky pro elektronické obchodování.

KAPITOLA 2

Plánování frameworku

Nyní, když toho víme více o úkolu, který před námi stojí, je na čase začít plánovat framework a zajistit si tak správný start. V této kapitole se dozvíme:

- O návrhových a architektonických vzorech v PHP, včetně vzorů:
  - MVC (Model-View-Controller)
  - Registr
  - Singleton
- Jak uspořádat rozšiřitelný framework
- Jak by měl framework přistupovat k nastavením webu a elektronického obchodu

Začneme návrhem frameworku a na základě našich úvah pak framework vytvoříme.

Návrh frameworku

Existuje mnoho různých způsobů, jak navrhnout a vytvořit framework. Výsledkem některých jsou velmi komplikované frameworky, výsledkem jiných zase jednoduché. V této knize rychle vytvoříme snadno použitelný a pochopitelný framework.

Tato kniha slouží jako příručka, která vám pomůže vyvinout vlastní framework, odlišný od toho vytvářeného v této knize, lépe vyhovující vašim potřebám, nápadům a preferencím. Důraz se v této knize klade na elektronický obchod, takže pokud už máte vlastní framework nebo preferujete použití některého z již existujících frameworků, získáte z této knihy představu o tom, jak integrovat podporu pro elektronický obchod do jakéhokoli frameworku.

Softwarové vzory

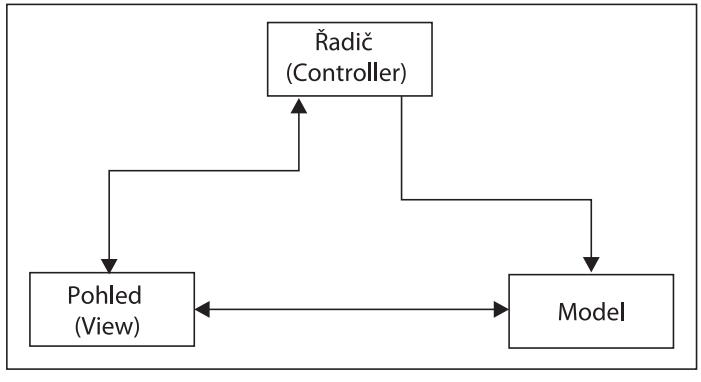
Existuje celá řada návrhových a architektonických vzorů poskytujících obecná řešení častých problémů v oblasti návrhu softwaru. Protože chceme vytvořit framework, jsou pro nás zajímavé zejména následující vzory:

- MVC (Model-View-Controller)
- Registr
- Singleton

Vzor MVC (Model-View-Controller)

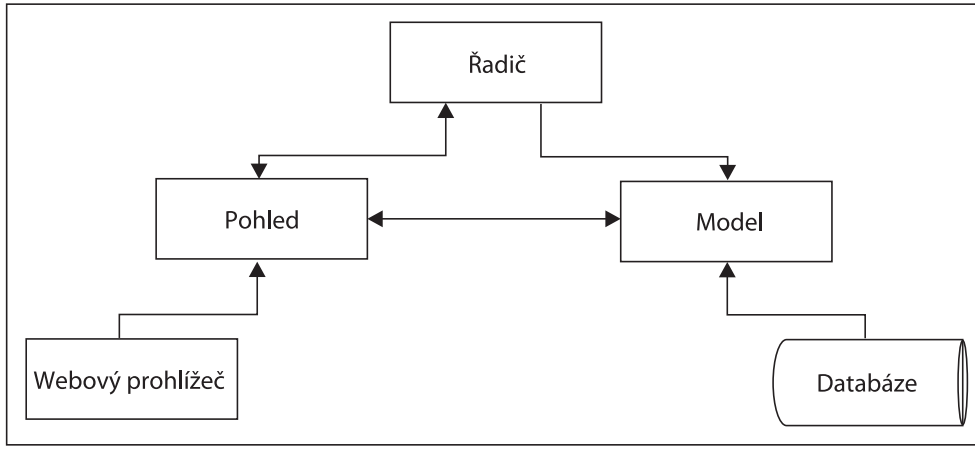
Architektonický vzor MVC představuje široce používané řešení oddělující uživatelské rozhraní od logiky aplikace. Uživatelské rozhraní aplikace (view neboli pohled) pracuje s daty (modelem) pomocí řadiče (controller), který obsahuje aplikační pravidla nezbytná pro manipulaci s daty odesílanými modelu a přicházejícími z něj.

Z pohledu elektronického obchodu uvažte zákazníka, který přidává produkt do nákupního košíku klepnutím na odkaz **Vložit do košíku** v pohledu (uživatelském rozhraní). Řadič tento požadavek zpracuje a interakcí s modelem (košíkem) zajistí přidání produktu do košíku. Obdobně se data z košíku prostřednictvím řadiče předají zpět do uživatelského rozhraní, kde se zobrazí počet produktů v košíku a hodnota jeho obsahu.



Protože vytváříme framework, který budou používat weby a webové aplikace, můžeme dále rozšířit reprezentaci vzoru MVC tak, aby reflektovala implementaci v takovémto frameworku. Jak už bylo řečeno, model reprezentuje data a ta se primárně ukládají v databázi. V našem frame-

worku však budeme mít sérii modelů, které načtou data a uloží je v sobě ve vhodnějším formátu, popřípadě usnadní manipulaci s daty. Do tohoto diagramu tak můžeme začlenit i databázi a zobrazit interakce mezi modely a databází. V prohlížeči se zobrazuje také konečný výsledek webové aplikace. Prohlížeč zobrazí pohledy a naše interakce (např. klepnutí myší nebo vyplněná pole) odešle nazpět řadiči. Do diagramu proto můžeme přidat také webový prohlížeč a znázornit tak jeho interakci s pohledy. Bude tak lépe patrné, jak pracuje návrhový vzor MVC v našem frameworku a jaké místo zaujímají jeho tři komponenty.

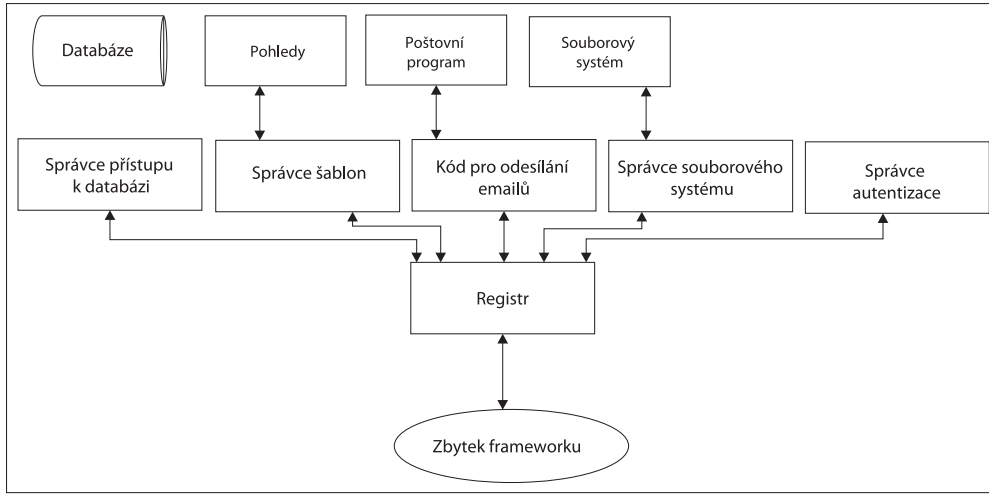


Registr

Návrhový vzor Registr umožňuje uložení kolekce objektů v rámci našeho frameworku. Potřeba registru je důsledkem abstrakce zavedené vzorem MVC. Každá skupina řadičů a modelů, které vytvoříme (např. produkty, nákupní košík nebo správa obsahu), bude provádět určité společné úkoly:

- Dotazování databáze
- Kontrola, zda je uživatel přihlášený, a pokud ano, načtení určitých uživatelských dat
- Odeslání dat pohledům ke zpracování (správa šablon)
- Odeslání e-mailů např. pro potvrzení objednávky zákazníka
- Interakce se souborovým systémem serveru, např. pro účely nahrání fotografií produktu na server

Většina systémů a frameworků abstrahuje tyto funkce do samostatných objektů a ten náš nebude v tomto ohledu žádnou výjimkou. Registr umožňuje udržet tyto objekty pohromadě. Registr je možné v rámci frameworku předávat a zajistit tak jednotný bod pro přístup ke klíčovým funkcím. Podívejme se, jak návrhový vzor Registry vypadá:



Framework pracuje přímo s registrem, který umožňuje přístup k dalším relevantním objektům. Tyto objekty mohou vzájemně spolupracovat s použitím samotného registru a oplývají funkcionalitou pro komunikaci s těmi částmi systému, které potřebují. Správce přístupu k databázi tak může přistupovat k databázi, správce šablon může přistupovat k šablonám uloženým v souborovém systému, kód pro odesílání e-mailů může přistupovat k šablonám e-mailů a také k systémové službě mail, správce souborového systému může přistupovat k souborovému systému a správce autentizace načítá a ukládá proměnné relace a soubory cookie, aby tak zachoval relaci autentizovaného uživatele mezi jednotlivými návštěvami webu.

Singleton

Existují situace, ve kterých chceme, aby existovala pouze jedna instance určité třídy. Typickým příkladem je správce spojení s databází. Více instancí by mohlo vést k odlišným výsledkům dotazů na databázi v závislosti na konfiguraci těchto instancí. Návrhový vzor Singleton tomu zabraňuje povolením existence pouze jedné instance třídy.

Uvedeným způsobem však návrhový vzor Singleton nepoužijeme. Místo toho ho použijeme k tomu, aby v rámci našeho frameworku vždy existovala pouze jedna instance registru.

Struktura

Další důležitou fází návrhu našeho frameworku je jeho struktura. Potřebujeme pro náš framework navrhnout vhodnou strukturu souborů. Struktura musí zohledňovat:

- Modely
- Pohledy (může být žádoucí integrovat možnost přepínat různé styly webu, samostatný adresář pro každou sadu šablon, stylů a pohledů tak může být dobrý nápad)

- Řadiče (vzhledem k tomu, že se k řadičům mohou pojit pomocné funkce uložené v dalších souborech, může být vhodné každý řadič umístit do vlastního adresáře)
- Řadiče pro administraci (pokud chceme do frameworku přidat nástroje pro administraci, neobejdeme se bez speciálních řadičů – bude se jednat o řadiče pro nejrůznější modely, přístupné pouze administrátorům a navržené specificky pro administraci)
- Registr
- Objekty registru
- Soubory nahrávané uživateli/administrátory na server
- Knihovny třetích stran
- Veškerý další kód

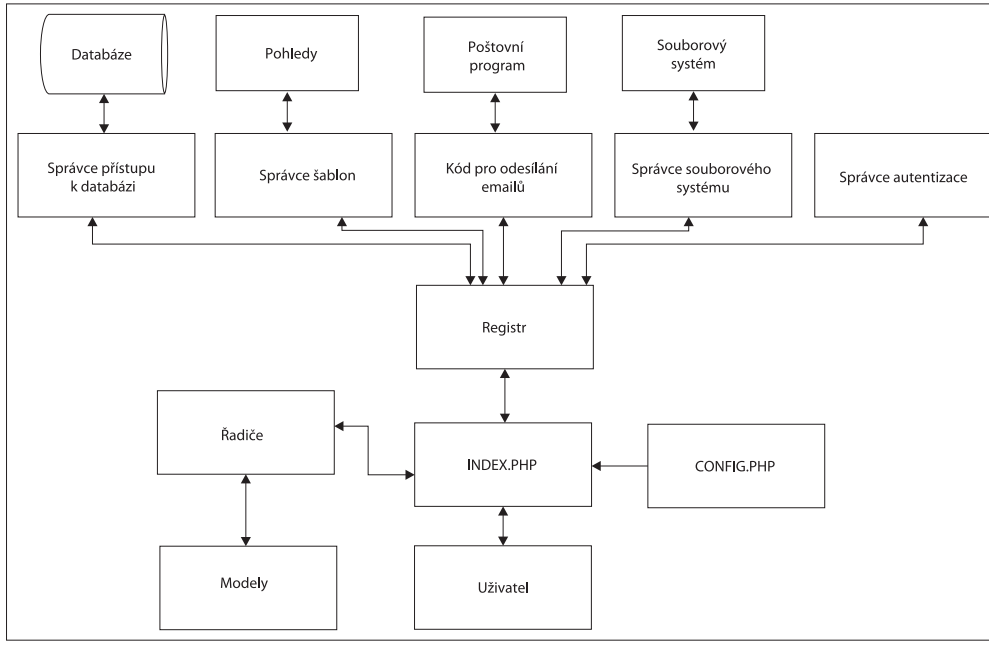
S ohledem na tuto strukturu frameworku může vhodná adresářová struktura vypadat následovně:

- Modely
- Pohledy
  - Pohled A (tj. adresář pro každou množinu pohledů)
  - Šablony
  - Obrázky
  - JavaScript
- Řadiče
  - Řadič A (tj. adresář pro každý řadič)
    - ŘadičA
    - ŘadičAAdmin
- Registr
  - Objekty
  - Databázové objekty
- Položky
- Soubory nahrané na server
  - Tuto část si detailněji objasníme, až přidáme produkty a obrázky do našeho frameworku
- Knihovny
- Další

Vytvoření frameworku

Strukturu frameworku máme vymyšlenou, je načase ho vytvořit. Začneme implementací výše uvedených architektonických a návrhových vzorů.

Celkový pohled na framework prozradí, že uživatel přistoupí k webu přes soubor `index.php`, který vytvoří instanci registru, vytvoří instance relevantních řadičů a předá jim objekt registru. Řadič vytvoří patřičné modely a ty, stejně jako řadiče, mohou pracovat s předaným registrem a vytvářet pohledy a manipulovat s nimi, jak je zapotřebí – tak jak ilustruje následující diagram:



Implementace softwarových vzorů

Existuje celá řada různých způsobů, jak uvedené vzory implementovat. Mnoho různých frameworků tyto vzory využívá a implementuje různým způsobem. Způsob implementace, který zvolíme my, je pouze jednou z mnoha možných metod. Tuto implementaci možná uznáte za vhodné změnit tak, aby lépe vyhovovala vašim potřebám nebo posloužila jako odrazový můstek pro vybudování vašeho vlastního frameworku. Pusťme se tedy do toho.

Vzor MVC

Vlastní implementace vzoru MVC je v tomto stádiu velmi obtížná, protože předpokládá, že jsme už ve fázi, kdy chceme implementovat hlavní funkce, které budou weby založené na našem frameworku používat. Tak daleko samozřejmě ještě nejsme. Zatím máme pouze adresářovou strukturu. V rámci ilustrace toho, jak tento vzor funguje, můžeme vytvořit nějaké základní modely, pohledy a řadiče, popř. nějaká rozhraní pro modely a pohledy, tak aby jejich struktura byla pro všechny modely a řadiče obdobná.

Vzor Registr

Implementace samotného registru je vcelku snadná. Složitější je to s objekty, které obsahuje. Registr potřebuje jednu metodu pro vytvoření objektu a jeho uložení pod určitým klíčem a druhou, která bere klíč jako parametr a vrací odpovídající objekt.

Registr může disponovat také některými užitečnými centrálními funkcemi, které je případně možné abstrahovat také do samostatných tříd (máte-li za to, že bude pro váš framework vhodnější tyto funkce přesunout do samostatných objektů, učiňte tak). Mezi některé z těchto funkcí patří tyto:

- Zpracování adresy URL příchozího požadavku, tak aby mohl soubor `index.php` požadavek nasměrovat správnou cestou.
- Sestavení adresy URL na základě skupiny parametrů, dotazového řetězce a použité metody pro generování URL (specifikující, jestli se na serveru používá modul `mod_rewrite`)
- Stránkování

Nastavení

Registr je naším primárním úložištěm pro nastavení a běžně používané objekty. Je tedy třeba zajistit možnost správy nastavení.

Data

Nastavení, která webové aplikace ukládají do databáze, mívají celou škálu podob, od rozsáhlých textů až po pravdivostní hodnoty typu vyjadřující stav zaškrtávacího pole. Takovéto flexibility se nedosahuje snadno. Nejjednodušší metodou je uložit nastavení do jedné tabulky ve formě párů klíč/hodnota, kde hodnota je typu `longtext`. Určitá nastavení se mohou ukládat také přímo v kódu. Registr musí mít možnost kopii těchto nastavení jednoduše uložit, aby s nimi mohl pracovat framework.

Kód

Následující kód představuje základ našeho registru, tvořeného dvěma poli (jedním pro objekty a druhým pro nastavení) a načítací/ukládací metodou pro každé z nich. Metoda `storeObject` kontroluje, jestli se jedná o databázový objekt, a pokud ano, připojí soubor s deklarací třídy z jiného adresáře.

```
/**
 * Pole objektů uložených v registru
 * @access private
 */
private static $objects = array();

/**
 * Pole nastavení uložených v registru
 * @access private
 */
```

```
private static $settings = array();

/**
 * Uloží objekt do registru
 * @param String $objekt název objektu
 * @param String $klic klíč, pod kterým se objekt v poli uloží
 * @return void
 */
public function storeObject( $object, $key )
{
  if( strpos( $object, 'database' ) !== false )
  {
    $object = str_replace( '.database', 'database', $object);
    require_once('databaseobjects/' . $object . '.database.class.php');
  }
  else
  {
    require_once('objects/' . $object . '.class.php');
  }
  self::$objects[ $key ] = new $object( self::$instance );
}

/**
 * Získá objekt z registru
 * @param String $klic klíč, pod kterým je objekt v poli uložený
 * @return object objekt
 */
public function getObject( $key )
{
  if( is_object ( self::$objects[ $key ] ) )
  {
    return self::$objects[ $key ];
  }
}

/**
 * Uloží nastavení do registru
 * @param String $data nastavení, které se má uložit
 * @param String $klic klíč, pod kterým se objekt v poli uloží
 * @return void
 */
public function storeSetting( $data, $key )
{
  self::$settings[ $key ] = $data;
}
```

```
/**
 * Získá nastavení z registru
 * @param String $klic klíč, pod kterým je nastavení v poli uložené
 * @return String nastavení
 */
public function getSetting( $key )
{
  return self::$settings[ $key ];
}
```

Vzor Singleton

Návrhový vzor Singleton není nijak obtížné implementovat, protože standardní třídu v jazyce PHP stačí jen nepatrně upravit, aby se zajistilo, že se vždy vytvoří pouze jedna její instance.

```
/**
 * Instance třídy
 * @access private
 */
private static $instance;

/**
 * Soukromý konstruktor zabraňující přímému vytvoření instance třídy
 * @access private
 */
private function __construct()
{
}

/**
 * Metoda singleton používaná pro přístup k instanci třídy
 * @access public
 * @return
 */
public static function singleton()
{
  if( !isset( self::$instance ) )
  {
    $obj = __CLASS__;
    self::$instance = new $obj;
  }
  return self::$instance;
}

/**
 * Znemožňuje klonování objektu - při pokusu o klonování vyvolá
```

```
 * chybu E_USER_ERROR
 */
public function __clone()
{
  trigger_error( 'Klonování registru je zakázané', E_USER_ERROR );
}
```

Konstruktor (používaný pro vytvoření instance této třídy) je deklarovaný jako soukromý, což znamená, že ho není možné volat vně třídy. To je důležité pro omezení toho, jak se dá vytvořit instance třídy, a umožňuje nám řídit počet vytvořených instancí.

Následuje metoda s názvem `singleton`, která se používá pro získání instance třídy. Pokud doposud žádná instance třídy neexistuje, vytvoří se nová instance a uloží se do proměnné `instance`. Pokud už je tato proměnná nastavená, vrátí se v ní uložený objekt (tj. samotná instance).

Pokud bychom se někdy pokusili o klonování objektu, dojde k zavolání funkce `trigger_error` a vyvolá se chyba `E_USER_ERROR`.

Objekty registru

Jak už bylo řečeno dříve, samotný registr je velmi přímočarou záležitostí. Horší je to s objekty, které se v registru budou ukládat a k nimž bude registr zajišťovat přístup. Jedná se mimo jiné o objekty:

- Správce přístupu k databázi
- Autentizace uživatelů
- Správce šablon
- Správce e-mailů

Pojďme se nyní podívat na to, jak tyto objekty vytvořit.

Databáze

Náš databázový objekt musí nabízet následující funkcionalitu:

- Připojení k databázi
- Správa několika spojení s databází
- Provádění dotazů
- Získání běžných informací o výsledcích dotazu, jako je například počet ovlivněných řádků nebo poslední vložený identifikátor
- Ukládání dotazů do mezipaměti tak, aby se výsledky mohly integrovat do pohledů (skrze správce šablon, kterému se předá reference na mezipaměť, díky čemuž bude moci zakomponovat výsledky do pohledu)
- Usnadnění běžných dotazů (např. vkládání, aktualizací a mazání) pomocí metod, které se postarají o zformátování patřičného dotazu

Přirozeně je toho mnohem více, co může databázový objekt provádět. Za malý okamžik se k tomu dostaneme.

Náš databázový objekt

Naše databázová třída abstrahuje funkce pro databázový systém MySQL od zbytku frameworku a spravuje spojení s databází:

```
<?php
/**
  * Třída pro správu přístupu k databázi: základní abstrakce
  *
  * @author Michael Peacock
  * @version 1.0
  */
class mysqldatabase {
  /**
   * Umožňuje více spojení s databází
   * každé spojení se uloží ve formě prvku pole, aktivní spojení
   * je uloženo v proměnné (viz dále)
   */
  private $connections = array();

  /**
   * Říká DB objektu, které spojení se má použít,
   * pomocí setActiveConnection($id) je možné to změnit
   */
  private $activeConnection = 0;

  /**
   * Provedené dotazy, jejichž výsledky se uložily do mezipaměti, primárně
   * pro potřeby správce šablon
   */
  private $queryCache = array();

  /**
   * Data uložená do mezipaměti pro pozdější použití, primárně pro potřeby
   * správce šablon
   */
  private $dataCache = array();

  /**
   * Počet provedených dotazů
   */
  private $queryCounter = 0;
```

```
/**
 * Záznam o posledním dotazu
 */
private $last;

/**
 * Konstruktor databázového objektu
 */
public function __construct() { }
```

Pojďme se postupně podívat na jednotlivé metody třídy:

- `newConnection` – Tato metoda vytvoří nové spojení s databázovým systémem. Od konstruktoru je oddělená ze dvou důvodů. Za prvé, pokud by její funkci přebral konstruktor, musel by přijímat parametry spojení, což by vyžadovalo, aby se vytvořil odděleně od ostatních objektů v registru. Za druhé, tato metoda umožňuje v jednom objektu udržovat několik spojení s databází.

```
/**
 * Vytvoří nové spojení s databází
 * @param String název hostitele
 * @param String uživatelské jméno
 * @param String heslo
 * @param String název databáze
 * @return int identifikátor nového spojení
 */
public function newConnection( $host, $user, $password, $database )
{
  $this->connections[] = new mysqli( $host, $user, $password,
    $database );
  $connection_id = count( $this->connections )-1;
  if( mysqli_connect_errno() )
  {
    trigger_error('Chyb při připojování k hostiteli. '
      . $this->connections[$connection_id]->error, E_USER_ERROR);
  }
  return $connection_id;
}
```

- `closeConnection` – Tato metoda ukončí aktivní spojení (vzhledem k tomu, že může být v objektu uloženo několik spojení s databázovým systémem, uchovává se aktuální spojení v samostatné proměnné).

```
/**
 * Ukončí aktivní spojení
 * @return void
 */
public function closeConnection()
```

```
{
  $this->connections[$this->activeConnection]->close();
}
```

- `setActiveConnection` – Tato metoda umožňuje nastavit aktivní spojení.

```
/**
 * Nastaví aktuální spojení s databází pro následující operace
 * @param int identifikátor nového spojení
 * @return void
 */
public function setActiveConnection( int $new )
{
  $this->activeConnection = $new;
}
```

- `cacheQuery` – Tato metoda umožňuje uložit výsledek dotazu k pozdějšímu zpracování – primárně, aby se v pohledu mohl zobrazit (v cyklu).

```
/**
 * Uloží výsledek dotazu do mezipaměti pro pozdější zpracování
 * @param String dotazový řetězec
 * @return index výsledku dotazu v mezipaměti
 */
public function cacheQuery( $queryStr )
{
  if( !$result = $this->connections
    [$this->activeConnection]->query( $queryStr ) )
  {
    trigger_error('Chyba při provádění dotazu a jeho ukládání do mezipaměti: '
      . $this->connections[$this->activeConnection]->error,
      E_USER_ERROR);
    return -1;
  }
  else
  {
    $this->queryCache[] = $result;
    return count($this->queryCache)-1;
  }
}
```

- `numRowsFromCache` – Tato metoda získá počet řádků, které daný výsledek dotazu uložený v mezipaměti obsahuje.

```
/**
 * Získá počet řádků výsledku uloženého v mezipaměti
 * @param int index výsledku dotazu v mezipaměti
 * @return int počet řádků
 */
```

```
public function numRowsFromCache( $cache_id )
{
  return $this->queryCache[$cache_id]->num_rows;
}
```

- `resultsFromCache` – Tato metoda získá výsledek dotazu uložený v mezipaměti.

```
/**
 * Získá výsledek dotazu uložení v mezipaměti
 * @param int index výsledku dotazu
 * @return array výsledek dotazu
 */
public function resultsFromCache( $cache_id )
{
  return $this->queryCache[$cache_id]->fetch_array(MYSQLI_ASSOC);
}
```

- `cacheData` – Tato metoda uloží do mezipaměti pole dat, stejně jako by se jednalo o výsledek dotazu. Vzhledem k tomu, že se jedná o databázovou třídu, můžeme ji použít také pro uložení běžných dat.

```
/**
 * Uloží data do mezipaměti pro pozdější použití
 * @param array data
 * @return int index dat v mezipaměti
 */
public function cacheData( $data )
{
  $this->dataCache[] = $data;
  return count( $this->dataCache )-1;
}
```

- `dataFromCache` – Tato metoda získá data uložená v mezipaměti.

```
/**
 * Získá data z mezipaměti
 * @param int index dat v mezipaměti
 * @return array data
 */
public function dataFromCache( $cache_id )
{
  return $this->dataCache[$cache_id];
}
```

- `deleteRecords` – Tato metoda vytvoří na základě názvu tabulky, podmínky a limitu, předaných ve formě parametrů, dotaz pro odstranění záznamů a provede jej.

```
/**
 * Odstraní záznamy z databáze
 * @param String tabulka, ze které se záznamy odstraní
```

```
 * @param String podmínka pro odstranění
 * @param int počet řádků, které se mají odstranit
 * @return void
 */
public function deleteRecords( $table, $condition, $limit )
{
  $limit = ( $limit == '' ) ? '' : ' LIMIT ' . $limit;
  $delete = "DELETE FROM {$table} WHERE {$condition} {$limit}";
  $this->executeQuery( $delete );
}
```

- `updateRecords` – Tato metoda vytvoří na základě názvu tabulky, pole změn (klíče pole představují názvy sloupců, hodnoty pak jejich nové hodnoty) a podmínky, předaných ve formě parametrů, dotaz pro aktualizaci záznamů a provede jej.

```
/**
 * Aktualizuje záznamy v databázi
 * @param String název tabulky
 * @param array pole změn sloupec => hodnota
 * @param String podmínka
 * @return bool
 */
public function updateRecords( $table, $changes, $condition )
{
  $update = "UPDATE " . $table . " SET ";
  foreach( $changes as $field => $value )
  {
    $update .= "`" . $field . "`='{$value}',";
  }

  // odstranění koncového znaku ","
  $update = substr($update, 0, -1);
  if( $condition != '' )
  {
    $update .= "WHERE " . $condition;
  }
  $this->executeQuery( $update );

  return true;
}
```

- `insertRecords` – Tato metoda vytvoří na základě názvu tabulky a pole dat (klíče pole představují názvy sloupců, hodnoty pak jejich hodnoty), předaných ve formě parametrů, dotaz pro vložení záznamu do databáze a provede jej.

```
/**
 * Vloží záznam do databáze
```

```
 * @param String název tabulky
 * @param array pole dat sloupec => hodnota
 * @return bool
 */
public function insertRecords( $table, $data )
{
  // inicializace proměnných pro uložení sloupců a hodnot
  $fields = "";
  $values = "";

  // vyplnění proměnných
  foreach ($data as $f => $v)
  {
    $fields .= "`$f`,";
    $values .= ( is_numeric( $v ) && ( intval( $v ) == $v ) ) ?
      $v."," :   "'$v',";
  }

  // odstranění koncového znaku ","
  $fields = substr($fields, 0, -1);

  // odstranění koncového znaku ","
  $values = substr($values, 0, -1);
  $insert = "INSERT INTO $table ({$fields}) VALUES({$values})";
  $this->executeQuery( $insert );

  return true;
}
```

- `executeQuery` – Tato metoda provede dotaz.

```
/**
 * Provede dotaz
 * @param String dotaz
 * @return void
 */
public function executeQuery( $queryStr )
{
  if( !$result = $this->connections[$this->activeConnection]->
    query( $queryStr ) )
  {
    trigger_error('Chyba při provádění dotazu: '.$this->
      connections[$this->activeConnection]->error, E_USER_ERROR);
  }
  else
  {
```

```
      $this->last = $result;
    }
  }
```

- `getRows` – Tato metoda vrací řádky výsledku dotazu.

```
/**
 * Získá řádky výsledku posledního provedeného dotazu
 * @return array
 */
public function getRows()
{
  return $this->last->fetch_array(MYSQLI_ASSOC);
}
```

- `affectedRows` – Tato metoda vrací počet řádků ovlivněných posledním provedeným dotazem.

```
/**
 * Získá počet řádků ovlivněných předchozím dotazem
 * @return int počet ovlivněných řádků
 */
public function affectedRows()
{
  return $this->$this->connections[$this->activeConnection]->
    affected_rows;
}
```

- `sanitizeData` – Tato metoda vyčistí data, tak aby je bylo možné vložit do dotazu SQL.

```
/**
 * Vyčistí data
 * @param String data, která se mají vyčistit
 * @return String vyčištěná data
 */
public function sanitizeData( $data )
{
  return $this->connections[$this->activeConnection]->
    real_escape_string( $data );
}
```

- `__destruct` – Destruktor, který ukončí všechna otevřená spojení s databázovým systémem.

```
 /**
  * Destruktur
  * ukončí všechna otevřená spojení s databázovým systémem
  */
  public function __destruct()
  {
    foreach( $this->connections as $connection )
```

```
        {
          $connection->close();
        }
      }
    }
    ?>
```

Rozšíření databázového objektu

Jak můžeme databázový objekt rozšířit?

- **Dědičnost** – můžeme vytvořit obecné databázové rozhraní, které pro všechny databázové objekty definuje určité základní metody. To umožňuje později snáze přejít k jinému typu databáze (např. přejít z MySQL na PgSQL nebo MSSQL).
- **Přenesení logiky z dotazů** – aby se dále zjednodušilo použití nejrůznějších databází, můžeme přenést logiku z dotazů do samotných databázových objektů. Namísto celých dotazů se objektům budou předávat parametry dotazů, jako jsou např. názvy tabulek, sloupců, sloupců, podle kterých se má řadit (definujících seřazení výsledků), typy sjednocení atd. Objekt následně doplní nezbytné části SQL syntaxe. Jedině tak je možné ve frameworku dosáhnout skutečné abstrakce databáze.
- **Ladicí informace** – můžeme zajistit protokolování výkonu našich dotazů, zaznamenávat pomalé dotazy a umožnit tak ladění a optimalizaci dotazů, které používáme.

Autentizace uživatelů

Naše třída pro autentizaci uživatelů musí:

- Zpracovat požadavky na přihlášení
- Ověřit přihlášení uživatele
- Odhlásit uživatele
- Spravovat informace o právě přihlášeném uživateli (v případě zájmu můžeme třídu rozšířit o použití objektu uživatele)

Ze všeho nejdřív budeme potřebovat naši třídu a některé její metody:

```
<?php
/**
 * Správce autentizace
 *
 *
 * @version 1.0
 * @author Michael Peacock
 */
class authentication {
  private $userID;
  private $loggedIn = false;
```

```
  private $admin = false;
  private $groups = array();
  private $banned = false;
  private $username;
  private $justProcessed = false;
  public function __construct() {}
```

Toto jsou základní vlastnosti, které budeme potřebovat. Dále vytvoříme metodu, která zajistí autentizaci uživatele. Tuto metodu bude volat náš framework poté, co se naváže spojení s databází. Metoda nejdříve zkontroluje, jestli může být uživatel přihlášený, a pokud ano, ověří toto přihlášení. V opačném případě ověří, zda se uživatel snaží přihlásit. V závislosti na situaci může metoda volat některou z pomocných metod. Jak metoda vypadá, ukazuje následující kód.

```
public function checkForAuthentication()
{
  if( isset( $_SESSION['phpecomf_auth_session_uid'] ) &&
    intval( $_SESSION['phpecomf_auth_session_uid'] ) > 0 )
  {
    $this->sessionAuthenticate( intval(
      $_SESSION['phpecomf_auth_session_uid'] ) );
  }
  elseif( isset( $_POST['ecomf_auth_user'] ) &&
    $_POST['ecomf_auth_user'] != '' &&
    isset( $_POST['ecomf_auth_pass'] ) &&
    $_POST['ecomf_auth_pass'] != '')
  {
    $this->postAuthenticate(
      PeacockCarterFrameworkRegistry::getObject('db')->
        sanitizeData( $_POST['ecomf_auth_user'] ),
          md5( $_POST['ecomf_auth_pass'] ) );
  }
  //echo $this->userID;
}
```

Přihlášeného uživatele můžeme autentizovat na základě dat z relace – pokud uložíme do relace identifikátor uživatele, můžeme ověřit, že je tento identifikátor platný a uživatel je přihlášený.

```
private function sessionAuthenticate( $uid )
{
  $sql = "SELECT u.ID, u.username, u.active, u.email, u.admin,
    u.banned, u.name, (SELECT GROUP_CONCAT( g.name SEPARATOR
    '-groupsep-' ) FROM groups g, group_memberships gm
    WHERE g.ID = gm.group AND gm.user = u.ID ) AS groupmemberships
    FROM users u WHERE u.ID={$uid}";
  PeacockCarterFrameworkRegistry::getObject('db')->
    executeQuery( $sql );
```

```
  if( PeacockCarterFrameworkRegistry::getObject('db')->
      numRows() == 1 )
  {
```

I když uživatel existuje, nemůžeme ho jednoduše přihlásit. Co když není účet uživatele aktivní, nebo je dokonce zablokovaný?

```
$userData = PeacockCarterFrameworkRegistry::getObject('db')-> getRows();
if( $userData['active'] == 0 )
{
  $this->loggedIn = false;
  $this->loginFailureReason = 'inactive';
  $this->active = false;
}
elseif( $userData['banned'] != 0)
{
  $this->loggedIn = false;
  $this->loginFailureReason = 'banned';
  $this->banned = false;
}
else
{
  $this->loggedIn = true;
  $this->userID = $uid;
  $this->admin = ( $userData['admin'] == 1 ) ? true : false;
  $this->username = $userData['username'];
  $this->name = $userData['name'];
```

Veškerá členství uživatelů ve skupinách vrací výše uvedený dotaz jako jeden sloupec. Hodnotu tohoto sloupce můžeme rozdělit na jednotlivé skupiny a uložit je do objektu.

```
    $groups = explode( '-groupsep-', $userData['groupmemberships'] );
    $this->groups = $groups;
  }
}
else
{
  $this->loggedIn = false;
  $this->loginFailureReason = 'nouser';
  if( $this->loggedIn == false )
  {
    $this->logout();
  }
}
```

Pokud se uživatel snaží o přihlášení, pokusíme se vyhledat jím zadanou kombinaci uživatelského jména a hesla v databázi a tím ji ověřit. Podobá se to předchozí metodě, pouze s tím rozdílem, že se používá uživatelské jméno a heslo zadané uživatelem namísto identifikátoru uživatele uloženého v relaci.

```
private function postAuthenticate( $u, $p )
{
  $this->justProcessed = true;
  $sql = "SELECT u.ID, u.username, u.email, u.admin, u.banned,
    u.active, u.name, (SELECT GROUP_CONCAT( g.name SEPARATOR
    '-groupsep-' ) FROM groups g, group_memberships gm WHERE
    g.ID = gm.group AND gm.user = u.ID ) AS groupmemberships
    FROM users u WHERE u.username='{$u}' AND u.password_hash='{$p}'";
  //echo $sql;
  PeacockCarterFrameworkRegistry::getObject('db')->
    executeQuery( $sql );
  if( PeacockCarterFrameworkRegistry::getObject('db')->
      numRows() == 1 )
  {
    $userData = PeacockCarterFrameworkRegistry::getObject('db')->
      getRows();
```

Stejně jako v případě předchozí metody musíme zkontrolovat, že je uživatel aktivní a jeho účet není zablokovaný.

```
    if( $userData['active'] == 0 )
    {
      $this->loggedIn = false;
      $this->loginFailureReason = 'inactive';
      $this->active = false;
    }
    elseif( $userData['banned'] != 0)
    {
      $this->loggedIn = false;
      $this->loginFailureReason = 'banned';
      $this->banned = false;
    }
    else
    {
      $this->loggedIn = true;
      $this->userID = $userData['ID'];
      $this->admin = ( $userData['admin'] == 1 ) ? true : false;
      $_SESSION['phpecomf_auth_session_uid'] = $userData['ID'];
      $groups = explode( '-groupsep-', $userData['groupmemberships'] );
      $this->groups = $groups;
    }
```

```
  }
  else
  {
    $this->loggedIn = false;
    $this->loginFailureReason = 'invalidcredentials';
  }
}
```

Odhlášení je možné realizovat pouhým odstraněním identifikátoru uživatele z relace.

```
function logout()
{
  $_SESSION['phpecomf_auth_session_uid'] = '';
}
```

V neposlední řadě potřebujeme také několik metod pro čtení vlastností, které budou vracet hodnoty nejrůznějších vlastností aktuálního uživatele:

```
  public function getUserID()
  {
    return $this->userID;
  }
  public function isLoggedIn()
  {
    return $this->loggedIn;
  }
  public function isAdmin()
  {
    return $this->admin;
  }
  public function getUsername()
  {
    return $this->username;
  }
  public function isMemberOfGroup( $group )
  {
    if( in_array( $group, $this->groups )
    {
      return true;
    }
    else
    {
      return false;
    }
  }
}
?>
```

Správce šablon

Funkci správce šablon je možné rozdělit do dvou částí – objektu pro správu vlastního obsahu (objekt `page`) a objektu šablony pro správu interakce s obsahem společně s analýzou jeho obsahu.

Podívejme se na kód uložený v souboru `template.class.php`:

```
<?php
/**
 * Views: Správce šablon
 * Obsah stránky a strukturu spravuje samostatný objekt page.
 *
 * @version 1.0
 * @author Michael Peacock
 */
class template {
```

Konstruktor třídy `template` vytváří instanci třídy `page`, která se uloží do vlastnosti `page` objektu `template`. To dovoluje objektu `template` manipulovat se stránkou tak, jak vyžaduje framework.

```
private $page;

  /**
   * Připojí soubor s definicí třídy page a vytvoří instanci této
   * třídy, která bude sloužit pro správu obsahu a struktury stránky
   */
  public function __construct()
  {
    include( FRAMEWORK_PATH . '/registry/objects/page.class.php');
    $this->page = new Page();
  }
```

V některých případech může být zapotřebí jednu šablonu vložit do druhé. Můžeme například chtít na každé stránce zobrazit souhrn nákupního košíku. Pokud ale v košíku nic není, můžeme chtít vložit jinou šablonu nebo jen nějaký text. Následující metoda vloží do šablony další šablonu, najde-li patřičnou značku.

```
/**
 * Přidá do stránky část šablony
 * @param String $znacka značka, do které se vloží část
 * šablony - např. {ahoj}
 * @param String $cast část šablony (cesta k souboru nebo jeho název)
 * @return void
 */
public function addTemplateBit( $tag, $bit )
{
  if( strpos( $bit, 'views/' ) === false )
  {
```

```
    $bit = 'views/' . PHPEcommerceFrameworkRegistry::getSetting('view')
    . '/templates/' . $bit;
  }
  $this->page->addTemplateBit( $tag, $bit );
}
```

Aby se usnadnilo vložení více šablon do jiné šablony, můžeme šablony a značky šablon uložit do pole, kterým se následně projde při analýze šablon.

```
/**
 * Načte části šablon stránky a vloží je do obsahu stránky
 * Aktualizuje obsah stránek
 * @return void
 */
private function replaceBits()
{
  $bits = $this->page->getBits();

  // cyklus přes části šablony
  foreach( $bits as $tag => $template )
  {
    $templateContent = file_get_contents( $template );
    $newContent = str_replace( '{' . $tag . '}', $templateContent,
      $this->page->getContent() );
    $this->page->setContent( $newContent );
  }
}
```

Protože budeme chtít do šablon vkládat dynamicky generovaná data, potřebujeme pro tento účel metodu. Ta nahradí všechny značky požadovanými hodnotami a dále zkontroluje, zda se jedná o běžná data, nebo se jedná o výsledky dotazů uložené v mezipaměti (popř. data uložená v mezipaměti). Jde-li o výsledek dotazu, nahradí metoda značku jeho výsledkem.

```
/**
 * Nahradí značky ve stránce požadovaným obsahem
 * @return void
 */
private function replaceTags()
{
  // získej značky ve stránce
  $tags = $this->page->getTags();

  // cyklus přes značky
  foreach( $tags as $tag => $data )
  {
    // pokud je značka pole, je zapotřebí víc než prosté
    // "vyhledej a nahraď"
```

```
    if( is_array( $data ) )
    {
      if( $data[0] == 'SQL' )
      {
        // jedná se o výsledek dotazu uložený v mezipaměti, značky
        // se nahradí tímto výsledkem
        $this->replaceDBTags( $tag, $data[1] );
      }
      elseif( $data[0] == 'DATA' )
      {
        // jedná se o data uložená v mezipaměti, značky
        // se nahradí daty z mezipaměti
        $this->replaceDataTags( $tag, $data[1] );
      }
    }
    else
    {
      // nahraď obsah
      $newContent = str_replace( '{' . $tag . '}', $data,
        $this->page->getContent() );

      // aktualizuj obsah stránky
      $this->page->setContent( $newContent );
    }
  }
}
```

Pokud jsme uložili výsledky dotazu na databázi do mezipaměti a přiřadili ho do proměnné šablony, je třeba těmito výsledky nahradit odpovídající značky šablony. To má za úkol následující metoda, která vyhledá obsah mezi značkami `<!-- START znacka -->` a `<!-- END znacka -->` a nahradí odpovídající značky v tomto bloku kódu výsledky dotazu.

```
/**
 * Nahradí obsah stránky daty z databáze
 * @param String $znacka značka definující oblast nahrazovaného obsahu
 * @param int $idMezipameti identifikátor dotazu v mezipaměti
 * @return void
 */
private function replaceDBTags( $tag, $cacheId )
{
  $block = '';
  $blockOld = $this->page->getBlock( $tag );

  // cyklus přes jednotlivé záznamy výsledku dotazu
  while ($tags = PHPEcommerceFrameworkRegistry::getObject('db')->
    resultsFromCache( $cacheId ) )
```

```
  {
    $blockNew = $blockOld;

    // vytvoří nový blok s vloženými výsledky
    foreach ($tags as $ntag => $data)
    {
      $blockNew = str_replace("{" . $ntag . "}", $data, $blockNew);
    }
    $block .= $blockNew;
  }
  $pageContent = $this->page->getContent();

  // odstraní oddělovač ze šablony => čistší kód HTML
  $newContent = str_replace( '<!-- START ' . $tag . ' -->'
    . $blockOld . '<!-- END ' . $tag . ' -->', $block, $pageContent );

  // aktualizace obsahu stránky
  $this->page->setContent( $newContent );
}
```

Následující metoda provádí totéž co metoda `replaceDBTags`, avšak s daty (na rozdíl od dotazů) uloženými v mezipaměti.

```
/**
 * Nahradí obsah stránky daty z mezipaměti
 * @param String $znacka značka definující oblast nahrazovaného obsahu
 * @param int $idMezipameti identifikátor dat v mezipaměti
 * @return void
 */
private function replaceDataTags( $tag, $cacheId )
{
  $block = $this->page->getBlock( $tag );
  $blockOld = $block;
  while ($tags = PHPEcommerceFrameworkRegistry::getObject('db')->
    dataFromCache( $cacheId ) )
  {
    foreach ($tags as $tag => $data)
    {
      $blockNew = $blockOld;
      $blockNew = str_replace("{" . $tag . "}", $data, $blockNew);
    }
    $block .= $blockNew;
  }
  $pageContent = $this->page->getContent();
  $newContent = str_replace( $blockOld, $block, $pageContent );
  $this->page->setContent( $newContent );
```

```
}
```

Tato metoda vrací objekt `page`, jehož veřejné metody můžeme následně přímo volat, bude-li to vně kontextu objektu šablony zapotřebí.

```
/**
 * Získá objekt stránky
 * @return Object
 */
public function getPage()
{
  return $this->page;
}
```

Naši stránku tvoří řada šablon. Je to tedy první věc, kterou musíme při vytvoření nového pohledu udělat – pohled sestavíme z několika šablon. Šablony se musí metodě předat ve správném pořadí (tj. záhlaví, hlavní obsah, zápatí), aby se správně zobrazily.

```
/**
 * Sestaví obsah stránky na základě několika šablon
 * umístění jednotlivých šablon se předávají ve formě argumentů
 * @return void
 */
public function buildFromTemplates()
{
  $bits = func_get_args();
  $content = "";
  foreach( $bits as $bit )
  {
    if( strpos( $bit, 'views/' ) === false )
    {
      $bit = 'views/' . PHPEcommerceFrameworkRegistry::getSetting('view')
      . '/templates/' . $bit;
    }
    if( file_exists( $bit ) == true )
    {
      $content .= file_get_contents( $bit );
    }
  }
  $this->page->setContent( $content );
}
```

To je obzvláště důležité, když v řadiči nebo modelu pracujete s jedním řádkem výsledků dotazu. Můžeme je chtít transformovat na značky šablony. K tomu slouží následující metoda, která navíc umožňuje opatřit značky prefixy, což pomáhá eliminovat konflikty v názvech.

```
/**
 * Převede pole dat na značky
```