PŘÍRODOVĚDECKÁ FAKULTA UNIVERZITY PALACKÉHO
KATEDRA INFORMATIKY

BAKALÁŘSKÁ PRÁCE

Aplikační rámec pro PHP v češtině

2010 Libor Pečinka

Anotace

Bakalářská práce je zaměřena na problematiku frameworků pro PHP. Čtenář je nejprve seznámen se skriptovacím jazykem PHP a situací v oblasti aplikačních rámců pro něj určených. Poté je na ukázkové aplikaci, která je stručně zanalyzována, provedeno srovnání dvou zavedených frameworků. Poslední část práce se věnuje implementaci vlastního aplikačního rámce pro českého uživatele a popisu jeho použití.

Děkuji vedoucímu práce Mgr. Janu Outratovi, PhD. za odborné vedení, rady a pomoc a všem, kteří se svými podněty podíleli na její výsledné podobě.

Obsah

Seznam obrázků

Seznam tabulek

1. Zadání bakalářské práce

Cílem práce (na vlastní téma) je vytvoření česky psaného PHP frameworku (jeho zdrojového kódu), tj. aplikačního rámce pro tvorbu (českých) dynamických webových stránek a webů pomocí jazyka PHP. Framework bude obsahovat šablony pro standardní prvky dynamického webu jako registraci a správu účtu uživatele, přihlašovací formulář, diskuze pod stránkou, CAPTCHA formulář, editační nástroje stránky a další. Součástí práce bude i demonstrace využití frameworku pro tvorbu jednoduché aplikace.

Dílčí požadavky

- česky psaný PHP framework
- šablony pro registraci a správu účtu uživatele, přihlašovací formulář, diskuze pod stránkou, CAPTCHA formulář, editační nástroje stránky a další
- demonstrace frameworku na jednoduché aplikaci včetně názorného popisu tvorby jeho kódu pomocí frameworku
- srovnání několika existující PHP frameworků: klady a zápory, požadavky na uživatele apod.
- zpracování dle webových standardů, důraz na čistotu kódu frameworku
- uživatelský (programátorský) manuál

2. PHP a frameworky

Termín framework, který se do češtiny často překládá jako aplikační rámec, označuje speciální případ knihovny poskytující uživateli určitou pokročilou funkcionalitu, která může být dále upravována. Od běžných knihoven a uživatelských aplikací se odlišuje zejména kontrolou řízení toku dat, standardním chováním a rozšiřitelností [1]. Webový framework slouží k usnadnění a zkrácení vývoje dynamických webových stránek a webových aplikací tím, že nabízí uživateli oddělení vrstev aplikace a často používané funkce jako je ošetření uživatelských vstupů, autentizace a autorizace nebo práce s databázovými systémy.

V úvodu kapitoly je čtenář seznámen se skriptovacím jazykem PHP, určeným převážně pro tvorbu webových stránek. Druhá část je věnována motivaci a průběhu vývoje aplikačních rámců pro PHP.

2.1. Vývoj PHP

Raná verze PHP, kterou tvořila sada několika skriptů v jazyce Perl, vznikla v roce 1994 [2], tehdy ovšem pouze pro osobní potřebu autora, dánského programátora Rasmuse Lerdorfa. O rok později, v červnu 1995, byla veřejnosti oficiálně oznámena první verze pod názvem PHP Tools (*Personal Home Page Tools*), přepsaná do jazyka C[1], která umožňovala zaznamenávat a prohlížet data o návštěvnosti stránky, omezit přístup určitých návštěvníků a vytvářet, zobrazovat a zpracovávat formuláře, v té době nepříliš běžnou součást webu. K použití PHP Tools potřeboval uživatel pouze právo na spouštění vlastních CGI skriptů.

V dubnu roku 1996 Lerdorf dokončil práci na historicky druhé oficiální verzi PHP spouštěné s využitím nově se vyvíjejícího webového serveru Apache, celým názvem pojmenované *PHP/FI Server–side HTML-Embedded Scripting Language* – zkratka FI znamená *Form Interpreter*. Poprvé zde byl použit název „skriptovací jazyk“. Byla přidána možnost využití uživatelských funkcí, rekurze, regulárních výrazů, spojení s několika druhy databází, náhrání souboru pomocí internetového prohlížeče na server, manipulace s HTTP hlavičkami a tvorba obrázků ve formátu GIF. Projekt jediného člověka[2], který se v listopadu 1997 dočkal další verze *PHP/FI 2.0*, se dostal do povědomí několika tisíc uživatelů po celém světě a přibližně 50 000 domén[3] jazyk umožňovalo využít. Ještě větší ohlas následoval s vydáním třetí verze, na níž už Lerdorf pracoval s dvěma izraelskými vývojáři – Andi Gutmansem a Zeevem Suraskim.

Červen 1998 znamenal velký mezník ve vývoji PHP. Po devíti měsících veřejného testování byla vydána verze 3.0, jejímž základem se stal přepracovaný

[1] část syntaxe Perlu se však zachovala dodnes - např. označení proměnných
[2] s několika přispěvateli
[3] tehdejší cca 1 % domén Internetu [3]

parser a která přinesla změnu významu zkratky PHP na *PHP: Hypertext Preprocessor*[4]. Přibyla podpora všech hlavních operačních systémů (Windows 95/NT, většina verzí Unixu a MacOS), webových serverů (včetně Apache, Netscape a IIS) a databázových systémů (např. MySQL, Oracle a datové zdroje ODBC). Úpravy prodělala syntaxe jazyka, která nově umožňovala elementární objektově orientované programování a která s drobnými změnami se zachovala až do současnosti. Jednou z velkých deviz nové verze byla díky úpravě rozhraní API snadná rozšiřitelnost, která umožnila vývojářům vytvářet nové moduly. V roce 2000 byl kvůli jejich stále rostoucímu počtu vytvořen repositář rozšíření a aplikací pro PHP s názvem PEAR[5]. Koncem roku 1998 bylo PHP 3.0 nainstalováno odhadem na 10 % webových serverů na světě [3].

Krátce po vydání třetí verze začali Suraski a Gutmans pracovat na přepsání jádra PHP s cílem zvýšit výkon aplikací a oddělit jazykovou vrstvu od webového serveru. O dva roky později, v květnu 2000, bylo oznámeno PHP 4.0 založené na zcela novém jádře – *Zend Engine*[6]. Narozdíl od parseru PHP 3, který zpracovával skript během čtení, byl skript nejprve přeložen a pak zpracován. Díky Zend Enginu mohly být zavedeny moduly pro ladění nebo zrychlení zpracování kódu. V novém PHP mohli uživatelé využít sessions[7], ukládání výstupu do vyrovnávací paměti, nové jazykové konstrukty, od verze 4.1.0 *superglobální proměnné* a od verze 4.3.0 rozhraní příkazové řádky a grafickou knihovnu GD. Jazyk bylo nyní možné provozovat na dalších druzích webových serverů, což zvýšilo rozšíření PHP na více než 20 % domén Internetu [3].

V červenci 2004 byla vydána verze 5 s jádrem Zend Engine 2.0, která obsahovala novou správu paměti s lepší podporou vícevláknových prostředí a efektivním uvolňováním paměti. Přestože byl základní objektový model přítomen již od třetí verze, chyběla podpora velké části rysů OOP[8]. Hlavním nedostatkem bylo předávání objektů hodnotou namísto odkazem, což snižovalo výkon a často vedlo k neočekávaným chybám[5]. PHP 5 přineslo přepracovaný systém objektů, který se více přiblížil standardním objektově orientovaným jazykům, implementaci zachytávání výjimek, lepší podporu UTF-8, snadnou manipulaci s XML dokumenty a integroval několik knihoven z repositáře PEAR.

Šestá verze PHP začala být vyvíjena v roce 2006. Kvůli nepředpokládaným problémům s plnou podporou kódování Unicode bylo v březnu 2010 rozhodnuto, že vývoj bude prozatím pokračovat v dílčích verzích PHP 5 [4]. Aktuální verze PHP v době psaní tohoto textu je 5.3.2.

[4] tzv. rekurzivní zkratka
[5] PHP Extension and Application Repository
[6] Zend je složenina křestních jmen autorů – Zeev a Andi
[7] v češtině se někdy používá pojem uživatelská relace nebo sezení
[8] Object-oriented programming – objektově orientované programování

2.2. Vlastnosti a syntaxe jazyka PHP

Interpretovaný jazyk PHP je primárně určen k tvorbě dynamických webových stránek, jeho kód se standardně vkládá přímo do HTML kódu. Skriptovací stroj zpracuje příkazy mezi tagy `<?php` a `?>`, případně `<script language="php"` a `</script>`, úvodní značka se může po povolení dodatečného nastavení krátkých tvarů zkrátit na `<?` nebo ve stylu jazyka ASP na `<%`. Velká část syntaxe byla převzata z jazyka C a Perl – výrazy, operátory, řídící struktury. Každá instrukce musí být ukončena středníkem, jednořádkové komentáře jsou uvozeny znaky `//` nebo `#` a víceřádkové uzavřeny mezi `/*` a `*/`.

PHP je slabě typovaný jazyk, nevyžaduje po programátorovi určení datového typu proměnné ani její definici. Jazyk podporuje osm primitivních typů: *boolean*, *integer*, *float* (totožné s *double*), *string*, *array*, *object*, *resource* a *NULL*. Datový typ proměnné je určen její hodnotou a ta může být v určitých situacích upravena (např. vyhodnocení retězce na typ *boolean* v podmínce nebo aplikace nějakého operátoru), převody jsou detailně popsány v dokumentaci, která patří především díky množství příkladů mezi nejlepší. Vývojář si může typ proměnné ověřit nebo manuálně měnit pomocí vestavěných funkcí či typování.

```
<?php
// proměnná je typu string
$a = '5';

// typ proměnné je změněn na float a je proveden součet
$a += 2.5;

// funkce ověří, zda je proměnná typu float
is_float($a)
?>
```

Rozsah platnosti proměnných je rozlišen na globální a lokální. Na rozdíl od např. jazyků rodiny C nejsou proměnné definované globálně přístupné uvnitř funkcí, pokud to však programátor vyžaduje, může si je zpřístupnit pomocí klíčového slova *global*. Od verze 4.1.0 jsou vývojářům k dispozici tzv. „superglobální“ pole, která obsahují informace o serveru, na němž je skript spuštěn, data z hlaviček HTTP požadavku apod. Jak sám název napovídá, jsou tyto proměnné přístupné na libovolném místě kódu, uživatel však nemůže zakládat další superglobální proměnné. PHP umožňuje využití statických proměnných, které jsou inicializovány pouze při prvním volání funkce a jejichž hodnota je zachována i po dokončení zpracování příkazů funkce.

Proměnná je uvozena znakem `$`, po němž následuje její jméno, ve kterém mohou být použita malá i velká písmena a-z, číslice a znaky 8bitových kódování v rozmezí 127 – 255. Jména nesmí začínat číslicí a jsou *case-sensitive*, tzn.

rozlišuje se velikost písmen. Jazyk PHP poskytuje dynamické odkazování na proměnné pomocí řetězce obsahujícího název odkazované proměnné a konstruktu `$$`.

```
<?php
$a = 1;
$b = 'a';

// vytiskne číslo 1
echo $$b;
?>
```

Konstanty mají globální platnost a jejich jména, která musí splňovat stejná pravidla jako u proměnných, je podobně jako v jiných jazycích doporučeno psát velkými písmeny. Hodnoty mohou být pouze typu *boolean*, *integer*, *float* nebo *string*. Kromě uživatelských a systémových konstant nabízí PHP i tzv. „kouzelné" konstanty, které obsahují údaje o zpracovávaném souboru a jeho kódu (řádek aktuálního skriptu, jméno volané funkce apod.).

Názvy funkcí jsou *case-insensitive*, tzn. nerozlišuje se velikost písmen[9], a musí splňovat stejné podmínky jako u proměnných. Volání funkce může být až na výjimky provedeno dynamicky podle hodnoty proměnné. PHP podporuje proměnný počet parametrů, programátor může určit implicitní hodnoty, které jsou přiřazeny, pokud je funkce volána s menším počtem parametrů. Ty mohou být předávány hodnotou i odkazem. Jazyk nabízí funkce pro získání počtu parametrů předaných při volání funkce a jejich hodnot. Od verze 5.3.0 jsou vývojářům k dispozici anonymní funkce.

```
<?php
// sečte všechny předané parametry a vrátí výsledek
function sečti()
{
    $parametry = func_get_args();

    $součet = 0;
    foreach ($parametry as $hodnota)
    {
        $součet += $hodnota;
    }
    return $součet;
}
```

[9] platí pro písmena anglické abecedy

```
$výsledek = sečti(198, 4, 29, 7);
?>
```

Objektový model PHP obsahuje většinu standardních rysů OOP, chybí však např. dědění z více tříd nebo přetěžování metod. U názvů tříd a metod nerozhoduje velikost písmen[10]. Datové a funkční členy objektů, které jsou rozlišeny na veřejné, chráněné a soukromé, jsou zpřístupněny přes operátor `->`. Instance třídy je vytvořena pomocí klíčového slova `new`, proměnná `$this` odkazuje na aktuální objekt. Podporována je přímá manipulace s objekty vrácenými z voláných metod a funkcí. Statické členy a konstanty jsou zpřístupněny buď pomocí klíčového slova `self`, pokud se jedná o členy vlastní třídy, nebo `parent` u rodičovské třídy, následovaným operátorem `::`. Předek třídy je určen klíčovým slovem `extends` za jejím názvem, implementace rozhraní pomocí `implements`, následovaným názvem třídy, resp. rozhraní. Systém výjimek je podobný jako v jiných jazycích. Základní třídou je *Exception*, která může být rozšířena odvozenými třídami. Verze PHP 5.3.0 a vyšší také podporuje jmenné prostory pro třídy, funkce a konstanty.

```
<?php
class Rodič
{
    protected $proměnná = 5;

    public function vypiš()
    {
        echo $this->proměnná;
    }
}

class Potomek extends Rodič
{
    public function vypiš()
    {
        echo 'Volám metodu rodiče... ';
        parent::vypiš();
    }
}

$objekt = new Potomek();
$objekt->vypiš();
?>
```

Operátory jsou několika druhů, zapisované v infixové notaci. Uživatel má k dispozici kromě základních unárních a binárních aritmetických, bitových, lo-

[10] platí pro písmena anglické abecedy

gických a přiřazovacích operátorů rozšířených o test datových typů proměnných, operátor pro ignorování chybových výstupů `@` nebo provádění systémových příkazů[11] uzavřených do dvojice znaků `‘`. Stejně jako v syntaxi jazyků rodiny C je podporován zkrácený zápis některých operátorů v kombinaci s přiřazením a ternární operátor `?:`. Typ objektu je možné ověřit operátorem `instanceof`.

Ve většině aplikací je nejpoužívanějším datovým typem proměnných řetězec, PHP je proto bohaté na funkce pro zpracování řetězců. Operátor `.` slouží k jednoduchému spojení řetězců. Definice hodnoty typu řetězec může být provedena čtyřmi způsoby: jednoduchými uvozovkami, dvojitými uvozovkami, kde je možné použít speciální sekvence znaků pro nový řádek apod. nebo výpis hodnoty proměnných, a tzv. *heredoc* a *nowdoc* syntaxí, které využívají operátor `<<<` a slouží především pro víceřádkové řetězce.

```
<?php
$typ = 'řetězci';

// obsah proměnné: Práce s~řetězci je v~PHP snadná
$text = "Práce s~$typ je v~PHP snadná";

// do proměnné jsou doplněny i konce řádků
$obsah = <<<TEXT
    Heredoc syntaxe slouží především
    pro řetězce, které jsou dlouhé
    přes několik řádků.
TEXT;
?>
```

Aplikace často bývá rozdělena do více souborů, které lze načítat pomocí příkazů `include` nebo `require`, případně od nich odvozených funkcí s příponou „_once“, jež zajistí, aby byl požadovaný soubor načten jen jednou za celý běh aplikace. Pokud je to povoleno, PHP také může pomocí výše uvedených funkcí jednoduše načítat obsah souborů ze vzdálených serverů.

[11] pokud je to povoleno

2.3. Historie aplikačních rámců pro PHP

Předobrazy dnešních frameworků v podobě knihoven se zobecněnými funkcemi začaly vznikat již koncem 20. století s prvními veřejnými verzemi PHP. Přestože kód PHP byl od svých nejranějších verzí vkládán přímo do HTML kódu, díky lepší přehlednosti a oddělení logické a prezentační vrstvy se staly populární tzv. „šablonovací systémy“. Za všechny jmenujme nejznámější Smarty, jehož syntaxe se stala defacto standardem. S nástupem PHP 4.0 postaveným na *Zend Enginu* se objevilo množství aplikačních rámců dostupných pro veřejnost, které byly často sestaveny z rozšíření a knihoven různých autorů. Mnohé z nich byly později přepsány do přívětivějšího prostředí páté verze PHP a jsou vyvíjeny i v současnosti. Mezi průkopníky na poli PHP frameworků patří aplikační rámec *Horde*, *Seagull*, *Zoop* nebo známé *Prado*.

PHP umožňuje i začátečníkům bez velkých zkušeností rychle vytvořit dynamickou internetovou stránku. S přibývající délkou kódu a potřebou jej zpětně upravovat se však vývoj webu zpomaluje. Požadavky uživatelů proto směřovaly k oddělení vrstev aplikace. Ideálním adeptem se stal návrhový vzor *Model-View-Controller*, poprvé použitý v jazyce Smalltalk již v roce 1979 [6], který pro širokou veřejnost znovu objevil framework Ruby on Rails postavený na jazyku Ruby. Vydání PHP 5 s upraveným přístupem k objektově orientovanému programování zákonitě znamenalo masivní rozvoj aplikačních rámců s podporou MVC.

Firma Zend Technologies, založená dvěma spolutvůrci současné podoby PHP Andi Gutmansem a Zeevem Suraskim, vyvíjí od roku 2005 *Zend Framework*, hojně používáný po celém světě. Velkou popularitu si získal rovněž aplikační rámec *Yii* od zakladatele výše zmíněného frameworku Prado, jehož první verze byla vydána v roce 2008. O dva roky dříve začal vývoj aplikačního rámce *Nette*, kolem nějž se vytvořila velká základna českých uživatelů. Mezi další známé frameworky patří *CakePHP*, dříve komerční projekt *Symfony*, velmi rychlá *Kohana* nebo *Code Igniter*.

Téměř každá větší aplikace v PHP je postavena na některém z aplikačních rámců, jež často implementují vlastnosti komerčních projektů nebo jiných programovacích jazyků a nabízí tak plnohodnotné prostředí pro vývoj webových aplikací. K dnešnímu dni dosahuje počet frameworků pro PHP několika desítek, většina je šířena zdarma.

Následuje ukázka použití aplikačního rámce *Nette* k vytvoření jednoduchého formuláře s textovým polem pro zadání jména, zatrhávacím políčkem a víceřádkovým polem pro zadání vzkazu. Framework automaticky formulář vykreslí, po odeslání ověří hodnoty polí podle zvolených pravidel a v případě, že není nalezena chyba, jsou hodnoty vypsány uživateli.

```
<?php
$form = new Form();
```

```
$form->addText('name', 'Jméno')
     ->addRule(Form::FILLED, 'Zadej své jméno');
$form->addCheckbox('promo', 'zasílejte mi reklamu');
$form->addTextArea('text', 'Vzkaz')
     ->addRule(Form::FILLED, 'Něco napiš');
$form->addSubmit('send', 'Odeslat');

if ($form->isSubmitted())
{
    if ($form->isValid())
    {
        echo 'Formulář byl odeslán. Vaše zadání bylo:<br />';

        $values = $form->getValues();
        Debug::dump($values);
        exit;
    }
}

echo $form;
?>
```

3. Analýza ukázkové aplikace

Úvodní etapou životního cyklu většiny softwarových systémů je analýza, která se zabývá studiem problému před tím, než jsou provedeny kroky vedoucí k jeho řešení. Úkolem analýzy je shromáždění požadavků na vyvíjený systém, porozumění mu a sestavení jeho specifikace. [7]

V každém z vybraných frameworků a následně i vlastním frameworku byla vytvořena aplikace jednoduchého blogu se správou obsahu, která ke své implementaci vyžaduje použití funkcionality, jež by měly aplikační rámce standardně poskytovat. Tato kapitola přiblíží analýzu aplikace, konkrétně požadavky a dva různé pohledy na systém, které se vzájemně kombinují – první zaměřený na funkcionalitu, druhý na datovou strukturu. Na závěr jsou zmíněny technologie použité k implementaci aplikace.

3.1. Přehled požadavků

Na aplikaci webového blogu je pohlíženo z pohledu obyčejného návštěvníka a přihlášeného správce obsahu. Blog je rozdělen do tří sekcí, k nimž má uživatel přístup z hlavní nabídky. Titulek stránky v liště okna se mění podle vybrané sekce, případně její podsekce.

3.1.1. Návštěvník

Na hlavní stránce webu je pro přehlednost zobrazen vždy jen určitý počet příspěvků řazených podle data vložení od nejnovějšího, z každého pak jen několik úvodních znaků obsahu. Zobrazené příspěvky lze vybrat omezením měsíce vydání, nebo určením štítku, kterými jsou příspěvky označeny. K posouvání mezi vybranými příspěvky slouží stránkování. Každý příspěvek lze otevřít na samostatné stránce, kde je zobrazen jeho celý text a komentáře návštěvníků s formulářem pro vkládání nového komentáře, pokud to není u daného příspěvku zakázáno. Návštěvník musí pro úspěšné odeslání zadat své jméno, text komentáře a bezpečnostní kód, který slouží k ověření lidského uživatele – CAPTCHA[12]. Kromě příspěvků je možné zobrazit stránku s textem o autorovi blogu.

3.1.2. Správce

Správa blogu je dostupná z hlavní nabídky po zadaní hesla administrátora, je možné zvolit také dlouhodobé přihlášení. Správce může vytvářet, upravovat a mazat příspěvky. Tvorba a úprava příspěvku je usnadněna editorem obsahu s pokročilými formátovacími funkcemi. U příspěvku je možné zakázat zobrazení a přidávání komentářů a označit jej štítky, podle nichž je možné příspěvky filtrovat. Nevyužité štítky nesmí zůstávat v databázi. Komentáře, které jsou zobrazeny

[12] Completely Automated Public Turing test to tell Computers and Humans Apart

vždy jen pro vybraný příspěvek, může správce upravovat, mazat a jednotlivě zakazovat jejich zobrazení. K dispozici je změna obsahu sekce o autorovi s využitím výše zmíněného editoru. V neposlední řadě může správce měnit své jméno zobrazené u příspěvků a heslo k přístupu do administrace. Ze správy blogu je možné se odhlásit, což zruší dlouhodobé přihlášení, pokud byla tato možnost zvolena.

3.2. Funkční model

Funkčně orientovaný přístup ve strukturované analýze pohlíží na systém jako na množinu funkcí. Nejčastěji používaným modelem je DFD - *Data Flow Diagram*, česky diagram datových toků, který umožňuje zobrazit systém jako síť procesů, jež plní dané funkce a předávají si mezi sebou data. Proces je jedinou komponentou, která mění data, k jejich ukládání slouží paměť. Vnějším činitelem je terminátor, který komunikuje se systémem – často je to lidský uživatel. Pro větší přehlednost bývá funkční model rozložen do diagramů několika úrovní. Nejvyšší úroveň, kde je celý systém zobrazen jako jediný proces, se nazývá kontextový diagram, který se někdy uvádí jako součást *modelu okolí*. Vnitřní části systému popisuje *model chování*, kam patří systémový DFD se zobrazením hlavních procesů, jež mohou být dále dekomponovány na samostaných diagramech. [7]

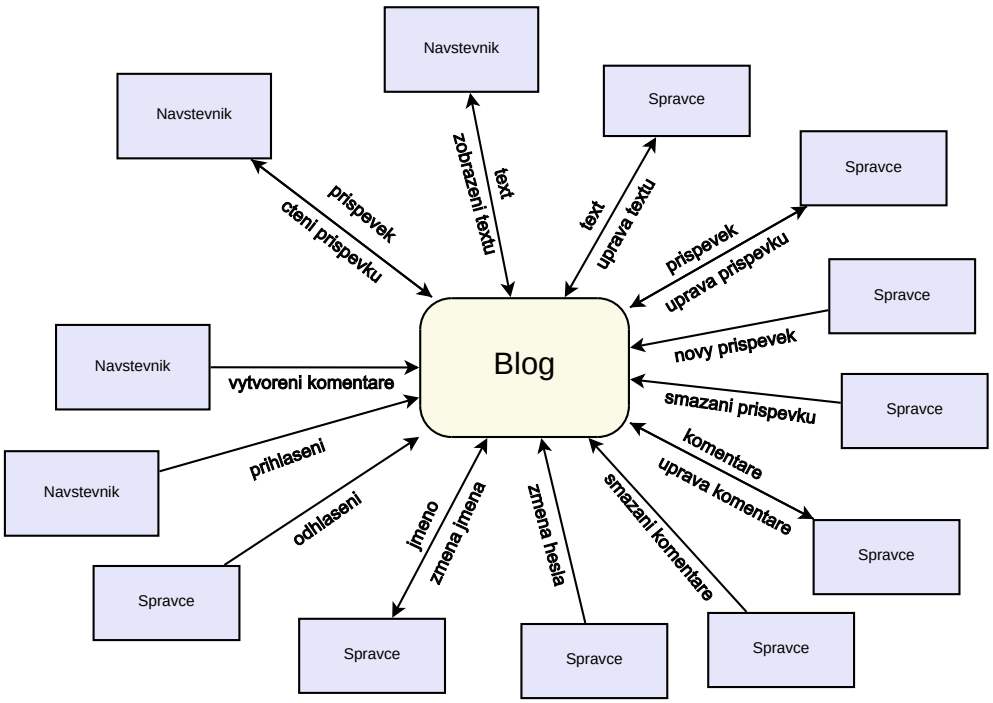
K tvorbě diagramů byl zvolen mutliplatformní program pro kreslení strukturovaných diagramu Dia[13], vydaný a šířený pod licencí GPL[14]. Proces je zobrazen jako zaoblený obdélník, uzavřený vyplněný obdélník představuje terminátor a paměť je znázorněna jako neuzavřený obdélník s tučným písmem. Cesty mezi komponentami označují toky dat, šipka určuje směr toku. Níže uvedené diagramy neobsahují kvůli omezení použitého software diakritiku, na přiloženém CD je však k dispozici kompletní sada DFD i s českými znaky.

Kontextový diagram zobrazený na obr. 1 znázorňuje celou aplikaci jako jeden proces, s nímž komunikují terminátory, které jsou v tomto případě dvou typů – návštěvník a správce. Cesty mezi procesem a terminátory představují výše popsané operace, jež může uživatel provádět. Běžný návštěvník si může nechat zobrazit různě filtrované příspěvky, napsat k nim komentář, prohlédnout si stránku s textem a přihlásit se do administrace blogu. Správce má práva na administraci příspěvků – jejich vytváření, úpravu a mazání – a jejich komentářů – ty může upravit nebo smazat. Dále může upravovat obsah stránky s textem o autorovi, změnit heslo pro vstup do administrace, jméno autora příspěvků a samozřejmě se z administrace odhlásit.

Seznam událostí pro terminátor **návštěvník**:

- návštěvník žádá zobrazení příspěvků

[13] http://live.gnome.org/dia
[14] GNU General Public License – http://www.gnu.org/licenses/gpl.html



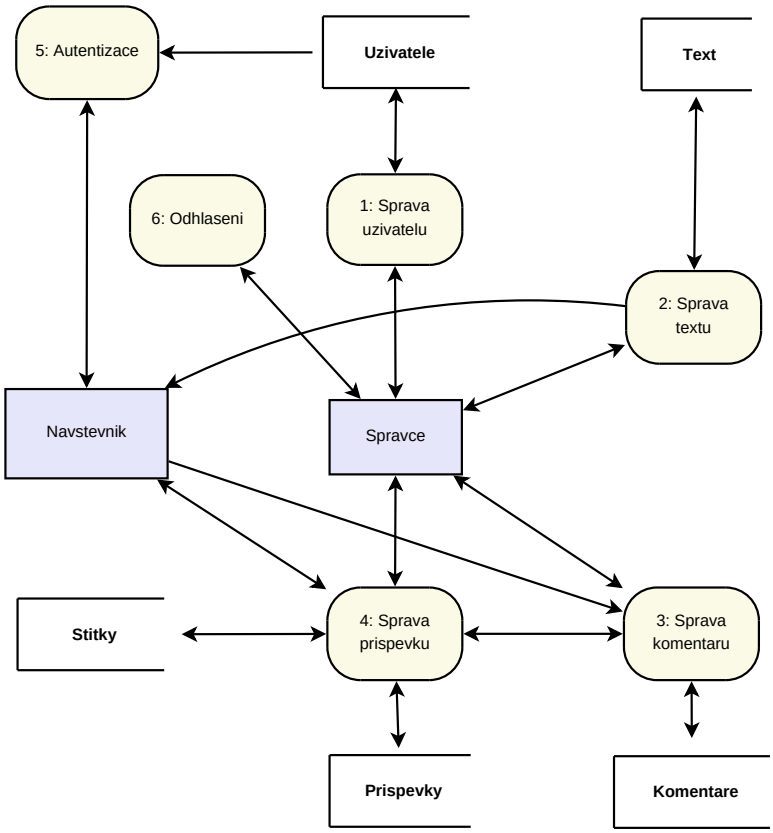
Obrázek 1. Kontextový diagram aplikace webového blogu.

- návštěvník vytváří komentář
- návštěvník žádá zobrazení textu
- návštěvník se přihlašuje do administrace

Seznam událostí pro terminátor **správce**:

- správce vytváří nový příspěvek
- správce upravuje příspěvek
- správce maže příspěvek
- správce upravuje komentáře příspěvku
- správce maže komentáře příspěvku
- správce upravuje stránku s textem
- správce mění jméno autora příspěvků blogu
- správce mění heslo pro vstup do administrace
- správce se odhlašuje z administrace

Na obr. 2 je systémový diagram datových toků, někdy nazývaný též DFD nulté úrovně, jenž zobrazuje aplikaci rozloženou na několik hlavních procesů. Pro přehlednost jsou procesy jednoznačně očíslovány a terminátory s nimi podle svého typu komunikují. Jednotlivé procesy pojmenované podle vykonávané činnosti jsou dekomponovány na vlastních diagramech, kam se přenáší číslování pomocí tečkové notace. Na obrázku nejsou pro zjednodušení a větší přehlednost znázorněny a pojmenovány všechny datové toky, ze stejného důvodu jsou také často obousměrné. Nižší úrovně hiearchie DFD aplikace nebudou v textu již dále rozebírány; pro zájemce jsou k dispozici na přiloženém CD.



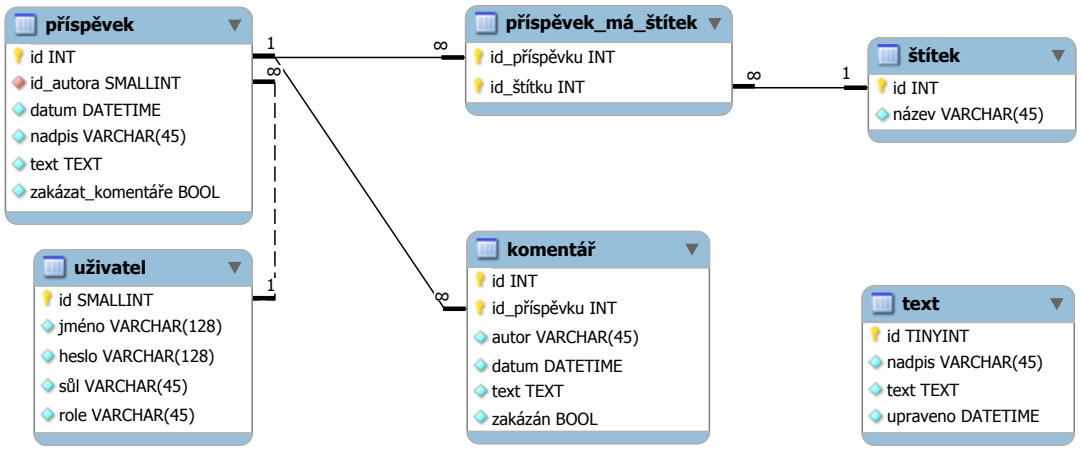
Obrázek 2. Systémový diagram datových toků aplikace webového blogu.

Systémový DFD obsahuje šest hlavních procesů, dva terminátory a pět pamětí. Nejvytíženějším procesem je **správa příspěvků**, s nímž může komunikovat nepřihlášený návštěvník i správce, v závislosti na požadavku spolupracuje s procesem **správa komentářů** a žádá si data z pamětí **příspěvky** a **štítky**. Naopak proces **správa uživatelů** načítá data pouze z paměti **uživatelé**, jež je jako jediná k dispozici dvěma procesům, a komunikuje s ním pouze terminátor správce. Druhým procesem, který může používat jen pro správce, je **odhlášení**.

3.3. Datový model

Datově orientovaný přístup návrhu systému má za úkol nalézt fundamentální datové struktury aplikace, které definují konceptuální model pro databázi systému. Transformace dat jsou pro něj méně podstatné. Datové modelování hledá entity, vztahy mezi nimi a jejich atributy. Nejpoužívanějším nástrojem je zde *entitně relační diagram*, který znázorňuje datový model systému. Na něm ukazuje neměnné atributy a strukturu dat a vyjadřuje vztahy, jež nejsou zachyceny ve funkčních modelech. K dosažení vhodné struktury se používá *normalizace*, která transformuje datový model do podoby, jež má předejít problémům a anomáliím při práci s daty. Seznam definic datových prvků systému popisuje datový slovník – konkrétně význam dat v tocích a pamětech na DFD, entity a atributy v ER diagramu a další klíčová slova ve specifikaci. [7]

Entitně relační diagram znázorněný na obr. 3 byl vytvořen pomocí vizualizačního nástroje pro návrh databází MySQL Workbench[15], šířeného pod licencí GPL[16]. Navrhovaná aplikace blogu patří z hlediska dat k jednodušším, proto se celý datový model vejde na jediný diagram.



Obrázek 3. Entitně relační diagram aplikace webového blogu.

V návrhu datového modelu aplikace nalezneme pět hlavních entitních typů – **příspěvek**, **štítek**, **komentář**, **uživatel** a **text**. Příspěvek blogu je jednoznačně určen primárním klíčem `id`. Jméno autora není kvůli snazšímu udržování aktuálního jména správce blogu uvedeno jako řetězec, nýbrž jako cizí klíč odkazující na entitní typ **uživatel**. Další atributy určují nadpis, obsah, datum vydání a povolení zobrazení příspěvku a vkládání komentářů.

Každý příspěvek může být označen libovolným počtem štítků, které slouží k filtrování podle tématu. Entitní typ **štítek** obsahuje dva atributy – primární

[15] http://wb.mysql.com/
[16] GNU General Public License - http://www.gnu.org/licenses/gpl.html

klíč `id` a `název`. Protože vztah mezi příspěvky a štítky je typu M:N, byl mezi tyto entitní množiny vložen vazební entitní typ `příspěvek\_má\_štítek`, který má jako jediné dva atributy primární klíče obou výše zmíněných entitních množin a vztah s nimi 1:N. Plná čára znázorňuje fakt, že vztahy jsou identifikující, což znamená, že jedna entita závisí na druhé – primární klíč entity **příspěvek** je zároveň primárním klíčem entity **příspěvek_má_štítek**; to samé platí pro entitu **štítek**.

Komentář je jednoznačně přiřazen vždy jednomu příspěvku, naopak příspěvek může mít komentářů více. Vztah mezi těmito entitami je tedy 1:N a je opět identifikující. Atributy komentáře představují jeho primární klíč, cizí primární klíč určující komentovaný příspěvek, jméno komentátora, datum vytvoření, textový obsah a zákaz jeho zobrazení.

Entitní typ **uživatel** obsahuje kromě primárního klíče jméno správce blogu a údaje potřebné k autentizaci přístupu do administrace aplikace. Konkrétně je to heslo, které je zašifrováno kombinací jednoduchých kryptografických hashovacích funkcí MD5[17] a SHA1[18], jež jsou součástí jak PHP, tak většiny databázových systémů. Protože u obou funkcí byly již před několika lety nalezeny kolize[19], je bezpečnost posílena použitím tzv. „soli“, kdy je k původnímu řetězci hesla přidán řetězec náhodných znaků, jež jsou uloženy v atributu `sůl`. Kontrola a vytvoření nového hesla se provádí podle následujícího vzorce; infixový operátor `.` spojuje řetězce.

```
SHA1(sůl . MD5(heslo . sůl))
```

Atribut **role** slouží k určení práv uživatele. Protože v ukázkové aplikaci blogu je použita pouze jedna entita typu **uživatel**, bude tento atribut (stejně jako primární klíč) plně využit až při případném rozšíření aplikace např. o registraci návštěvníků. Vztah mezi entitními typy **uživatel** a **příspěvek** je 1:N. Protože žádná z entit není závislá na druhé, není vztah identifikující, což znázorňuje přerušená čára.

Entitní množina **text** obsahuje údaje pro textovou sekci blogu – nadpis sekce, její obsah a datum poslední úpravy. Stejně jako u entitního typu **uživatel** je v ukázkové aplikaci použita jediná entita **text** a atribut `id` nemusí být proto plně využit.

Pro bezproblémovou manipulaci s daty byl uvedený datový model transformován do 3. normální formy, což znamená, že splňuje následující podmínky: [7]

- *1. NF* – entitní množina neobsahuje opakující se skupiny atributů a všechny její komponenty jsou atomické

[17] Message-Digest algorithm verze 5
[18] Secure Hash Algorithm
[19] funkce vrací pro dva různé řetězce stejný otisk

- *2. NF* – každý neklíčový atribut je plně funkčně závislý na každém kandidátním klíči[20]
- *3. NF* – každý neklíčový atribut je netranzitivně závislý na každém kandidátním klíči

3.4. Použité technologie

Jazyk PHP byl použit pro tvorbu většiny zdrojových kódů. Výstup aplikace ve formátu HTML uživatel prohlíží v internetovém prohlížeči, který zpracovává i dodatečné skripty v jazyce *JavaScript*. Ty byly použity pro lepší interakci webové stránky s návštěvníkem, jemuž jsou zobrazeny potvrzovací dialogy, jež zabraňují odeslání požadavku na server, pokud to není nutné, a také usnadněna úprava příspěvků blogu pomocí WYSIWYG[21] editoru *TinyMCE*[22] volně šířeného pod licencí LGPL[23]. Aplikace používá šablonu *EarthlingTwo* dostupnou zdarma na stránce freecsstemplates.com a šířenou pod licencí CC-BY[24].

[20] atribut, který je u každého záznamu unikátní
[21] What You See Is What You Get – volně přeloženo jako „dostaneš, co vidíš“
[22] http://tinymce.moxiecode.com/
[23] GNU Lesser General Public License – http://www.gnu.org/licenses/lgpl.html
[24] Creative Commons Attribution 2.5 – http://creativecommons.org/licenses/by/2.5

4. Srovnání existujících frameworků

V současné době je k dispozici velké množství aplikačních rámců pro PHP, které jsou poskytovány zdarma. Během posledních let bylo provedeno několik srovnání, jež byla většinou zaměřena na výkon frameworků, nikoliv na složitost použití a poskytovanou funkcionalitu, což bude hlavní záměr této kapitoly. Pro účely testování byly vybrány dva aplikační rámce, které se v současnosti v Česku těší asi největší oblibě – „domácí produkt“ *Nette Framework* a celosvětově známý *Zend Framework*.

Začínající uživatel obou výše zmíněných frameworků by měl disponovat základními znalostmi jazyka PHP a znát principy objektově orientovaného programování.

4.1. Nette Framework

Autor:	David Grudl
Internetové stránky:	http://nette.org/
Licence:	CC BY-SA[25]
Testovaná verze:	0.9.3 (vydaná 3. 2. 2010)
Podporovaná verze PHP:	5.2 a vyšší
Podpora vlastností:	
Návhový vzor MVC	✓
Databázové rozhraní	✗
Šablonovací systém	✓
Cachování	✓
Správa formulářů	✓
Autentizace uživatele	✓
Překlad textů	✗
Ladící prostředí	✓
Česká dokumentace	✓

Tabulka 1.

Aplikační rámec Nette od českého autora Davida Grudla si za čtyři roky své existence získal pro svou jednoduchost a rychlost učení velkou popularitu, především v České republice. Framework je pravidelně aktualizován a několikrát

[25] Creative Commons Attribution-ShareAlike (Uveďte autora-Zachovejte licenci) http://creativecommons.org/licenses/by-sa/3.0/cz/

ročně je vydána nová verze. Podle nezávislého testu serveru root.cz z 11. září 2008 je to jeden z nejrychlejších frameworků pro PHP [8]. Pro jeho nasazení na vývoj internetových prezentací a aplikací se rozhodli autoři webu prezidenta ČR Václava Klause klaus.cz nebo vydavatelství Mladá Fronta pro weby svých tištěných periodik (mfplus.cz, sedmicka.cz, E15.cz).

Nette je k dispozici ke stažení v několika verzích: pro PHP 5.2 bez a s prefixy tříd[26] a pro PHP 5.3, které podporuje jmenné prostory[27]. Ve stáhnutém balíku uživatel najde kromě samotného frameworku dabázovou knihovnu, test požadavků pro běh aplikačního rámce, základní kostru nové aplikace a několik příkladů použití. Kostra aplikace (tzv. skeleton) obsahuje adresářovou strukturu se základními soubory, standardní chybová hlášení a předpřipravené přihlášovaní pro uživatele. Pro ostré nasazení aplikace na serveru se může hodit jednosouborová minimalizovaná verze Nette. Navíc není třeba používat všechny standardní rysy, framework lze využít pouze pro specifické problémy – např. tvorba a zpracování formulářů – nebo v kombinaci s jinými aplikačními rámci, kde se hodí použít verzi s prefixy tříd.

Internetové stránky frameworku nabízí přehlednou dokumentaci. Začínající uživatelé mají k dispozici úvodní tutoriál, který je na příkladu aplikace jednoduché návštěvní knihy seznámí s použitím Nette. Rysy aplikačního rámce jsou podrobněji popsány v Příručce programátora, některé však nepříliš detailně a bez názorných příkladů. Uživateli tak nezbývá než hledat návod na použití některých tříd a funkcí na diskuzním fóru nebo samostatně zkoušet. Samozřejmostí je API[28] reference, která je v angličtině a k dispozici rovněž ke stažení. Vzhledem k tomu, že je framework v neustálém vývoji, může uživatel občas narazit na některé nefunkční, nepřesné nebo zastaralé části dokumentace, která je pro základní použití aplikačního rámce sice dostačující, při hledání nápovědy ke složitějším prvkům však často nesplní účel.

Jednou z předností Nette je vysoká úroveň zabezpečení aplikace. Automaticky jsou ošetřovány veškeré uživatelské vstupy, výstupy skriptů jsou standardně upravovány podle kontextu (HTML, JavaScript, XML…), je minimalizována možnost zcizení nebo nahrazení uživatelských sessions. Také nabízí možnost přidat ochranu proti útokům z jiných webů. Uživatel je navíc veden k čistému kódu ohlašováním PHP chyb úrovně `E_NOTICE`, které mohou nastat např. při použití nedeklarované proměnné, což není většinou považováno za chybu. Toto je v běžném nastavení PHP vypnuto, framework si však sám nastavení konfiguruje (pokud je to povoleno).

Aplikační rámec se snaží automaticky podle IP adresy serveru rozlišit prostředí, ve kterém je aplikace spuštěna – produkční nebo vývojářské –, uživatel

[26] pro použití společně s jinými knihovnami a frameworky

[27] v době psaní tohoto textu většina webhostingů PHP 5.3 nenabízí

[28] Application Programming Interface – rozhraní pro programování aplikací

se tak nemusí starat o úpravu zdrojového kódu při přenášení aplikace do ostrého provozu. Podle prostředí se rozhoduje o konfiguraci (časová zóna, znaková sada, typ a údaje pro připojení k databázi atd.), jež se zapisuje do souboru *config.ini* v adresáři aplikace, a zda budou zveřejněna chybová hlášení, která jsou zobrazena v grafickém prostředí *debuggeru* s názvem Laděnka. Uživateli je kromě popisu chyby či výjimky, názvu souboru a řádku s chybou ukázán náhled na obsah souboru, při jehož zpracování nastala chyba, zásobník volání funkcí a metod aplikace, proměnné prostředí, načtené soubory, hlavičky požadavku i odpovědi HTTP a další informace podle druhu chyby. Občas se mohou objevit nepřesná chybová hlášení nebo zásobník volání neobsahuje odpovídající položky. V nové verzi frameworku je při zapnutém ladění zobrazen malý panel s informacemi o době generování výstupu, množství využité paměti a seznamem chyb nízké úrovně (`E_WARNING` a `E_NOTICE`), jež nejsou nově zapsány na výstup skriptu.

Nette používá návrhový vzor *Model-View-Presenter*, který je odvozen ze známějšího vzoru MVC, a narozdíl od něj klade větší důraz na odstínění vrstev aplikace. Hlavní roli má *presenter*, který od *modelu* žádá data, a v upravené podobě je poskytuje na zobrazení komponentě *view*, jež je v Nette založena na vlastním šablonovacím systému. *Model* by neměl vůbec vědět o existenci *presenteru* a *view*, jež může být podle zvolené koncepce pasivní, kdy zpracovává data předaná *presenterem*, nebo aktivní a načítá je samo z *modelu*.

Šablonovací systém Nette má klasickou syntaxi složených závorek, navíc však umožňuje zápis pomocí tzv. *n:atributů*, jež zpřehledňují kód při použití podmínek a iterací. Viz. následující příklad.

```
<li n:foreach="$items as $item">{$item}</li>
```

Uvnitř složených závorek je použita lehce upravená syntaxe jazyka PHP, kdy jsou vynechávány především závorky u řídících příkazů (cykly, podmínky) a lze použít speciální konstrukty pro Nette jako aplikování *helperů* s parametry na výstup proměnných, které lze navíc zřetězit. Na výstupu je zobrazen nejdříve *layout*, do nějž jsou na určená místa doplněny obsahové bloky generované jednotlivými *view* nebo komponentami. Aby se skripty negenerovaly při každém požadavku složitě znovu, disponuje aplikační rámec vyrovnávací pamětí (*cache*), již lze mimo ukládání šablon využít pro libovolná data. Protože je funkcionalita cache standardně zapnuta a neexistuje mechanismus pro její úplné vypnutí, mohou při vývoji nastat nečekané problémy.

O načítání souborů s třídami, případně rozšiřování konfigurační direktivy PHP *include_path*[29], se uživatel nemusí starat díky třídě *Robot Loader*, jež automaticky prochází adresářový strom aplikace a zaznamenává cesty k souborům a v nich definovaným třídám. Aby nedocházelo ke zpomalování aplikace, výsledky hledání

[29] seznam cest, které funkce PHP pro práci se soubory procházejí, pokud není určena absolutní cesta k souboru

se ukládají a následně jsou načítány bez nutnosti procházet adresáře při každém spuštění. Při přidávání nových tříd je nutné cache vymazat.

Framework nabízí mnoho vestavěných tříd, např. pro práci s FTP[30], URI[31], HTTP hlavičkami, AJAXem[32] nebo pro odesílání e-mailových zpráv. Mnohé z nich nepocházejí původně od autora, ale od velmi aktivní a stále rostoucí komunity, a postupem času byly do aplikačního rámce integrovány. Uživatelská komunita kolem Nette vyprodukovala rovněž množství doplňků, pluginů a komponent rozšiřujících framework, které jsou na webu k dispozici ke stažení v několika kategoriích. S aktualizacemi aplikačního rámce bohužel některé z komponent přestávají být funkční a uživateli si je musí případně upravit.

Samotné Nette nedisponuje funkcemi pro práci s databázemi, je proto nutné použít rozšíření třetí strany[33]. Autor doporučuje svou vlastní knihovnu pro PHP 5 s názvem „dibi“, na niž je odkazováno již v úvodních tutoriálech. Na vlastních webových stránkách dibi je API dokumentace, chybí však podrobnější příklady a vysvětlení. Dibi zvládá bez problémů různá kódování databází, včetně všech českých.

Pro vývojáře vícejazyčných webových aplikací sice framework nabízí detekci preferovaného jazyka uživatele podle hlaviček HTTP požadavku prohlížeče, funkce pro překlad textu však nejsou implementovány a uživatel musí spoléhat na různá rozšíření. Doporučuje se využít rozšíření na základně lokalizačního nástroje gettext[34], které je však staršího data a není plně kompatibilní s aktuální verzí Nette. Webové stranky navíc neposkytují žádný ucelený návod pro začátečníky, použití proto není jednoduché.

Další silnou stránkou frameworku je směrování požadavků na příslušné *presentery* ve spolupráci s modulem serveru Apache *mod_rewrite*, který slouží ke zpracování a přepisování URL[35]. *Routování* je dvousměrné – zpracovává se podle něj požadavek i generují adresy odkazů. Uživatel definuje cesty směrování ve vstupním souboru aplikace před jejím spuštěním pomocí regulárních výrazů obohacených o sekvence určující nepovinné parametry, jejich standardní hodnotu a přiřazení do proměnných. Ty jsou dostupné ve funkcích obsluhujících požadovanou akci, je však nutné je ve správném pořadí zadat jako parametry do definice funkce. Pokud není určena žádná cesta, vytvářené URL odkazů mají klasickou podobu s proměnnými oddělenými znakem `&`.

Největší devizou Nette je tvorba a zpracování webových formulářů. Při přidávání jednotlivých prvků se zadává také jejich popisek (*label*) a volitelné množství

[30] File Transfer Protocol – protokol rodiny TCP/IP pro přenos souborů mezi počítači
[31] Uniform Resource Identifier – „jednotný identifikátor zdroje“
[32] Asynchronous JavaScript and XML
[33] orig. Third-party software component
[34] http://www.gnu.org/software/gettext/
[35] Uniform Resource Locator – „jednotný lokátor zdrojů“

ověřovacích filtrů, k nimž je zadán vlastní text chybového hlášení. Uživatel může rovněž určit, aby validace proběhla jen za splnění určité podmínky. U základních typů ověření (typ zadané hodnoty, prázdné pole…) se na výstupu vygeneruje i obslužný kód v jazyce JavaScript, který vypíše chybovou zprávu ještě před odesláním dat na server a šetří tak čas uživatele aplikace. Vykreslení celého formuláře včetně chybových hlášení je možné jediným příkazem v šabloně. Standardně se provede do tabulky, uživatel si však může zvolit vlastní způsob pomocí úpravy *dekorátorů* nebo vypsáním jednotlivých prvků samostatně, kde už počet řádků kódu narůstá. Zpracování odeslaných dat se provede ve specifikované funkci při vytváření formuláře, framework si sám poradí s případně zapnutou direktivou *magic_quotes_gpc*[36]. Formuláře jsou automaticky chráněny proti XSS[37] a je možné přidat i zabezpečení proti CSRF[38].

Kódování výstupu není možné nastavit a je vždy v 8bitovém UTF. Autor argumentuje především zpracováním dat z formulářů, kde může uživatel zadat vstup v libovolné znakové sadě nehledě na kódování stránky. Vývojáři, kteří používají v kódu českou diakritiku, mohou narazit u názvů komponent, které Nette kontroluje regulárním výrazem `[a-zA-Z0-9_]+`, což neodpovídá standardům PHP.

Framework Nette je vhodný pro různorodé typy webových aplikací, velké projekty však budou pravděpodobně postrádat některé pokročilé prvky, které bude muset vývojář sám implementovat, nebo použít některé z existujících řešení. Uživatelé ocení jednoduchost použití a strmou křivku učení rysů aplikačního rámce. Nespornou výhodou pro českého uživatele je hojně navštěvované komunitní fórum v češtině.

Test rámce Nette byl proveden na ukázkové aplikaci blogu, jejíž analýza byla provedena na straně 17. Zdrojové soubory jsou k dispozici na přiloženém CD, na internetové adrese `http://bc.lipe.cz/nette/` je běžící aplikace k nahlédnutí.

Následující ukázka kódu ukazuje, jak ve frameworku Nette probíhá autentizace uživatele. První část představuje metodu volanou po odeslání formuláře, jež se pokusí ověřit přihlašovací údaje pomocí metody `authenticate` objektu, který je instancí třídy v nastavení určené pro autentizaci. Tato metoda, zobrazená v druhé části příkladu, provede načtení přihlašovacích údajů z databáze a porovná je s odeslanými hodnotami. Pokud dojde k chybě, je vyvolána výjimka, která je zachycena a převedena na chybové hlášení zobrazené formulářem. Jinak je vytvořen objekt identity uživatele, jenž je dále použit při autorizaci požadavku vyžadujícího oprávnění. Volitelně je nastaveno automatické přihlášení.

Metoda umožňující vlastní ověření údajů je snadno implementována i začínajícím vývojářem. Nevýhodou může být nutnost určit samostatnou třídu s metodou ověření v konfiguračním souboru aplikace a používání povinných objektů.

[36] automatické *escapování* příchozích dat pro PHP skripty; nedoporučuje se používat
[37] Cross-Site Scripting – metoda narušení využitím neošetřeného vstupu
[38] Cross-Site Request Forgery – „mezistránkové padělání požadavku“

Nastavení platnosti automatického přihlášení je díky možnosti použití textového formátu intuitivní.

```
public function loginFormSubmitted($form)
{
    try {
        $values = $form->values;
        $this->getUser()->authenticate('admin',
                                        $values['password']);
        if ($values['remember']) {
            $this->getUser()->setExpiration('+ 14 days', FALSE);
        } else {
            $this->getUser()->setExpiration('+ 20 minutes', TRUE);
        }
        $this->getApplication()->restoreRequest($this->backlink);
        $this->redirect('Admin:');
    }
    catch (AuthenticationException $e) {
        $form->addError($e->getMessage());
    }
}

...

public function authenticate(array $credentials)
{
    $uživatel = dibi::fetch("SELECT * FROM [uživatel]" .
                            "WHERE [role] = 'admin' LIMIT 1");

    $heslo = Uživatel::zašifrujHeslo($credentials['password'],
                                     $uživatel->sůl);

    if ($uživatel->heslo !== $heslo)
    {
        throw new AuthenticationException
            ("Neplatné heslo.", self::INVALID_CREDENTIAL);
    }

    return new Identity($uživatel->jméno, NULL, $uživatel);
}
```

4.2. Zend Framework

Autor:	Zend Technologies
Internetové stránky:	http://www.zendframework.com/
Licence:	New BSD License
Testovaná verze:	1.10.4 (vydaná 28. 4. 2010)
Podporovaná verze PHP:	5.2 a vyšší
Podpora vlastností:	
Návhový vzor MVC	✓
Databázové rozhraní	✓
Šablonovací systém	✗
Cachování	✓
Správa formulářů	✓
Autentizace uživatele	✓
Překlad textů	✓
Ladící prostředí	✗
Česká dokumentace	✗

Tabulka 2.

Zend Framework je robustní aplikační rámec pro profesionální webové aplikace postavený na návrhovém vzoru *Model-View-Controller*, který je velmi rozšířený po celém světě a je na něm postaveno několik tisíc webů [9]. Autorem je firma Zend Technologies, založená spolutvůrci současného jádra PHP, jež kromě frameworku vyvíjí řadu nástrojů pro vývoj internetových aplikací, včetně multiplatformního Zend Serveru poskytujícího jednoduchou instalaci webového a databázového serveru, PHP a několika dalších nástrojů pro optimalizaci běhu skriptů. Partnery vývoje jsou i další firmy jako známý Google či Microsoft, které poskytly rozhraní ke svým službám, aby mohly být využity uživateli tohoto frameworku Zend.

Aplikační rámec je k dispozici ke stažení ve dvou balících – menší obsahuje pouze soubory frameworku, větší navíc příklady, testy, jazykové mutace chybových hlášení a různá rozšíření. Příklady použití jsou však většinou zaměřeny na pokročilé prvky, které běžný uživatel nevyužije, a naopak chybí ukázka běžných aplikací. V obou distribucích nechybí dávkový soubor využívající PHP CLI[39], který je možné použít pro tvorbu nového projektu, konfiguraci webové aplikace, vytváření souborů částí MVC, tvorbu tabulek pro databázi SQLite apod. Zakládání nových souborů pomocí tohoto nástroje je automaticky propojí s již exis-

[39] Command Line Interface – rozhraní příkazové řádky

tujícími částmi aplikace, často však dochází k nechtěným úpravám textu, např. odstranění odsazení nebo smazání znaků, v jejichž důsledku nemusí jít skript přeložit a vývojář musí chyby ručně najít a opravit. Pokud se navíc uživatel rozhodne použít českou diakritiku, dojde k poškození[40] XML souboru s informacemi o projektu, který je naštěstí používán výhradně dávkovým souborem, a další akce nad aplikací pomocí tohoto nástroje již nejsou možné nebo není vytvořený soubor použitelný.

Začínající uživatel má možnost seznámit se s aplikačním rámcem pomocí krátkého tutoriálu, v němž je popsáno zprovoznění Zend Frameworku, tvorba nového projektu a základní architektura aplikací. Hlavní část dokumentace, která je kromě angličtiny k dispozici částečně přeložená v pěti dalších jazycích, je zpracována formou referenční příručky, v níž jsou nejprve popsány často používané funkce frameworku jako automatické načítání souborů s třídami, systém šablon, autorizace a autentizace, hledání nebo stránkování. Následují detailní popisy použití jednotlivých tříd, nezřídka se však jedná pouze o nástin hlavních rysů a konkrétní postup pro vytvoření požadovaného prvku aplikace musí uživatel hledat na externích webových stránkách. Poslední částí je API dokumentace, která je rozdělena podle verzí aplikačního rámce a je k dispozici i ke stažení. Vyhledávání na webových stránkách Zend Frameworku občas nevrací požadované výsledky, proto může být rychlejší použít hledání přes specializovanou službu.

Aplikační rámec Zend je od počátku vyvíjen jako modulární. Komponenty jsou navrhovány s co nejmenším počtem vzájemných závislostí, uživatelé tak nemusí využít všechny funkce, které framework nabízí. K dispozici je řada robustních knihoven pro běžné i složitější prvky aplikace – komponenty pro práci s datem, formuláři, HTTP požadavky a odpověďmi, RSS, soubory, službami Google, ověřováním práv uživatele a další, které rozšiřují hlavní části frameworku. Jak již bylo zmíněno, dokumentace často zmiňuje jen základní rysy komponenty a vývojář se i s jednoduchým problémem napoprvé potýká zbytečně dlouho. Příkladem může být systém správy identit uživatelů a jejich oprávnění pro určité akce, který je sice velmi propracovaný, ale v běžné aplikaci by se hodila spíše jeho odlehčená verze se základní funkcionalitou. Kolem Zend Frameworku se během několika let jeho vývoje utvořila velká komunita, která má možnost zasílat své návrhy na vylepšení a přidávat rozšíření na samostatné stránce. Před přidáním do distribuce frameworku musí nové rozšíření projít několika fázemi kontrol a úprav, než je schváleno vývojářským týmem aplikačního rámce.

Různá nastavení aplikace rozdělená podle prostředí, v němž je aplikace spouštěna – např. údaje pro připojení k databázovému systému, povolení hlášení chyb nebo zapnutí a vlastnosti tzv. „zdrojů“ (uživatelské relace, cache, e-mail, lokalizace atd.) – se zapisují do konfiguračního souboru. Zatímco cesta k adresáři frameworku je určena při vytvoření nového projektu pomocí dávkového souboru,

[40] znaky jsou převedeny na entity

pokud chce uživatel využít automatické načítání i pro vlastní soubory, musí složitě pátrat po odpovídajícím kódu pro nastavení této funkcionality. Načítání funguje na systému předpon názvů tříd – vývojář musí určit základní adresář, jmenný prostor aplikace, kterým začíná každý název třídy, a u každého typu zdroje určit relativní cestu, kde se mají soubory hledat. Třída musí být pojmenována ve tvaru *Application_Zdroj_Název*, znak _ v názvu slouží jako oddělovač adresářů. Tímto Zend Framework nahrazuje standardní jmenné prostory OOP, jež plánuje podporovat od své verze 2.0.0. Česká diakritika není v názvech souborů a tříd podporována.

Návrhový vzor MVC nabízí potřebné odstínění aplikační, datové a prezentační logiky. *Controller* obsahuje akce, na které je skript přesměrován při odpovídajícím požadavku ze vstupního souboru[41], v němž jsou upravována společná nastavení pohledů, přesměrování, automatického načítaní apod. Akce připraví data z *modelu* pro výstup, který je proveden určeným *view* a vložen na určené místo do *layoutu* – společného HTML kódu všech stránek aplikace. Zend Framework nemá vlastní šablonovací systém, uživatel však může využít libovolné řešení třetí strany. Pohledy používají klasickou syntaxi PHP, k níž aplikační rámec navíc poskytuje řadu tzv. „pomocníků“, které lze rozšířit o vlastní funkce. Užitečná je zejména správa připojovaných souborů na straně uživatele aplikace – kaskádových stylů nebo skriptů v jazyce JavaScript –, informací o stránce, tvorba menu, drobečkové navigace, odkazů nebo mapy webu.

Směrování požadavků, které odpovídá i za tvorbu adres odkazů, je zapnuto automaticky a standardní formát adres je *ovladač/akce/parametr1/hodnota1/parametr2/hodnota2...*. Uživatel může definovat vlastní pojmenované *routy* pro přesměrování, jejichž název je nutné uvádět při vytváření odkazů v šablonách. Pokud není název určen, je použita standardní cesta ve výše uvedeném tvaru, která automaticky přenáší všechny argumenty z aktuální adresy, což nemusí být vždy žádoucí. K dispozici je několik typů cest, v nichž mohou být použity regulární výrazy nebo pojmenované parametry s implicitními hodnotami. Správa směrování je pro malé aplikace zbytečně složitá a z důvodu nutnosti uvádět název požadované cesty jako parametr pomocníka pro tvorbu odkazů nepříliš flexibilní.

Systém vytváření a zpracování formulářů ve frameworku Zend usnadní uživateli velkou část práce. Všem prvkům, jejichž název může obsahovat i českou diakritiku a mezi nimiž nechybí třeba CAPTCHA, lze nastavit standardní HTML parametry, určit ověření nebo úpravu vstupu a styl jejich vykreslení. Formulář lze zobrazit do pohledu jediným příkazem, kdy jsou prvky vykresleny podle nastavených *dekorátorů*, které mohou být rozšířeny o vlastní styly. Někdy je vhodnější vykreslit každým prvek formuláře nebo jeho část samostatně, protože aplikační

[41] tzv. bootstrap

rámec občas negeneruje validní HTML kód, např. u výše zmíněného ověření lidského uživatele to však způsobuje chyby.

Zend Framework nabízí řadu filtrů pro úpravu a validátorů pro kontrolu vstupu od základní kontroly typů, číselného rozsahu a formátu data po kontrolu formátu čísla kreditní karty, IP adresy, čárových kódů a číslování knih ISBN. Chybová hlášení při neúspěšné validaci jsou sice přednastavena, programátor je ale může upravit ve vstupním souboru inicializací nového překladače nebo přidat vlastní zprávu při manuálním ověřování vstupu. Aplikační rámec poskytuje volitelnou ochranu proti CSRF, nevypořádá se však se zapnutou direktivou *magic_quotes_gpc* pro vstupní data skriptu, která proto musí být pro bezproblémový běh vypnuta.

Framework disponuje vlastní knihovnou pro komunikaci s databázemi využívající rozšíření PHP, která podporuje všechny známé databázové systémy – MySQL, MSSQL, SQLite, PostgreSQL, Oracle, IBM DB2, IDS a Firebird/Interbase. Jednotné rozhraní umožňuje uživateli napsat jeden kód pro kompletní správu databáze a případnou změnu DBS je možné provést s minimálními úpravami. Aplikační rámec Zend poskytuje pro dotazy nad databází velkou míru abstrakce, u jednoduchých webů vývojář dokonce nemusí vůbec přijít do styku s SQL příkazy. Objekt vrácený po výběru z tabulky obsahuje odkazy na objekty reprezentující její řádky, tento přístup však není ideální pro informace získané z více tabulek. Poskytovaná funkcionalita je sice velmi bohatá, je možné např. definovat závislosti mezi tabulkami a vyhledávat sdružené záznamy z jiných entit, tento způsob používání však zvyšuje množství dotazů na databázi. Základní znaková sada je UTF-8, ale i data z tabulek ve starším českém kódování Windows-1250 nebo ISO-8859-2 jsou zpracována bez problémů.

Vícejazyčné webové aplikace najdou ve frameworku Zend silnou podporu. Jazyk uživatele může být určen z hlaviček HTTP požadavku a podle něj je pak nastaveno prostředí pro funkce závislé na specifikaci jazyka. Výběr jazyka může být omezen, stejně tak lze nastavit implicitní hodnotu nebo automatickou detekci vypnout. Třída pro překlad textu podporuje řadu formátů – od klasických polí, přes XML a CSV[42] po gettext. Standardní funkce obsahují data z projektu CLDR[43] jako jsou názvy v kalendáři nebo formát data[44]. Framework také umožňuje překlad částí adresy URL v závislosti na zvoleném jazyku.

Zend Framework je vhodný pro velké internetové aplikace, kterým nabízí širokou podporu nejrůznějších komponent, určených i pro velmi specifické úkoly. Robustnost aplikačního rámce může být naopak překážkou pro jednoduché projekty, jež převážnou část poskytované funkcionality nevyužijí, a musejí se občas potýkat se zbytečně velkou složitostí použití standardních komponent.

[42] comma-separated values – hodnoty oddělené čárkami

[43] Unicode Common Locale Data Repository – http://cldr.unicode.org/

[44] překlad měsíců do češtiny je přizpůsoben pro druhý/čtvrtý pád

Framework Zend byl otestován na ukázkové aplikaci blogu, jež byla představena na straně 17. Kompletní zdrojové soubory jsou k dispozici na přiloženém CD, funkční aplikace je k nalezení na internetové adrese http://bc.lipe.cz/zend/.

Ukázka kódu zobrazená níže je vyňata z metody obsluhující požadavek na přihlášení do správy blogu. Autentizační databázový adaptér, který je součástí aplikačního rámce, vyžaduje novou instanci třídy správy databáze. Po jeho inicializaci je nastavena tabulka uživatelů, sloupce s hodnotami, způsob porovnání hesla a jsou mu předány údaje odeslané z formuláře. Dalším objektem je konečně ověřena totožnost uživatele aplikace a pokud kontrola proběhla v pořádku, je volitelně nastaveno automatické přihlášení a uživatel je přesměrován do administrace.

```
$config = $this->getInvokeArg('bootstrap')
               ->getResource('db')->getConfig();
$db = new Zend_Db_Adapter_Pdo_Mysql($config);

$model = new Application_Model_Uzivatel;
$identity = $model->vraťId();

$adapter = new Zend_Auth_Adapter_DbTable($db);
$adapter->setTableName('uživatel');
$adapter->setIdentityColumn('id');
$adapter->setCredentialColumn('heslo');
$adapter->setCredentialTreatment(
                'SHA1(CONCAT(sůl, MD5(CONCAT(?, sůl))))');

$adapter->setIdentity($identity);
$adapter->setCredential($loginForm->getValue('password'));

$auth = Zend_Auth::getInstance();
$result = $auth->authenticate($adapter);

if ($result->isValid())
{
    if ($loginForm->remember->checked) {
        Zend_Session::rememberMe();
    }
    else {
        Zend_Session::forgetMe();
    }
    $this->_helper->redirector('index');
    return;
}
```

Ze samotného kódu je čtenáři patrné, že tento postup není příliš intuitivní. Dokumentace aplikačního rámce navíc podrobný návod na implementaci přihlášení uživatele nenabízí, vývojář jej musí složitě hledat v externích zdrojích zabývajících se problematikou Zend Frameworku. Objekt autentizačního databázového ovladače sice nabízí pokročilou funkcionalitu, pro potřeby implementace jednoduchého přihlášení je však jeho použití zbytečně složité.

5. Implementace vlastního frameworku

Aplikační rámec, pojmenovaný Simona, je určen pro českého uživatele, kterému se snaží přizpůsobit použitím jeho rodného jazyka ve zdrojovém kódu i dokumentaci. Přestože mají vývojáři webových aplikací k dispozici celou řadu frameworků, jen málo z nich poskytuje českou dokumentaci a zdrojový kód žádného z nich nebyl doposud napsán kompletně v češtině. Uživatel proto nemusí disponovat velkými znalostmi angličtiny a může se soustředit na psaní programu, jehož kódu bude po jazykové stránce bez problémů rozumět. Častou předností aplikačních rámců je podpora velkého množství různých funkcí a pokročilých vlastností, což však bývá vykoupeno vyšší obtížností užití a delší dobou učení. Simona se proto zaměřuje na co nejjednodušší použití a rychlou realizaci webových aplikací.

Jednou z podmínek zádání bakalářské práce byl česky psaný kód. Jazyk PHP již po několik verzí podporuje psaní zdrojového kódu s českými znaky, po úvaze ale bylo od použití diakritiky upuštěno. Adresa URL může obsahovat pouze určité znaky anglické abecedy[45], čeští vývojáři jsou proto nuceni vynechat diakritiku, nebo se smířit se zakódováním některých znaků, čímž se adresa stává pro uživatele webové aplikace špatně čitelnou[46]. Požadavek v adrese URL je v aplikaci přeložen na proměnné a z nich je určen název ovladače a akce, které jej zpracují. Kvůli zvolenému systému načítání souborů frameworkem (viz. strana 38) a zachování co nejvyšší kompatibility napříč operačními systémy, je kód aplikačního rámce psán bez diakritiky.

Zdrojové kódy jsou psány s důrazem na čitelnost, délka řádku nepřekračuje až na nutné výjimky 100 znaků. Kód určený pro zpracování skriptovacím strojem PHP je vkládán mezi dvojici značek `<?php` a `?>`, v souborech, které obsahují pouze kód PHP, však není ukončující značka povinná. Jejím vynecháním je navíc zamezeno nechtěnému vložení koncové mezery na výstup. Soubory jsou kvůli přenositelnosti mezi platformami uloženy v kódování UTF-8.

Jazyk PHP má u jednoduchých aplikací výhodu rychlého dosažení výsledku a jeho uživatelé proto často tíhnou k psaní kódu procedurálním stylem bez větší organizace. u složitějších aplikací však s rostoucí dobou vývoje nastává problém s orientací ve zdrojovém kódu a jeho úpravami. Framework se snaží udržet přehlednost kódu aplikace použitím paradigmatu objektově orientovaného programování a oddělením aplikační, datové a prezentační logiky. k tomuto účelu byl implementován návrhový vzor MVC.

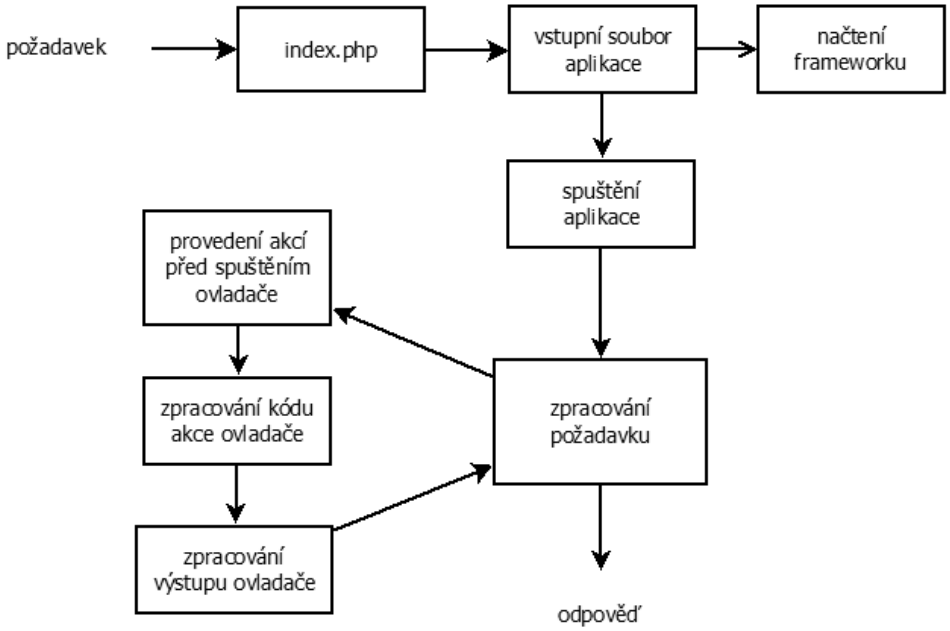
Zpracování požadavku na webovou aplikaci využívající aplikační rámec Simona začíná kontrolou verze PHP, nastavením některých direktiv PHP, kódo-

[45] podle RFC 1738 - http://www.apps.ietf.org/rfc/rfc1738.html

[46] v současnosti už většina internetových prohlížečů automaticky převádí zakódované sekvence, stále hojně používaný Internet Explorer však nikoliv

vání znaků a automatického načítání souborů a tříd. Framework využívá část funkcí jazyka PHP, které jsou dostupné až od jeho verze 5.2.0, což je minimální verze požadovaná pro provoz Simony. Kvůli práci s řetězci v kódování UTF-8 je vyžadováno také rozšíření *mbstring*.

5.1. Životní cyklus aplikace



Obrázek 4. Životní cyklus aplikace v aplikačním rámci Simona.

Cesta požadavku uživatele webové stránky využívající aplikační rámec Simona začíná ve vstupním souboru (typicky *index.php*), který nastaví konstanty cest k adresářům se zdrojovými kódy a načte vstupní soubor aplikace. Zde je aktivován framework, nastaveno běhové prostředí a spuštěno vykonávání kódu aplikace. Zpracování požadavku probíhá v cyklu, který umožňuje předávání řízení mezi ovladači bez odeslání nového požadavku na server. z každého požadavku na aplikaci je určen ovladač a jeho akce, která požadavek zpracuje a vrátí odpověď. Je-li odpovědí požadavek na předání řízení, proběhne nové zpracování požadavku, pokud je to výstup pro uživatele nebo požadavek na přesměrování, je odpověď odeslána prohlížeči a běh aplikace ukončen.

5.2. Načítání souborů a tříd

Jak bylo zmíněno v kapitole o vlastnostech PHP na straně 14, přidávání souborů do zdrojového kódu je možné pomocí příkazu `include`, případně jeho variant. u aplikací, které jsou rozděleny do velkého množství souborů by však kód obsahoval spoustu řádků s upřesněním cesty k načítaným souborům. Při požadavku o načtení souboru PHP automaticky prochází cesty určené v direktivě *include_path*, kterou lze funkcí `set_include_path` upravit. ke stávajícím cestám definovaným systémem přidá Simona cesty k adresáři aplikace, frameworku a externích knihoven.

Při inicializaci objektů tříd, které nejsou načteny, je standardně vyvolána kritická chyba. Od verze 5 PHP nabízí uživateli možnost definice funkce `__autoload`, která je volána, pokud není třída definována [5]. Aplikační rámec se v této funkci pokusí načíst soubor se třídou podle jejího názvu, který je rozdělen na části oddělenými znakem _. z nich je sestavena cesta k souboru s příponou *php* a ten, pokud existuje, je načten, jinak je oznámena chyba. Následně je provedena kontrola existence požadované třídy nebo rozhraní, pokud proběhne neúspěšně je aplikace ukončena s vytisknutím chybového hlášení. Po skončení funkce se provede opětovná inicializace objektu.

```
function __autoload($trida)
{
    $soubor =
        str_replace('_', ODDELOVAC_ADRESARU, $trida) . '.php';

    require_once $soubor;

    if (!class_exists($trida, false) &&
        !interface_exists($trida, false))
    {
        die("Chyba! Třída '$trida' nebyla nalezena " .
            "v předpokládaném souboru '$soubor'!");
    }
}
```

Třídy frameworku i aplikace na něm postavené musí být pro své úspěšné automatické načítání umístěny ve výše zmíněných adresářích uvedených v direktivě *include_path* nebo jejich podadresářích za splnění podmínky, že název adresáře je uveden jako předpona třídy a znak pro systémový oddělovač adresářů je zaměněn za znak _. Např. třída `Ovladac_Zakladni` musí být umístěna v souboru *Zakladni.php* v adresáři *Ovladac*, který je umístěn v kořenovém adresáři aplikace, frameworku nebo externích knihoven.

5.3. Prostředí a nastavení aplikace

Statická třída `Prostredi` uchovává nastavení a informace o prostředí, ve kterém je aplikace spuštěna, včetně objektu jí samotné. Prostředí se rozlišuje na *vývojové* a *produkční* a pokud není manuálně nastaveno, framework se je podle IP adresy serveru pokusí automaticky určit. Části adresy serveru jsou porovnávány se známými vzory adres lokálních sítí a pokud není nalezena shoda, je prostředí nastaveno jako produkční. ve vývojovém prostředí se zobrazují chybová hlášení, která by v produkčním prostředí mohla ohrozit bezpečnost aplikace a místo toho jsou proto uložena do souborů anebo odeslána na určený e-mail správce. `Prostredi` také nastavuje a poskytuje cesty k adresářům pro odkládací soubory, chybová hlášení a soubory s pohledy.

Uživatel může určit nastavení aplikace pomocí konfiguračního souboru ve formátu INI se standardní syntaxí, který je zpracován pomocí funkce jazyka PHP `parse_ini_file`. Hodnoty nastavení mohou být rozděleny do kategorií podle prostředí, přičemž lze použít dědění hodnot z jiného prostředí. v případě shodných konfiguračních proměnných jsou zděděné hodnoty přepsány. Načteny jsou vždy pouze hodnoty aktivního prostředí do nově vytvořené instance třídy `Nastaveni`, která provádí aktivaci nastavení a uchovává hodnoty konfiguračních proměnných. Pokud proměnná neodpovídá žádné z předepsaných hodnot, uvedených v API dokumentaci, není provedena aktivace nastavení a hodnota je pouze uložena k případnému pozdějšímu použití.

5.4. Zpracování požadavku

Nejdůležitější část aplikačního rámce tvoří zpracování požadavku uživatele webové aplikace. Nejprve se z údajů požadavku určí ovladač s odpovídající akcí, která je určena pro jeho zpracování. Následně proběhne volání metody objektu ovladače, jež provede uživatelský kód určený ke zpracování před spuštěním akce a poté už volá samotnou akci. Po ukončení akce se zpracuje její výstup a je vrácen objekt odpovědi. Jeho typ určí, zda bude uživateli odeslán výstup, oznámení o přesměrování nebo bude řízení předáno jiné části aplikace, což je dosaženo pomocí cyklu, v němž je volán uživatelský kód aplikace. Pokud by došlo k opakovanému volání stejného požadavku, je po určitém počtu průchodů cyklem vyvolána výjimka, která ukončí aplikaci.

Následující ukázka kódu zobrazuje tělo metody `spust` třídy `Aplikace`, která je volána ze vstupního souboru aplikace.

```
$smerovani = $this->vratSmerovani();
$pozadavek = $smerovani->vratPozadavek();
$opakovani = 0;

do
{
    if ($opakovani++ > self::OPAKOVANI_MAXIMUM)
    {
        throw new
            Vyjimka('Příliš mnoho opakování cyklu aplikace.');
    }

    $this->ovladac      = $pozadavek->vratOvladac();
    $odpoved_ovladace   = $this->ovladac->spust($pozadavek);

    // předání nového požadavku
    if ($odpoved_ovladace instanceof Pozadavek)
    {
        $pozadavek = $odpoved_ovladace;
    }
    else
    {
        break;
    }
}
while (TRUE);

// přesměrování
if ($odpoved_ovladace instanceof Pozadavek_Presmerovani)
{
    $odpoved_http = $this->vratOdpoved();
    $odpoved_http->presmeruj($odpoved_ovladace->vratUrl(),
                             $odpoved_ovladace->vratKod());
}
```

Uživatelský požadavek na aplikaci je analyzován ve třídě `Pozadavek_Http` postavené na návrhovém vzoru *singleton*, který zajišťuje existenci její jediné instance a byl zvolen z důvodu náročnosti analýzy položek požadavku. Data požadavku jsou tříděna, filtrována a ošetřena podle druhu a typu jejich odeslání (GET, POST, soubory), adresa URL je rozdělena na části ve třídě `Pozadavek_Url`. Hlavičky požadavku jsou vraceny na požádání.

Framework Simona umožňuje díky použití modulu serveru Apache *mod_rewrite* tvorbu odkazů s parametry požadavku oddělenými lomítky, tzv.

cestu. Vývojář může pomocí lehce upravené syntaxe regulárních výrazů určit tvar cesty a proměnné, které mají být z adresy získány. Pokud uživatel zadá URL adresu ve tvaru cesty[47], jsou formáty definovaných cest porovnávány se zadanou adresou a je-li nalezena shoda, jsou nastaveny proměnné požadavku. Pokud není ovladač určen, je ke zpracování požadavku určen základní ovladač aplikace, to samé platí u akce. Není-li nalezena třída s určeným ovladačem nebo neexistuje metoda určené akce, je řízení předáno tzv. „chybovému ovladači“.

Uživatelský kód obsluhy požadavku obstarávají tzv. ovladače[48], resp. akce v nich definované. Každý ovladač je potomkem abstraktní třídy `Ovladac_Abstraktni`, která obsahuje metodu `spust` (viz. níže), v níž začíná zpracování požadavku na uživatelské úrovni. Hlavní část probíhá v bloku `try` – `catch` zachytávajícím objekt výjimky třídy `Vyjimka_Storno`, která značí přerušení běhu ovladače, např. při přesměrování. Po skončení kódu akce ovladače se provede zpracování jejího výstupu, což je nejčastěji vytištění šablony, a vrácení odpovědi. Hlavička odpovědi, jež určuje typ výstupu a znakovou sadu UTF-8, je nastavena při inicializaci instance třídy `Aplikace`.

```
public function spust(Pozadavek $pozadavek)
{
    $this->pozadavek = $pozadavek;

    try
    {
        $this->predSpustenim();
        $akce = $this->pozadavek->vratAkci();
        $this->$akce();
        $this->zpracujVystup();
    }
    // přerušení běhu ovladače
    catch (Vyjimka_Storno $e)
    {
    }

    return $this->odpoved;
}
```

[47] tzn. ne pomocí klasického tvaru *?proměnná1=parametr2&proměnná2=parametr2*
[48] orig. controller

5.5. Výstup aplikace

Nejčastější odpovědí většiny webových aplikací je vrácení výstupu k vytištění prohlížečem. Aplikační rámec Simona díky implementaci návrhového vzoru MVC umožňuje oddělení prezentační logiky do samostatných souborů, tzv. šablon. Každá akce ovladače má typicky vlastní šablonu, která slouží k výpisu v ní vytvořených dat. Aby byl zachován jednotný vzhled aplikace, je možné nastavit tzv. *layout*, jenž na určeném místě vytiskne obsah šablony akce. Třída `Layout` je odvozená od třídy `Sablona` a poskytuje metody pro přiřazení titulku HTML stránky, vložení souborů s kaskádovými styly nebo uživatelskými skripty. Pokud šablona layoutu neobsahuje standardní strukturu HTML stránky, je automaticky vytvořena. Stejně tak jsou přidány značky pro titulek stránky a vyhrazeny bloky pro obsah stránky a vkládání souborů s kaskádovými styly a uživatelskými skripty.

Po dokončení kódu akce ovladače jsou v případě standardní odpovědi do šablony doplněny přiřazené proměnné a je provedeno její zpracování. Framework nabízí uživateli pro přehlednější zápis jednoduchý šablonovací systém s klasickou syntaxí složených závorek známou ze známějších nástrojů. Záměna značek šablonovacího systému pomocí regulárních výrazů za odpovídající PHP kód interpretovatelný překladačem zabírá podle jejich množství a rozsahu značnou část doby potřebné k vrácení odpovědi uživateli, proto Simona přeloženou šablonu uloží na disk a při požadavku na její překlad nejprve zkontroluje, zda souboru s přeloženou šablonou již neexistuje. u požadavků, které mají vždy stejný výstup, má vývojář možnost použití vestavěné *cache*. k implementaci bylo použito rozšíření *Cache_Lite* z repositáře PEAR, nad nímž aplikační rámec vytváří jednotné rozhraní pro snazší použití.

Níže uvedený kód představuje metodu pro zpracování šablony.

```
public function zpracuj()
{
    $prelozeny_soubor = $this->preloz();

    extract($this->data, EXTR_OVERWRITE);
    ob_start();
    require $prelozeny_soubor;
    $obsah = ob_get_clean();

    return $obsah;
}
```

Seznam podporovaných značek, jehož kompletní výčet je uvedený v programátorské příručce na straně 51, obsahuje kromě vypsání proměnných s ošetřenými HTML entitami, podmínek nebo zápisu cyklů pro iteraci nad polem hodnot, také

vkládání odkazů. z jednoduchého zápisu dvojice `Ovladac:Akce` a volitelně dalších parametrů je podle definovaných cest vytvořena odpovídající URL adresa. Po získání hodnot ze značky je zavolána metoda, která porovnává vývojářem definované cesty pro směrování požadavku s předanými parametry. Pokud je nalezena shoda, je vrácen odkaz ve formátu určeném cestou, jinak klasický zápis parametrů a jejich hodnot oddělených znakem `&`.

5.6. Získávání dat

Poslední částí návrhového vzoru MVC je po ovladači a šabloně model, který obsahuje logiku získání dat. Uživatel může v ovladači nastavit objekt modelu, jinak ale aplikační rámec nenabízí žádnou funkcionalitu pro práci s modely a je na vývojáři, jak získávání dat pro aplikaci naimplementuje. Simona využívá databázovou knihovnu *dibi*[49] od autora frameworku Nette, jež poskytuje jednoduché a intuitivní rozhraní pro práci s různými druhy databáze. Nad touto knihovnou je vytvořeno české aplikační rozhraní ve třídách `Databaze` a `Databaze_Vysledek`.

5.7. Formuláře

Formuláře jsou základním prvkem internetových stránek a i přes rozšíření nových technologií jsou základním prvkem interakce uživatele s webovou aplikací. Tvorba a zpracování formulářů patří mezi běžné činnosti vývojářů, kteří najdou v aplikačním rámci dobrou podporu této funkcionality. Všechny formulářové prvky jsou odvozeny od abstraktní třídy `Formular_Prvek`, jež poskytuje metody pro výpis do šablony, nastavení typu prvku, jeho atributů pro HTML značku, hodnoty a jejího ověření. Atributy a jejich hodnoty jsou ověřeny podle typu prvku, uživatel nemůže nastavit atribut HTML značky, který prvek nepodporuje, čímž je zajištěna validita výstupu. Prvku může být již při vytvoření jeho objektu nastaven štítek, jenž je s prvkem provázán a s ním zároveň předáván na výstup.

Každý prvek formuláře je možné předat na výstup díky přetížené metodě PHP `__toString`, která vytiskne značky štítku i samotného prvku včetně všech atributů, jimž jsou v případě nutnosti doplněny standardní hodnoty, pomocí funkce, která odpovídá za výstup skriptu. Vypsaný řetězec je upraven pomocí nastaveného *dekorátoru*, jenž může být i uživatelsky definovaný. Další možností je vypsání jednotlivých částí zvlášť – štítek, prvek a chybové hlášení.

Prvky jsou vkládány do objektu formuláře, který navíc umožňuje hromadné nastavení hodnot prvků, ověření odeslání formuláře a nastavení ochranného prvku proti útokům typu CSRF[50]. Ten je vkládán do HTML kódu jako skryté pole, které obsahuje zašifrované unikátní číslo, jež je uloženo také do proměnné uživatelského

[49] je zahrnuta ve frameworku v adresáři *Rozsireni/dibi*

[50] Cross-Site Request Forgery – „mezistránkové padělání požadavku“

sezení. Po odeslání formuláře je porovnána přijatá hodnoty s vygenerovaným kódem a pokud se nerovnají, je vypsána chyba, jak ukazuje kód níže.

```
$sezeni = Sezeni::vrat($jmeno);
if (isset($sezeni->kod))
{
    $kod = $sezeni->kod;
}
else
{
    $kod = $sezeni->kod =
        md5(md5($jmeno) . sha1(uniqid('', TRUE)));
}

$skryte_pole = new Formular_Skryte($jmeno, $kod);
$skryte_pole->pridejOvereni(Overeni::ROVNO, $zprava, $kod);

$this->pridejPrvek($skryte_pole);
```

Přetížená metoda `__get` umožnuje odkazování na prvky formuláře, jako by to byly veřejné proměnné objektu, což usnadňuje zápis kódu aplikace. Formulář se všemi prvky včetně seznamu chyb lze vypsat do šablony jednoduše předáním objektu formuláře na výstup. Při volání metody `over` objektu formuláře proběhne ověření všech jeho prvků. Je možné nastavit některý ze základních ověřovacích mechanismů, nebo určit vlastní funkci či statickou metodu třídy.

5.8. Autentizace a autorizace uživatele

Framework Simona nabízí vývojářům jednoduchou implementaci vlastního přihlašování uživatelů a přístupu do chráněných částí aplikace. Třída `Prihlaseni` v případě úspěšného ověření zadaných přihlašovacích údajů pomocí uživatelské funkce vytvoří uživatelské sezení, které je přístupné v celé aplikaci. Volitelně mohou být nastaveny údaje uchovávané v sezení nebo oprávnění přihlášeného uživatele. Pokud není konstruktoru objektu přihlášení předán vytvořený formulář, je automaticky vytvořen standardní přihlašovací formulář se jménem, heslem a zatrhávacím polem pro volbu automatického přihlášení. Každý z prvků lze zakázat bližším určením v parametru konstruktoru.

Zapamatování uživatele aplikace je implementováno pomocí souboru *cookie* uchovávaného na počítači uživatele. Framework do něj uloží všechny údaje sezení a hlavičky určující prohlížeč uživatele a přijímanou znakovou sadu zašifrované pomocí standardizovaného kódovacího algoritmu base64[51] a posunutím znaků

[51] podle RFC 2045 – http://www.apps.ietf.org/rfc/rfc2045.html#sec-6.8

výsledného řetězce o daný počet. Při načtení cookie s daty automatického přihlášení je provedeno rozšifrování a porovnání hodnoty prohlížeče a znakové sady s hlavičkami požadavku uživatele. Pokud hodnoty neodpovídají, je cookie smazána.

```
$jmeno      = $uzivatel->vratJmeno();
$opravneni  = serialize($uzivatel->vratOpravneni());
$udaje      = serialize($uzivatel->vratUdaje());

$prohlizec  = Pozadavek_Http::vrat()->vratHlavicku('user-agent');
$kodovani   = Pozadavek_Http::vrat()
                ->vratHlavicku('accept-charset');

$hodnota = base64_encode($jmeno . self::COOKIE_ODDELOVAC .
               $prohlizec . self::COOKIE_ODDELOVAC . $opravneni .
               self::COOKIE_ODDELOVAC . $kodovani .
               self::COOKIE_ODDELOVAC . $udaje);

return Nastroje::posunZnaky($hodnota, self::COOKIE_SIFRA_POSUN);
```

Ověření platnosti sezení je prováděno při každém požadavku obdobně, v tomto případě je však použita kombinace hashovacích funkc pro zakódování některých hlaviček požadavku s uživatelským jménem.

5.9. Ladění kódu

Ve vývojovém prostředí zobrazí Simona při výskytu chyb zpracování kódu nebo nezachycených výjimkách uživatelsky přívětivé informace o chybě či výjimce. Kromě čísla a popisu chyby je zobrazen úsek zdrojového kódu souboru, ve kterém k chybě došlo, se zvýrazněným řádkem s chybou a pokud je to možné zásobník volání. Při vyvolání výjimky je získán z metody `getTrace` objektu výjimky, který je v PHP vždy odvozena od třídy `Exception`, jinak pomocí funkce `debug_backtrace`. Většina chybových zpráv jádra PHP je přeložena, aby jim český uživatel bez problémů rozuměl. Pokud je aplikace v produkčním prostředí, nejsou kvůli bezpečnosti chybová hlášení vypisována uživateli, ale ukládána do souboru a na e-mail správce aplikace je zasláno upozornění.

Framework definuje vlastní výjimky odvozené od základní třídy `Vyjimka`, vývojář tak nemusí používat vestavěnou třídu výjimek. Jazyk PHP nepodporuje určování základních typů parametrů funkcí a metod[52], Simona proto pro zabránění nechtěných chyb testuje na prvních řádcích kódu metod datové typy předaných

[52] tzv. *type hinting*, PHP umožňuje funkci vynutit si jen třídu předaného objektu nebo požadovat pole

proměnných a v případě nesprávného typu vyvolá výjimku `Vyjimka_Argument`. Pro její vytvoření jsou potřeba parametry určující jméno chybného parametru a jeho požadovaný typ. Konstruktor (ukázka kódu je zobrazena níže), získá informace o volané metodě, pořadovém čísle parametru a typu předané proměnné pomocí funkcionality reflexe[53] jazyka PHP.

```
$volani         = self::getTrace();
$trida          = $volani[0]['class'];
$metoda         = $volani[0]['function'];
$parametr       =
      new ReflectionParameter(array($trida, $metoda), $argument);

$pozice         = $parametr->getPosition();
$typ_promenne   = gettype($volani[0]['args'][$pozice]);
$pozice++;

$this->message = "$trida:$metoda() - Chybný argument " .
                 "'$$argument' na pozici $pozice. Očekáván " .
                 "typ '$typ', předán typ '$typ_promenne'.";
```

[53] někdy označované také jako introspekce [5]

6. Programátorská příručka

6.1. Podpora vlastností

Návhový vzor MVC	✓
Databázové rozhraní	✓
Šablonovací systém	✓
Cachování	✓
Správa formulářů	✓
Autentizace uživatele	✓
Překlad textů	✗
Ladící prostředí	✓
Česká dokumentace	✓

Tabulka 3.

6.2. Konfigurace aplikace

Nastavení aplikace lze provést v INI souboru, jehož umístění je určeno parametrem statické metody `nactiNastaveni` třídy `Prostredi` volané ve vstupním souboru aplikace. Framework rozlišuje několik parametrů nastavení, která budou aktivována, zapsaných po řádcích v klasické syntaxi *parametr* = *hodnota*. Pokud není parametr rozeznán, je jeho hodnota pouze uložena. Parametry nastavení mohou být určeny jen pro některá vývojové prostředí, která mohou přebírat nastavení jiných prostředí pomocí zápisu `[prostredi < predek]`. Nastavení uvedená v „předkovi“ budou případně přepsána hodnotami v aktuálním prostředí.

Rozeznávané parametry nastavení:

- casove_pasmo
- hlaseni_chyb
- hlaseni_chyb_spusteni
- hlaseni_chyb_email
- ovladac_chyb
- ovladac_zakladni
- akce_zakladni

- sezeni_jmeno
- db_ovladac
- db_server
- db_uzivatel
- db_heslo
- db_databaze
- db_znakova_sada
- db_pripojeni_po_startu

Nastavení lze znepřístupnit pomocí zakomentování celého řádku znakem `#`, případně `;`. Následuje příklad konfiguračního souboru, který nastavuje pro prostředí produkce a vývoje odlišné hodnoty hlášení chyb. Konfigurační parametry produkčního prostředí jsou zděděny vývojovým prostředím a hodnota proměnné `hlaseni_chyb` přepsána na hodnotu `E_ALL`.

```
[produkce]
hlaseni_chyb = 0
hlaseni_chyb_spusteni = Off
hlaseni_chyb_email = "pecinkal@inf.upol.cz"
casove_pasmo = "Europe/Prague"

[vyvoj < produkce]
hlaseni_chyb = E_ALL
testovaci_hodnota = 123
#casove_pasmo = "Europe/Paris"
```

6.3. Struktura adresářů

V kořenovém adresáři domény jsou soubory a adresáře přístupné uživatelům, ať už přímo, nebo zprostředkovaně aplikací. Standardně je zde i adresář se všemi zdrojovými kódy aplikace. Pokud je doména hostována na webovém serveru Apache, je dobré do něj zakázat uživatelům přístup pomocí direktivy `deny from all` zapsané v souboru *.htaccess* umístěném v tomto adresáři. v adresáři se zdrojovými kódy aplikace musí být přítomny adresáře *Ovladac* a *Model*, volitelně pak adresář s pohledy a adresáře pro ukládání dočasných souborů a chybových hlášení.

Klasická struktura aplikace je vidět na ukázkové aplikaci (viz. strana 55). Adresářovou strukturu je však možné volit libovolně, protože framework Simona vždy využívá konstanty a proměnné definované ve vstupních souborech aplikace, případně určuje umístění souborů s třídami podle jejich názvu (viz. strana 38).

6.4. Spuštění aplikace

Při odeslání požadavku je prvním zpracovaným souborem nejčastěji *index.php*. v něm je nutné definovat konstantu `WWW_CESTA` s absolutní cestou ke kořenovému adresáři aplikace, `APLIKACE_CESTA` určující adresář se skripty aplikace, `FRAMEWORK_CESTA` s cestu k souborům frameworku a `KNIHOVNY_CESTA` určující adresář externích knihoven. Následně by měl být vložen vstupní soubor aplikace v adresáři se skripty – obvykle *vstup.php*.

Zde proběhne inicializace aplikačního rámce vložením souboru *Start.php* z adresáře frameworku. Potom už může proběhnout nastavení adresářů zmíněných v předchozí podkapitole a načtení souboru s nastavením pomocí statických funkcí třídy `Prostredi`. Níže je uveden typický příklad použití:

```
Prostredi::nastavAdresarOdkladani(
        APLIKACE_CESTA . ODDELOVAC_ADRESARU . 'docasne');
Prostredi::nastavAdresarPohledu(
        APLIKACE_CESTA . ODDELOVAC_ADRESARU . 'pohledy');
Prostredi::nastavAdresarLadeni(
        APLIKACE_CESTA . ODDELOVAC_ADRESARU . 'chyby');
Prostredi::nactiNastaveni(
        APLIKACE_CESTA . ODDELOVAC_ADRESARU . 'nastaveni.ini');
```

Následně je možné nastavit *layout* a definovat cesty pro směrování požadavků a vytváření odkazů. Uvedený příklad ukazuje definici cesty, jejíž formát odpovídá např. adrese URL *http://www.domena.cz/seznam/strana-2*. Prvním argumentem funkce `pridejCestu` je řetězec určující formát, kde dvojité hranaté závorky ohraničují nepovinnou část a zápis `<promenna vyraz>` určuje proměnnou[54], do které bude přiřazena hodnota z odpovídajícího místa adresy URL, přičemž část `vyraz` je nepovinná. Druhý parametr nastavuje proměnné, které budou přiřazeny do požadavku. Tímto způsobem je možné určit standardní hodnotu proměnné v nepovinné části adresy nebo ovladač, který má zpracovat požadavek, i když není v adrese uveden.

```
$aplikace   = Prostredi::vratAplikaci();
$smerovani  = $aplikace->vratSmerovani();

$smerovani->pridejCestu(
    '<akce [a-z]+>/[[/strana-<strana [0-9]+>]]',
    array('ovladac' => 'Prispevky',
          'akce'    => 'zakladni',
          'strana'  => 1)
);
```

[54] omezení funkcí pro práci s regulárními výrazy umožňuje použít jen alfanumerické znaky

Posledním krokem je spuštění aplikace zavoláním metody `spust` objektu aplikace, který je k dispozici ve všech částech aplikace voláním `Prostredi::vratAplikaci()`.

6.5. Ovladače a akce

Po spuštění aplikace jsou požadavky směrovány na jednotlivé ovladače, které jsou umístěny v adresáři *Ovladac*. Jméno třídy ovladače musí odpovídat názvu souboru s předponou `Ovladac_`, třída musí být odvozena od abstraktní třídy `Ovladac_Abstraktni` a obsahovat metody pojmenované podle akcí. Dobrou praktikou je odvodit ovladače od vlastní abstraktní třídy vstupního ovladače, která definuje kód společný pro všechny části aplikace, např. nastavení proměnných pro šablonu layoutu. Každý ovladač může přepisovat metody rodiče, které jsou automaticky volány před spuštěním akce, před a po vypsání výstupu. Pro zachování přednastavené funkcionality je nutné v přepsané metodě vždy volat metodu předka.

```
abstract class Ovladac_Vstup extends Ovladac_Abstraktni
{
    protected function predSpustenim()
    {
        parent::predSpustenim();
        Databaze::pripoj();
    }
    protected function predVystupem()
    {
        parent::predVystupem();
        $this->layout->nastavTitulek('Moje aplikace');
    }
}
class Ovladac_Prispevky extends Ovladac_Vstup
{
    protected function predSpustenim()
    {
        parent::predSpustenim();
        if ($this->uzivatel === NULL ||
            !$this->uzivatel->maOpravneni('sprava prispevku'))
        {
            $this->presmeruj('Admin:prihlaseni');
        }
        $this->nastavModel(new Model_SpravcePrispevku);
    }
}
```

Při psaní kódu ovladače může vývojář využít metody třídy `Ovladac_Abstraktni`. Šablona akce není standardně aktivní, před přiřazováním proměnných je nutné nejdříve zavolat metodu `nastavPohled` s volitelným parametrem určujícím soubor se šablonou. Pokud není parametr zadán, je soubor určen podle ovladače a akce. Poté je možné přiřazovat proměnné k vytištění v šabloně pomocí konstruktu `$this->pohled`.

```
$this->nastavPohled();
$this->pohled->prispevky = $prispevky;
$this->pohled->strankovac = $strankovac;
```

K dispozici jsou i metody pro přesměrování či předání požadavku, kde je možné použít krátký zápis `Ovladac:Akce`, ve volitelném druhém parametru předat další hodnoty a v dalším nepovinném argumentu určit HTTP kód přesměrování.

```
$this->presmeruj(':zobrazit', array('id' => $id));
```

Aplikační rámec také umožňuje předávání zpráv přes požadavky pomocí jejich uložení v uživatelských sezení. Volitelný druhý parametr metod `pridejZpravu` umožnuje určit, po kolik požadavků bude text uchován. Zprávy jsou zobrazeny automaticky v šabloně layoutu, pokud je pro ně vyhrazen blok se jménem `zpravy_aktualni`.

6.6. Šablonovací systém

Zápis prezentační logiky je usnadněn použitím jednoduchého šablonovacího systému, který umožňuje náhradu základních příkazů a tím poskytuje přehlednější kód. Níže následují všechny podporované zkratky, které jsou při zpracování šablony převedny na interpretovatelný PHP kód. Detailní popis jednotlivých zástupců je k dispozici v API dokumentaci.

- `{$promenna[|funkce]}`
- `{!$promenna[|funkce]}`
- `{*komentar*}`
- `{=vyraz}`
- `{!=vyraz}`
- `{?vyraz}`
- `{foreach $pole, [$klíč, ]$prvek} ... {/foreach}`

- `{if podmínka} ... {elseif podmínka} ... {else} ... {/if}`
- `{ifset $proměnná} ... {elseifset $proměnná} ... {/if}`
- `{test $proměnná}`
- `{soubor cesta[, promenna => hodnota,...]}`
- `{odkaz formát[, promenna => hodnota,...]}`
- `{blok jméno} ... {/blok jméno}`
- `{#jméno_bloku}`

Před vytištění proměnné mohou být použity dodatečné funkce. Je možné určit již existující funkci nebo statickou metodu uživatelské třídy, která je přidána ke zpracování pomocí metody `pridejFunkci` objektu šablony. První parametr určuje jméno funkce, jež je možné použít v šabloně, druhý už samotná funkce jako řetězec nebo pole, např. `array('Prispevky', 'zkratNadpis')`. Parametry předávané vkládanému souboru nebo k vytvoření odkazu jsou oddělené čárkami a název proměnné nemusí být uzavřen v uvozovkách.

6.7. Formuláře

Vytváření a zpracování formulářů je v aplikačním rámci Simona jednoduché a intuitivní. Do vytvořeného objektu formuláře je možné vložit libovolný prvek, jejichž kompletní seznam včetně parametrů konstruktoru je k dispozici v API dokumentaci. Většina prvků vyžaduje jako první parametr své jméno, pomocí nějž je na prvek odkazováno, následuje text štítku a pole s určením atributu a dalších vlastností. Při požadavku o vypsání prvku je vykreslen s použitím základního dekorátoru, pokud mu není nastaven jiný metodou `nastavDekorator` s parametrem názvu třídy dekorátoru, která musí implementovat rozhraní `Formular_Dekorator`.

Každému prvku je možné přidat ověřovací funkce, které jsou volány při kontrole odeslaných hodnot. Metodě třídy prvků `pridejOvereni` jejako první parametr nutné předat ověřovací funkci, následovanou chybovou zprávou při neplatné hodnotě a libovolným počtem parametrů předaných funkci. Kromě základních ověřovacích metod určených konstantami třídy `Overeni` jako je `NEPRAZDNE`, `EMAIL`, `REGULARNI_VYRAZ` nebo `DELKA_MIN` lze určit uživatelskou funkci, stejně jak bylo zmíněno v předchozí podkapitole.

```
$formular = new Formular();
$formular->nastavCssTriduChyb('error');

$formular->pridejText('jmeno', 'Jméno:',
                array('maxlength' => 45, 'size' => 45))
         ->pridejOvereni(Overeni::NEPRAZDNE,
                         'Zadejte své jméno.');

$formular->pridejOdeslat('ulozit', 'Uložit')
         ->nastavDekorator('Rozsireni_Dekorator_Tlacitko');

$formular->pridejOchranu(NULL, 'ochrana');
```

Metoda formuláře `jeOdeslan` testuje, zda byl formulář odeslán, při zadání volitelného parametru se kontroluje, zda byl formulář odeslán stisknutím určeného tlačítka. Pro ověření odeslaných hodnot je k dispozici metoda `over`, která u každého prvku formuláře volá jeho ověřovací funkce. k načtení odeslaných hodnot bez ověření slouží metoda `nactiHodnoty`.

```
if ($formular->jeOdeslan('ulozit') &&
    $formular->over())
{
   ...
}
```

6.8. Autentizace a autorizace

Simona nabízí velmi jednoduchou implementaci přihlašování uživatele. Konstruktoru objektu třídy `Prihlaseni` je předána ověřovací funkce a volitelně nastaveno chybové hlášení a podoba přihlašovacího formuláře, případně předán již vytvořený formulář, který musí obsahovat buď textové pole pro zadání uživatelského jména nebo pole pro vyplnění hesla. Poslední parametr určuje dobu v milisekundách, po kterou má být zachována cookie s daty pro automatické přihlášení.

```
$prihlaseni = new Prihlaseni(array('Ovladac_Admin', 'overeni'),
                        NULL, array('uzivatel' => FALSE), 3600);
if ($prihlaseni->zpracuj())
{
    $this->uzivatel->nastavJmeno('Administrator');
    $this->pridejZpravu('Byl jste úspěšně přihlášen.');
    $this->presmeruj(':zakladni');
}
```

Ověřovací funkce přihlášení, které je předána hodnota uživatelského jména a hesla, může kromě jednoduché hodnoty typu *boolean* vracet i asociativní pole hodnot, které budou přiřazeny do uživatelského sezení. Pod klíčem `opravneni` je možné předat pole řetězců určující zdroje, na které má uživatel oprávnění – zda je uživatel autorizován je možné ověřit pomocí metody `maOpravneni` jeho objektu. Klíč `udaje` určuje data, jež budou uložena do uživatelského sezení.

```
public static function overeni($uzivatel, $heslo)
{
    if ($heslo !== 'admin123')
    {
        return FALSE;
    }
    if ($uzivatel == 'Karel')
    {
        return array('opravneni' => array('mazat prispevky'));
    }
    else
    {
        return TRUE;
    }
}
```

6.9. API dokumentace

Kompletní seznam všech tříd a metod frameworku včetně jejich parametrů a popisu nalezne čtenář v API dokumentaci, která byla vytvořena s pomocí nástroje phpDocumentor[55]. Protože je standardně určen pro tvorbu anglických dokumentací, šablony a některé zdrojové kódy musely být upraveny, aby výsledná dokumentace k aplikačnímu rámci byla z co největší části v českém jazyce. Některé značky u tříd či metod však z technických důvodů musely zůstat v originále. PhpDocumentor automaticky zpracuje pro něj určené komentáře v předaných zdrojových kódech a vytvoří dokumentaci ve zvoleném formátu. Pro snazší navigaci uživatele, prohlížení na internetu, možnosti úpravy podoby výstupu a podporu českého kódování byl zvolen formát HTML.

Aplikace je v dokumentaci rozdělena na tzv. balíky, které obsahují jednotlivé soubory a třídy. Po výběru balíku z nabídky v levé části obrazovky je možné procházet jeho rozhraní a třídy, nebo si z nabídky vpravo nahoře nechat zobrazit strom tříd a seznam všech prvků balíku. Poslední možností je abecedně roztříděný kompletní výčet souborů, tříd, rozhraní, metod, proměnných a konstant s jejich stručným popisem.

[55] http://www.phpdoc.org/

Specifikace třídy obsahuje její stručný popis, umístění v hiearchii, pokud je odvozena od jiné třídy, a přehled proměnných a metod, u nichž je odkaz na řádek v souboru se zdrojovým kódem, kde byl člen třídy definován. Definice metody obsahuje její návratovou hodnotu, datové typy předávaných parametrů a určení, zda jsou volitelné. Popis metody stručně shrnuje její účel a je uveden i stupeň přístupnosti. Pokud je metoda přepsána v potomkovi třídy, do dokumentace je přidán křížový odkaz v obou třídách.

6.10. Ukázková aplikace

V aplikačním rámci Simona byla implementována ukázková aplikace blogu, analyzovaná na straně 17. Na přiloženém CD jsou k dispozici zdrojové soubory a na internetové adrese `http://bc.lipe.cz/simona/` je k nahlédnutí běžící aplikace.

Závěr

Výsledkem práce je srovnání dvou frameworků pro skriptovací jazyk PHP na jednoduché aplikaci webového blogu, které proběhlo v několika kategoriích s popisem vlastností a obtížnosti použití aplikačních rámců. Na základě srovnání byl vytvořen framework s zdrojovým kódem psaným v češtině určený pro české vývojáře, který přebírá a kombinuje některé vlastnosti z aplikačních rámců pro PHP, dostupných veřejnosti již několik let, a uživateli tak poskytuje plnohodnotný nástroj pro vývoj webové aplikace.

Reference

[1] *Wikipedia – The Free Encyclopedia* [online]. [cit. 2010-06-24] URL: <http://en.wikipedia.org/>.

[2] Lerdorf, Rasmus; Tatroe, Kevin. *Programming PHP* ISBN 1-56592-610-2, First edition, O'Reilly Media, March 2002.

[3] *History of PHP* [online]. [cit. 2010-06-24] URL: <http://www.php.net/manual/en/history.php.php>.

[4] *php.internals: PHP 6* [online] [cit. 2010-07-15] URL: <http://news.php.net/php.internals/47120>

[5] Gutmans, Andi; Bakken, Stig Saether; Rethans, Derick. *Mistrovství v PHP 5* ISBN 80-251-0799-X, 1. vydání, CP Books, a.s., 2005.

[6] Reenskaug, Trygve. *THING-MODEL-VIEW-EDITOR* [online]. [cit. 2010-06-24] URL: <http://heim.ifi.uio.no/~trygver/1979/mvc-1/1979-05-mvc.pdf>.

[7] Ráček, Jaroslav. *Strukturovaná analýza systémů.* ISBN 80-210-4190-0, 1. vydání, Masarykova univerzita, 2006.

[8] *Velký test PHP frameworků* [online]. [cit. 2010-06-30] URL: <http://www.root.cz/clanky/velky-test-php-frameworku-2008/>.

[9] *Zend Framework by the Numbers* [online]. [cit. 2010-07-12] URL: <http://www.zendframework.com/about/numbers>.

Rejstřík