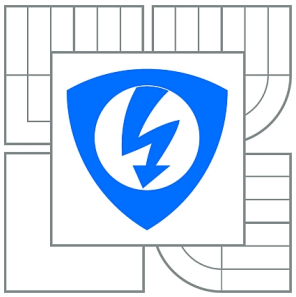
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

DEPARTMENT OF CONTROL AND INSTRUMENTATION

VYUŽITÍ FRAMEWORKŮ PRO NÁVRH DATABÁZOVÉ WWW APLIKACE.

THE FRAMEWORKS FOR WWW APPLICATIONS.

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

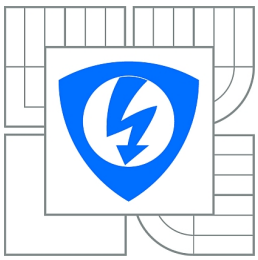
AUTOR PRÁCE — Bc. DANIEL MARINOV

AUTHOR

VEDOUCÍ PRÁCE — Ing. RADOVAN HOLEK, CSc.

SUPERVISOR

BRNO 2012



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav automatizace a měřicí techniky

Diplomová práce

magisterský navazující studijní obor
Kybernetika, automatizace a měření

Student:	Bc. Daniel Marinov	***ID:***	106619
Ročník:	2	***Akademický rok:***	2011/2012

NÁZEV TÉMATU:

Využití frameworků pro návrh databázové www aplikace.

POKYNY PRO VYPRACOVÁNÍ:

1. Seznamte se s prostředím Zend Framework s plným využitím MCV architektury.
2. Zaměřte se na procesy, které umožní automatizovat vytváření částí výsledné aplikace.
3. Navrhněte vhodný soubor tříd a metod pro prostředí Zend Framework, které umožní automatizované generování částí výsledné aplikace.
4. Soubor tříd a metod ověřte vygenerováním vzorové aplikace, využívající nově vytvořené třídy a metody.
5. Srovnejte přínosy Vašeho řešení při automatickém genrování modulů aplikace proti předchozím možnostem prostředí Zend Framework.

DOPORUČENÁ LITERATURA:

Maslakowski M., Naučte se MySQL za 21 dní. Praha: Computer Press, 2001. 478 s. ISBN 80-72226-448-6.
Brázda J., PHP 5 začínáme programovat. Praha: Grada Publishing a.s., 2006. 244 s. ISBN 80-247-1146-X.
Bohm, M. Zend Framework - Programujeme webové aplikace v PHP. vyd. Brno: Komputer Press, 2010. 416 s. ISBN 978-80-251-2965-4.
Dle pokynů vedoucího práce a vlastního výběru.

Termín zadání:	6.2.2012	***Termín odevzdání:***	21.5.2012

Vedoucí práce: Ing. Radovan Holek, CSc.
Konzultanti diplomové práce:

doc. Ing. Václav Jirsík, CSc.
Předseda oborové rady

UPOZORNĚNÍ:

Autor diplomové práce nesmí při vytváření diplomové práce porušit autorská práva třetích osob, zejména nesmí zasahovat nedovoleným způsobem do cizích autorských práv osobnostních a musí si být plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č.40/2009 Sb.

Abstrakt

Cílem této práce je seznámit se s technologiemi nutnými k sestavení internetové aplikace, která bude částečně generována prostřednictvím generátoru. Těmito technologiemi jsou především PHP framework s MVC architekturou, XML a databázový systém MySQL. Cílem není pouze seznámení se s technologiemi ale i vytvořit generátor.

Klíčová slova

MySQL, PHP, Zend Framework

Abstract

The aim of this work is to introduce with the technology necessary to build Internet application, which will be partially generated by the code generator. These technologies are primarily PHP framework with MVC architecture, XML and MySQL database system. The goal is not only introduce with the technology but also the preparation of the generator.

Keywords

MySQL, PHP, Zend Framework

Bibliografická citace:

MARINOV, D. Využití frameworků pro návrh databázové aplikace. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2012. 59s. Vedoucí diplomové práce Ing. Radovan Holek, CSc.

Prohlášení

„Prohlašuji, že svou diplomovou práci na téma Využití frameworků pro návrh databázové www aplikace jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.
Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

V Brně dne: **18. května 2012** ……
podpis autora

Poděkování

Děkuji vedoucímu diplomové práce Ing. Radovanu Holkovi, CSc. za účinnou metodickou, pedagogickou a odbornou pomoc a další cenné rady při zpracování mé diplomové práce.

V Brně dne: **18. května 201**2 ……

podpis autora

OBSAH

Seznam Obrázků

Seznam výpisů kódu

1 ÚVOD

V této práci navrhnu a implementuji generátor pro Zend Framework, který bude umožňovat automatické generování části kódu aplikace na základě informací získaných z databáze a od programátora. Vygenerovaný kód bude představovat kostru aplikace, kterou bude programátor dále rozvíjet.

Hlavní výhodou je tedy usnadnění práce programátora tím, že nebude nutné ručně implementovat všechen rutinní kód, ale nepozměněný generovaný kód netvoří plně funkční aplikaci, takže je nutná další práce programátora.

Práce nezahrnuje pouze vytvoření generátoru kódu, ale i připravení dalších obecných součástí aplikace doplňujících uživatelskou knihovnu.

1.1 Využité technologie

Za základní technologie v prostředí internetu můžeme pokládat nejčastěji používané. Těmito technologiemi jsou PHP a MySQL, ale rozhodně nelze říci, že pouze s pomocí těchto dvou technologií si vystačí vývojář. Škála možností, kterými můžeme tvořit internetové aplikace je značně široká. Proto uvedu pouze seznam dnes nejvíce skloňovaných technologií:

PHP

MySQL

Oracle

SQL server

CSS

XML

Ruby

JavaScript

AJAX

XSL

XHTML

jQuery

2 PHP

Počátky PHP se datují rokem 1995, kdy jeden nezávislý programátor jménem Rasmus Lerdorf vyvinul jistý skript Perl/CGI, který mu umožňoval zjistit, kolik návštěvníků četlo jeho online résumé. Jeho skript protokoloval informace o návštěvnících a zobrazoval na webové stránce počet návštěvníků. Protože web na rozdíl od toho jak ho známe dnes, byl tehdy ještě v plenkách a neexistovaly žádné takové nástroje; proto tento čítač vyvolal záplavu emailů se žádostmi o Lerdorfovi skripty. Lerdorf proto začal svou sadu nástrojů dávat k dispozici, a opatřil ji titulem Personal Home Page (PHP).

Halasné dožadování se sady nástrojů PHP přimělo Lerdorfa, aby začal vyvíjet dodatky k PHP. Jeden z nich měl převádět data předaná do formuláře HTML do symbolických proměnných, aby je uživatelé mohli exportovat na jiné systémy. Aby toho dosáhl, pokračoval ve vývoji v kódu C, a Perl opustil. Průběžné přidávání dodatků do sady nástrojů PHP kulminovalo v listopadu 1997 vydáním PHP 2.0, neboli Personal Home Page – Form Interpreter (PHP-FI). Důsledkem trvale rostoucí popularity PHP bylo, že verze 2.0 byla provázána řadou nejrůznějších zobecnění a zdokonalení od programátorů po celém světě. [4]

2.1 PHP 5 [4]

Verze 5 je další zlomový předěl v evoluci jazyka PHP. I když měla předchozí hlavní vydání enormní počet nových knihovních dodatků, verze 5 přesto zdokonaluje existující funkcionalitu. Přidává také několik nových schopností, které běžně provázejí architektury vyzrálého programovacího jazyka:

- Zdokonalená výbava objektově orientovaného programování
- Zpracování výjimek ve stylu try/catch
- Zdokonalené zpracování řetězců
- Zdokonalená podpora XML a webových služeb
- Vlastní podpora SQLite

2.2 Framework

Framework je soustava bloků programového kódu, který vytváří podporu při programování a vývoji jiných aplikací. Můžeme tedy říci, že kód obsahující framework je z velké části kódem nejčastěji používaným při vytváření běžných aplikací dané technologie. Framework je možné použít jako knihovnu kódu, ale zároveň framework může zavádět do aplikace nejrůznější architektonické vzory a dokonce může tvořit jádro naší aplikace v nejobecnější podobě.

Cílem frameworku je převzetí běžných, rutinních činností programátora tak aby urychlil vývoj aplikace.

Nevýhodou frameworku je vždy maximální robustnost kódu, která může vést ke snížení rychlosti aplikace. Za nevýhodu se dá také považovat delší doba prvního nasazení způsobená nutností studovat nový rozsáhlý systém frameworku. Opravdová síla tkví v opakovaném používání na rozsáhlých projektech.

2.2.1 Zend Framework

Mezi hlavní výhody Zend frameworku patří:

Množství komponent pro rutinní úlohy

Vysoká efektivita práce při větších projektech

Velká rozšiřitelnost

Kvalitní dokumentace

Profesionální podpora

Velké množství uživatelů

Licenční politika

AJAX Web 2.0

2.2.2 Prado

Tento framework je založený na komponentovém přístupu a programování událostí. Podporuje širokou škálu databází, podobně jako další zmíněné frameworky. Jeho přístup programování pomocí komponent a událostí trochu může mást při pokusu o návrh MVC aplikace. Dokumentaci k tomuto frameworku hodnotím jako méně přehlednou a obsáhlou v porovnání s dalšími frameworky.

2.2.3 Netee

Netee framework má široké možnosti, ale není zatím tak robustní jako jiné frameworky se silnějšími partnery nebo delší historií. U tohoto frameworku byla dlouhou dobu jeho slabinou nevalná dokumentace, která však byla v české verzi díky rodišti autora. Mezi jeho výhody patří zpracování šablon, které u jiných frameworků mnohdy chybí.

2.2.4 Pear

Archiv PEAR uchovává na jednom místě mnoho standardně používaného kódu, čímž ulehčí práci vývojářům. Kód sdružený v tomto archivu je rozdělen do nejrůznějších balíčků podle svého zaměření. Tyto balíčky mohou mít vzájemné závislosti, které je nutné respektovat. Každý balíček PEAR obsahuje popis všech jeho funkcí s jednotnou strukturou.

2.2.5 Porovnání knihoven kódu

Vzájemné porovnávání jednotlivých frameworků v tomto případě není příliš nutné, protože jejich společnou výhodou je možnost snadné implementace vlastních nebo převzatých knihoven kódu. Při tvoření nové aplikace je tak možné vytvořit zcela nový framework skládáním jednotlivých tříd z různých zdrojů samozřejmě s dodržením závislostí. Takovýto nový framework je možné navrhnout s větším úsilím i s podporou MVC návrhového vzoru.

Jedna z často vytýkaných nevýhod frameworků je jejich rychlost což je nejčastějším důvodem k vytvoření vlastního frameworku s méně rozsáhlou implementací zaměřenou pouze na daný projekt.

2.3 Objektově relační mapování (ORM)

Rozhodl jsem se kapitolu o ORM umístit do kapitoly PHP, protože tvoří jakýsi logický most mezi relační databází a objektově orientovaným jazykem jako je PHP.

Způsobů jak si každý může vytvořit vlastní ORM je celá řada, ale vždy se jedná o ukládání a roztřízení dat z relační databáze do objektů. Objekty by samozřejmě měli svými metodami umožňovat změnu takto získaných dat a zpětné uložení do databáze.

Výhodou tohoto přístupu je, že pokud si dobře napíšete ORM, už nemusíte pracovat se SQL, ale pracujete s metodami objektů ORM, které v sobě zapouzdřují databázové dotazy. V konečném důsledku to znamená, že při použití ORM nemusíte k žádnému projektu vytvářet databázové dotazy, ale používáte metody, které už většinu rutinních operací provádí samostatně. Rutinními operacemi zde myslím především správu cizích a primárních klíčů ve všech tabulkách. Hlavní výhodou, kterou zde popisuji, tu již nepřímo několikrát zazněla a její obecnost ORM. Stačí pouze jedno dobře napsané ORM pro jakoukoli databázi.

V dnešní době jsou již součástí téměř každého frameworku třídy, které svými instancemi jsou určeny k reprezentaci dat relační databáze (tabulky). MVC model je následně sestaven pomocí těchto objektů, které jako vstupní informaci vyžadují všechny primární a cizí klíče obsažené v dané tabulce.

Hlavní nevýhodou kvalitního ORM je, že Vám značně zbobtná aplikace, čímž se stává pomalejší. U specializovaných aplikací jakými mohou být, aplikace s extrémními požadavky na rychlost nemusí být schopno ORM vyhovět.

3 JAVASCRIPT V ZEND FRAMEWORKU

3.1 Dojo a Jquery

Dojo je javaScriptový framework, který má přímou podporu v Zend Frameworku prostřednictvím komponenty Zend_Dojo. Komponenta Zend_Dojo se stará o připojení javaScriptových souborů a inicializaci objektů, což umožňuje přistupovat k celé řadě view helperů i rozšíření Zend_Form. [2]

Kromě oficiální podpory Dojo Toolkitu poskytuje Zend Framework i komponentu na práci s jQuery. Tato komponenta – ZendX_JQuery – je vyvíjena komunitou. Kromě toho je možné Zend Framework použít i s jinými javaScriptovými knihovnami, jako jsou Prototype, YUI nebo další. [2]

Pro použití jQuery v Zend Frameworku je nutné knihovnu doplnit o adresář ZendX, který je dostupný při stažení plné verze Zend Frameworku. Adresář zendX obsahuje zejména propojení jQuery s elementy a další rozšíření, ale neobsahuje vlastní soubor s jQuery ten je nutné získat odděleně ze stránek www.jquery.com [8]. Soubor je nutné umístit do adresáře public/js.

3.1.1 Použití

Většina internetových aplikací dnes využívá technologii javaScript a proto i generátor provádí inicializaci komponent Zend Frameworku využívající tuto technologii.

Ukázková Aplikace pracuje s články, a proto vyžaduje textový editor pracující v prostředí internetu umožňující podobné operace, které jsou obvyklé v kancelářských aplikacích. Všechny tyto editory vyžadují použití některé javaScriptové knihovny.

Druhým příkladem využití komponent Zend Frameworku podporující javaScript je okno kalendáře umožňující standardním pohodlným způsobem vyplňovat datum.

4 DATABÁZOVÝ SYSTÉM

V následujícím textu uvedu pouze databázový systém MySQL, který je použit v ukázkové aplikaci. MySQL je jedna z nejrozšířenějších databází v prostředí internetu a to především dík spojení jazyka PHP a otevřené licenční politiky.

4.1 MySQL

Název SQL je zkratkou sousloví Structured Query Language. Je to akronym používaný výhradně v souvislosti s komunikací s databázemi. Každý větší databázový systém používá dotazovací jazyk SQL a ani MySQL není výjimkou. [1]

MySQL je nejoblíbenější otevřenou databázovou aplikací a velmi často se používá právě společně s jazykem PHP. Software MySQL se skládá z databázového serveru, klientské aplikace a z několika dalších pomocných programů.

5 ARCHITEKTONICKÉ VZORY

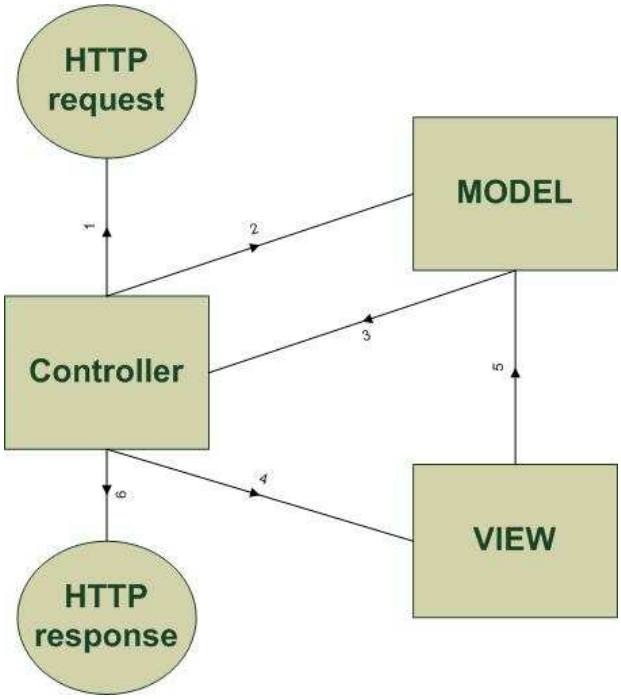
Mezi architektonické vzory patří právě MVC, označení architektonický vzor vyplývá ze skutečnosti, že MVC je tvořeno návrhovými vzory.

5.1 MVC architektura

Model View Controller architektura (MVC) je nový přístup k návrhu a implementaci internetových aplikací. Jedná se o rozdělení aplikace do tří bloků: data (model), vzhled (view) a řízení (controller). Každý z těchto bloků spoluvytváří aplikaci.

5.2 Principy MVC architektury v Zend Frameworku

Obrázek 1 představuje proces zpracování http požadavku na server.



Obrázek 1: Průběh požadavku v MVC architektuře [2]

Začátek každého požadavku je odeslání HTTP požadavku na server. Tento požadavek je řadičem přijat a dále zpracováván. [2]

Z požadavku řadič zjistí, který modul musí být v případě potřeby osloven, a požaduje od něho údaje. [2]

V některých případech předá řadič modelu údaje získané od uživatele a požaduje jejich uložení. V tomto případě obdrží řadič od modelu odpověď o úspěchu respektive neúspěchu, vykonané operace. [2]

Řadič zjistí pohled potřebný na výpis údajů a předá mu údaje, které mají být vypsány.[2]

V závislosti na implementaci se může stát, že se pohled dožaduje údajů od modelu, respektive model informuje pohled o změnách údajů. [2]

Na závěr vytvoří pohled výpis údajů (například ve formě HTML stránky) a řadič ho pošle zpět do prohlížeče uživatele ve formě HTTP odpovědi. [2]

5.3 Funkce modelu v MVC

Model v MVC architektuře tvoří most přístupu k datům a vytváří rozhraní mezi systémem úložiště a vlastní aplikací. Prostřednictvím modelu je umožněno aplikaci přistupovat k datům a popřípadě je měnit, takovým způsobem aby aplikace nemusela mít bližší informace o fyzickém uložení a bezprostředních aplikačních vrstvách úložiště. Důsledkem tohoto přístupu je jednotný přístup k datovým úložištím různého druhu.

Důležité je si uvědomit, že model nemusí pouze umožňovat přístup k databázi, ale k jakémukoli úložišti jakým může být:

Databáze

Textový soubor

RSS

Webová služba

Řadič ani pohled nemají žádné tušení, jak a v jaké podobě jsou fyzicky ukládána data nebo jakým způsobem komunikuje model s úložištěm.

Řadič a pohled, ale musí správně interpretovat data přijatá od modelu a model musí správně reagovat na žádosti o data od řadiče.

5.4 Funkce pohledu v MVC

Pohled je ta část MVC aplikace, která vytváří prezenční vrstvu. Hlavním úkolem pohledu je přijímat data od řadiče a zobrazovat jejich upravenou podobu uživateli. Přijímaná data musí mít odpovídající formát, ale informace odkud data pocházejí nebo jak byla získána, pohledu nepřísluší.

Pohled může mít vlastní prezenční logiku, například pro výpis seznamů v tabulce nebo pro změnu barev v řádcích tabulky. Také je možný přístup k metodám objektů, které vracejí údaje. Pohled by však měl být úplně izolovaný od ostatní business logiky,

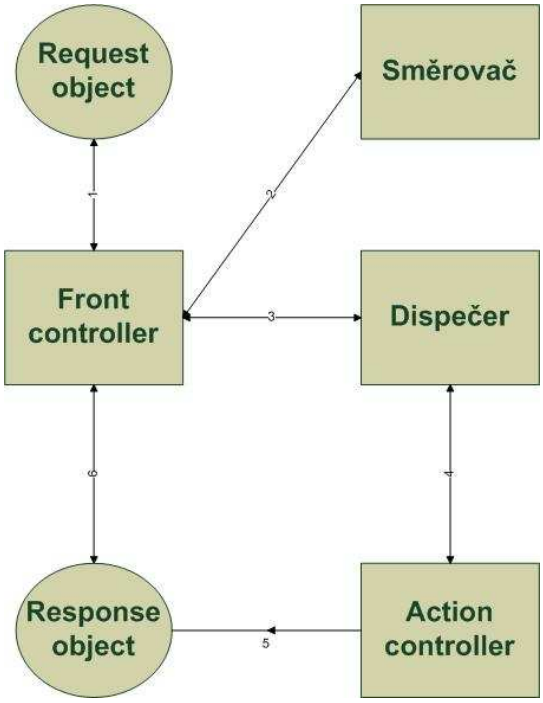
která například aktualizuje nebo maže údaje, odesílá emaily nebo přihlašuje a odhlašuje uživatele.

Výpis pohledu v MVC aplikaci se ve většině případů realizuje ve formě HTML stránky, která je odeslána do prohlížeče uživatele, který ji potom zobrazí. Tento výpis může být i ve formě CSV nebo XML souboru, které jsou uživateli nabídnuty ke stažení. Pohled též může vytvořit nějakou grafiku nebo PDF soubor, které jsou odeslány do prohlížeče. Nakonec může též pohled vypsat RSS. Jakákoliv forma výpisu je do prohlížeče uživatele odeslána v HTTP odpovědi (HTTP response). [2]

5.5 Funkce řadiče v MVC

Řadič je zodpovědný za řídící vrstvu v MVC architektuře. Požadavek na řadič je vykonán odesláním HTTP žádosti (HTTP request) z prohlížeče uživatele. Z požadavku řadič zjistí, který model musí být v případě potřeby osloven a který pohled má vykonat výpis údajů. Odpověď je na konci odeslána do prohlížeče uživatele ve formě HTTP odpovědi (HTTP request).

Řadič v MVC aplikaci se skládá z více dílčích komponent, které implementují různé návrhové vzory. Nejznámější z nich jsou front controller, action controller a návrhový vzor příkaz (command pattern). V závislosti na implementaci může být interakce mezi těmito dílčími komponentami více či méně komplexní. Na obrázku 2 je zobrazen průběh požadavku v řadiči. Stejně je to realizováno i v Zend Frameworku. Klíčovou komponentou je Front Controller, který příjme přicházející požadavek, zpracuje ho a osloví další dílčí komponenty. [2]



Obrázek 2: Průběh požadavku v řadiči [2]

HTTP požadavek uživatele je zapouzdřený request v objektu, a proto je možné k údajům v tomto požadavku přistupovat pomocí tohoto centrálního objektu. V průběhu zpracování mohou k request objektu přistupovat směrovač, dispečer a action controller ať už pro čtení nebo zápis. [2]

V dalším kroku předá front controller zpracování směrovači. Ten odvodí z požadavku, jaká akce nebo jaké akce mají být vykonány. [2]

Následně front controller pověří dispečera, aby vykonal další zpracování požadavku. [2]

V rámci cyklu volá dispečer postupně action controller za účelem vykonání uživatelem požadovaných akcí. Action controller může být v případě potřeby osloven i model. [2]

Výsledek je předán response objektu. [2]

Na závěr získá front controller opět kontrolu nad řízením a vyzve response objekt, aby odeslal odpověď do prohlížeče uživatele ve formě HTTP odpovědi. [2]

6 NASTAVENÍ A STRUKTURA APLIKACE

6.1 Adresářová struktura aplikace

Adresářová struktura aplikace využívající Zend Framework má standardní adresářovou strukturu a generátor kódu ji využívá s menším doplněním adresářů. Hlavní adresář aplikace má následující strukturu:

- Application
- Configs
- Controllers
- Forms
- Layouts
- scripts
- Models
- DbTable
- Form
- Views
- Helpers
- Scripts
- Controller name
    - Data
    - docs
    - Library
    - Public

Application - adresář obsahuje hlavní část aplikace vytvořenou uživatelem, ve které se nalézají následující podadresáře.

Configs - obsahuje především soubor configs.ini, který nese inicializační nastavení pro Zend Framework.

Forms - obsahuje, jak název napovídá formuláře, nicméně v aplikacích vytvářených pomocí tohoto generátoru obsahuje pouze ini soubory sloužící k sestavení formulářů.

Controllers - obsahuje soubory se třídami reprezentující controllery.

Layouts - nese informace pro vytvoření layoutu (vizuálního rozdělení stránky).

Models - patří mezi nejobsáhlejší adresáře, protože v podadresáři DbTable jsou uloženy třídy reprezentující databázové entity (tabulky) a jejich vzájemné vztahy.

Forms - obsahuje třídy formulářů, sestavujících výsledné formuláře na základě informací v konfiguračních ini souborech uložených v adresáři Form.

Views - obsahuje základní pohledy pro každý controller uložené v adresáři scripts. Podadresář helpers obsahuje view helpery.

Data - slouží k ukládání logů a dalších průběžných informací aplikace.

Library - knihovna aplikace

Public - obsahuje spouštěcí soubor (index.php), css styly, javaScripty a další uživatelsky orientované skripty.

6.2 Nastavení Zend Frameworku

Komponenta Zend_Application zajišťuje zavádění aplikace pro zdroje, jako jsou databázové adaptéry a Bootstrap třída. Prostřednictvím této komponenty se také provádí nastavení samotného PHP. [2]

Vlastní nastavení prostředí je možné provádět třemi způsoby, kde první možností je veškeré nastavení umístit do konfiguračního souboru application/configs/application.ini. Druhý způsob využívá Bootstrap třídy, kde v metodách objektu dochází k inicializaci a poslední možností je kombinace obou způsobů.

6.3 Nastavení konfiguračního souboru

Výpis 6.1 zobrazuje konkrétní částečný výpis nastavení konfiguračního souboru, ve kterém je nastaveno zobrazování chybových hlášení, definování cest, sestavení databázového adaptéru a další nastavení.

```
phpSettings.display_startup_errors = 1
phpSettings.display_errors = 1
includePaths.library = APPLICATION_PATH "/../library"
bootstrap.path = APPLICATION_PATH "/Bootstrap.php"
bootstrap.class = "Bootstrap"
appnamespace = "Application"
resources.frontController.controllerDirectory = APPLICATION_PATH "/controllers"
resources.frontController.params.displayExceptions = 1

resources.db.adapter = "Pdo_Mysql"
resources.db.params.username = "root"
resources.db.params.password = "root"
resources.db.params.dbname = "myweb"
resources.layout.layoutPath = APPLICATION_PATH "/layouts/scripts/"
```

Výpis 6.1: Konfigurační soubor

Z výpisu 6.1 již je zřejmý možný rozsah nastavení, který podrobněji ozřejmí dokumentace Zend Frameworku.

6.4 Bootstrap třída

Použití Bootstrap třídy je reprezentováno zejména souborem application/Bootstrap.php, který zajišťuje konfiguraci Zend Frameworku. Z výpisu 6.2 je patrné, že jednotlivé bloky nastavení mohou být umístěny v jednotlivých inicializačních metodách, které jsou prováděny při vytváření objektu. [2]

```
class Bootstrap extends Zend_Application_Bootstrap_Bootstrap
{
        protected function _initConfig()
        {
        $autoloader=Zend_Loader_Autoloader::getInstance();
        $autoloader->getRegisteredNamespaces("MyWeb_");
        }
}
```

Výpis 6.2: Bootstrap třída.

V tomto výpisu je Bootstrap třída s metodou registrující jmenný prostor.

7 DATABÁZE APLIKACE

7.1 Konvence databáze

Aplikace, která využívá vytvořený generátor, musí splňovat některé konvence, kterými jsou především správné pojmenování tabulek a jejich sloupců.

- Všechny názvy tvořící součást databáze mohou obsahovat alfanumerické znaky a znak podtržítka.
- Každé jméno musí začínat písmenem.
- Spojovací tabulka řešící vazbu M:N má název skládající se z názvů spojovaných tabulek spojených prostřednictvím slova has. Například: ArticleHasSubject spojující tabulky Article a Subject.
- Sloupec primárního klíče je vždy pojmenován id_jménoTabulky. Výjimku tvoří primární klíč tabulky spojovací, kde název se skládá ze jména spojované tabulky a jména jejího primárního klíče. Například: article_id_article a subject_id_subject.
- Jméno sloupce cizího klíče je utvořeno ze jména cizí tabulky a jejího primárního klíče. Například: article_id_article.

7.2 Speciální jména tabulek

Speciální tabulky jsou tabulky, které je schopný generátor rozpoznat v databázi na základě jejich pojmenování a následně vygenerovat některé části kódu, které usnadní další implementaci celé aplikace. Tyto části kódu jsou specializovanější a využitelné pouze v kooperaci s danými tabulkami a proto jsou generovány pouze za existence určitých tabulek v databázi. V případě, že tabulky neexistují, nedojde k vygenerování těchto specializací, ale aplikaci to významně neomezí.

Na prvním místě se jedná o tabulky State, NameState a TabelDatabase se strukturou a vzájemným propojením popsaným v části 11.4 a na obrázku 8. Existence těchto tabulek vede ke změně SQL dotazů modelů, které začnou kontrolovat stav, a také bude možné v programu využívat funkce na ověření existence stavu.

V druhém případě se jedná o tabulku role s vazbou M:N, která umožní využívat třídy pro autentizaci a funkce pracující s pojmem role. Administrace uživatelských rolí využívá komponent Zend_Acl a Zend_Auth.

8 IMPLEMENTACE GENERÁTORU

V této kapitole je popsaný výsledek návrhu generátoru. Cílem snažení je vytvořený generátor kódu php pro Zend Framework. Generátor na základě validní databáze vygeneruje php kód, který se spojí s již uživatelem vytvořenou základní adresářovou strukturou projektu v Zend Frameworku. Generátor umožňuje jak vygenerovat celý kód ekvivalentní databázové struktuře tabulek, tak jednotlivé soubory pro jednotlivé tabulky. Například je možné vygenerovat pouze dané view pro danou databázovou entitu.

Generátor generuje kód pro určitou strukturu projektu, která je pevně dána, a proto používání pro jiné návrhové struktury projektu je značně neefektivní.

8.1 Základní struktura generátoru

Generátor je tvořen sadou tříd reprezentující jednotlivé generované třídy a soubory. Nad každou takovou třídou nebo jejich skupinou je definována abstraktní třída zajišťující společné funkcionality pro své potomky. Každá takováto abstraktní třída je potomkem třídy MyWeb_File_File, která sjednocuje funkcionality na vyšší vrstvě, což je především ukládání do souboru. Třída MyWeb_File_File je potomkem třídy CreatorFile, která zajišťuje možnost automatického vytváření souborů pro všechny entity databáze, ale i tvorbu jednotlivých souborů. Dále je tu třída MyWeb_DbTable_DbTable, která reprezentuje databázovou tabulku. Zbývající třídy doplňují vygenerovaný kód.

- MyWeb
  - Controller
    - Pagin
      - ActionSetup.php
      - Authentication.php
      - ConverterDate
      - Paginator
  - DbTable
    - CreatorDbTable.php
    - DbTable.php
    - Exception.php
  - File
    - Controller
      - Abstrakt
      - Controller
    - DbTable
      - Abstrakt
      - DbTable
    - Form
      - Abstrakt, AbstractIni, Edit, EditIni, Login, LoginIni, Logout, New, NewIni, Search, SearchIni
    - Languages
      - Languages
    - Model
      - Abstrakt
      - Model
    - View
      - New.php, Edit.php, List.php, …
    - CrreatorFile
    - File
    - Exception
  - JavaScript
    - Popup
      - Abstrakt
      - Main
      - Slave
  - Loader
    - ModelLoader
  - Log
    - ApplicationLogger
  - Model
    - Abstrakt
    - Database
    - Interface
    - Exception
  - Acl
  - Form
  - Plugin

Výpis 8.1: Adresářová struktura

8.2 Třída MyWeb_File_File

Třída MyWeb_File_File zajišťuje vytvoření odpovídajících adresářů a u uložení php souborů. Cesta pro vytvoření adresáře nebo php souboru je zadána uživatelem nebo je použita cesta defaultně nastavená pro daný typ souboru.

Existuje několik typů souborů, které jsou odlišeny svým jmenným prefixem a tedy i umístěním v adresářové struktuře. Prefixy souborů jsou odvozeny zejména z logického rozdělení generované aplikace, která je podřízena MVC architektuře.

Seznam Základních prefixů:

- Prázdný prefix
- New
- Edit
- Search
- List
- Index
- Controller
- List2

8.2.1 Třída CreatorFile

Třída CreatorFile zajišťuje automatické nebo ruční vytváření jednotlivých souborů aplikace. Vstupní proměnnou je pole obsahující seznam souborů, které se mají vygenerovat pro daný seznam databázových tabulek reprezentovaných instancí třídy DbTable. Na základě takto definovaného vstupu jsou vytvářeny jednotlivé soubory, případně složky prostřednictvím tříd odvozených z výše popsané třídy MyWeb_File_File.

8.2.2 Třídy odvozené ze třídy MyWeb_File_File

Tyto třídy reprezentují již konkrétní generované soubory a všechny jsou si svou strukturou podobné, a proto popíši pouze třídu modelu, která je nejobsáhlejší. Základním společným rysem většiny těchto tříd je využití komponenty Zend_CodeGenerator, která umožňuje generování PHP kódu pomocí objektově orientovaného rozhraní.

Seznam tříd:

- Controller
- DbTable
- Form
  - Edit, Login, Logout, Search, List, New a ini třidy
- Model
- Languages
- View
  - Edit, Index, List, List2, Login, Logout, New, Search

8.2.2.1 Třída MyWeb_File_Model_Model

Instance třídy je zodpovědná za vytvoření modelu, pro který je nutné vytvářet databázové dotazy nebo metody umožňující jejich dynamické generování. Informace o databázi, jednotlivých tabulkách a jejich vzájemných vztazích jsou neseny v instancích třídy DbTable a třídami odvozenými ze třídy Zend_Db_Table_Abstract. Základními metodami pro vytváření modelu jsou create_join a createContent.

Metoda create_join vytváří join dotazu z informací obsažených v objektu třídy DbTable. Zatímco metoda createContent zajišťuje vlastní vygenerování php kódu třídy modelu, který je doplněn informacemi z konkrétní třídy odvozené z Zend_Db_Table_Abstract.

8.3 Log

Do různých metod aplikace je možné vložit řádky výpisu 8.2, které zajistí výpis hlášení a případné uložení výpisu do textového souboru data/logs/log.txt.

```
MyWeb_Log_ApplicationLogger::addLog($this,3,4);
MyWeb_plugin::showMe(MyWeb_Log_ApplicationLogger::showLog(),'LOG2');
```

Výpis 8.2: Použití logu

Zend Framework samozřejmě disponuje vlastní alternativou.

8.4 Model

Struktura modelu se skládá z rozhraní a dvou abstraktních tříd. Rozhraní je definováno nad abstraktními třídami. Při vzniku potřeby využívat nové úložiště musí být vytvořena nová abstraktní třída, která musí implementovat použité rozhraní. Výpis 8.3 ukazuje implementované rozhraní.

```
interface MyWeb_Model_Interface   {public function getStorage();
public function insert(array $data);
public function update(array $data, $id);
public function delete($id);
public function deleteMNroe($listMNtable,$column);//odstrani zaznamy z MN tabulky pro dane id
public function fetchEntry($id);
public function fetchEntries($data);
public function getFetchEntrySelect($data);
public function getIdState($stateName, $tableName); //vrati id stavu (pokud existuji tabulky)
public function getIdRole($roleName);                    //vrati id role (pokud existuji tabulky)
public function createWhere($select,$data,$mainTable);//vytvori sql where z dat odelaneho formulare
public function isExists($column, $value,$id);}
```

Výpis 8.3: Interface modelu

První abstraktní třída implementující rozhraní obsahuje metody pro prací s různými úložišti. Druhá abstraktní třída využívá třídu Zend_Db_Table reprezentující databázovou tabulku, nad kterou jsou implementovány metody rozhraní. Třídy s modely jsou uloženy v adresáři application/models což není standardní cesta a proto je nutné zajistit jejich načítání samostatnou třídou, která dědí ze třídy Zend_Loader_PluginLoader.

Účel metod použitých v modelu je zřejmý z názvu a popisu ve výpisu 8.3.

8.5 Autentizace a řízení přístupu uživatelů

Autentizace a schopnost řízení přístupu bylo z větší části převzato ze zdroje [2], kde je využito komponent Zend_Auth a Zend_Acl.

Zend_Acl je zodpovědná za autorizaci a poskytuje odpověď na otázku: Jaká oprávnění má aktuální uživatel? Zend_Auth zajišťuje autentizaci a odpovídá na otázku: Kdo je aktuální uživatel? [2]

K přihlášení do aplikace se používá přihlašovací formulář. Použitý formulář se sestává z dvou vstupních políček username a passwd, a tlačítka submit_login.

Při použití komponent Zend_Auth a Zend_Acl spolu se Zend_Controller se dá do cíle dojít více cestami. Na autorizaci a autentizaci je možné použít přímo Bootstrap soubor nebo vytvořit zásuvný modul, který zajistí veškeré operace s tím spojené. Poslední možností je sestrojení action helperu, který se stará o přihlášení a odhlášení.[2]

V aplikaci je použit zásuvný modul zajišťující autorizaci a autentizaci uživatelů. Výhodou tohoto řešení je možnost využít jej na více projektech. [2]

Autentizace využívá databázového adaptéru Zend_Auth_Adapter_DbTable, který získává data právě ze speciální tabulky user a role. [2]

8.5.1 Přístupové seznamy

Přístupové seznamy (anglicky Access Kontrol List, ACL) je možné vytvářet pomocí Zend_Acl. Tato komponenta je zodpovědná za autorizaci a obsahuje informace, kdo může co dělat. [2]

Přístupový seznam vytvořený pomocí komponenty Zend_Acl se skládá z následujících prvků: uživatelská role, zdroje, oprávnění a pravidla. Příklad takto vytvořeného přístupového seznamu zobrazuje výpis 8.4.

Ukázková aplikace využívá přístupové seznamy definované pomocí ini souboru, který je umístěn v application/configs/acl/default.ini.

Třída MyWeb_Acl je odvozená ze třídy Zend_Acl zajišťuje načtení konfiguračního souboru Acl a byla převzata ze zdroje [2].

```
[roles]
1 = guest
2 = editor
3 = administrator

[ressources]
1 = article
2 = user
3 = cement

[rules]
;********administrator********
allow.administrator.article = all
allow.administrator.user = all
allow.administrator.comment = all
```

Výpis 8.4: Konfigurační soubor Acl

8.6 Loader

Třída MyWeb_Loader_ModelLoader pouze umožňuje načítání modelů ze složky models, která není standardně určena pro ukládání modelů. Tato třída byla také převzata ze zdroje [2].

8.7 Plugin

Plugin je třída umožňující metodou showMe vypisovat jednotlivá hlášení z kódu podle jejich typu. Funkce je používána například místo příkazu echo.

8.8 Třída DbTable

Třída DbTable je prvotním hlavním nositelem informací o databázi a jejich jednotlivých tabulkách bez informací o vzájemných vztazích tabulek. Propojení s databází je nezávislé na adaptéru a jeho informacích využívaných Zend Frameworkem. Hlavní funkcí třídy je získání a uspořádání informací ekvivalentních příkazu describe použitém pro všechny tabulky obsažené v databázi.

Na základě informací obsažených v těchto objektech mohou být v jednom z prvních kroků generování vytvořeny třídy odvozené z Zend_Db_Table_Abstract, které jsou hlavním nositelem informace o databázi a to i o vzájemných vztazích tabulek.

8.9 Generování javaScriptu

JavaScript je součástí generované aplikace, aby bylo možné, zajistit funkcionalitu vygenerované aplikace umožňující výběr informace ve vyskakovacím okně, která se při výběru přenese do okna hlavního. Ve skutečnosti se nejedná o generování javaScriptu ale o generování php kódu, který generuje javaScript. Tento postup umožňuje provést nastavení generování javaScriptu například v controlleru a vygenerovaný kód předat pohledu. Tento javaScript využívá knihovnu javaScriptu umístěnou v public/js, kde mohou být umístěny další skripty javaScriptu.

Generovaný php kód je tvořen dvěma soubory, kde jeden je určen pro hlavní okno (tj. okno ze kterého je voláno vyskakovací okno) výpis 8.5 a druhý pro okno vyskakující (List2) výpis 8.6.

Generován je i třetí soubor, který obsahuje pouze odkazy s parametrem id odkazu příslušející elementu formuláře.

Bližší informace o nastavení javaScriptu v controlleru jsou popsány výše.

```
$(function(){
	$.initWindowMsg();
	var childWin;

	jQuery.isSubstring = function(haystack, needle) {
	return haystack.indexOf(needle) !== -1; };

	var pathname;
	$(document).ready(function() {
		pathname = window.location.pathname;
	});

	if( $.isSubstring(pathname, "new") && $.isSubstring(pathname, "article")) {
		$('#openWin0').click(function(){
		childWin = window.open('list2','okno', "width=500, height=500, location=no,
		menubar=no, scrollbars=no, status=no, toolbar=no");
		if (window.focus) {childWin.focus()}});
	};
	$.windowMsg("childMsg0", function(message) {
		$('#State_id_stateValue').val(message);
	});

	$.windowMsg("childHidden00", function(message) {
		$('#State_id_stateInputValue').val(message);
	});

	$.windowMsg("childHidden01", function(message) {
		$('#nameState_nameInputValue').val(message);
	});
```

```
$.windowMsg("childHidden02", function(message) {
	$('#State_id_stateInputName').val(message);
});

$.windowMsg("childHidden03", function(message) {
	$('#nameState_nameInputName').val(message);
});});
```

Výpis 8.5: Generovaný javaScript pro main okno

```
$(function(){
		$.initWindowMsg();
		// maximalni pocet odkazu v list2 pro odeslani hodnoty do formulare je:
		//		maximalni pocet radku tabulky list2 = 30
		//		maximalni pocet sloupcu tabulky s odkazy = 4
		//		celkem = 30 * 4 = 120


	// upon click of a button, send corresponding text input value to parent window
	//funkce slouzi k overeni existence substringu
	jQuery.isSubstring = function(haystack, needle) {
	return haystack.indexOf(needle) !== -1; };

	var pathname;
	// funkce naplni promenou pathname hodnotou url
	$(document).ready(function() {
		pathname = window.location.pathname;
		//document.write(pathname);
	});



	$('#button0').click(function(){
		value = $('#in0').val();
		$.triggerParentEvent("childMsg0", value);
	});
```

Výpis 8.6: Generovaný javaScript pro slave okno

Vygenerovaný javaScript využívá knihovnu public/windowmsg/jquery.windowmsg-1.0.js, která je dostupná na adrese http://www.sfpeter.com/2008/03/communication-between-browser-windows-with-jquery-my-new-plugin/ [7].

Z výpisu 8.5 a 8.6 je patrná funkce javaScriptu, která pouze při dané události odesílá data ze strany odesilatele např. list2 na stranu příjemce např. edit. Na straně příjemce je pro každý „vysílací kanál“ jedna funkce očekávající vstup. Identifikace každého „kanálu“ je zajištěna pomocí identifikátorů.

8.10 View helpery

View helper je návrhový vzor, který zajišťuje rozšíření pohledu. Zend_View implementuje tento návrhový vzor a disponuje infrastrukturou k vytvoření a správě view helperů. Kromě toho Zend Framework nabízí již řadu vytvořených view helperů. [2]

V této krátké kapitole je uveden popis nejčastěji používaných helperů aplikací.

8.10.1 Zend_View_Helper_Table

Tento view helper slouží pro snadnější napojení komponenty Zend_Paginator ke generátoru.

```
$paginator = $this->PagerToPaginator($this->pager,$this->select);// prevod vstupnich dat pro pager
$head = NULL; //jmena sloupcu pokud je null jmena sloupscu jsou jak v databazi
//$head['nameHeader']=array(        $translate->translate('ID'),
                                    $translate->translate('Title'),
                                    $translate->translate('Date'),
                                    $translate->translate('Username'),
                                    $translate->translate('State'));
$selectData = array('id_table1','table1_column1','table1_column2','table1_column3','table3_id_table3');
$links['id_table1']['link'] = $this->url(array('controller' => $request->getControllerName(),
                                                'action' => 'edit'), $request->getModuleName(), true);
$links[current($selectData)]['get'][0] = current($selectData);
$columnValue = NULL; // funkce prida hidden pole k odkazovane bunce tabulky List2
$hidden = NULL; //prida hidden pole (vyuzito v List2)
$caption = array($translate->translate('table1'));
print $this->Table($paginator,$selectData,$columnValue,$head,$hidden,$caption,$links);
print $this->paginationControl($this->pager,'Sliding','paginator.phtml');
```

Výpis 8.7: Použití View helperu Table

Základní využití tohoto helperu je zobrazeno na výpisu 8.7 ze kterého jsou patrné vstupní informace pro vytvoření tabulky. Do pole links je uložena cesta pro odkaz v tabulce vytvořený view helperem url a klíč pole links, je tvořen jménem sloupce, ve kterém má být daný odkaz. Klíč get obsahuje jméno sloupce, jehož hodnota má být předávána prostřednictvím odkazu. Ve zbývajícím řádku výpisu je vytvoření tabulky se stránkováním prostřednictvím komponenty Zand_Paginátor.

Výpis 8.8 zobrazuje nastavení tabulky z controlleru, kde je vyžadován model, odeslané hodnoty a request objekt.

```
$paginator = new MyWeb_Controller_Plugin_Paginator();
$paginator->setModel($modelTable3);
$paginator->setData($this->getRequest()->getParams());
$paginator->setRequest($this->getRequest());
```

Výpis 8.8: Nastavení tabulky ze strany controlleru

Součástí vytvořené tabulky jsou i odkazy ve jménech sloupců umožňující seřazení záznamů.

8.10.2 View helpery pro výpis HTML hlavičky

Zend Framework obsahuje také standardní helpery k výpisu HTML hlaviček. Mezi nejpoužívanější patří:

- DocType - Vypíše typ dokumentu.
- HeadLink - Vypíše link tag pro css soubor.
- HeadScript - Vypíše script tag.
- HeadStyle - Vypíše style tag.

```
$this->headLink()->appendStylesheet($this->DinamicPath() .'css/form.css' ,'all');
```

Výpis 8.9: Výpis Html hlavičky pro css

8.10.3 View helper url

Tento view helper patří k hojně používaným, protože zajišťuje vytvoření cesty pro odkazy. Parametry tohoto helperu jsou jména controlleru, modulu a akce. Příklad jeho použití předvádí výpis 8.10.

```
$this->url(array('controller' => $this->controller, 'action' => $this->action);
```

Výpis 8.10: View helper url

8.11 CSS

Pro základní zobrazení formulářů, tabulek a layout jsou definovány jednoduché sady kaskádových stylů. Soubory obsahující kaskádové styly jsou uloženy v adresáři application/public/css. Popis tříd a id jednotlivých tagů nebude dále uveden, protože je snadno zjistitelný výpisem kódu stránky nebo zobrazením patřičné třídy generátoru.

8.12 Filtry a validátory

Komponenta Zend_Form neslouží jen k vytvoření a vykreslení formulářů, ale podporuje i filtrování a validaci údajů. V případě potřeby je možné využít třídu Zend_Filter i vlastní filtry, které implementují rozhraní Zend_Filter_Interface. Totéž platí i pro

Zend_Validate i pro vlastní validátory, které implementují rozhraní Zend_Validate_Interface.[2]

8.13 Filtry

Každému formulářovému prvku je možné přiřadit pomocí metody addFilter libovolný filtr a je přitom jedno zda ho přiřadíte před předáním prvku formuláři nebo poté co jste ho přiřadili. Kromě toho je možné pomocí metody addFilter přiřadit více filtrů. Výpis 8.11 demonstruje přiřazení filtru prvku formuláře definovaného prostřednictvím třídy a konfiguračního souboru. [2]

```
elements.article_text.Options.filters.trim.filter = "StringTrim";
$article_text ->addFilter(new Zend_Filter_Alpha(True));
```

Výpis 8.11: Přidání filtru v ini souboru formuláře

8.14 Validátory

Každému elementu je možné přiřadit pomocí metody addValidator také validátor. Práce s validátory je téměř totožná jako s filtry.

9 GENEROVANÉ SOUBORY

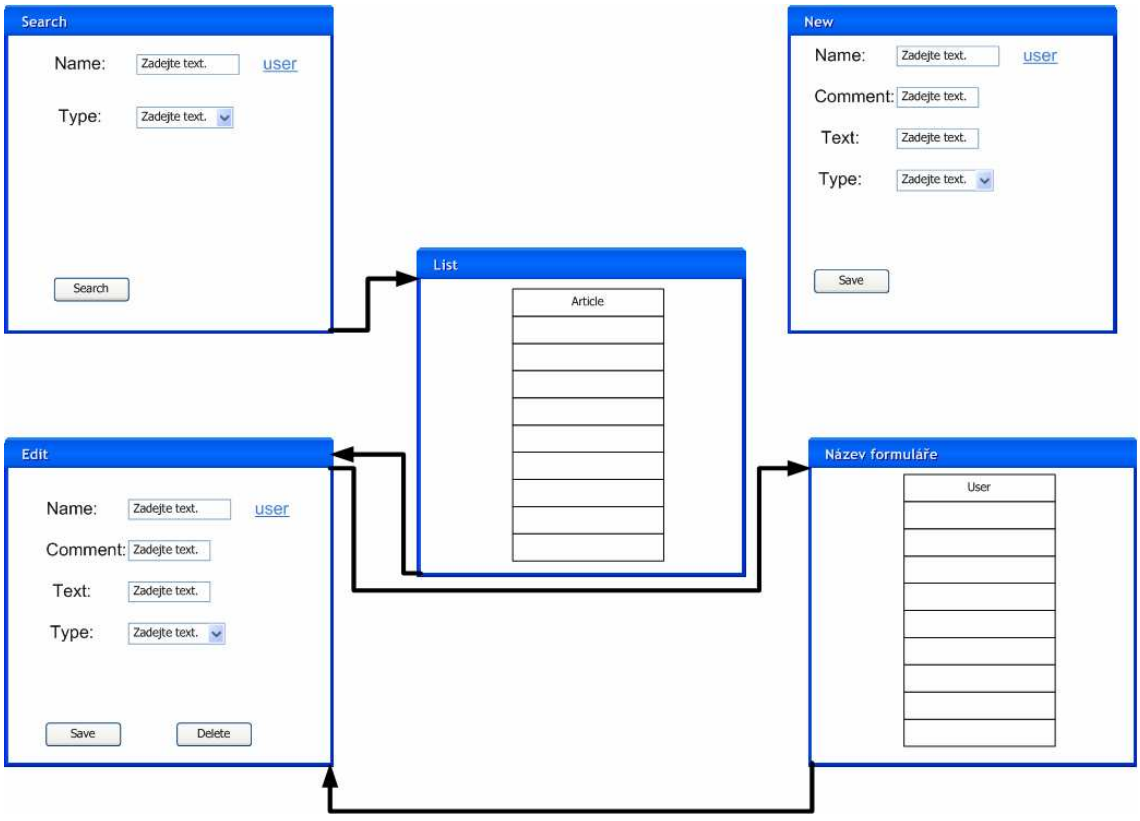
Každý generovaný soubor je možné dále upravovat a některé soubory to vyžadují. Generuje se:

- formEditIni, formSearchIni
- formNewPhp, formEditPhp, formSearchPhp
- model
- modelTable
- controller
- viewList, viewNew, viewEdit, viewIndex, viewSearch

9.1 Struktura vygenerované aplikace

Generování rozličných aplikací jedním generátorem vyžaduje vytvoření teoretické šablony projektů, která standardizuje prvky jednotlivých projektů takovým způsobem, že každý projekt je realizovatelný jako kolekce bloků s vysokou opakovatelností.

Uživatel se v aplikaci může pohybovat po cestě zobrazené na obrázku 3.



Obrázek 3: Základní struktura aplikace

Na obrázku jsou patrné čtyři základní bloky, ve kterých se uživatel pohybuje:

- Search - formulář pro vyhledávání a jeho obsluha
- List - stránka zobrazující výsledky hledání
- List2 - seznam hodnot s možností volby
- Edit - detail záznamu s možností jeho editace
- New - formulář pro vytvoření nového záznamu

Tato struktura bloků je protknuta celou aplikací a generátor je pro ni navrhnut.

9.2 Controller

Třída controlleru je vytvořena generátorem v základní podobě s metodami:

- init – metoda volná při vytvoření objektu
- indexAction
- listAction
- list2Action
- editAction
- searchAction
- newAction

Všechny metody zprostředkovávají zobrazení a zpracování dat pro view, které představují reakce na uživatelovi akce. Každá z těchto metod má k dispozici vlastní view vyjma metody init. Toto upořádání vychází z obrázku 3. Metody controlleru tedy akce jsou v dalších kapitolách podrobněji popsány, protože programátor musí s takto vygenerovaným kódem dále pracovat, nicméně se předpokládá znalost PHP a Zend Frameworku, proto nebude popis manuálem pro začátečníka.

9.2.1 List action

Na výpisu 9.1 je ukázka typické metody actionList, kde je načten model, pomocí něhož jsou získány požadované záznamy na základě vstupních údajů. Získané záznamy jsou následně předány komponentě Zend_Paginator určené pro vytváření stránkovaného výpisu tabulky záznamů. Výsledná data jsou předána pohledu k zobrazení.

```
$model = MyWeb_Model_Database::loadModel('');

$list = $model->fetchEntries($this->getRequest()->getParams());
$paginator = Zend_Paginator::factory($list);
$page = (int) $this->getRequest()->getParam('page',1);

$paginator->setCurrentPageNumber($page);
$this->view->paginator = $paginator;
$this->view->list = $list;
```

Výpis 9.1: List action

9.2.2 Edit action

Metoda edit je již o něco komplikovanější. Opět dojde k načtení modelu a prostřednictvím modelu je načten formulář.

```
$modelArticle = MyWeb_Model_Database::loadModel('Article');
$form = $modelArticle->getForm('update');
```

Výpis 9.2: Načtení modelu a formuláře

Prostřednictvím formuláře jsou k dispozici dvě další cesty a to update a delete, které volají příslušné metody modelu a předávají jim validovaná data. Na závěr může dojít k přesměrování na další stránku. Na výpisu je zobrazen částečný kód pro zpracování update.

```
if ($this->getRequest()->isPost() && false !== $this->getRequest()->getPost('send',false)){
        if ($form->isValid($this->getRequest()->getPost())){
                $modelArticle->update($attribs,$articleId);
                $this->_redirect('/Article/list/');
        }
}
```

9.2.2.1 Nastavení JavaScriptu

Nastavení javaScriptu v controlleru se vyskytuje ve dvou variantách jednou pro list2 a po druhé pro pohled získávající data z list2. V tomto případě bude popsáno nastavení v akci edit tedy nastavení pro získání dat z list2.

```
//********** Nastaveni javascriptu pro popup window *****************
$jsPopupMain = MyWeb_JavaScript_Popup_Abstract::createJsPopupMain();
$jsPopupMain->setSourceController('article');
$jsPopupMain->setSourceAction('edit');
$jsPopupMain->addLinkPopupWindow('okno', 'list2');
$jsPopupMain->addIdInputs('State_id_stateValue');
$jsPopupMain->addControllerLinkOpenWindow('state');

//lepsi je pro kazde individualni nazev bez indexace
$jsPopupMain->addIdHidden(0,'State_id_stateInputValue');
$jsPopupMain->addIdHidden(0,'nameState_nameInputValue');
$jsPopupMain->addIdHidden(0,'State_id_stateInputName');
$jsPopupMain->addIdHidden(0,'nameState_nameInputName');
$jsPopupMain->levelDirectory(4);//nastaveni adresarove urovne + predavana promenne
$js=$jsPopupMain->js();
$jsPopupMain->setNameFile('mainNew');//nastaveni jmena okna
$jsPopupMain->writeFileJs();//zapsani do souboru
```

```
$jsPopupMain->setNameFileLink('mainNewLink');
$jsPopupMain->addParamGetLinkOpenPopup(0,'tableDatabase_table', 'article');
//$jsPopupMain->writeFileJsLinks();//zapsani do souboru
$jsOpenLinks = $jsPopupMain->jsOpenLinks();//vytvoreni js pro odkazy

$this->view->jsOpenLinks = $jsOpenLinks;//predani pohledu js odkazu
$this->view->javascriptPopupMain = $js;//predani pohledu js list2
```

Výpis 9.3: Nastaveni javaScriptu edit

V prvním kroku je vytvořena proměnná jsPopupMain, která zprostředkovává nastavení generování javaScriptu. Metoda setSourceController() nastaví jmenný název aktuálního controlleru. Metoda setSourceAction() nastaví aktuální akci. Metody addLinkPopupWindow() a addControllerLinkOpenWindow() vytvoří odkaz s id vedoucí na popup okno. Metoda addIdHidden() přidá pole očekávající příchozí hodnoty z list2 u kterých záleží na pořadí odeslání. Metoda addParamGetLinkOpenPopup() nastavuje parametry pole odkazu, které vyžaduje list2 pro správné sestavení návratových hodnot. Zbývající funkce nastavení jsou popsány ve výpisu 9.3.

9.2.3 Search action

Metoda searchAction opět jako předcházející metody načte model a s jeho pomocí i načte příslušný formulář.

```
$modelArticle = MyWeb_Model_Database::loadModel('Article');
$form = $modelArticle->getForm('search');
```

Výpis 9.4: Načtení modelu a formuláře

Základní úlohou této metody je provést validaci vstupních údajů z formuláře a odeslat je prostřednictvím pole GET metodě list příslušného controlleru.

9.2.4 List2 action

Metoda list2Action je variantou metody listAction, která je určena pro vytvoření seznamu položek zobrazovaném v popup okně. Tento seznam je obvykle dostupný prostřednictvím odkazu umístěném ve formuláři vedle formulářového políčka, jehož možné hodnoty se nalézají v seznamu v novém okně.

Každá položka tohoto seznamu obsahuje odkaz, který uzavře okno a přenese zvolenou hodnotu do políčka formuláře.

Hlavní odlišností této akce je nastavení javaScriptu, které umožňuje takovéto chování aplikace. Nastavení javaScriptu zajistí vytvoření a nastavení příslušného objektu určeného k tomuto účelu.

```
//********** Nastaveni javascriptu pro popup window
/*$jsPopupSlave = MyWeb_JavaScript_Popup_Abstract::createJsPopupSlave();
$jsPopupSlave->setIdInput('in');
$jsPopupSlave->addIdHidden('inid');
$jsPopupSlave->addIdHidden('inid');
$jsPopupSlave->addIdHidden('inid');
$jsPopupSlave->addIdHidden('inid');
$jsPopupSlave->setIdLink('button');
$js=$jsPopupSlave->js();

$jsPopupSlave->setNameFile('slaveList2');
//$jsPopupSlave->writeFileJs();*/
$this->view->javascriptPopupSlave = $js;
$this->view->select = $modelTable2->createFetchEntrySelect
($this->getRequest()->getParams());
$this->view->pager = $paginator->getPaginator();
```

Výpis 9.5: Nastaveni javaScriptu

9.2.4.1 Nastaveni JavaScriptu

Nastavení javaScriptu je zahájeno vytvořením proměnné jsPopupSlave. Metoda setIdInput() nastaví název id, kterým je identifikován každý řádek seznamu list2 pomocí přidaných číselných indexů k id. Metoda addIdHidden() přidá k vlastní hodnotě řádku hidden pole obsahující hodnoty přidružené. Metoda setIdLink() vytvoří id odkazu pro každý řádek seznamu a na závěr je pomocí metody js() vygenerován javaScript, který je předán pohledu.

Pro kontrolu generovaných skriptů nebo pro externí uložení je možné využít metodu writeFileJs().

9.3 Model

Model představuje jeden ze základních kamenů MVC aplikace stejně tak jako controlleru a View.

9.3.1 Implementace modelu

Struktura modelu se skládá z rozhraní a dvou abstraktních tříd. Rozhraní je definováno nad abstraktními třídami což zajišťuje jejich konzistentnost pro další vývoj. Při vzniku potřeby využívat nové úložiště musí být vytvořena nová abstraktní třída, která musí implementovat použité rozhraní. Výpis 9.6 ukazuje implementované rozhraní.

```
interface MyWeb_Model_Interface
{
	public function getStorage();
	public function insert(array $data);
	public function update(array $data, $id);
	public function delete($id);
	public function fetchEntry($id);
	public function fetchEntries($data);
	public function isExists($column, $value,$id);
}
```

Výpis 9.6: Interface modelu

První abstraktní třída implementující rozhraní obsahuje metody pro prací s různými úložišti. Druhá abstraktní třída využívá Zend_Db_Table reprezentující databázovou tabulku, nad kterou jsou implementovány metody rozhraní. Třídy s modely jsou uloženy v adresáři application/models což není standardní cesta a proto je nutné zajistit jejich načítání samostatnou třídou, která dědí ze třídy Zend_Loader_PluginLoader.

9.3.2 Třida Zend_DbTable_Abstract

Objekty této třídy reprezentují databázové tabulky a jejich vzájemné vztahy. Sada těchto tříd tedy může reprezentovat kompletní přístup k databázi. Příklad takové třídy je zobrazen na výpisu 9.7, ze kterého je patrné, že jedinými vstupními údaji nutnými k vytvoření třídy jsou data obsažená v databázi a proto mohou být třídy generovány bez dalšího zásahu programátora.

```
class Application_Model_DbTable_Article extends Zend_Db_Table_Abstract
{
protected $_name = 'article';
protected $_primary = array('id_article');

protected $_referenceMap = array(
                        'Application_Model_DbTable_User'=>array(
                                'columns'=>'User_id_user',
                                        'refTableClass'=>'Application_Model_DbTable_User',
                                        'refColumns'=>'id_user'
                                        ),
                        'Application_Model_DbTable_State'=>array(
                                        'columns'=>'State_id_state',
                                        'refTableClass'=>'Application_Model_DbTable_State',
                                        'refColumns'=>'id_state'
                                        ),
                        );
}
```

Výpis 9.7: Zend_Db_Table_Abstract

Klíč columns v poli Application_Model_DbTable_User představuje jméno sloupce s cizím klíčem, klíč refColumns obsahuje jméno primárního klíče cizí tabulky a klíč refTableClass nese jméno třídy reprezentující cizí tabulku.

9.4 View

View v Zend Frameworku jsou umístěny v adresářích s názvy controllerů application/views/scripts/jméno_controlleru. View jsou opět rozdělena do základních bloků podle metod příslušného controlleru:

- Index
- Edit
- List
- List2
- New
- Search

Všechna view zjišťují především připojení kaskádových stylů, javaScriptů a zobrazení formulářů což je ukázáno na výpisu 9.8.

```
$this->headLink()->appendStylesheet($this->DinamicPath() .'css/form.css' ,'all');
print $this->form->render();
```

Výpis 9.8: View

Výjimečný v tomto ohledu je view List, který využívá view helper Table, který zapouzdřuje komponentu Zand_Paginator pro tvorbu tabulek.

9.5 Formuláře

Zend Framework poskytuje prostřednictvím komponenty Zend_Form nástroj k vytvoření, vykreslení a zpracování formulářů. Díky této komponentě je také možné provádět filtrování a validaci údajů i upload souborů.

9.5.1 Zend_Form

Každý Zend_Form se skládá z atributů, dekoratérů a elementů. Atributy například udávají, jak a na jakou adresu mají být odeslány údaje z formuláře, a dekoratéry se starají o zobrazení formulářových prvků. Kromě toho mohou být formulářové prvky seskupeny a formulář může též obsahovat vnořené formuláře.

Formulářové prvky také obsahují atributy, dekoratéry a doplňující údaje, například možnosti při výběru ze seznamu, nebo zda je políčko povinné či ne. Kromě toho je možné každému prvku formuláře přiřadit libovolný počet Zend_Filter objektů filtrujících údaje a Zend_Validate objektů na validaci údajů.[2]

9.5.2 Zend_Form tvorba formulářů

Formulář je možné vytvořit různými způsoby. Nejjednodušší cestou je inicializace objektu třídy Zend_Form a přidání formulářových prvků do tohoto objektu. Můžete také vytvořit třídu, která je potomkem třídy Zend_Form, a při její inicializaci přiřadit prvky do formuláře. Třetí možností je kombinace Zend_Form a Zend_Config. [2]

V projektu je použita pouze možnost využívající Zend_Config v kombinaci s rozšířením třídy Zend_Form a proto bude dále popsána pouze tato možnost. Příklad takového konfiguračního souboru ukazuje výpis 9.9.

```
elements.article_text.Type = text
elements.article_text.Options.required = true
elements.article_text.Options.label = article_text
elements.article_text.Options.size = 45
elements.article_text.Options.maxlength = 45

elements.article_text.Options.validators.length.validator = "StringLength"
elements.article_text.Options.validators.length.options.1 = 3
elements.article_text.Options.validators.length.options.2 = 45
elements.article_text.Options.validators.alnum.validator = "Alnum"
```

Výpis 9.9: Konfigurační soubor formuláře

Atributy formuláře jsou definovány na nejvyšší úrovni. Všechny formulářové prvky jsou definovány pomocí parametru elements, za kterým následuje název prvku, například artikle_txt. Dále následuje definice typu prvku a definice možností výběru. [2]

Načteni konfiguračního souboru a případné doplnění sady elementů je ukázáno na výpisu 9.10.

```
class Application_Model_Form_ArticleNew extends MyWeb_Form
{
   public function init()
   {
      $root = 'index.php';
      $request = Zend_Controller_Front::getInstance()->getRequest();
      $getValue = $request->getParams();
      $path = explode($root, $request->getRequestUri());
      $this->setAction("new");
      $this->setMethod("post");
      $this->setAttrib("id", "form_article_new");
      $config = new Zend_Config_Ini(APPLICATION_PATH . "/forms/articleNew.ini");
      $this->setConfig($config);
      $article = $this->_model->fetchEntry(NULL);
      //*****************************************************************************
      $url = new Zend_View_Helper_Url();
            $link = '<a href = "'.$url->url(array('controller' => 'user', 'action' => 'list2'), 'default', true)
         .'/sourceController/article/sourceAction//"'.' Onclick = "window.open(\". $url->url(
```

```
        array('controller' => 'user', 'action' => 'list2'), 'default',
        true).'/sourceController/article/sourceAction//\".');   return false">user</a>';
    $element = $this->getElement('User_id_userValue');
    $element->setLabel('User');
    $element->setDescription( $link );
    $element->setDecorators(array('ViewHelper',array('Description',array('escape'=>false,'tag'=> false)),
                    array('HtmlTag', array('tag' => 'dd')),
                    array('Label', array('tag' => 'dt')),
                    'Errors'));
    $element->setAttrib('disabled',"");

    if (key_exists('User_id_userValue', $getValue)){
        $element->setValue($getValue['User_id_userValue']);
    }

    $User_id_userID = new Zend_Form_Element_Hidden('User_id_userID');
    $User_id_userID->setValue(NULL);
    if (key_exists('User_id_user', $getValue)){
        $User_id_userID->setValue($getValue['User_id_user']);
    }
        // vytvoří formulářový prvek odesílací tlačítko
    $submitInsert = new Zend_Form_Element_Submit("send", array("label" => "new"));
        // přidá formulářové prvky do formuláře
    //$this->addElements(array($option_state,$hidden_role));
        $this->addDisplayGroup(
        array('id_article','article_text','article_dateTime','User_id_user','State_id_state','article_content'),
        'formRoleEdit',array('legend' => 'Formular'));
         $this->addElements(array($submitInsert));
}}
```

Výpis 9.10: Vytvoření formuláře

9.5.3 Formulář

Takto dlouhý výpis kódu je zde uveden, protože je generátorem vygenerovaný a programátor je nucen s ním dále pracovat.

V tomto výpisu je nejzajímavější část oddělená komentářem, ve které se nalézá doplnění prvku se jménem 'User_id_userValue' o odkaz vedoucí na list2, protože hodnota prvku je získávána prostřednictvím cizího klíč z další tabulky. Hodnota tohoto prvku je nastavena na hodnotu z globálního pole get pokud existuje klíč se jménem prvku v tomto poli. Prvek hidden je vytvořen za účelem uložení hodnoty cizího klíče. Ve zbývající části výpisu je formulář doplněn tlačítky a jednotlivé prvky formuláře (elementy) jsou seskupeny.

10 POUŽITÍ GENERÁTORU

10.1 Postup vytvoření projektu

Prvním krokem pro vytvoření projektu je vytvoření standardní adresářové struktury aplikace využívající Zend Framework. Adresáře je možné vytvořit ručně, ale snadnější je použit nástroj Zend Tool, který poskytuje přímo Zend Framework. Zend Tool je blíže představen ve dvanácté kapitole. Ovládání Zend tool probíhá skrze příkazovou řádku.

```
C:\Program Files\ Apache2.2\htdocs\zf\bin>
C:\Program Files\ Apache2.2\htdocs\zf\bin>zf create project
Please provide a value for $path
zf>C:\Program Files\Apache Software Foundation\Apache2.2\htdocs\eclipse\project
```

Výpis 10.1: Založení projektu pomocí Zend Tool

Druhým krokem založení projektu je:

- nastaveni konfiguračního souboru application/configs/application.ini
- vytvoření layoutu v adresáři application/layouts/scripts
- doplnění:
  - view helperů application/views/helpers
  - css \public\css (form,styles,table)
  - javaScript \public\js
  - knihovny \library\MyWeb
  - data /data

11 UKÁZKOVÁ APLIKACE

Ukázková aplikace byla vytvořena pro ověření funkčnosti generátoru a případnému odhalení nedostatků.

11.1 Popis ukázkové aplikace

Aplikace byla vytvořena jako jednoduchý systém pro vytváření a prezentaci článků.

11.2 Struktura aplikace

Aplikace se skládá ze tří základních bloků, které reprezentují databázové entity:

- Article
- Comment
- User

Kód byl vytvořen generátorem, a proto se každý blok skládá z částí:

- Seach
- List
- List2
- Edit
- New
- Formuláře
- View
- Controller
- Konfigurační soubory
- Model

Kromě běžné implementace generované šablony byly doplněny některé části navíc specifické pro danou aplikaci. Tyto části budou popsány dále.

11.2.1 Blok User

Tento blok je zodpovědný za správu entity user, která obsahuje data jednotlivých uživatelů. Blok navíc obsahuje v controlleru akce související s přihlášení a odhlášením uživatele a některé související pohledy. Tyto specifické akce byly částečně generovány na základě specifičnosti názvu databázové entity.

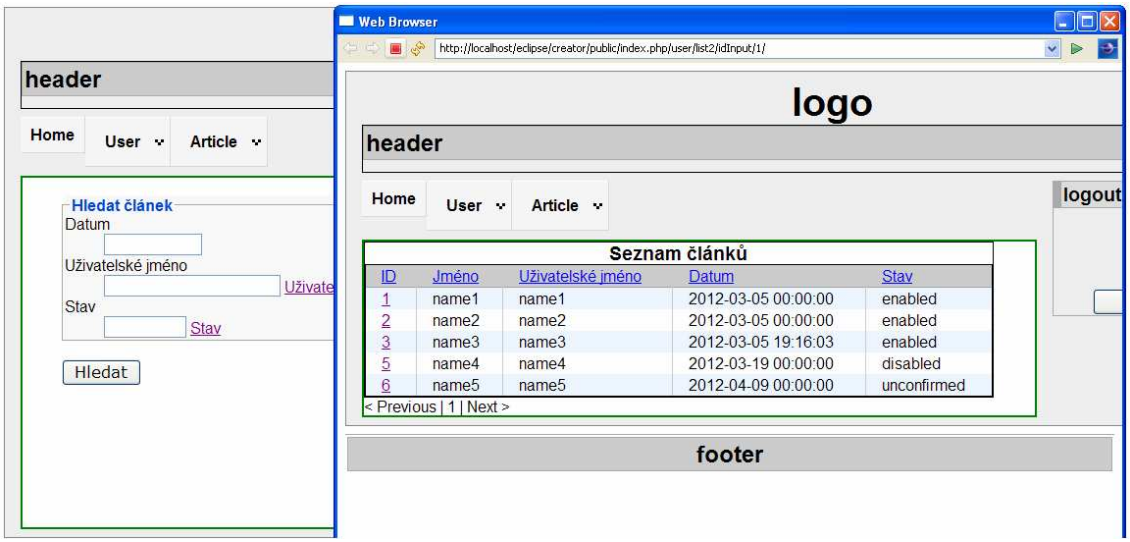
11.2.2 Blok Article

Blok article zajišťuje vytváření, editování a prohlížení článků pro uživatele.

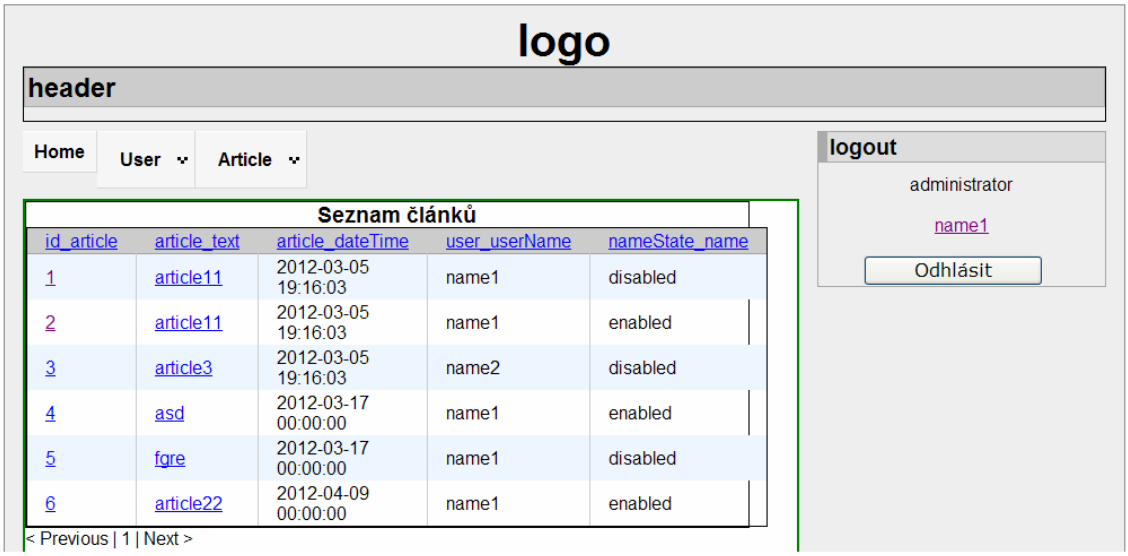
Blok se vyznačuje dvěma odlišnostmi od vygenerované šablony a to především implementací WYSIWYG editoru, který zajišťuje právě vytváření a editaci článků.

Druhou odlišností je implementace akce controlleru show, která umožňuje zobrazení článků. S vytvořením nové akce souvisí i vytvoření odpovídajících pohledů.

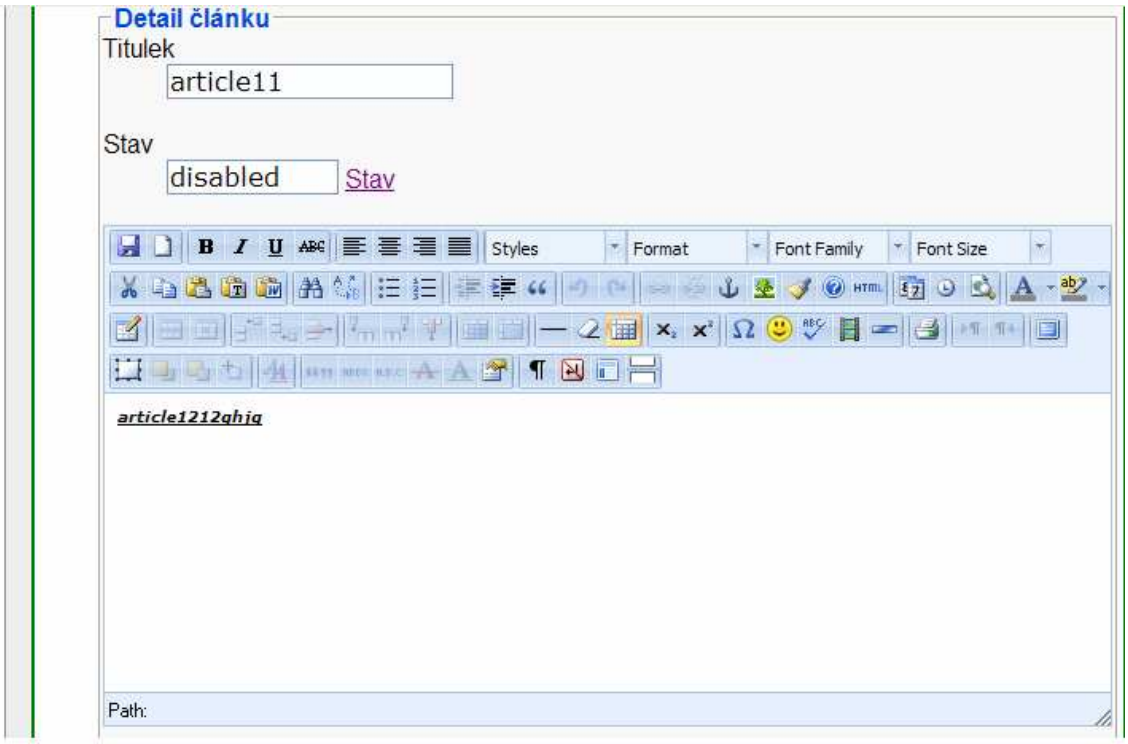
Obrázek 4: Vyhledávání článků



Obrázek 5: List2 pro seznam uživatelů



Obrázek 6: Nalezený seznam článků



Obrázek 7: Edit článku

11.2.3 Navigace

Navigace tvoří samostatný blok doplňující aplikaci využívající view helperů zejména Navigation a Menu. Navigace není generována.

Vzhled a funkce drop down menu je zajištěna pomocí knihovny získané ze stránky http://www.lwis.net/free-css-drop-down-menu/ . [6]

11.2.4 Formuláře aplikace

Standardní vzhled formulářů je docílen pomocí nastavených dekorátorů jejichž nastavení zajišťuje třída MyWeb_Form a následné použití CSS. Pro doplněné elementy formuláře jsou specifikovány dekorátorý přímo ve třídě pro daný formulář.

11.2.5 Uživatelské role v aplikaci

V aplikaci byl použit jednoduchý systém uživatelských rolí umožňující řízený přístup ke zdrojům aplikace.

Seznam uživatelských rolí:

- Guest
- Administrator
- Editor

Role administrator nemá v systému omezení. Guest může pouze prohlížet články a registrovat uživatele. Role editor může zobrazovat seznam uživatelů, vytvářet články a editovat již vytvořené články. Tato role má také všechna práva, která má guest.

11.2.6 Databázové stavy

V implementované aplikaci byl použit pro většinu prvků dvoustavový životní cyklus, který je tvořen stavy

- Disabled
- Enabled

V případě stavu uživatelů byl přidán stav unconfirmed, který zajišťuje nutnost potvrzení nově registrovaného uživatele administrátorem, kde administrátor má absolutní kontrolu nad stavy všech uživatelů systému.

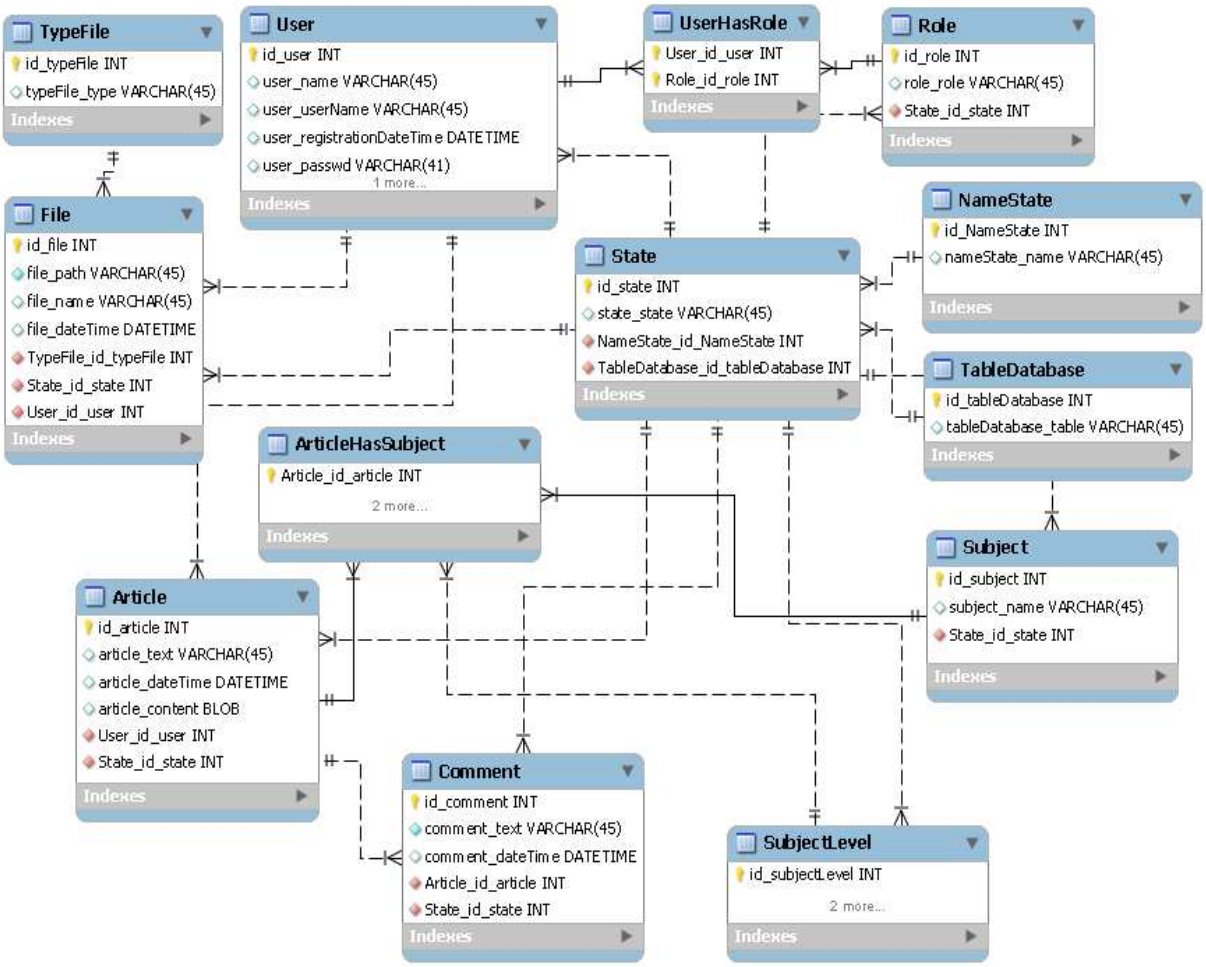
11.3 Databáze aplikace

Návrh databáze se skládá ze třech základních entit, které jsou identické s bloky aplikace popsané výše. Při prvním pohledu na schéma databáze je patrný mnohem větší rozsah databáze, než je zde popsán, nicméně je skutečně realizována pouze ta část, která je popsána. Nevyužitá část databáze slouží zejména pro testování generátoru.

Entita article sdružuje informace o článcích vytvořených uživateli, kde těla článků jsou uložena ve sloupci article_content, který je datového typu blob. Každý článek nese odkaz na uživatele uloženého v tabulce user.

Informace uživatelů jsou uloženy v tabulce user, která je propojena s tabulkou role vazbou M:N, která není v systému reálně využita.

Poslední tabulkou je comment, která uchovává komentáře k jednotlivým článkům.



Obrázek 8: Schéma databáze ukázkové aplikace

11.4 Část databáze reprezentující stav

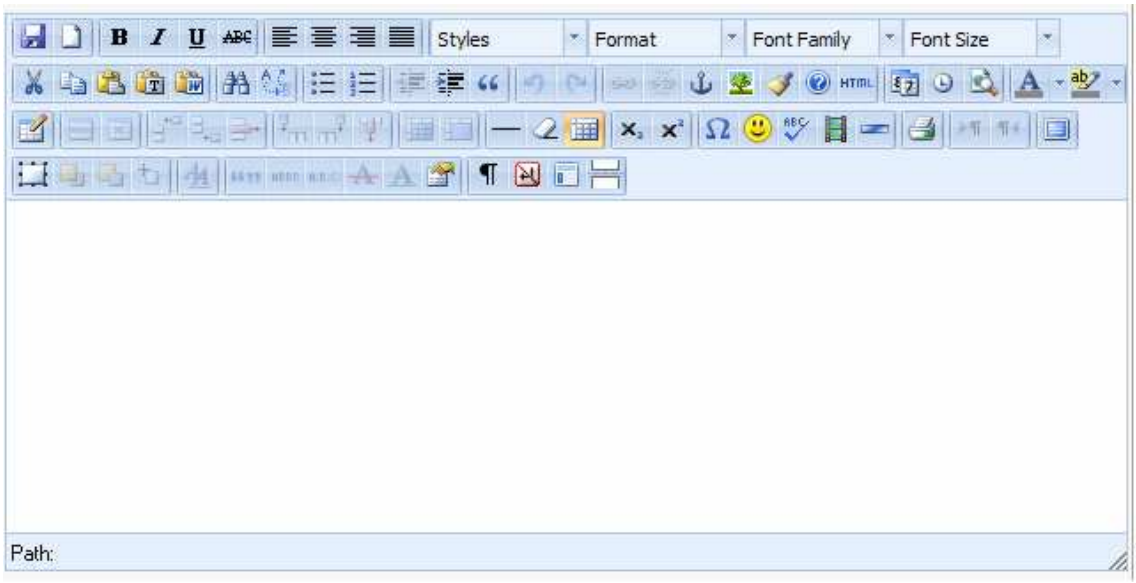
Tato část databáze je reprezentována tabulkami:

- State
- NameState
- TabelDatabase

Tabulka state je spojena se všemi tabulkami databáze, což umožňuje přiřadit stavy každé tabulce. Pro rozlišení stavů jednotlivých tabulek existuje propojení s tabulkou tableDatabase, která obsahuje jmenné názvy všech tabulek databáze.

11.5 Sozfo

Implementaci WISWIG editoru jsem využil již připravený element pro Zend Framework, který je dostupný pod licencí BSD ze stránek http://code.google.com/p/sozfo/ [5]. Pro využití tohoto formulářového elementu je nutné doplnit knihovnu o adresář Sozfo, který zajišťuje implementaci.



Obrázek 9: Editor článků

Editor v této implementaci má k dispozici standardní sadu vzhledů a motivů, kde nastavení se provádí pomocí konfiguračního souboru configs/tinyMCE.ini. Editor obsahuje širokou paletu funkcí a volitelnou sadu panelů nástrojů.

12 ZEND TOOL

Zend Tool je nástroj pro vytvoření základní adresářové struktury vyžadované Zend Frameworkem. Ovládací rozhraní je zprostředkováno příkazovým řádkem.
Zend Tool umožňuje, také vytvářet soubory jednotlivých tříd, které jsou náležitě umístěny do adresářové struktury. Prostřednictvím tohoto nástroje je také možné vytvářet metody. Bohužel ve starších verzích se objevovalo mnoho chyb znemožňující jeho plnohodnotné využití. Nicméně při vytvoření projektu bez dalších nároků se chyby nevyskytují.

13 OBSAH CD/DVD

CD obsahuje dva adresáře generator a ukazkova_aplikace.

V adresáři ukazkova_aplikace je umístěna ukázková aplikace, tedy aplikace jejíž jádro bylo vygenerováno a dále byla upravena ručně do výsledné podoby. Databáze ukázkové aplikace je umístěna v souboru ukazkova_aplikace/creator/application/database, kde se nalézá soubor database.mwb pro aplikaci MySQL Workbench 5.1 OSS a soubor full_db.sql pro aplikaci MySQL Query browser umožňující rychlé naplnění databáze. Poslední soubor obsahuje zálohu databáze.

V druhém adresáři je umístěn projekt s jednoduchou aplikací, která byla vygenerována bez dalších ručních úprav.

Na přiloženém CD/DVD je také umístěn vývojový nástroj eclipse.

14 ZÁVĚR

Hlavním cílem této práce bylo navržení a realizování generátoru kódu pro snadné vytvoření internetové aplikace založené na php frameworku a databázovém systému MySQL.

První část práce byla zaměřena na analýzu a porovnání dostupných technologií zejména PHP frameworků. Na základě tohoto porovnávání byl zvolen PHP Zend Framework, který vyhovoval nejvíce kritériím.

V druhé logické části práce byla navrhnuta generovaná šablona, která zapouzdřuje většinu struktur aplikací běžně se v prostředí internetu vyskytujících. Tato šablona podřízená architektonickému vzoru MVC se skládá z částí: Search, List, List2, Edit a New.

V předposlední třetí části byl navrhnut a realizován generátor kódu přímo propojený se třídami vlastního Zend Frameworku. Realizovaný generátor je schopný na základě informací získaných z databáze a s minimem informací od programátora vytvořit větší část aplikace, která nadále vyžaduje dílčí ruční úpravy. Vygenerovaná aplikace respektuje MVC architektonický vzor a obecnou šablonu návrhu aplikace.

Poslední pátá část popisuje realizaci ukázkové aplikace s využitím vytvořeného generátoru. Ukázková aplikace je zjednodušený systém umožňující spravovat články prostřednictvím internetové aplikace. Ukázková aplikace byla doplněna řadou specifických funkcionalit vyžadovaných zaměřením aplikace.

Krátký vývoj ukázkové aplikace poukázal na řadu specializací internetových aplikací, které navržený generátor nezohledňuje, ale vygenerovaná aplikace umožňuje jejich začlenění nebo přetvarování původního návrhu. Generátor také potvrdil, že dokáže řádově zkrátit dobu vývoje aplikace.

V případě snahy o další vývoj v této oblasti je správným směrem schopnost generátoru číst strukturu databáze, ve které by byl umístěn ekvivalent přístupového seznamu Acl což může vést k vygenerování plnohodnotné aplikace.

15 LITERATURA

[1] ULLMAN, L. PHP a MySQL Názorný průvodce tvorbou dynamických WWW stránek. Brno: Computer Press, 2004. 534 s. ISBN 80-251-0063-4.

[2] BÖHMER, M. Zend Framework – Programujeme webové aplikace v PHP. Brno: Computer Press, 2010. 416 s. ISBN 978-80-251-2965-4.

[3] Reference Guide [online]. Dostupné z: < http://framework.zend.com/manual/en/>.

[4] JASON GILMORE, W. VELKÁ KNIHA PHP MySQL 5. KOMPENDIUM ZNALOSTÍ PRO ZAČÁTEČNÍKY I PROFESIONÁLY. BRNO: Zoner press, 2007, s 864. ISBN 80-86815-53-6

[5] Sozfo [online]. Dostupné z: < http://code.google.com/p/sozfo/>.

[6] Drop-down-menu [online].Dostupné z: <http://www.lwis.net/free-css-drop-down-menu/>.

[7] jQuery plugin for communication between browser windows [online]. Dostupné z: <http://www.sfpeter.com/2008/03/communication-between-browser-windows-with-jquery-my-new-plugin/>.

[8] JQuery [online]. Dostupné z: < www.jquery.com >.

16 SEZNAM ZKRATEK

Acl (Access Kontrol List)

PHP (Personal Home Page)

SQL (Structured Query Language)

XML (Extensible Markup Language)

CSS (Cascading Style Sheets)

AJAX (Asynchronous JavaScript and XML)

XHTML (extensible hypertext markup language)

MVC (Model-view-controller)

ORM (Object-relational mapping)

PDF (Portable Document Format)

CSV (Comma-separated values)

HTML (HyperText Markup Language)

HTTP (Hypertext Transfer Protocol Secure)

17 SEZNAM PŘÍLOH

Příloha 1. CD/DVD