VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

# KOMPONENTY WEBOVÝCH APLIKACÍ V PHP S TRANSPARENTNÍ VAZBOU NA DATA

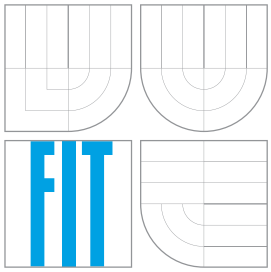
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. PETR LOUKOTA

BRNO 2014

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

# KOMPONENTY WEBOVÝCH APLIKACÍ V PHP S TRANSPARENTNÍ VAZBOU NA DATA

PHP WEB APPLICATION COMPONENTS WITH A TRANSPARENT DATA BINDING

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. PETR LOUKOTA
AUTHOR

VEDOUCÍ PRÁCE — Ing. RADEK BURGET, Ph.D.
SUPERVISOR

BRNO 2014

# Abstrakt

Tato diplomová práce se zabývá návrhem a implementací rozšíření zvoleného rámce pro tvorbu webových aplikací v jazyce PHP o definici šablon stránek v XML s možností implementace vlastních rozšiřujících značek v PHP. Rámce jsou nástroje sloužící pro snadnou tvorbu webových aplikací a tato práce popisuje také systémy definic a využití šablon v existujících rámcích. Součástí návrhu je rovněž možnost definice vazby mezi stavem prvku uživatelského rozhraní a daty webové aplikace. Dále je popsána tvorba ukázkových knihoven komponent demonstrujících funkčnost řešení. Ve svém závěru pak práce shrnuje dosažené výsledky a věnuje se možnostem dalšího vývoje.



# Abstract

This Master's thesis deals with the design of implementation of the extension of given framework for making web applications in PHP language by a definition of website templates in XML with the possibility of implementing own additional tags in PHP. Frameworks are tools used for simple creation of web applications, and this thesis also describes systems of definitions and the utilization of templates in existing frameworks. The part of the design is also a possibility of definition of relation between state of the element of user interface and web application data. Further, the creation of sample libraries of components demonstrating functionality of the solution is described. In the conclusion, the thesis summarizes achieved results and deals with possibilities of further development.



# Klíčová slova

framework, Nette, PHP, XML, šablona, rozšíření, web

# Keywords

framework, Nette, PHP, XML, template, extension, web


# Citace

Petr Loukota: Komponenty webových aplikací v PHP s transparentní vazbou na data, diplomová práce, Brno, FIT VUT v Brně, 2014

# Komponenty webových aplikací v PHP s transparentní vazbou na data

## Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Radka Burgeta Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Petr Loukota
20. května 2014

## Poděkování

Rád bych poděkoval vedoucímu mé diplomové práce Ing. Radkovi Burgetovi Ph.D. za věcné připomínky k návrhu rozšíření a rychlou odbornou pomoc při řešení problémů.

# Obsah

# Kapitola 1

# Úvod

Cílem této diplomové práce je navrhnout rozšíření zvoleného rámce pro tvorbu webových aplikací o definici šablon stránek v XML s možností implementace vlastních rozšiřujících značek v PHP. V návrhu je dále uvažována možnost definice vazby mezi stavem prvku uživatelského rozhraní a daty webové aplikace. Tato práce popisuje nejen systém definice a využití šablon stránek v existujících rámcích a samotný návrh rozšíření, ale také implementaci a testování. Dále se rovněž zabývá tvorbou ukázkových knihoven komponent demonstrujících funkčnost řešení a tvorbou ukázkové webové aplikace. Závěrem jsou zmíněny také možnosti dalšího vývoje.

Kapitola 2 se zabývá analýzou existujících rámců pro tvorbu webových aplikací s důrazem na systém definice a využití šablon stránek v těchto rámcích.

Popis technologií, které využívá zvolený rámec pro tvorbu webových aplikací, a tedy jsou využity i pro implementaci navrženého rozšíření, je k dispozici v kapitole 3. Tato kapitola se věnuje technologiím HTTP, HTML a XHTML, XML včetně používání jmenných prostorů, DOM, PHP včetně možností zpracování XML dokumentů a AJAX.

V kapitole 4 je popsán návrh rozšíření zvoleného rámce. První část kapitoly se zabývá výběrem vhodného rámce a způsobem zpracování XML v jazyce PHP. Kapitola je dále rozdělena na uživatelskou část, ve které je možné nalézt, jakým způsobem uživatel bude uvádět definice XML elementů, a část zpracování těchto definic na straně rozšířeného frameworku. Součástí této kapitoly je také návrh vazby mezi stavem prvku uživatelského rozhraní a daty aplikace a rovněž návrh ukázkových knihoven komponent a webové aplikace.

Kapitola 5 se zabývá postupem implementace a implementačními problémy, které bylo třeba řešit. Je zde uvedeno, jakým způsobem jsou XML šablony zpracovávány, jak je implementována vazba prvků uživatelského rozhraní na data aplikace, jak jsou reprezentovány jmenné prostory, jaké výjimky navržené rozšíření vyhazuje a další.

Kapitola 6 je věnována testování. Popisuje, v jakém prostředí byl návrh rozšíření zvoleného rámce vyvíjen a testován, dále se zabývá způsobem testování v daném rámci, debugováním PHP kódu a odhalenými nedostatky.

V závěru práce se věnuji souhrnu dosažených výsledků a zabývám se také možnostmi dalšího vývoje navrženého rozšíření.

# Kapitola 2

# Analýza existujících rámců pro tvorbu webových aplikací

Tato kapitola se zabývá analýzou existujících rámců pro tvorbu webových aplikací. U každého rámce je zmíněn jeho stručný popis včetně historie vývoje, dále se věnuji postupu instalace každého rámce a důraz je pak kladen především na systém definice a využití šablon stránek v každém rámci. Kapitola se věnuje čtyřem populárním a v dnešní době často využívaným rámcům, a to rámcům CakePHP, Nette Framework, Zend Framework a Symfony. V jejím úvodu jsou také vysvětleny pojmy Model-View-Controller a Model-View-Presenter.

Díky analýze existujících rámců bude možné zvolit vhodný rámec pro návrh a implementaci rozšíření o definici šablon stránek v XML. Uživatelé využívající zvolený rámec budou mít tedy možnost definovat vlastní rozšiřující značky v PHP a též jim bude umožněno využívat definice vazby mezi stavem prvku uživatelského rozhraní a daty webové aplikace.

## 2.1 Architektury MVC a MVP

MVC (Model-View-Controller) je softwarová architektura, která slouží pro rozdělení řídicí logiky aplikace, jejích dat a uživatelského rozhraní do tří nezávislých komponent. Ty je pak možné jednotlivě modifikovat s minimální nutností úprav ostatních. MVC je dnes často chápána také jako návrhový vzor. Poprvé tuto architekturu popsal Trygve Reenskaug, a to v roce 1979 [6].

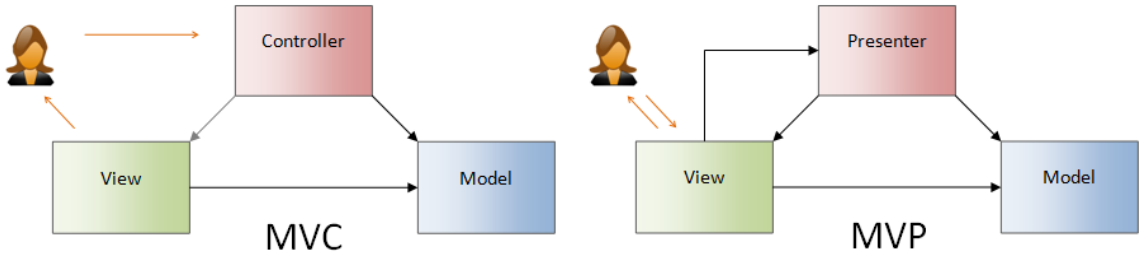
Komponentami této architektury jsou model, pohled a řadič. Model reprezentuje data, se kterými aplikace pracuje, pohled je pak převádí do podoby, ve které jsou prezentována uživatelům. Mezi těmito komponentami se nachází řadič, který reaguje na události od uživatele a zajišťuje změny v modelu, rovněž pak žádá pohled o jeho vykreslení [30]. Uživatel tedy požádá o vykonání nějaké akce, která je zachycena řadičem. Ten na akci zareaguje změnou modelu nebo přímým ovlivněním pohledu a pohled pak změny zobrazí uživateli. Celý proces se neustále opakuje [1].

Architektura Model-View-Controller byla poprvé použita v jazyce Smalltalk, dnes je spojena především s tvorbou webových aplikací. Využívají ji populární rámce jako JavaServer Faces, CakePHP, Zend Framework a Symfony [20].

MVP (Model-View-Presenter) je softwarová architektura, která se vyvinula z architektury MVC. Komponenty jsou opět tři, a to model, pohled a presenter. Model vykonává totéž co u předchozí architektury, pohled pak navíc zpracovává akce uživatele, například po stisknutí myši volá danou metodu presenteru. Presenter manipuluje s modelem a za-

jišťuje aktualizaci pohledu. U architektury MVP jsou uživatelské vstupy a výstupy plně kontrolovány pohledy, pohled má navíc přímou vazbu na presenter [1].

Hlavní rozdíly mezi architekturami MVC a MVP jsou zobrazeny na obrázku 2.1.




*Obrázek 2.1: Schéma architektury MVC a MVP [1]*

## 2.2 CakePHP

CakePHP[1] je rámec pro tvorbu webových aplikací napsaný v jazyce PHP. Tento framework je založen na softwarové architektuře MVC a vznikl v roce 2005. Inspirací se pro CakePHP stal framework Ruby on Rails[2], využívá tedy mnoho z jeho myšlenek a mnoho z jeho užitečných vlastností také implementuje [24]. CakePHP verze 1.0 byl vydán v květnu roku 2006 a v současné době je k dispozici verze 2.4.6.

CakePHP má srozumitelnou strukturu, díky čemuž je orientace v kódu snadná a vývoj aplikací rychlý, což přispělo k jeho popularitě. Framework lze propojit s nejpoužívanějšími databázemi (MySQL, PostgreSQL, Microsoft SQL Server, SQLite), k dispozici je velké množství nástrojů použitelných například pro validaci vstupních dat nebo ověřování uživatelů. CakePHP zajišťuje zabezpečení aplikace, lze s jeho pomocí snadno pracovat s cookies, sezeními a dalšími. Na domovských webových stránkách tohoto rámce je k dispozici také rozsáhlá anglická dokumentace [2] a je možné nalézt i hotové ukázkové aplikace.

### 2.2.1 Instalace

CakePHP vyžaduje PHP verze 5.2.8 a vyšší a postup instalace je popsán v jeho dokumentaci. Adresářová struktura tohoto frameworku je zachycena v kódu 2.1. Framework je umístěn v adresáři *lib/Cake*. Po příchodu na URL adresu instalace tohoto rámce se zobrazí úvodní stránka s informacemi, která nastavení nejsou v pořádku. Připojení k databázi je možné nastavit v souboru *app/Config/database.php.default*, který je pak nutné přejmenovat na *app/Config/database.php*. V souboru *app/Config/core.php* je také nutné změnit hodnoty *Security.salt* a *Security.cipherSeed*.

[1] http://cakephp.org/
[2] http://rubyonrails.org/

```
cakephp/          // rozbalený archív
   app/           // webová aplikace
      Config/     // konfigurační soubory
      Console/    // konzolové příkazy a úkoly
      Controller  // kontrolery a jejich komponenty
      Lib         // knihovny, které nepocházejí z třetích stran
      Locale      // lokalizace
      Model       // modely, chování, zdroje dat
      Plugin      // doplňky, rozšíření
      Test        // testy
      tmp         // dočasné soubory
      Vendor      // knihovny a třídy třetích stran
      View        // pohledy, layouty, helpery, chybové stránky
      webroot     // kořenový adresář webové aplikace
      .htaccess
      index.php
   lib/
      Cake        // samotný framework
   plugins/       // doplňky a rozšíření
   vendors/       // knihovny třetích stran
```

*Kód 2.1: Adresářová struktura CakePHP [3]*

### 2.2.2 Definice šablon stránek v HTML a PHP

Definici šablon stránek lze v CakePHP provádět v adresáři *app/View*. Zde se nachází několik dalších adresářů. V *Layouts* jsou umístěny definice stránek obalujících určitý pohled. Pohledy se nacházejí v adresářích s názvy dle názvů řadičů, zobrazují se na základě určitých akcí a podle těchto akcí jsou také pojmenovány. V adresáři *Elements* lze dále definovat menší, znovupoužitelné části kódu, které se obvykle zobrazují v pohledech. Chybové stránky se umísťují do adresáře *Errors* a v adresáři *Helpers* lze nalézt PHP soubory definující třídy s logikou, jež je v pohledové části často využívána. Šablony v CakePHP mají příponu *.ctp* [5].

Šablony se v tomto rámci píší v jazyce HTML, případně XHTML a jsou doplněny PHP kódem. Tvorbu webových aplikací usnadňují helpery. CakePHP obsahuje několik výchozích helperů, například pro snadnou práci s formuláři, s JavaScriptem, se sezeními, s textem a dalším. Je také možné definovat helpery vlastní. Každý z nich je pak v pohledu dostupný jako veřejná vlastnost daného pohledu [4].

Kód 2.2 ukazuje definici šablony stránky v tomto frameworku.

```
<table>
    <tr>
        <th>Id</th>
        <th>Titulek</th>
        <th>Autor</th>
        <th>Obsah</th>
    </tr>
    <!-- HTML kód doplněný PHP kódem -->
    <?php foreach ($pages as $page) { ?>
    <tr>
        <td><?php echo $page['Page']['id'] ?></td>
        <td><?php echo $page['Page']['title'] ?></td>
        <td><?php echo $page['Page']['author'] ?></td>
        <td><?php echo $page['Page']['content'] ?></td>
    </tr>
    <?php } ?>
</table>
```

*Kód 2.2: Ukázka definice pohledů v CakePHP*

## 2.3 Nette Framework

Nette Framework[3] je nástroj pro tvorbu webových aplikací v jazyce PHP 5. Jeho původním autorem je David Grudl[4], o jeho vývoj se nyní stará organizace Nette Foundation[5], která vznikla v roce 2009. Nette je založen na softwarové architektuře MVP [31]. Nette Framework je svobodný software a každý se může zapojit do jeho vývoje, v současné době má v České republice nejaktivnější komunitu.

Vývoj Nette Frameworku byl zahájen v roce 2004, tehdy v PHP verze 4. Ačkoliv ještě neměl název Nette, disponoval už tenkrát vlastní databázovou vrstvou a komponentovým modelem velmi podobným tomu současnému. Několik částí tohoto rámce pro tvorbu webových aplikací bylo zveřejněno v roce 2006 a celý framework byl představen na konci roku 2007, a to na Konferenci PHP frameworky podzim 2007. Poté byl Nette zveřejněn. Oficiální vydání se pak uskutečnilo na konci ledna 2008, a to při příležitosti Konference zima 2008. Tato verze nesla označení 0.7. Během dalšího vývoje pak bylo vydáno také PHP 5.3.0 s podporou jmenných prostorů, na které byl Nette Framework připraven. Nette 2.0 pak byl vydán čtyři roky po vydání první verze. Současnou verzí tohoto rámce je Nette 2.1.2 [19].

V České republice Nette Framework využívá řada významných webů, například web bývalého prezidenta České republiky Václava Klause[6] a Česko-Slovenské filmové databáze[7]. K tomuto rámci je také k dispozici rozsáhlá česká dokumentace a řada doplňků a komponent.

[3] http://nette.org/cs/
[4] http://davidgrudl.com/
[5] http://nettefoundation.com/
[6] http://www.klaus.cz/
[7] http://www.csfd.cz/

### 2.3.1 Instalace

Instalace Nette Frameworku je popsána v dokumentaci. V současné době je k dispozici verze pro PHP 5.3 a vyšší, ale také verze pro PHP 5.2. Adresářová struktura staženého rámce je zachycena v kódu 2.3. Celý rámec se nachází v adresáři *Nette*, jeho minimalizovaná verze pak v *Nette-minified*.

Tento rámec obsahuje několik nástrojů, které jsou umístěny v adresáři *tools*. Rovněž je připraveno několik ukázkových aplikací. Před jejich vyzkoušením je možné ověřit, zda server, na kterém je Nette nahrát, splňuje minimální požadavky, a to pomocí nástroje Requirements-Checker. Je-li vše v pořádku, je možné se ihned pustit do vytváření vlastní webové aplikace. Nastavení připojení k databázi se provádí v konfiguračním souboru *app/config/config.local.neon*.

```
nette/                // rozbalený archív
   API-reference/     // offline verze API dokumentace
   client-side/       // skript pro validaci formulářů
   examples/          // příklady, rovnou spustitelné
   Nette/             // samotný framework
      Application/
      Caching/
      ...
      loader.php
   Nette-minified/    // minimalizovaná verze frameworku
      nette.phar
   sandbox/           // předpřipravený projekt
      app/
      log/
      temp/
      tests/
      vendor/
      www/
   tools/             // pár užitečných nástrojů
      Code-Checker/
      Code-Migration/ // (jen v distribuci pro PHP 5.3)
      Debugger-Editor/
      Requirements-Checker/
```

*Kód 2.3: Adresářová struktura Nette Frameworku [32]*

### 2.3.2 Šablonovací systém Latte

Šablonovací systém, který je obsažen v Nette Frameworku, nese název Latte. Šablony stránek jsou umístěny v adresáři *app/templates* a jeho podadresářích a mají příponu *.latte*. Nette je založen na MVP architektuře a část View představují právě šablony. Presenter nejprve na základě požadavku uživatele vyvolá příslušnou aplikační logiku a poté požádá šablonu o vykreslení výsledku.

Šablony se v tomto rámci píší v jazyce HTML, případně XHTML, což je možné nastavit v konfiguraci. Navíc však lze využít makra a helpery, které Nette poskytuje a které psaní šablon usnadňují. Makra jsou speciální značky zapisované buď ve složených závorkách,

nebo jako n:makra, například *n:if="..."*, přičemž platí, že každé párové makro operující nad jedním elementem lze zapsat jako n:makro. Pomocí maker je možné provádět výpis proměnných a výrazů, ladění, vytvářet podmínky a cykly, využívat AJAX a další. V dokumentaci je dostupný seznam výchozích Latte maker [34], přičemž je možné také vytvářet makra vlastní.

Helpery jsou funkce, které je možné používat přímo v šablonách. Tyto funkce slouží k úpravě nebo formátování dat do výsledné podoby. V dokumentaci je rovněž k dispozici kompletní soupis helperů, které lze v Nette Frameworku využít [33]. Pomocí většiny z nich je možné provádět modifikace řetězců, například jejich oříznutí nebo odstranění bílých znaků. Výchozí helpery v tomto rámci však zvládnou i změnit velikost písma nebo formátovat datum do požadovaného tvaru. Použití maker a helperů je ukázáno v kódu 2.4.

```
<table>
    <thead>
        <tr>
            <th>Čas vytvoření</th>
            <th>Úkol</th>
            <th>Přiřazeno</th>
        </tr>
    </thead>
    <tbody>
        <!-- použití makra foreach -->
        {foreach $tasks as $task}
        <tr>
            <!-- použití helperu pro formátování data -->
            <td>{$task->created|date:'j. n. Y'}</td>
            <td>{$task->text}</td>
            <td>{$task->user->name}</td>
        </tr>
        {/foreach}
    </tbody>
</table>
```

*Kód 2.4: Použití maker a helperů v definici šablony stránky [31]*

## 2.4 Symfony

Symfony![8] je framework pro tvorbu webových aplikací v jazyce PHP 5 postavený na softwarové architektuře MVC. Byl inspirován dříve existujícími rámci, například Ruby on Rails. Původně byl představen agenturou Sensio Labs, která jej využívala pro tvorbu webových stránek pro jejich klienty. Rámec byl pak publikován v roce 2005 a dnes patří mezi frameworky, které si získaly popularitu po celém světě. Používá ho například internetový

[8] http://symfony.com/

portál Yahoo![9] nebo systémy Drupal[10] a phpBB[11]. Aktuální verze nese označení 2.4.2.

Tento rámec je rozdělen na jednotlivé komponenty, které lze využívat samostatně. Díky komponentě *Console* je dokonce možné ovládat aplikaci frameworku Symfony z příkazového řádku. Komunita je tvořena tisíci vývojáři a kromě výchozí funkcionality je možné získat řadu rozšíření nazývaných bundles. Díky nim je možné například zrychlit načítání obrázků nebo umožnit přihlašování pomocí sociální sítě Facebook. Na oficiálních webových stránkách je dále k dispozici rozsáhlá dokumentace [9], každá nová funkce je dokumentována také na blogu[12] tohoto rámce.

### 2.4.1 Instalace

Adresářová struktura Symfony po rozbalení do kořenového adresáře webového serveru je zachycena v kódu 2.5. Framework je možné stáhnout z oficiálních webových stránek a v dokumentaci je popsán také způsob instalace [8]. Pomocí *web/config.php* je možné ověřit konfiguraci serveru, uvítací stránka je pak připravena na adrese *web/app_dev.php*.

```
Symfony/              // rozbalený archív
   app/               // konfigurace aplikace
      cache/
      config/
      logs/
      Resources/
   bin/
   src/               // zdrojové soubory aplikace
      Acme/
         DemoBundle/
            Controller/
            Resources/
            ...
   vendor/            // knihovny třetích stran
      doctrine/
      symfony/
      ...
   web/               // kořenový adresář webové aplikace
      app.php
      ...
```

*Kód 2.5: Adresářová struktura frameworku Symfony [8]*

[9] http://www.yahoo.com/
[10] https://drupal.org/
[11] https://www.phpbb.com/
[12] http://symfony.com/blog/

### 2.4.2 Šablonovací systém Twig

Šablona stránky je vykreslena na základě požadavku řadiče a Symfony verze 2 nabízí dvě možnosti zápisu šablon. Může jít o jazyk HTML v kombinaci s bloky PHP kódu, rovněž lze však využít šablonovací systém Twig[13]. V takovém případě je HTML doplněno speciálními značkami a kód šablony je kratší a přehlednější. Ukázka zápisu je zachycena v kódu 2.6.

Šablonovací systém Twig rozlišuje dva druhy značek. Uvnitř konstrukce {{ ... }} je možné provést výpis proměnné nebo výrazu, konstrukce {% ... %} zajišťuje možnost provádění nějaké operace. Blokové komentáře se uvádějí mezi značky {# ... #}. Twig dále disponuje filtry, kterými lze modifikovat text ještě před tím, než je zobrazen. Příkladem může být blok {{ *title*|*upper* }}. Na webových stránkách tohoto šablonovacího systému jsou vypsány všechny značky a výchozí filtry, které je možné použít. Mezi další vlastnosti patří například dědění šablon nebo automatické escapování [7] [18].

Definice šablon stránek se ve frameworku Symfony obvykle nacházejí v adresáři *app/Resources/views/* a soubory mohou mít tři různé přípony – *index.html.twig* značí HTML formát s využitím šablonovacího systému Twig, v *index.html.php* jsou použity konstrukce jazyka PHP a poslední možností je použití CSS (Cascading Style Sheets), tedy *index.css.twig*. Další možnosti definice šablon stránek v tomto rámci jsou důkladně popsány v dokumentaci [7].

```
<h1>{{ page_title }}</h1>
<ul id='navigation'>
    {% for item in navigation %}
        <li><a href='{{ item.href }}'>{{ item.caption }}</a></li>
    {% endfor %}
</ul>
```

*Kód 2.6: Ukázka definice šablon stránek ve frameworku Symfony [7]*

## 2.5 Zend Framework

Zend Framework[14] je rámec pro tvorbu webových aplikací implementovaný v jazyce PHP 5. Jde o zcela objektově orientovaný framework využívající softwarovou architekturu MVC. Jeho jádro se skládá z jednotlivých modulů, které lze využívat samostatně. Vytvořením nebo použitím dalších, již existujících modulů lze pak doplnit novou funkcionalitu, například podporu dalších typů databází [17]. Vývoj Zend Frameworku byl zahájen v roce 2005, současná verze nese označení 2.3.0 a pro svou funkčnost vyžaduje PHP verze 5.3 a vyšší.

Zend Framework 2 se vyvinul z verze 1 a využívá většiny možností, které PHP 5.3 nabízí. Například na rozdíl od původní verze plně podporuje jmenné prostory. Obě dvě verze mezi sebou tedy nejsou kompatibilní [61]. Tento rámec umožňuje programátorům používat komponenty, mezi kterými existují jen minimální závislosti. Ty slouží pro práci s filtry, umožňují validaci uživatelských dat, autorizaci, autentizaci, správu e-mailů, tvorbu PDF souborů a další.

[13] http://twig.sensiolabs.org/
[14] http://framework.zend.com/

Zend Framework podporuje několik databázových systémů, například MySQL, PostgreSQL, SQLite a Oracle. Dále jsou k dispozici helpery, tedy pomocné funkce, které programování webových stránek ještě více usnadňují. Tento rámec nabízí několik předdefinovaných helperů, je však i možné definovat helpery vlastní. Helpery jsou v Zend Frameworku dvojího typu, a to helpery akcí využívané v řadičích a helpery pohledů využívané v pohledech [23].

Za tímto frameworkem stojí rozsáhlá komunita programátorů. Na jeho oficiálních webových stránkách lze nalézt mnoho návodů a ukázek kódů, k dispozici je ale především i rozsáhlá dokumentace, kterou lze stáhnout mimo jiné ve formátu PDF.

### 2.5.1 Instalace

Postup instalace Zend Frameworku je popsán v dokumentaci i na webových stránkách. Framework kromě PHP 5.3 a vyšší verze vyžaduje také povolenou funkci *mod_rewrite*. Po jeho stažení je možné jej rozbalit do kořenového adresáře webového serveru a dále získat kostru ukázkové aplikace [59].

### 2.5.2 Třída Zend_View

Zend Framework využívá pro práci s pohledy třídu *Zend_View*. Ta představuje šablonovací systém, ve kterém lze využít helpery pohledů, výstupní filtry a escapování proměnných a jako šablonovací jazyk zvolit buď jazyk PHP, anebo vytvořit instanci jiného šablonovacího systému a manipulovat s ním v rámci definice pohledu. Instanci *Zend_View* vytváří řadič, který jí přiřadí proměnné a poté požádá o vykreslení pohledu voláním metody *render()*. Proměnné jsou pak v definici pohledu dostupné pomocí *$this->variable*. Dle samotné definice je pak generován patřičný výstup. Příklad možné definice pohledu v Zend Frameworku je zachycen v kódu 2.7 [60].

Možnost využití alternativního šablonovacího systému je rovněž popsána v dokumentaci [62], k dispozici jsou dva možné způsoby – pomocí definice pohledu nebo využitím *Zend_View_Interface*. V tomto rámci je nezbytné provádět escapování proměnných pomocí funkce *escape()*.

```
<table>
    <tr>
        <th>Author</th>
        <th>Title</th>
    </tr>
    <!-- HTML kód doplněný PHP kódem -->
    <?php foreach ($this->books as $key => $val): ?>
    <tr>
        <!-- escapování proměnných -->
        <td><?php echo $this->escape($val['author']) ?></td>
        <td><?php echo $this->escape($val['title']) ?></td>
    </tr>
    <?php endforeach; ?>
</table>
```

*Kód 2.7: Ukázka definice pohledů v Zend Frameworku [60]*

# Kapitola 3

# Použité technologie

Následující kapitola shrnuje technologie, které byly využity pro implementaci rozšíření zvoleného rámce pro tvorbu webových aplikací v jazyce PHP. Technologie byly voleny s ohledem na požadavky výsledného řešení. Protože se tato diplomová práce věnuje rozšíření již existujícího rámce, jde zejména o technologie, které tento rámec využívá a které jsou běžně používány pro vývoj webových aplikací.

U každé technologie je uveden její stručný popis, zabývám se také vznikem dané technologie a její aktuální verzí. Tato kapitola nejprve popisuje protokol HTTP, dále jazyky HTM a XHTML, XML, PHP a v neposlední řadě se zmiňuje i o zajištění asynchronní komunikace pomocí AJAX. Jazyku XML se věnuji podrobněji s důrazem zejména na jmenné prostory, neboť tato diplomová práce se zabývá možností definice šablon stránek právě v tomto jazyce. Dále popisuji také možnosti parsování XML dokumentů v jazyce PHP.

## 3.1 Protokol HTTP

HTTP (Hypertext Transfer Protocol) je v současnosti nejpoužívanější protokol pro přenos dat mezi klientem a serverem. Přestože je určen pro výměnu hypertextových dokumentů ve formátů HTML, díky standardu MIME (Multipurpose Internet Mail Extensions) jej dnes je možné využít i pro přenos libovolných jiných souborů [11].

Činnost tohoto protokolu je založena na principu dotaz-odpověď. Klient požadující nějaký dokument zašle požadavek serveru a server na takovou žádost odpoví. Požadavek mimo jiné obsahuje identifikátor požadovaného dokumentu, server pak za odpovědí takový dokument klientovi zašle. Protokol HTTP přitom nazýváme bezstavovým protokolem, neboť při dalším požadavku klienta server nepozná, zda nový požadavek souvisí s předchozím či nikoliv [11].

První verze tohoto protokolu byla označena HTTP/0.9 a jednalo se o vůbec nejjednodušší možnou implementaci. Server na obdržený požadavek klienta rovnou zasílal požadovaný dokument. Tato verze protokolu vznikla v roce 1991. Verze HTTP/1.0 pak pochází z roku 1996. Ta mimo jiné začala využívat internetový standard MIME pro identifikaci typu dokumentu. Aktuální verze protokolu HTTP však vznikla ještě o rok později a je označena HTTP/1.1. Tato verze je stabilní, proto se již dále nevyvíjí [52]. V roce 1999 byla přepsána a její specifikace je dána normou RFC 2616 [26].

### 3.1.1 Hypertextový dokument

Hypertext je text, který obsahuje odkazy a není tedy lineární. Tento pojem poprvé použil počítačový vědec Ted Nelson, a to v roce 1965. Hypertext popsal jako nelineární text s rozvětvenou strukturou, která se skládá z textových bloků propojených různými spojeními. Hypertextovým dokumentem tedy rozumíme takový dokument, který v sobě zahrnuje běžný text doplněný odkazy na další hypertextové dokumenty [48] [22].

### 3.1.2 Identifikace zdroje

Jednotlivá místa v prostředí Internetu většinou poskytují nějaká data. Jsou tedy zdroji dat, proto je obvykle nazýváme zdroje. Každý zdroj má přitom svůj jednoznačný identifikátor, přičemž existují dva typy identifikátorů:

- Identifikátory specifikující místa v síti, tedy závislé na umístění v síti. Takové identifikátory nazýváme URL (Uniform Resource Locator).
- Identifikátory pojmenovávající zdroje, tedy nezávislé na umístění v síti. Takovým identifikátorům říkáme URN (Uniform Resource Name).

Obě tyto skupiny identifikátorů pak nazýváme URI (Uniform Resource Identifier), jde tedy o souhrnné označení. URL v hypertextovém prostředí pak představuje adresu síťového zdroje. Ta mimo TCP/IP adresy obsahuje také jméno protokolu a další dodatečné informace, které nejčastěji bývají uváděny ve formě parametrů. URL adresy slouží pro propojení s ostatními stránkami [11].

## 3.2 Jazyky HTML a XHTML

HTML (Hypertext Markup Language) je značkovací jazyk, který se stal základním jazykem využívaným pro tvorbu webových stránek. Jedná se o nejznámější aplikaci univerzálního značkovacího jazyka SGML (Standard Generalized Markup Language), kde jednotlivým značkám jazyka je přiřazena sémantika hypertextového dokumentu v prostředí Internetu [12].

Dokument ve formátu HTML je hypertextovým dokumentem. HTML je charakteristické množinou značek a atributů, které dávají význam jednotlivým částem dokumentu. Značky v jazyce HTML přitom mohou být párové nebo nepárové. Zatímco nepárové značky mají specifický význam v místě dokumentu, na kterém se nacházejí, párové značky strukturují obsah dokumentu nacházející se mezi počáteční a koncovou značkou, a to dle významu dané značky a jejích atributů. [55]

Webové prohlížeče zobrazují HTML dokumenty uživatelům, přičemž značky jazyka určují, jak má webová stránka vypadat. Dokumenty v tomto jazyce mají předepsanou strukturu, která je následující:

- <!DOCTYPE> deklarace se v dokumentu musí nacházet ještě před značkou *<html>*. Nejedná se přitom o HTML značku, nýbrž o informaci pro prohlížeč, v jaké verzi je HTML dokument napsán. Například pro dokument psaný v HTML5 se uvádí deklarace *<!DOCTYPE html>* [54].
- Značka *<html>* je párová, nazývá se kořenový element a reprezentuje celý dokument v HTML.

- Hlavička dokumentu uzavřená mezi počáteční a koncovou značkou <*head*> a </*head*> obsahuje metadata, která se vztahují k celému dokumentu. Jedná se o definici názvu a titulku dokumentu, kódování, klíčových slov a dalšího [21].
- Tělo dokumentu mezi značkami <*body*> a </*body*> obsahuje vlastní obsah dokumentu.

První verze HTML byla vydána v roce 1991 a od té doby vzniklo několik dalších verzí. Například v roce 1995 to byla verze označená jako HTML 2.0, o dva roky později pak verze HTML 3.2. Verze HTML 4.01, která opravuje chyby předchozího HTML 4.0, do jejíž specifikace přibyly nové značky a která je dodnes nejčastěji používanou verzí jazyka HTML, byla vydána v roce 1999. Od roku 2007 se také připravuje HTML 5, ve kterém bude možné využít další nové značky například pro přehrávání videí. Specifikace této verze však doposud není dokončena a v současné době ani podpora HTML 5 ze strany webových prohlížečů není úplná [55] [41].

Pod pojmem XHTML (Extensible Hypertext Markup Language) rozumíme variantu jazyka HTML, která je aplikací jazyka XML (Extensible Markup Language). Původně se předpokládalo, že jazyk HTML bude XHTML nahrazen, nakonec se tomu tak ale nestalo. Aktuální verze XHTML označená jako XHTML 1.1 pochází z roku 2010. XHTML je dnes podporováno hlavními webovými prohlížeči, přičemž XHTML se od HTML příliš neliší. XHTML je téměř identické s HTML 4.01, jde o jeho striktnější a čistější verzi [43] [56].

## 3.3 Jazyk XML

XML (Extensible Markup Language) je jazyk sloužící pro reprezentaci strukturovaných dat, a to ve formě jednoduchého textu. XML dokumenty mohou mít libovolnou strukturu a je možné v nich používat libovolně pojmenované značky. Jazyk XML je vyvíjený a standardizovaný konsorciem W3C a jeho poslední verzí je pátá revize verze 1.0 pocházející z roku 2008 [42]. Jazyk je snadno rozšiřitelný, byl odvozen od univerzálního značkovacího jazyka SGML jeho zjednodušením [53] a je určen především pro výměnu dat mezi aplikacemi.

Od jazyka XML se vyvinulo několik dalších technologií. Mezi ně patří například technologie XPath [45] nebo XSLT (eXtensible Stylesheet Language Transformations) [47], která slouží pro transformaci XML dokumentu na jiný XML, HTML či textový dokument. Dále jde například o jazyk XQuery [46], díky němuž lze nad daty ve formátu XML pokládat dotazy.

Jazykem, který slouží pro popis schématu XML dokumentu, je DTD (Document Type Definition). Schéma je určeno k definici metadat XML. Metadata určují, jaké prvky a atributy mohou být v XML dokumentu použity, co mohou obsahovat, jak je lze kombinovat a další. Schéma XML dokumentu tedy udává syntaxi nového značkovacího jazyka. Hlavní výhodou XML schématu je podpora datových typů, díky níž lze jednoduše vymezit přípustný obsah XML dokumentu. Nad XML schématem pak lze provádět transformace pomocí XSLT a manipulovat s ním pomocí DOM, neboť je rovněž psáno v XML [14].

### 3.3.1 Syntaxe XML

Syntaxe jazyka XML se v několika ohledech odlišuje od syntaxe jazyka HTML. Každý element, který není prázdným elementem, musí mít ukončovací značku – v HTML je možné ji v některých případech vypustit. Dále je třeba důsledně dodržovat vnořování značek a uvádět vždy nejprve ukončovací značku poslední počáteční značky. Každý element tedy musí

být celý obsažen v jiném elementu. Názvy značek jsou pak také závislé na velikosti písmen a každý XML dokument musí obsahovat kořenový element. Dále je třeba, aby hodnoty atributů byly zapsány v uvozovkách [58].

```
<?xml version="1.0" encoding="UTF-8" ?>
<korenovy>
    <element atribut="hodnota">
        <VNORENY>XML dokument</VNORENY>
    </element>
</korenovy>
```

*Kód 3.1: Ukázka XML dokumentu*

### 3.3.2 Jmenné prostory

Protože XML je jazykem rozšiřitelným, je možné si snadno definovat vlastní značky a atributy a poté je používat v určitém významu. Názvy elementů v rámci XML dokumentu však musí být jedinečné a jednoznačné. Zejména při spojení dvou a více XML dokumentů do jednoho proto často dochází ke konfliktům, kdy dvě značky mají stejná jména, ale jiný obsah a význam. Konflikt ukazuje kód 3.2. XML parser v takovém případě není schopen s rozdíly naložit.

```
<table>
    <tr>
        <td>První řádek, první sloupec</td>
        <td>První řádek, druhý sloupec</td>
    </tr>
</table>

<table>
    <name>Jméno</name>
    <width>Šířka</width>
    <length>Délka</length>
</table>
```

*Kód 3.2: Konflikt v XML dokumentu [57]*

Jmenné prostory jsou způsobem, jak zabránit konfliktům jmen elementů v XML dokumentech. XML značky mají díky prefixům různé názvy a ke konfliktům nedochází. Ke každému použitému prefixu musí být definován jmenný prostor.

Jmenný prostor je dán atributem *xmlns* počáteční značky daného elementu. Deklarace má pak syntaxi *xmlns:prefix="namespaceURI"*. Je-li tento atribut uveden, vztahuje se na všechny elementy, které jsou v daném elementu obsaženy. Všechny elementy uvnitř tohoto elementu jsou tedy asociovány se stejným jmenným prostorem.

Jmenný prostor může být deklarován buď v elementu, kde je použit, nebo v kořenovém elementu, jak ukazuje kód 3.3 [57].

```
<root xmlns:h="http://www.w3.org/TR/html4/"
      xmlns:f="http://www.w3.org">

<h:table>
    <h:tr>
        <h:td>První řádek, první sloupec</h:td>
        <h:td>První řádek, druhý sloupec</h:td>
    </h:tr>
</h:table>

<f:table>
    <f:name>Jméno</f:name>
    <f:width>Šířka</f:width>
    <f:length>Délka</f:length>
</f:table>

</root>
```

*Kód 3.3: XML dokument včetně definice jmenných prostorů [57]*

Z příkladu je patrné, že se žádný deklarovaný jmenný prostor nevztahuje na element *root*. Rovněž z něj vyplývá, že je možné deklarovat více jmenných prostorů současně. Některý z prefixů lze rovněž uvést už u kořenového elementu, čímž pak i daný element náleží do deklarovaného jmenného prostoru. Prefixy navíc mohou mít stejný název a přesto ukazovat na jiný jmenný prostor, a to díky opětovné deklaraci, jak je ukázáno v příkladu 3.4. Podobně je možné dvěma různým prefixům přiřadit stejnou URI [16].

```
<h:root xmlns:h="http://www.w3.org/TR/html4/">

<h:table>
    <h:tr>
        <h:td>První řádek, první sloupec</h:td>
        <h:td>První řádek, druhý sloupec</h:td>
    </h:tr>
</h:table>

<h:table xmlns:h="http://www.w3.org">
    <h:name>Jméno</h:name>
    <h:width>Šířka</h:width>
     <h:length>Délka</h:length>
</h:table>

</h:root>
```

*Kód 3.4: Opětovná deklarace jmenného prostoru [16]*

URI každého jmenného prostoru zaručuje jednoznačnou identifikaci tohoto jmenného prostoru. Hodnotou atributu *xmlns* přitom nejčastěji bývá URL, může jí však být i URN.

XML parser samotnou adresu při zpracování XML dokumentu nevyužívá, není tedy důležité, zda se na ní nachází popis využívaného XML formátu. Přesto je URL obvykle považováno za adresu webové stránky obsahující informace o daném jmenném prostoru [57].

Aktuální verzí specifikace jmenných prostorů pro XML 1.0 je třetí revize z roku 2009 [49], pro XML 1.1 je to druhá revize z roku 2006 [50].

### 3.3.3 Výchozí jmenný prostor

Díky výchozímu jmennému prostoru je možné se vyhnout zapisování prefixů u každé značky, která do daného prostoru patří. Výchozí jmenný prostor se opět vztahuje na všechny potomky, pokud některý z nich není explicitně předeklarován. Syntaxe deklarace takového prostoru je pak *xmlns="namespaceURI"*. Výchozí jmenný prostor je také možné zrušit, a to zápisem *xmlns=""* [57] [16].

### 3.3.4 Jmenné prostory atributů

Atributy elementů nepatří do žádného jmenného prostoru. Jmenný prostor jim však může být přiřazen, a to opět uvedením prefixu před jménem atributu. Atribut může patřit do jiného jmenného prostoru než značka, u které je atribut uveden. Rovněž platí, že žádný element nesmí obsahovat více atributů stejného názvu a obdobně ani více atributů stejného názvu patřících do stejného jmenného prostoru [49].

## 3.4 Rozhraní DOM

Pod pojmem DOM (Document Object Model) rozumíme objektový model dokumentu. Jde o objektově orientovanou reprezentaci dokumentů ve formátu HTML a XML, díky níž můžeme k danému dokumentu přistupovat jako ke stromu a modifikovat tak jeho obsah. DOM je platformě a jazykově nezávislé rozhraní, které je standardizováno konsorciem W3C. Standardizace proběhla kvůli dřívější nekompatibilitě původních rozhraní v různých webových prohlížečích. DOM umožňuje programům a skriptům k dokumentům přistupovat dynamicky a modifikovat jejich obsah, styl i strukturu [51].

Standard DOM je specifikací rozhraní, které může být implementováno v libovolném programovacím jazyce. Přestože zmíněný standard definuje vazbu na jazyky JavaScript a Java, existují i implementace DOM pro jazyky C++, PHP, Perl, Python a mnoho dalších. První verze DOM specifikace byla vydána v roce 1998. Postupem několika let pak úpravami a rozšiřováním vzniklo několik dalších verzí, které zachovávají zpětnou kompatibilitu. Existující verze specifikace jsou dnes rozděleny do několika úrovní – DOM Level 0 až DOM Level 3. Každá aplikace, která podporuje určitou DOM úroveň, pak musí implementovat nejen všechny její povinné moduly, ale i moduly všech nižších úrovní. Specifikace DOM Level 3 byla publikována v roce 2004 a přinesla další rozšíření, například Load and Save specifikující proces čtení a serializace DOM stromu. V současné době se pracuje také na DOM Level 4, poslední koncept byl vydán v listopadu 2013 [44] [12].

## 3.5 Jazyk PHP

PHP (Hypertext Preprocessor, dříve Person Home Page) je skriptovací programovací jazyk, který slouží pro tvorbu dynamických webových aplikací a za tímto účelem je dnes také hojně používán. Skripty psané v PHP jsou vykonávány na straně serveru, což tento jazyk odlišuje

například od rovněž často využívaného JavaScriptu [29], který se provádí na klientské straně. PHP kód je tedy nejprve proveden a teprve jeho výsledek je uživateli zobrazen.

PHP skripty se zapisují mezi značky *<?php* a *?>*, a to nejčastěji do běžného HTML dokumentu. HTTP server je pak doplněn o procesor jazyka PHP. Pokud klient požaduje běžný soubor (HTML dokument, obrázek apod.), server jej zašle v odpovědi. Jedná-li se však o PHP dokument, je nejprve předán procesoru jazyka PHP. Ten vykoná vložený PHP kód a vrátí výsledek v HTML, teprve ten je pak zaslán klientovi [13].

Syntaxe jazyka PHP je inspirována jinými jazyky, jako jsou Perl, C nebo Java. PHP je platformě nezávislý jazyk a dnes podporuje řadu knihoven například pro práci se soubory, XML dokumenty či databázovými systémy (MySQL, Oracle a dalšími). Jde o jazyk dynamicky typovaný, v němž pole jsou asociativní. Je možné v něm využívat funkcí operačního systému, což na druhou stranu může způsobovat nekompatibilitu. PHP je dnes podporován prakticky všemi hostingovými službami.

Současnou verzí PHP je PHP 5.5.10. Bylo vydáno 6. března 2014, podporovány jsou však stále i starší verze. PHP 5.4.26 vyšlo ve stejný den jako nejnovější verze a PHP 5.3.28 bylo vydáno 12. prosince 2013 [37].

### 3.5.1 Podpora objektové orientace

Základy objektové orientace se objevily už v PHP 3, které bylo vydáno v polovině roku 1998. Tehdy se však jednalo pouze o jakýsi doplňující nápad. Stejně tak tomu zůstalo i po vydání PHP 4 v roce 2000, kdy bylo možné vytvořit třídu a byla zde podpora jednoduché dědičnosti v jedné úrovni. Teprve PHP 5 vydané v roce 2004 přineslo výrazné vylepšení objektové orientace. Přibyla podpora objektů a objevily se i modifikátory přístupu, které v dřívějších verzích PHP zcela chyběly. PHP 5 rovněž přišlo se Standardní PHP knihovnou SLP (Standard PHP Library), která nabízí mnoho předpřipravených tříd a rozhraní. PHP 5.1 tuto knihovnu ještě rozšířilo [28] [25].

### 3.5.2 Podpora XML

PHP dnes podporuje knihovny pro práci s XML dokumenty, díky kterým je možné tyto dokumenty pohodlně zpracovat. První podpora XML se objevila již v PHP 3, tehdy se však jednalo o knihovnu, jejíž používání nebylo příliš pohodlné a která se nehodila pro zpracování XML dokumentů se složitějšími strukturami.

PHP verze 4 se snažilo podporu XML vylepšit, a tak přibyla především možnost načtení celého XML dokumentu do paměti jako DOM strom. Knihovna pro práci s DOM podporovala i dotazování pomocí XPath, avšak rozhraní této knihovny se v jednotlivých verzích PHP 4 měnilo, přičemž ani přesto nebylo v souladu se standardem definovaným konsorciem W3C. PHP 4 pak přišlo také s možností provádění XSLT transformací, a to postupným přidáním dvou knihoven. Ani ty však mezi sebou nebyly kompatibilní.

Teprve PHP 5 rozhraní knihoven sjednotilo, kvůli čemuž ale bylo při přechodu z PHP verze 4 na PHP 5 třeba přepsat prakticky všechny skripty pracující s XML. Všechna rozhraní byla přepracována tak, aby odpovídala daným standardům. Nedostatky původního návrhu pak byly odstraněny až v PHP 5.1 [27].

### 3.5.3 Knihovna SimpleXML

SimpleXML je jedna z knihoven, které slouží pro zpracování XML dokumentů v PHP. Dokument XML načítá celý do paměti a reprezentuje jej strukturou objektů, přičemž jejich

jména odpovídají názvům elementů. Díky tomu jsou informace snadno přístupné. V případě, že element obsahuje další potomky, není mapován na řetězec, ale na pole objektů, které lze zpracovat v jediném cyklu. K potomkům daného elementu se lze dostat prostřednictvím členských proměnných, atributy jsou pak reprezentovány asociativním polem.

SimpleXML je díky své jednoduchosti velmi oblíbená knihovna a hodí se především pro práci s menšími XML dokumenty, které mají jednoduchou a pravidelnou strukturu. Práce s dokumenty, které mají smíšený obsah a využívají jmenné prostory, již není tak bezproblémová [27].

```
<?php
// vytvoření struktury objektů
$xml = simplexml_load_string("dokument.xml");

// přístup k elementům
$xml->channel->title;

// přístup k atributům
$xml["version"];

// zpracování elementů v cyklu
foreach ($xml->channel->item as zprava) {
    // zpracování
}
?>
```

*Kód 3.5: Práce s knihovnou SimpleXML [27] [39]*

### 3.5.4 Rozhraní SAX

SAX parser patří k nejstarším rozhraním pro práci s XML dokumenty. Původně byl vytvořen pro jazyk Java, později se však v upravených podobách objevil i v dalších jazycích. Jeho počátky v PHP sahají až do dob PHP 3, proto nemá objektové rozhraní a pracuje se s ním podobně jako s připojením k databázi nebo se soubory. Na rozdíl od SimpleXML a od rozhraní DOM, kterému se budu věnovat v další části kapitoly, SAX nenačítá celý XML dokument do paměti, a proto se hodí i pro zpracovávání velkých dokumentů. XML je čteno sekvenčně a pro každý prvek (počáteční a koncové značky, komentáře apod.) je vyvolána určitá událost, kterou lze obsloužit.

Rozhraní SAX je tedy poměrně komplikované. Protože ke zpracování XML dokumentů dochází nepřímo v obsluze událostí, je třeba definovat funkce pro jejich obsluhu a zaregistrovat je ve vytvořeném parseru [27].

```php
<?php
// vytvoření parseru
$parser = xml_parser_create("utf-8");

// nastavení rozlišování velikosti písmen
xml_parser_set_option($parser, XML_OPTION_CASE_FOLDING, false);

// nastavení funkcí pro obsluhu událostí
// startElement(), endElement() a characters() musí být definovány dále
xml_set_element_handler($parser, "startElement", "endElement");
xml_set_character_data_handler($parser, "characters");

// otevření XML dokumentu
$fp = fopen("dokument.xml", "r");

// čtení bloků textu
while ($x = fread($fp, 4096)) {
    // předání bloku parseru ke zpracování
    if (!xml_parse($parser, $x, feof($fp))) {
        // syntaktická chyba v XML dokumentu
    }
}

// uvolnění paměti
xml_parser_free($parser);
?>
```

*Kód 3.6: Práce s rozhraním SAX [27]*

### 3.5.5 Rozhraní DOM v PHP

DOM je standardní rozhraní pro práci s XML dokumenty definující, jakým způsobem se XML mapuje na hierarchii objektů v paměti. Částem dokumentů pak odpovídají objekty a využitím vlastností a metod lze zjistit názvy uzlů, jejich obsahy, seznamy objektů, které reprezentují jejich potomky, a další.

Dokumentu odpovídá objekt, který je instancí třídy *DOMDocument*, u elementů jde o třídu *DOMElement*, pro atributy je to třída *DOMAttr* a textové uzly jsou reprezentovány objekty, které jsou instancí třídy *DOMText*. Všechny navíc mají společného předka, třídu *DOMNode*, a proto mají některé společné vlastnosti a metody. Používání těchto vlastností pro výběr určitých částí dokumentů však není úplně vždy pohodlné, a tak se v praxi často využívá jazyk XPath, kterým se zabývám v následující části této kapitoly [38] [27].

```php
<?php
// vytvoření DOM reprezentace XML dokumentu
$doc = new DomDocument();
$doc->load("dokument.xml");

// výběr všech elementů s názvem item
$polozky = $doc->getElementsByTagName("item");

// zpracování elementů item
foreach ($polozky as $polozka) {
    // zpracování
}

// přístup k textové hodnotě prvního elementu s názvem item
$doc->getElementsByTagName("item")->item(0)->textContent;
?>
```

*Kód 3.7: Práce s XML dokumentem pomocí DOM [27]*

### 3.5.6 Jazyk XPath

XPath (XML Path Language) je jazyk, který umožňuje pokládání dotazů nad stromovou reprezentací dokumentu. Dotazy mají jednoduchou syntaxi a vybírají určitou část XML dokumentu nebo konkrétní hodnotu. XPath dotazování je deklarativním přístupem, kdy programátor pouze určí, co chce získat, nikoliv jak se k takovému výsledku má aplikace dopracovat. Tím se práce s XML dokumenty pomocí XPath liší od výše uvedených příkladů. Podobně je tomu například v SQL. Aby bylo možné pokládání XPath dotazů v jazyce PHP, je nejprve třeba XML dokument načíst do paměti jako klasický DOM strom [27] [45].

```php
<?php
// vytvoření DOM reprezentace XML dokumentu
$doc = new DomDocument();
$doc->load("dokument.xml");

// vytvoření objektu, který dovoluje dotazování v XPath
$path = new DOMPath($doc);

// vyhodnocení XPath dotazu
$vysledek = $path->evaluace("/rss/channel/title");
?>
```

*Kód 3.8: Práce s XML dokumentem pomocí XPath [27]*

### 3.5.7 Rozhraní XMLReader

Rozhraní XMLReader překonává nevýhodu rozhraní SAX. Zatímco SAX zahrnuje naši aplikaci proudem událostí, které je třeba obsloužit, XMLReader předá aplikaci data pouze v případě, že o to aplikace požádá. Jedná se o tzv. pull model a tedy pull parser. Díky tomu je kód aplikace mnohem přehlednější [27].

```php
<?php
// vytvoření parseru
$reader = new XMLReader();
$reader->open("dokument.xml");

// zpracování elementů v cyklu
while ($reader->read()) {
    // obsluha elementu link
    if ($reader->nodeType == XMLReader::ELEMENT &&
        $reader->name == "link") {
        // obsluha
    }
}
?>
```

*Kód 3.9: Práce s rozhraním XMLReader [27] [40]*

## 3.6 Technologie AJAX

AJAX (Asynchronous JavaScript and XML) je technologie, díky níž je možné měnit obsah částí webových stránek, aniž by stránka musela být opětovně načtena celá. Nejedná se o nový programovací jazyk, ale o způsob využívání již existujících standardů. Pojmem AJAX tedy chápeme několik technologií, které se využívají současně. Jde především o HTML, CSS, DOM, JavaScript a objekt XMLHttpRequest, který slouží pro asynchronní komunikaci se serverem. AJAX se dnes využívá pro tvorbu interaktivních webových stránek [15].

Pojem AJAX pochází z roku 2005, kdy jej v článku *Ajax: A New Approach to Web Applications* představil Jesse James Garrett [10]. Ten je také dnes považován za autora tohoto pojmu. O popularitu této technologie se zasloužily například aplikace společnosti Google, které ji začaly využívat. Mezi nevýhody technologie AJAX patří především větší objem přenášených dat, a to z důvodu vyššího počtu HTTP požadavků.

# Kapitola 4

# Návrh rozšíření systému definice šablon stránek

V této kapitole se zabývám návrhem rozšíření zvoleného rámce pro tvorbu webových aplikací v jazyce PHP o definici šablon stránek v XML s možností implementace vlastních rozšiřujících značek v PHP. Volba rámce a návrh rozšíření vychází z analýzy existujících rámců popsané v kapitole 2. Návrh byl rovněž konstruován tak, aby co nejlépe vyhovoval požadavkům zadání diplomové práce.

Úvod kapitoly je věnován volbě vhodného rámce dle analýzy a seznámení s možnostmi definic šablon stránek v různých rámcích. Dále se zabývám volbou vhodného způsobu zpracování XML dokumentů v PHP. V úvodu samotného návrhu je pak uveden postup instalace navrhovaného rozšíření zvoleného rámce a také doplněná adresářová struktura frameworku. Návrh je dále rozdělen na dvě části. První část je uživatelská a popisuji v ní, jakým způsobem by uživatel při programování webových aplikací měl uvádět definice vlastních elementů využívaných v definicích šablon stránek, jak by měl volit názvy tříd a metod a tak dále. Součástí této části kapitoly je též návrh vazby mezi stavem prvku uživatelského rozhraní a daty webové aplikace, návrh ukázkových knihoven komponent a ukázkové webové aplikace a je také popsáno, jakým způsobem bude implementována podpora technologie AJAX. V druhé části se pak zabývám zpracováním definic XML elementů, tedy způsobem, jakým bude pracovat parser šablon stránek pro zpracování XML elementů, jak bude zakomponován do existujícího rámce a jakou funkcionalitu pomocných tříd bude využívat.

## 4.1 Volba rámce a způsobu zpracování XML dokumentů

Analýza existujících rámců odhalila jejich přednosti a nedostatky. Vyplynula z ní především důležitá fakta týkající se definice šablon stránek. Zatímco framework CakePHP a Zend Framework umožňují definovat šablony stránek pomocí HTML kódu doplněného kódem PHP, rámec Nette využívá šablonovací systém Latte a rámec Symfony systém Twig. Protože Latte a Twig jsou srovnatelné, i když Twig má lepší podporu vývojových prostředí [18], v ukázkách obou z nich v kapitole 2 lze nalézt podobnosti. Díky šablonovacím systémům je zápis kódu rychlejší a přehlednější, proto jsem se rozhodl výběr omezit na některý z těchto dvou rámců. Existují však i další důvody. Například Zend Framework před výpisem hodnot proměnných vyžaduje jejich escapování, zatímco Nette Framework toto řeší automaticky.

Z výše zmíněných jsem poté zvolil Nette Framework jako vhodný rámec pro tvorbu webových stránek a tuto diplomovou práci. Mezi hlavní důvody patří především rozsáhlá

dokumentace dostupná v českém jazyce, aktivní komunita a také popularita tohoto frameworku v České republice.

V kapitole 3 jsou popsány možné způsoby zpracování XML dokumentů v jazyce PHP. Z analýzy vyplývá, že SimpleXML je vhodný jen pro jednoduché dokumenty, které nevyužívají jmenné prostory. SAX pak sice nenačítá celý dokument do paměti, ovšem nedisponuje objektově orientovaným rozhraním. Oba způsoby jsem proto vyhodnotil jako nevhodné pro použití v této diplomové práci. Ze zbylých popsaných možností jsem se rozhodl pro rozhraní DOM, které je standardním rozhraním a jeho implementace existuje pro řadu programovacích jazyků.

## 4.2 Instalace rozšíření

Aplikace se v Nette Frameworku nachází v adresáři *app*. Zde je také umístěn soubor *bootstrap.php*, který provádí konfiguraci, načítá konfigurační soubory ve formátu NEON[1] a slouží také k volání statických metod a změně hodnot statických vlastností tříd. Na oficiálních stránkách frameworku lze nalézt několik rozšíření, které se aktivují právě v tomto souboru voláním statické metody, a tímto způsobem se bude aktivovat také mnou navrhované rozšíření.

V souboru *bootstrap.php* se rovněž vytváří instance třídy *Nette\Configurator*, která mimo registraci adresáře pro zaznamenávání chyb a adresáře pro dočasné soubory vytváří instanci třídy *Nette\Loaders\RobotLoader*. RobotLoader se v tomto rámci stará o načítání tříd z adresářů, přičemž Nette Framework v současné verzi ve výchozím nastavení načítá třídy z adresáře *app* a adresáře *vendor/others*. Z tohoto důvodu bude právě v druhém z uvedených adresářů umístěno rozšíření pro definici šablon stránek v XML. RobotLoader se pak postará o načtení tříd, a tak bude kdykoliv možné vytvořit jejich instanci.

V *app/bootstrap.php* lze dále provádět další konfiguraci aplikace, a to především změnou hodnot statických vlastností tříd. Konkrétním příkladem může být třída, ve které lze změnit maximální délku vypisovaného řetězce při ladění a další, tedy *Nette\Diagnostics\Debugger*. V tomto souboru se bude provádět i změna statických vlastností jedné ze tříd mnou navrhovaného rozšíření, a to voláním statických metod. Půjde o možnost mazat paměť cache při každém načtení stránky a možnost ponechání komentářů v XML šablonách.

Presentery, které jsou v Nette Frameworku reprezentovány třídami, musejí dědit z třídy *Nette\Application\UI\Presenter*. Úpravou této třídy je dle dokumentace možné registrovat vlastní filtr pro zpracování definic šablon stránek. Aby uživatel nemusel přepisovat tuto třídu, bude v rámci mého rozšíření k dispozici vlastní třída presenterů, která bude implementovat registraci filtru pro zpracování šablon stránek a případnou další funkcionalitu. Každý presenter webové aplikace pak bude dědit z mnou definované třídy a teprve ta bude dědit z výše uvedené třídy presenterů.

Instalaci rozšíření je tedy možné shrnout do tří kroků:

1. Uživatel do adresáře *vendor/others* nahraje adresář *Xml* obsahující navrhované rozšíření.
2. V souboru *app/bootstrap.php* aktivuje definice šablon stránek v XML přidáním kódu *Xml\Engine::register($configurator)*.

[1] http://ne-on.org/

3. V adresáři *app/presenters* zajistí, typicky pouze změnou definice třídy *BasePresenter*, aby každý presenter dědil z třídy *Xml\UI\Presenter*.

## 4.3 Adresářová struktura

Rozšířením Nette Frameworku o možnost definice šablon stránek v XML dojde i k rozšíření z hlediska adresářové struktury. Vývoj webové aplikace probíhá z pravidla v adresáři *app*, kde uživatel vytváří vlastní modely, presentery, šablony, komponenty a další. Právě na tomto místě bude mít proto možnost uvést definice elementů, které bude v šablonách stránek psaných v jazyce XML používat. Definice budou představovat kód šablonovacího systému Latte, který bude pro každý daný XML element vygenerován.

Definice elementů bude uživatel uvádět například v adresáři nazvaném *xml*. Tento adresář se bude nacházet přímo v adresáři *app*, záměrně nenavrhuji jeho umístění v adresáři *app/templates*. V něm jsou totiž další adresáře, a to pro každý presenter. Toto řešení by pro větší webové aplikace bylo nepřehledné. Název adresáře i adresář samotný je pouze doporučením. Ve skutečnosti se třídy s definicemi XML elementů mohou nacházet kdekoliv v adresáři *app*.

V kódu 4.1 je zachycena adresářová struktura webové aplikace v Nette Frameworku rozšířená o adresář *xml*.

```
components/   // komponenty
config/       // konfigurace
model/        // modely
presenters/   // presentery
router/       // routování
templates/    // šablony stránek
xml/          // definice XML elementů
```

*Kód 4.1: Adresář webové aplikace rozšířený o definice XML elementů*

Nette Framework se nachází v adresáři *vendor/nette*, kde jsou další adresáře reprezentující jmenné prostory tohoto rámce. Se mnou navrhovaným rozšířením, jak plyne již z výše uvedené instalace rozšíření, přibude jmenný prostor nazvaný *Xml*. V tom budou umístěny veškeré třídy zajišťující funkcionalitu tohoto rozšíření. Této části návrhu se budu blíže věnovat od kapitoly 4.11.

## 4.4 Šablony stránek v XML

Následující podkapitoly se zabývají uživatelskou částí rozšíření, tedy tím, co se nachází v adresáři *app*.

Šablony stránek využívající šablonovací systém Latte a nacházející se v adresáři *app/templates* mají příponu *.latte*. Ta je definována ve třídě, ze které dědí každý presenter, tedy třídě *Nette\Application\UI\Presenter*. Názvy adresářů a souborů šablon stránek jsou odvozeny od názvů presenterů a bez zásahu do struktury Nette Frameworku je obtížné je změnit. Uživatel by touto změnou navíc přišel o koncept, na který je zvyklý. Zároveň by však s instalací mého rozšíření měla být zachována možnost využívat šablony stránek psané v Latte. Nejvhodnějším řešením se tak jeví použití odlišných přípon souborů.

Soubory definic šablon stránek v XML budou mít stejné názvy jako dosavadní soubory šablon stránek, avšak jejich přípona bude *.xml*. Soubory *.latte* tak mohou být při využívání mého rozšíření beze změny zachovány, čímž bude možné přepínat mezi standardním šablonovacím systémem Nette Frameworku a definicemi šablon stránek v XML. To se provede jedinou změnou v souboru *app/bootstrap.php*, a sice zakomentováním řádku *Xml\Engine::register($configurator)*.

## 4.5 Adresáře a třídy jmenných prostorů

V kapitole 3 jsem se mimo jiné věnoval jazyku XML s důrazem na využívání jmenných prostorů. V rámci mého rozšíření hrají jmenné prostory významnou roli. Uživatel definuje šablony stránek v jazyce XML, přičemž pro každý element uvede definici, podle níž bude vygenerován příslušný kód šablonovacího systému Latte. Uživatel tedy bude mít možnost využívat veškerou funkcionalitu Nette Frameworku. Definice každého elementu přitom bude očekávána v určité třídě, a to právě v závislosti na jmenném prostoru, do kterého daný element spadá. Z tohoto pohledu lze rozlišovat elementy, které nenáleží žádnému jmennému prostoru, elementy spadající do výchozího jmenného prostoru a elementy spadající do uživatelem definovaných jmenných prostorů.

Uživatel pro každý jmenný prostor vytvoří třídu a umístí ji kamkoliv do adresáře *app*, například do výše navrhovaného adresáře *xml*. Třída může mít libovolný název a může také patřit do libovolného jmenného prostoru. Každá třída bude představovat jeden jmenný prostor a pro XML elementy náležící do daného prostoru pak budou volány metody z odpovídající třídy.

Třídy budou umístěny v PHP souborech, přičemž je doporučeno dodržovat obecné konvence, tedy aby se v jednom souboru nacházela pouze jedna třída a název souboru odpovídal názvu třídy. Toto je ovšem zcela na uživateli. Konkrétní metoda pro konkrétní XML element bude vždy volána z třídy obsahující URI jmenného prostoru, do kterého daný XML element patří.

Příklad tříd reprezentujících jmenné prostory je zachycen v kódu 4.2. V souboru *noElements.php* se nachází třída, jejíž metody budou volány pro XML elementy nenáležící do žádného jmenného prostoru. V souboru *defaultElements.php* se nachází třída, jejíž metody budou volány pro elementy náležící do výchozího jmenného prostoru. Poslední z uvedených souborů pak obsahuje třídu s metodami, které budou volány pro XML elementy náležící do jmenného prostoru h.

```
noElements.php        // elementy žádného jmenného prostoru
defaultElements.php   // výchozí jmenný prostor
hElements.php         // elementy jmenného prostoru h
```

*Kód 4.2: Příklad tříd reprezentující jmenné prostory*

## 4.6 URI jmenných prostorů

Každý jmenný prostor je identifikován pomocí URI, která se u elementů v jazyce XML uvádí jako hodnota atributu *xmlns*, jak bylo popsáno v kapitole 3. Přestože se jako prefix elementů uvádí název jmenného prostoru, jmenný prostor jako takový je jednoznačně identifikován právě pomocí URI. Proto i třídy reprezentující daný jmenný prostor budou obsahovat URI každého jmenného prostoru, a to v podobě veřejné vlastnosti třídy.

Definice XML elementů, které nespadají do žádného jmenného prostoru, budou očekávány v třídě s veřejnou vlastností *xmlns* nastavenou na hodnotu prázdného řetězce. Taková třída musí existovat vždy, když se v definici šablon stránek bude nacházet alespoň jeden element nenáležící do žádného jmenného prostoru.

Třídy ostatních jmenných prostorů včetně výchozího budou obsahovat veřejnou vlastnost *xmlns* s hodnotou URI, kterou je jmenný prostor identifikován. Právě podle této vlastnosti budou vyhledávány třídy, ve kterých budou očekávány definice XML elementů náležících do daných jmenných prostorů.

Vyhledávání tříd podle atributu *xmlns* bylo zvoleno z důvodu, že každému jmennému prostoru může být nejprve přiřazena jedna URI a později URI jiná, jak vyplynulo z analýzy problematiky uvedené v kapitole 3. Nebylo by tedy možné třídy vyhledávat podle jména, neboť je nutné, aby mohlo existovat teoreticky nekonečně mnoho tříd pro každý prefix jmenného prostoru.

Praktická ukázka definice šablony stránek v XML s využitím jmenného prostoru *h*, který má v rámci šablony dvě různé URI, je zachycena v kódu 4.3. Je patrné, že uživatel pro takový jmenný prostor musí vytvořit dvě třídy a v každé z nich definovat hodnotu vlastnosti *xmlns*. Třídy samozřejmě musí mít různé názvy, a tak je možné, aby například jedna z nich začínala znakem _. Jak však bylo uvedeno, název třídy může být zcela libovolný, třídy rovněž mohou patřit do odlišných jmenných prostorů.

```
<!-- šablona stránek v XML -->
<h:table xmlns:h="http://www.w3.org/TR/html4">
   <h:tr>
      <h:td>První řádek, první sloupec</h:td>
      <h:td>První řádek, druhý sloupec</h:td>
   </h:tr>
</h:table>
<h:table xmlns:h="http://www.w3.org">
   <h:name>Jméno</h:name>
   <h:width>Šířka</h:width>
   <h:length>Délka</h:length>
</h:table>

<?php
// třída hElements v souboru app/xml/hElements.php
class hElements {
   public $xmlns = "http://www.w3.org/TR/html4";
}

// třída _hElements v souboru app/xml/_hElements.php
class _hElements {
   public $xmlns = "http://www.w3.org";
}
?>
```

*Kód 4.3: Praktická ukázka využití jmenných prostorů v definici šablon stránek v XML*

## 4.7 Definice XML elementů

Z návrhu vyplývá, že každému jmennému prostoru identifikovanému pomocí URI bude odpovídat jedna třída. V této třídě budou očekávány definice XML elementů, a to v podobě metod třídy. Metody budou veřejné a budou ve svém názvu obsahovat jméno daného XML elementu doplněné sufixem *Element*. Každá metoda bude navíc přijímat jeden argument, kterým bude konkrétní DOM element, jenž bude odpovídat jednomu elementu z definice šablony stránek v XML.

Metody tříd jmenných prostorů budou DOM element modifikovat a v některých případech vracet hodnotu. Pro každý XML element šablony stránek tedy bude volána metoda a element bude tímto způsobem transformován dle uživatelem definovaných metod. Výsledkem zpracování šablony stránek v XML pak bude šablona stránek v šablonovacím systému Latte, která bude v rámci Nette Frameworku dále zpracována běžným způsobem.

Pro každý XML element musí být ve třídě daného jmenného prostoru definovaná daná metoda. V případě, že metoda pro daný XML element nebude nalezena, bude volána metoda *otherElements()* téže třídy. Díky tomuto přístupu bude možné zpracovávat elementy, pro které uživatel nevytvořil příslušnou metodu, jednotným způsobem. Současně nebude nutné vytvářet metody pro elementy, které uživatel nechce modifikovat.

Aby práce s DOM elementy byla pro uživatele co nejjednodušší, bude každá třída odpovídající danému jmennému prostoru dědit z třídy *Xml\DOMHelper*. Tato třída bude nabízet pomocné metody pro práci s DOM elementy, díky nimž bude možné snadno změnit název elementu, manipulovat s atributy elementu a další. Některé metody budou navíc z nepárových elementů dělat elementy párové, a to zcela automaticky. Příkladem takové metody je metoda *addTextNode()*.

Třída *Xml\DOMHelper* bude mimo jiné implementovat následující metody:

- Metoda *DOMElement renameElement(DOMElement $element, string $name, string $nsPrefix = "")* změní název daného elementu a vrátí takto modifikovaný DOM element.
- Metoda *DOMAttr setAttribute(DOMElement $element, string $name, string $value)* přidá atribut k danému elementu a vrátí přidaný atribut.
- Metoda *Boolean removeAttribute(DOMElement $element, string $name)* odstraní atribut.
- Metoda *string getAttributeValue(DOMElement $element, string $name)* vrátí hodnotu daného atributu.
- Metoda *DOMElement addTextNode(DOMElement $element, string $text, $before = NULL)* přidá textový uzel k danému elementu a vrátí přidaný uzel.

Příklad možné definice třídy pro jmenný prostor *h* je zobrazen v kódu 4.4. Tato třída poskytuje metodu pro zpracování XML elementu *nadpis*, ostatní XML elementy šablony stránek z tohoto jmenného prostoru nebudou změněny.

```
<!-- šablona v XML -->
<h:nadpis hodnota="Moje stránky" />

<?php
// třída jmenného prostoru h
class hElements extends Xml\DOMHelper {
   public $xmlns = "http://www.w3.org/TR/html4";

   public function nadpisElement($element) {
      $element = $this->renameElement($element, "h1");
      $hodnota = $this->getAttributeValue($element, "hodnota");
      $this->addTextNode($element, $hodnota);
      $this->removeAttribute($element, "hodnota");
      return $element;
   }

   public function otherElements($element) {
   }
}
?>
```

*Kód 4.4: Příklad možné definice třídy pro zpracování XML elementů*

## 4.8 Vazba mezi stavem prvku uživatelského rozhraní a daty aplikace

Nette Framework výrazným způsobem usnadňuje tvorbu a zpracování formulářů a veškeré možnosti jsou popsány v dokumentaci. Při tvorbě aplikací v tomto rámci se formuláře obvykle vytvářejí v presenterech, a to pomocí třídy *Nette\Application\UI\Form*, přičemž samotné přidání formuláře do presenteru probíhá pomocí tzv. továrny. Formulář je chápat jako komponenta. Je možné mu přidat různé typy prvků, přiřadit jim názvy, výchozí hodnoty, určit, která vstupní pole jsou povinná, jaké hodnoty mohou obsahovat a tak dále. Rovněž lze uvést název metody, která je volána po úspěšném odeslání formuláře.

K vykreslení formuláře dochází na základě definice v šabloně stránky, přičemž šablonovací systém Latte tento proces výrazným způsobem usnadňuje. V definicích šablon stránek lze rovněž využít proměnné, které jsou šabloně předány z presenteru, a to pomocí zápisu *$this->template->variable = $value*, a to nejčastěji v metodě *beforeRender()*.

Požadavkem zadání této diplomové práce je vazba mezi stavem prvku uživatelského rozhraní a daty webové aplikace. Uživatel tedy v presenteru předá šabloně například instanci nějaké třídy, jejíž některá z vlastností bude v šabloně použita jako hodnota některého ze vstupních polí formuláře. V případě, že tato hodnota bude uživatelem změněna a formulář odeslán, mělo by dojít k nahrazení hodnoty také v příslušné třídě. Každému vstupnímu poli samozřejmě může náležet hodnota z jiné třídy.

O vytvoření vazby mezi stavem prvku formuláře a daty aplikace se bude starat parser. Uživatel mu k vytvoření vazby dá pokyn voláním metody *createRelation()* z třídy *Xml\DOMHelper*, které předá XML element reprezentující kořenový uzel formuláře, a navrácením příslušné návratové hodnoty definované v třídě *Xml\DOMHelper*. Parser pak

ve všech vstupních polích vyhledá případné vazby na data s tím, že bude uvažovat i možné změny syntaxe Latte maker, a tyto vazby uloží do paměti cache. Ukládání do cache je nezbytné, neboť Nette Framework šablony stránek zpracovává pouze jedenkrát a při dalších požadavcích už ke zpracování nedochází. Dále bude nutné registrovat metodu, která bude volána po odeslání formuláře a která na základě vazeb uložených v paměti cache aktualizuje příslušné hodnoty. Z tohoto důvodu bude uživatel formuláře v presenterech vytvářet jako instanci třídy *Xml\UI\Form*, které předá instanci presenteru, namísto původní *Nette\Application\UI\Form*. Třída se pak postará o registraci metody volané po odeslání formuláře.

Příklad vytvoření formuláře v Nette Frameworku s transparentní vazbou na data webové aplikace je k dispozici v kódu 4.5. Z uvedeného příkladu je patrné, že pro vazbu mezi stavem prvku formuláře a daty aplikace je nutné vyhnout se v šabloně zápisu pomocí Latte, ale uvést XML elementy, které by mohly být zpracovány parserem. V příkladu je využita jedna z ukázkových knihoven komponent, kterým se věnuji v kapitole 4.10. Uživatel pak v metodě *formElement()* pro zpracování XML elementu *form* zavolá metodu *setRelation()* a předá jí tento element. Díky volání této metody pak parser zajistí vazbu mezi jednotlivými prvky formuláře a daty webové aplikace.

```
<?php
// libovolná třída definovaná v rámci Nette Frameworku
class userManager {
   public $username = "admin";
   public $password = "admin";
}

// metody presenteru pro vytvoření formuláře a předání dat šabloně
protected function createComponentSignInForm() {
   $form = new Xml\UI\Form($this);
   $form->addText('username', 'Jméno:');
   $form->addPassword('password', 'Heslo:');
   $form->addSubmit('login', 'Přihlásit se');
   $form->onSuccess[] = $this->signInFormSubmitted;
   return $form;
}
public function beforeRender() {
   $this->template->userManager = new userManager;
}
?>

<!-- šablona stránky s formulářem -->
<html:form name="signInForm">
   <html:input name="username" value="{$userManager->username}" />
   <html:input name="password" value="{$userManager->password}" />
   <html:input name="send" label="no" />
</html:form>
```

*Kód 4.5: Příklad vytvoření formuláře s transparentní vazbou na data webové aplikace*

## 4.9 Podpora technologie AJAX

Latte makra se v Nette Frameworku standardně zapisují mezi složené závorky, což je z pohledu XML dokumentu běžný text. Tento rámec má makra nejen pro podmínky, cykly a další, ale také pro technologii AJAX. Toto makro se jmenuje *snippet* a obsah, který má být pomocí AJAX aktualizován, se zapisuje mezi značky {*snippet obsah*} a {*/snippet*}.

V rámci rozšíření tohoto frameworku zajistím, aby bylo možné libovolný XML element transformovat na Latte makro. K této transformaci bude sloužit metoda *createMacro()*, která bude dostupná ve třídě *DOMHelper*. Té se předá textový řetězec, kterým se nahradí počáteční značka XML elementu, a řetězec, kterým se nahradí koncová značka XML elementu. O transformaci XML elementu na makro se pak postará parser. Uživatel mimo volání zmíněné metody ještě parseru vrátí návratovou hodnotu z třídy *DOMHelper*.

Tímto způsobem bude možné zajistit transformaci libovolného XML elementu na Latte makro. Protože syntaxi Latte maker je možné měnit, a to dynamicky v šablonách stránek, bude parser také rozpoznávat změny syntaxe a makra, na která budou XML elementy transformovány, uzavírat do správných závorek dle aktuální syntaxe Latte maker.

Příklad definice XML elementu pro podporu technologie AJAX je k dispozici v kódu 4.6. Stejným způsobem bude v rámci definic XML elementů možné transformovat jakýkoliv XML element na jakékoliv Latte makro.

```
<nette:snippet name="obsah">
   <!-- obsah stránky aktualizovaný pomocí AJAX -->
</nette:snippet>

<?php
class netteElements extends Xml\DOMHelper {
   public function snippetElement($element) {
      $value = $this->getAttributeValue($elem, "name");
      $this->createMacro($elem, "snippet ". $value, "/snippet");
      return $this::MACRO_CREATED;
   }
}
?>
```

*Kód 4.6: Definice XML elementu pro podporu technologie AJAX*

## 4.10 Ukázkové knihovny komponent

V rámci této diplomové práce vytvořím dva jmenné prostory, ve kterých definuji několik XML elementů demonstrujících funkčnost navrženého rozšíření. Prvním bude jmenný prostor zahrnutý v knihovně nazvané html. Díky ní bude v definicích šablon stránek v jazyce XML možné jednoduchým způsobem vytvořit formulář, jak je ukázáno v kódu 4.5, a to včetně transparentní vazby na data. Podobně snadno bude také možné vytvořit tabulku.

Druhý jmenný prostor bude obsažen v knihovně pojmenované *nette*. Tato knihovna bude obsahovat definice XML elementů, které budou transformovány na Latte makra, a bude pracovat způsobem, který je zachycen v kódu 4.6. Knihovna bude podporovat všechna makra, která jsou v Nette Frameworku k dispozici.

Ukázkovou aplikací demonstrující funkčnost navrženého řešení bude jednoduchý blog

s možností správy článků a komentářů a nejméně dvěma uživatelskými rolemi, který bude využívat obě dvě ukázkové knihovny. Veškeré formuláře navíc budou využívat transparentní vazbu formulářů na data.

Kód 4.7 zachycuje jednu z možností, jakým způsobem bude možné využít rozšíření tohoto rámce o definice šablon stránek v XML. Uživatel bude mít například možnost definovat XML element tak, aby jej v šablonách stránek mohl používat dvěma různými způsoby.

```
<!-- dva možné způsoby použití XML elementu -->
<h:nadpis>Moje stránky</h:nadpis>
<h:nadpis h="Moje stránky" />

<?php
// třída jmenného prostoru h
class hElements extends Xml\DOMHelper {

   public function nadpisElement($element) {
      $h = $this->getAttributeValue($element, "h");
      if ($h == "") {
         // ...
      }
      else {
         // ...
      }
   }
}
?>
```

*Kód 4.7: Ukázka možnosti dvou různých definic XML elementu*

## 4.11 Třída presenterů

Následující podkapitoly se budou věnovat zpracování definic šablon stránek v XML na straně rozšířeného Nette Frameworku.

O registraci filtru (parseru) pro zpracování XML elementů se bude starat třída presenterů *Xml\UI\Presenter*, která bude dědit z třídy *Nette\Application\UI\Presenter*. Dle dokumentace je registraci vlastního filtru pro zpracování šablon stránek doporučeno provádět právě ve třídě presenterů, a to přepsáním metody *templatePrepareFilters()*, která je definována ve třídě *Nette\Application\UI\Control*. Filtry jsou v Nette aplikovány v pořadí, ve kterém jsou registrovány, a proto bude nezbytné nejprve registrovat filtr pro zpracování XML elementů a teprve poté standardní Latte filtr.

Aby byla zachována koncepce Nette Frameworku, budu jako filtr šablon stránek v rámci rozšíření registrovat novou instanci třídy *Xml\Engine*, šablonovací systém Latte to provádí obdobně. Instance třídy pak bude volána jako funkce, čímž dojde k provedení magické metody *__invoke()*.

Metody *formatTemplateFiles()* a *formatLayoutTemplateFiles()* v rámci presenterů vyhledávají soubory šablon stránek, přičemž jde o soubory s příponami *.latte*. Přepsáním těchto metod bude v rámci rozšíření tohoto frameworku pro tvorbu webových aplikací možné vyhledávat soubory s příponami *.xml*.

## 4.12 Zpracování definic šablon stránek v XML

O zpracování šablon stránek se bude starat několik tříd, všechny budou umístěny v adresáři *vendor/others/Xml*. Třída *Xml\Engine* registrovaná jako filtr bude vytvářet instance nejméně dvou dalších tříd – *Xml\Parser* a *Xml\Compiler*. Tyto třídy budou plnit hlavní funkci při zpracování definic šablon stránek v jazyce XML.

Vstupem parseru bude šablona stránky v podobě, v jaké ji uvedl uživatel, a jeho výstupem bude modifikovaný DOM strom. Třída tedy nejprve vytvoří DOM strom daného XML dokumentu a poté bude volat uživatelem definované metody dle jmenných prostorů, do kterých konkrétní elementy patří. Podle těchto definic poté dojde k postupné modifikaci DOM stromu. Jakmile parser zpracuje všechny uzly stromu, vrátí upravený DOM strom. Bude nezbytné, aby si parser vytvářel a uchovával instance tříd, jejichž metody budou pro dané XML elementy volány. Rovněž bude zajišťovat rozpoznávání změn syntaxe Latte maker, vyhledávání vazeb formulářů na data a transformace XML elementů na Latte makra.

Compiler bude provádět tvorbu dokumentu na základě DOM stromu. Jeho vstupem tedy bude upravený DOM strom a výstupem šablona stránky v šablonovacím systému Latte, která bude dále zpracována, jak je v Nette Frameworku běžné. Compiler bude rovněž zajišťovat odstranění prefixů XML elementů, atributů deklarujících jmenné prostory a kořenového uzlu dokumentu.

Třídy pro zpracování šablon stránek v XML budou dále využívat další třídy, například půjde o třídu *Xml\XMLNamespace*. Každá její instance bude pro parser představovat jeden deklarovaný jmenný prostor.

# Kapitola 5

# Implementace

Tato kapitola popisuje postup a způsob implementace rozšíření zvoleného rámce, které bylo navrženo v kapitole 4, a věnuje se implementačním problémům, které bylo třeba řešit. Její úvod je věnován možnosti konfigurace navrženého rozšíření, dále se zabývám implementací parseru, reprezentací jmenných prostorů, implementací vazby prvků uživatelského rozhraní na data a dalším. Návrh rozšíření byl v průběhu implementace upravován, což je odůvodněno v kapitole 6.

## 5.1 Konfigurace rozšíření

Základní třídou rozšíření zvoleného rámce o definici šablon stránek v jazyce XML je třída *XML\Engine*. Ta obsahuje několik statických metod, jejichž voláním je možné provádět konfiguraci navrženého rozšíření. Volání probíhá typicky v souboru *bootstrap.php*.

Nejdůležitější metodou je metoda *register()*, po jejímž volání dojde k registraci parseru pro zpracování XML šablon, jak je popsáno v kapitole 5.2. Této metodě je třeba předat konfigurátor, tedy instanci třídy *Nette\Configurator*, která se vytváří v souboru *bootstrap.php*. Je třeba tak učinit proto, aby rozšíření mohlo využít RobotLoader pro vyhledávání tříd, čemuž se více věnuji v kapitole 5.6.

Další statickou metodou, kterou je možné volat, je metoda *cleanCache()*. Ta se stará o mazání paměti cache při každém požadavku, a to za účelem vývoje a testování nejen navrženého rozšíření, ale především XML šablon, což bude provádět uživatel. Paměť cache není mazána ihned při volání dané metody, ale až později, aby metodu bylo možné využít i v případě využití transparentní vazby prvků uživatelského rozhraní na data, která je ukládána právě do paměti cache. Způsobu mazání paměti cache se věnuji v kapitole 5.9.

Poslední statickou metodou, kterou může uživatel využít, je metoda *showComments()*. Ta zajišťuje zachování veškerých komentářů v XML šablonách. Ve výchozím stavu jsou komentáře parserem odstraněny, což uživateli umožňuje si XML šablony řádně okomentovat, aniž by se komentáře zobrazovaly ve výsledném HTML kódu.

Třída *Xml\Engine* disponuje i dalšími statickými metodami, které jsou volány v rámci rozšíření. Jde například o metodu pro zjištění, zda má být registrován parser pro zpracování XML šablon stránek nebo zda má být smazána paměť cache.

## 5.2 Registrace parseru pro zpracování XML šablon

Každý presenter musí dědit z třídy *Xml\UI\Presenter*, jak bylo navrženo v kapitole 4. Tato třída se poté stará o registraci parseru pro zpracování XML šablon. Ta probíhá v metodě *templatePrepareFilters()*, která přepisuje metodu zděděnou z rodiče. Protože v Nette Frameworku jsou filtry šablon stránek aplikovány v pořadí, v jakém jsou registrovány, bylo nezbytné nejprve registrovat parser pro zpracování XML a teprve poté Latte parser. Parser XML šablon je registrován pouze v případě, že je rozšíření aktivované, tedy že v souboru *bootstrap.php* byla volána metoda *register()*.

Kromě registrace parseru se třída *Xml\UI\Presenter* v případě, že je rozšíření aktivované, stará také o to, aby definice šablon stránek nebyly očekávány v souborech s příponou *.latte*, ale příponou *.xml*. To je provedeno přepsáním metod *formatLayoutTemplateFiles()* a *formatTemplateFiles()*. Pokud rozšíření není aktivované, metody volají metody definované v rodiči.

Díky kontrole, zda je či není rozšíření aktivované, je tedy možné v souboru *bootstrap.php* přepínat mezi šablonami definovanými v Latte a šablonami definovanými v jazyce XML, aniž by uživatel musel provádět jiné změny kódu.

## 5.3 Parser s rozpoznáváním změn syntaxe maker

Třída *Xml\Engine* vytváří instanci třídy *Xml\Parser* a *Xml\Compiler*, přičemž první ze zmíněných tříd dostane na vstup XML šablonu jako řetězec, ze kterého vytvoří DOM strom, rekurzivně jej zpracuje a zavolá uživatelem definované metody pro modifikaci jednotlivých uzlů, a druhá ze zmíněných tříd převede modifikovaný DOM strom zpět na řetězec.

Hlavní funkcionalitu třídy *Xml\Parser* zajišťuje metoda process(). Ta je volána rekurzivně pro každý uzel DOM stromu. Metoda nejprve zjistí, do jakého jmenného prostoru patří daný element, a vytvoří si instanci příslušné, uživatelem definované třídy. Tomu se více věnuji v kapitole 5.6. Poté z dané třídy zavolá metodu dle názvu elementu a následně kontroluje návratovou hodnotu metody.

- Pokud uživatel žádnou hodnotu nevrátil, parser zkontroluje, zda daný element v DOM stromu stále existuje. Uživatel totiž mohl uzel odstranit nebo například zavolat metodu pro přejmenování elementu, která ve skutečnosti vytváří nový uzel, a zapomenout o tom dát parseru vědět návratovou hodnotou. Pokud uzel neexistuje, parser vyhodí výjimku a uživatele upozorní, ve které metodě došlo k chybě. Kontrola, zda daný uzel DOM stromu existuje, se provádí rekurzivním průchodem celého stromu a porovnáním každého uzlu s aktuálně zpracovávaným uzlem, a to pomocí metody *DOMNode::isSameNode()*.
- Vrátí-li uživatelem definovaná metoda hodnotu *FALSE*, parser opět vyhodí výjimku a uživatele upozorní na nesprávné použití XML elementu v šabloně. Tento mechanismus tedy uživateli dovoluje kontrolovat správné používání definovaných XML elementů v šablonách. V kódu 5.1 je zachycen příklad využití této návratové hodnoty.
- Pokud uživatel vrátí konstantu *NODE_REMOVED* nebo *IGNORE_CHILDREN* z třídy *Xml\DOMHelper*, parser přeskočí daný uzel. Uživatel jej v obslužné metodě buď odstranil, nebo si nepřeje zpracovávat jeho potomky.
- Vrátí-li uživatel konstantu *MACRO_CREATED* z třídy *Xml\DOMHelper*, parser z daného XML elementu vytvoří Latte makro. Vrácení této konstanty musí předcházet vo-

lání metody *createMacro()*. Transformací XML elementu na Latte makro se zabývám v kapitole 5.7.

- Návratová hodnota *FORM_RELATION* z třídy *Xml\DOMHelper* zajistí, že parser pro každý *input* element vyhledá a uloží vazbu tohoto elementu na data. Tomu se více věnuji v kapitole 5.8.
- Pokud uživatel vrátí jinou hodnotu, parser ji považuje za nový uzel DOM stromu, kterým nahradí aktuálně zpracovávaný uzel.

```
public function h1Element($elem) {
   if ($this->hasAttribute($elem, "name")) {
      ...
      return;
   }

   return FALSE;
}

<!-- nesprávné použití, parser vyhodí výjimku -->
<h1></h1>
<!-- správné použití -->
<h1 name="nazev"></h1>
```

*Kód 5.1: Ukázka využití návratové hodnoty FALSE*

Po kontrole návratových hodnot následuje procházení potomků daného uzlu, přičemž pro každý z nich je rekurzivně volána metoda *process()*.

Jestliže daný uzel DOM stromu není element, ale uzel typu *XML_COMMENT_NODE*, parser voláním statické metody *isShowComments()* z třídy *Xml\Engine* zjistí, zda mají být komentáře v XML šablonách zachovány či nikoliv a podle toho daný uzel zachová či odstraní.

Parser dále rozpoznává změny syntaxe Latte maker, které je v Nette Frameworku možné provádět dynamicky v šablonách, jak je uvedeno v dokumentaci. Toto rozpoznávání parser využívá při transformaci XML elementů na Latte makra, kdy makro obalí do závorek dle aktuální syntaxe, a také při vyhledávání vazeb prvků uživatelského rozhraní na data. Za tímto účelem vyhledává parser změny syntaxe v každém textovém uzlu DOM stromu a také eviduje změny pomocí atributu *n:syntax* u libovolného elementu, kterým je rovněž možné syntaxi změnit.

Jakmile parser zpracuje celý dokument, vrátí modifikovaný DOM strom.

## 5.4 Převod DOM stromu na řetězec

O převod DOM stromu na řetězec se stará třída *Xml\Compiler*. Ta zajišťuje také odstranění kořenového elementu dokumentu, neboť v XML musí existovat jediný takový element a uživatel ve většině případů právě jej využije pro deklaraci jmenných prostorů, ale také odstranění deklarací u ostatních elementů a odstranění prefixů elementů. Ty ve výsledném HTML kódu nemají využití a jsou nežádoucí. Třída odstranění provádí tak, že každý XML

element v DOM stromu nahradí elementem novým, jehož název neobsahuje prefix, a provede kopii všech atributů a potomků. Jelikož deklarace jmenných prostorů ve skutečnosti nejsou reprezentovány atributy, nezkopírují se a tím dojde k jejich odstranění.

Samotný převod DOM stromu na řetězec provádí metoda *DOMDocument::saveHTML()*. Třída *Xml\Compiler* dále zajišťuje správné kódování českých znaků a HTML entit ve výsledném řetězci, k čemuž využívá funkce *html_entity_decode()* a *rawurldecode()*.

## 5.5 Třída DOMHelper

Každá uživatelem definovaná třída, ze které jsou volány metody pro modifikaci daného XML elementu dle jmenného prostoru a jména elementu, by měla dědit z tříd *Xml\DOMHelper*, jak bylo uvedeno v kapitole 4. Tato třída poskytuje konstanty, které uživatel může využít zejména jako návratové hodnoty, a metody, které výrazným způsobem usnadňují modifikaci DOM stromu.

Kromě konstant, které byly zmíněny v kapitole 5.3, nabízí třída *Xml\DOMHelper* konstanty *NETTE_NAMESPACE_URI* a *NETTE_NAMESPACE*, které uživatel může využít při deklaraci jmenného prostoru s prefixem *n*. Deklaraci jmenného prostoru, který uživatel nedeklaroval zápisem přímo v XML šabloně stránky, lze provést pomocí metod *isNamespaceDeclared()* a *declareNamespace()*.

Mezi další metody, které lze využít a které standardní DOM rozhraní neposkytuje, patří například metoda *renameElement()* pro přejmenování elementu nebo metoda pro přesun potomků daného elementu k jinému elementu nazvaná *moveChildren()*. Metody třídy *Xml\DOMHelper* plně podporují práci se jmennými prostory, je možné získat hodnotu atributu ležícího v nějakém jmenném prostoru, jednoduchým způsobem přidat XML elementu potomka a další.

Kód 5.2 ukazuje příklad přidání potomka danému elementu.

```
public function bodyElement($elem) {
   $before = $elem->lastChild;
   $this->addElement($elem, "p", ["n:attr" => "hodnota"], "Text", $before);
}
```

*Kód 5.2: Ukázka přidání potomka XML elementu pomocí třídy Xml\DOMHelper*

## 5.6 Reprezentace jmenných prostorů

Pro každý XML element je volána uživatelem definovaná metoda dle názvu elementu z třídy dle jmenného prostoru elementu. Je tedy třeba, aby parser k daným jmenným prostorům vyhledával odpovídající třídy a vytvářel a spravoval jejich instance.

Třída, z níž má být volána metoda pro daný element, je vyhledávána dle URI jmenného prostoru, do kterého element patří, jak bylo navrženo v kapitole 4. URI je uvedena jako veřejná vlastnost uživatelem definované třídy. Protože parser nezná název třídy, danou třídu vyhledává mezi všemi třídami existujícími v adresáři *app*. Třída tedy může mít libovolný název a patřit do libovolného jmenného prostoru. Parser k vyhledání třídy využívá konfigurátor, který v Nette Frameworku umožňuje vytvořit a využít vlastní instanci RobotLoaderu. Ten umí načíst všechny třídy v daném umístění. Parser pak reflektováním tříd kontroluje existenci veřejné vlastnosti *xmlns* a případně vytváří instanci třídy a porov-

nává hodnotu této vlastnosti s danou URI. Jestliže je nalezeno více tříd s danou URI, je vyhozena výjimka, neboť URI je jednoznačnou identifikací daného jmenného prostoru.

Jmenný prostor je reprezentován třídou *Xml\XMLNamespace*, přičemž parser si uchovává pole objektů této třídy. V případě, že pro jmenný prostor s danou URI nemá vytvořenu třídu, vytvoří ji a URI jí předá. Třída podle této URI vyhledá v adresáři *app* uživatelem definovanou třídu. Pokud parser později narazí na XML element, který patří do jmenného prostoru definovaného stejnou URI, instanci znovu nevytváří a třídu nevyhledává, protože už ji má uloženu v poli objektů reprezentujících jmenné prostory.

## 5.7 Transformace XML elementů na Latte makra

Uživatel má možnost jakýkoliv XML element transformovat na Latte makro, a to způsobem, který byl popsán v kapitole 4. Nejprve je nutné zavolat metodu *createMacro()* z třídy *Xml\DOMHelper* a předat jí daný element. Metoda transformaci neprovádí ihned, ale pouze k objektu elementu přidá dvě nové vlastnosti – na jaký text se má transformovat počáteční a koncová značka. O samotnou transformaci se stará parser, kterému uživatel vrátí konstantu *MACRO_CREATED*. Transformaci XML elementu na makro není možné provést ihned, neboť uvnitř elementu může dojít ke změně syntaxe Latte maker a koncová značka by byla obalena do nesprávných závorek. Parser proto provede transformaci počáteční značky a teprve až zpracuje všechny potomky a zná aktuální syntaxi Latte maker, transformuje i koncovou značku.

Transformaci XML elementů na Latte makra využívá knihovna nette, která pro každé makro definuje XML element, který lze v XML šablonách využít a který bude transformován na makro. Metody v této knihovně také využívají návratovou hodnotu *FALSE*, díky které bude uživatel upozorněn výjimkou, pokud XML element v šabloně použije nesprávně.

## 5.8 Vazba mezi stavem prvku uživatelského rozhraní a daty aplikace

Vazbu mezi stavem prvku uživatelského rozhraní a daty webové aplikace je možné využít tak, jak bylo navrženo. Třída *Xml\UI\Form*, jejíž instanci uživatel při tvorbě formuláře vytváří, dědí z třídy *Nette\Application\UI\Form*. V konstruktoru je registrována metoda *xmlFormApplyRelation()*, která se zavolá jako první po úspěšném odeslání formuláře a která aplikuje vazbu na data, tedy uživatelem vyplněné hodnoty uloží do příslušných proměnných. V metodě presenteru, kterou si uživatel registruje jako metodu, která se má po odeslání formuláře zavolat, pak není třeba získávat z formuláře vyplněné hodnoty. Tyto hodnoty jsou již uloženy v příslušných proměnných, na které byly navázány.

O vyhledávání vazeb prvků na data se stará parser. Tomu uživatelem definovaná metoda vrátí konstantu *FORM_RELATION*, které předchází volání metody *createRelation()* z třídy *Xml\DOMHelper*. Vazba konkrétního formuláře na data je reprezentována třídou *Xml\UI\Relation*, které parser předá kořenový uzel formuláře a úroveň zanoření tohoto elementu. V každé nižší úrovni, tedy ve všech potomcích pak parser rozpoznává XML elementy s názvem *input* a v jejich atributech *value* vyhledává vazby na data. Parser navíc počítá se změnami syntaxe Latte maker a poradí si i se složitějšími případy, kdy se jeden element váže na více proměnných. Jednotlivé vazby jsou ukládány do pole nazvaného *relations* v třídě *Xml\UI\Relation*.

Jakmile parser narazí na koncovou značku daného formuláře, uloží nalezené vazby do paměti cache. Nette Framework umožňuje snadnou práci s pamětí cache. Ukládání vazeb je nutné proto, že tento rámec každou šablonu zpracovává pouze jednou a poté už ji pouze načítá z cache. Spolu s ní jsou tedy načítány i uložené vazby prvků uživatelského rozhraní na data aplikace.

O aplikaci vazeb na data se po úspěšném odeslání formuláře stará metoda nazvaná *xmlFormApplyRelation()* z třídy *Xml\UI\Presenter*. Tato metoda na základě jména daného formuláře (jména komponenty) načte z paměti cache uložené vazby a každou z nich aplikuje, tedy uloží vyplněnou hodnotu do dané proměnné. Možnost aplikace vazby na metodu, tedy uložení hodnoty do proměnné vracené metodou (např. *$trida->metoda()*) není podporována z důvodu složité implementace a především nulového praktického využití.

## 5.9 Mazání paměti cache

Za účelem vývoje a testování XML šablon je ve třídě *Xml\Engine* k dispozici metoda nazvaná *cleanCache()*, jejímž voláním ze souboru *bootstrap.php* uživatel může mazat cache při každém požadavku. XML šablony tak budou zpracovávány při každém požadavku, což usnadní jejich vývoj. Protože vazby prvků uživatelského rozhraní na data webové aplikace se ukládají do paměti cache, metoda mazání neprovádí ihned, pouze zajistí, aby cache byla smazána později. Paměť se pak vymaže v metodě *processSignal()* ve třídě *Xml\UI\Presenter*, která je volána až poté, co jsou vazby na data uložené v cache aplikovány. Tento přístup umožňuje snadný vývoj XML šablon a současně využívání vazeb prvků uživatelského rozhraní na data aplikace.

## 5.10 Výjimky

Výjimky, které rozšíření zvoleného rámce o definici šablon stránek v jazyce XML vyhazuje, jsou definovány v souboru *Xml/exceptions.php*. Třída *Xml\Engine* vyhazuje výjimku v případě, že metodě *register()* není předán konfigurátor. Parser vyhazuje výjimku, pokud uživatelská metoda měla vrátit hodnotu a nevrátila ji, tedy například v případě, že uživatel nahradil uzel DOM stromu za uzel nový a tento nový uzel parseru nevrátil. Parser také disponuje detekcí zacyklení a vyhodí výjimku, pokud jeden a ten stejný uzel zpracovává podruhé.

Třída *Xml\XMLNamespace* vyhazuje výjimku, pokud nenajde uživatelskou třídu reprezentující daný jmenný prostor nebo pokud tříd s danou URI nalezne více. Pokud parser při vyhledávání vazby prvku uživatelského rozhraní na data aplikace narazí na volání metody, rovněž vyhodí výjimku, stejně tak provede v případě, že některá z uživatelských metod vrátí hodnotu *FALSE*.

# Kapitola 6

# Testování

Následující kapitola se věnuje testování navrženého rozšíření zvoleného rámce pro tvorbu webových aplikací v jazyce PHP. V jejím úvodu se věnuji prostředí, ve kterém bylo rozšíření o definici šablon stránek v XML vyvíjeno a testováno, rovněž se zabývám možnostmi testování, které při vývoji aplikací v Nette Frameworku nabízí sám tento rámec. V další části se věnuji testovacím XML dokumentům a následně popisuji způsob, jakým jsem při vývoji rozšíření prováděl debugování PHP kódu. V poslední části kapitoly se zabývám tvorbou testů pomocí testovacího frameworku Nette Tester a nedostatky, které jsem v průběhu testování odhalil.

## 6.1 Vývojové a testovací prostředí

Vývoj a testování navrženého rozšíření Nette Frameworku o definici šablon stránek v PHP s možností implementace vlastních rozšiřujících značek v PHP byly prováděny na systému Windows v prostředí NetBeans 7.4[1], později jsem využil verzi 8.0 s podporou PHP 5.5 a frameworku Nette Tester. K běhu webové aplikace na lokálním počítači jsem využil balík xampp[2], který obsahuje aktuální verzi PHP, Apache, MySQL, phpMyAdmin a další nástroje.

Jako výchozí webový prohlížeč jsem při vývoji rozšíření zvoleného rámce používal prohlížeč Firefox v aktuální verzi, avšak volba prohlížeče neměla na implementaci samotného rozšíření zásadní vliv. Ukázková webová aplikace pak byla testována také v prohlížeči Internet Explorer, Google Chrome a Opera.

V prohlížeči Firefox lze využít nástroj Firebug[3], který v dnešní době patří mezi populární vývojové nástroje tohoto prohlížeče. Firebug umožňuje sledování HTML a CSS kódu a také kódu psaného v jazyce JavaScript. Podporu PHP do tohoto rozšíření dodává doplněk FireLogger[4], díky kterému je možné vypisovat do konzole obsahy proměnných a podobně. Tento nástroj jsem v několika případech využil pro rychlé ladění, Nette Framework navíc disponuje třídou, která ladění usnadňuje. V několika případech (zejména pro zjištění řetězce, který vrací parser XML šablon stránek) jsem také využil možnost uložení obsahu

[1] https://netbeans.org/
[2] https://www.apachefriends.org/index.html
[3] https://getfirebug.com/
[4] http://firelogger.binaryage.com/

proměnné do souboru, a to z důvodu chybného zobrazování znaků s diakritikou v nástroji Firebug.

Za účelem vývoje a testování jsem dále vypnul paměť cache, aby Nette Framework každou šablonu zpracovával při každém požadavku. Provedl jsem tak voláním vlastní statické metody v souboru *bootstrap.php*, která odstraňuje veškerý obsah cache.

## 6.2 Knihovna Nette\Diagnostics\Debugger

*Nette\Diagnostics\Debugger* je třída dostupná v rámci Nette Frameworku, která usnadňuje odhalování chyb, výpis proměnných, měření času, zajišťuje logování chyb a další. Třída je známá pod názvem Tracy a provádí vizualizaci výjimek. V případě chyby v PHP kódu se tedy uživateli zobrazí stránka, která napoví, kde se chyba nachází a jak ji opravit. Mimo to dodává třída v ladícím módu také plovoucí panel nazvaný Debugger Bar. Více o třídě *Nette\Diagnostics\Debugger* se lze dočíst v dokumentaci. [35]

Protože jsem vyvíjel rozšíření pro Nette Framework a měl zapnutý ladící mód, každou chybu v PHP kódu mi tento rámec zobrazoval pomocí Tracy. Také jsem využíval další možnosti, které tato knihovna nabízí. Zejména se jednalo o jednoduchou možnost vypisování obsahu proměnných do nástroje FireLogger zmíněného v kapitole 6.1. To je možné provádět pomocí statické metody *Debugger::fireLog()*, případně lze využít metodu *Debugger::dump()*. Rovněž lze provádět konfiguraci, a to pomocí změn hodnot statických vlastností, ideálně v souboru *bootstrap.php*. Přiřazením hodnoty vlastnosti *Debugger::$maxLen* lze například změnit maximální délku vypisovaného řetězce, *Debugger::$maxDepth* pak nastaví maximální hloubku vypisovaného objektu.

## 6.3 Testovací XML dokumenty

Jako součást rozšíření jsem vytvořil knihovnu nazvanou html, díky které je možné jednoduchým zápisem v jazyce XML vytvořit formulář (včetně transparentní vazby na data) a tabulku, a knihovnu nette, která definuje XML elementy pro všechna makra, která lze v Nette Frameworku využít. Obě knihovny jsem testoval částmi XML, které jsem vkládal do šablony stránky a nechával je zpracovávat parserem. Později jsem pro každou knihovnu vytvořil samostatný XML dokument, přičemž každý z nich testuje funkcionalitu odpovídající knihovny.

Dokument *tests/html.xml* testuje knihovnu html. Je-li tento XML dokument vložen na vstup parseru pro zpracování šablon stránek v XML, výstupem parseru je HTML kód odpovídající kódu, který je v XML dokumentu zakomentovaný a slouží tak jako referenční výstup.

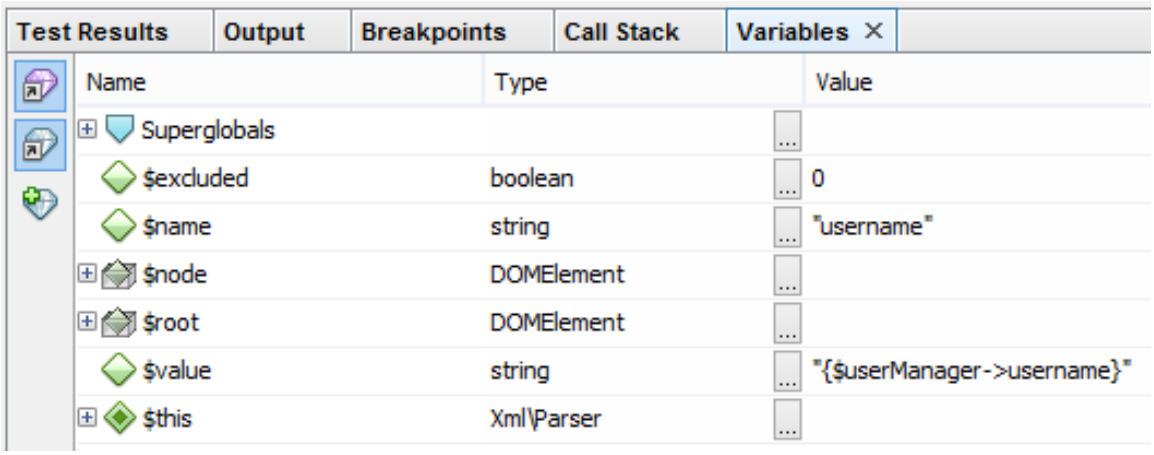
Dokument *tests/nette.xml* pak testuje knihovnu nette. Tento dokument obsahuje XML elementy pro všechna makra, která jsou v Nette Frameworku k dispozici, a zakomentovaná makra pro kontrolu správnosti transformace.

Testovací XML dokumenty neslouží jako automatizované testy, ale jako rychlá kontrola funkčnosti obou knihoven a zároveň ukázka správného použití těchto knihoven.

## 6.4 Xdebug

Pro vývoj a testování implementovaného rozšíření pro Nette Framework jsem využil rozšíření pro PHP nazvané Xdebug[5]. Díky němu je možné provádět debugování PHP kódu. Toto rozšíření jsem nainstaloval a zprovoznil na lokálním počítači a využíval jsem je v prostředí NetBeans. Při vývoji jsem tak měl možnost provádět zejména krokování PHP kódu a sledování obsahu proměnných. K debugování jsem využíval interní prohlížeč, který je v NetBeans k dispozici.

Obrázek 6.1 zachycuje ukázku debugování PHP kódu v prostředí NetBeans.




*Obrázek 6.1: Debugování v prostředí NetBeans pomocí Xdebug*

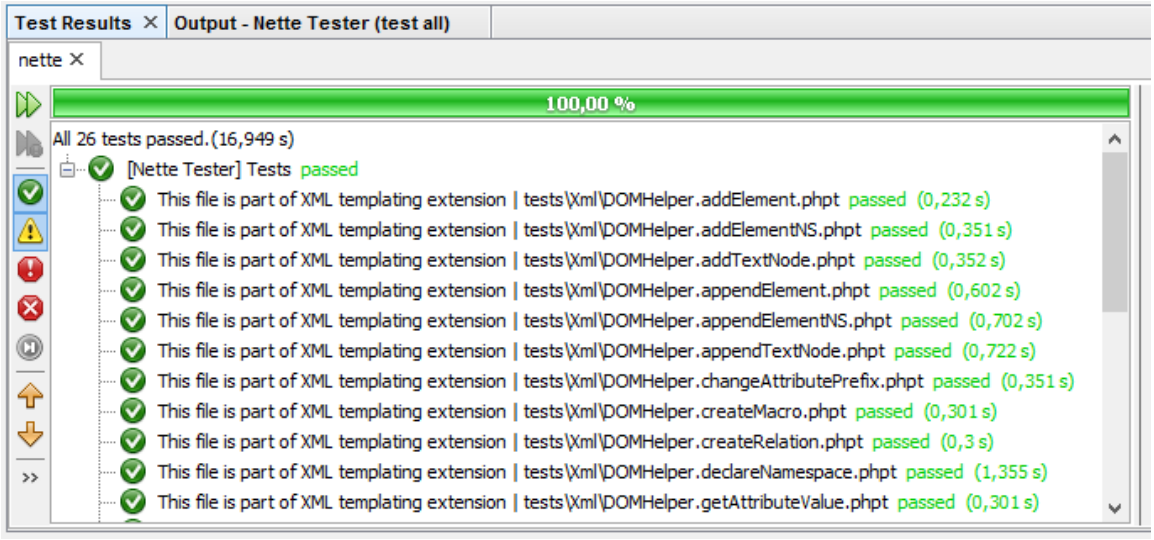
## 6.5 Nette Tester

Nette Tester je podobně jako PHPUnit[6] testovací framework, který slouží k automatizovanému spouštění testů. V rámci tohoto frameworku je k dispozici třída *Assert*, která v případě selhání daného testu čitelně vypíše, která metoda selhala, jakou hodnotu měla vrátit a jakou hodnotu vrátila. Více informací o možnostech použití Nette Testeru včetně postupu instalace je možné nalézt v dokumentaci [36].

Nette Tester jsem využil pro testování navrženého rozšíření. Testy jsou umístěny v adresáři *tests* a dále podadresářích dle jmenných prostorů tříd, do kterých patří testované třídy. V rámci rozšíření jsem vytvořil testy pro všechny metody třídy *Xml\DOMHelper*, které je možné nalézt v adresáři *tests/Xml*. Každý soubor je pojmenován dle názvu třídy a metody, kterou testuje, a každý test má příponu *.phpt*. V rámci jednoho testu dochází k testování příslušné metody zpravidla několikrát, a to s různými vstupními parametry. Testy byly vytvářeny tak, aby v případě odhalení chyby byly snadno rozšiřitelné.

Na obrázku 6.2 je zachyceno testování navrženého rozšíření pomocí Nette Testeru, a to v prostředí NetBeans 8.0, které disponuje podporou tohoto testovacího rámce.

[5] http://xdebug.org/
[6] http://phpunit.de/




*Obrázek 6.2: Testování pomocí frameworku Nette Tester v prostředí NetBeans 8.0*

## 6.6 Odhalené nedostatky

Zmíněné ladící nástroje jsem využíval v průběhu celého vývoje, pomohly mi tedy odhalit a opravit nedostatky mého rozšíření. Třída *Nette\Diagnostics\Debugger* mě upozorňovala na výjimky a pomohla mi také s vyhazováním vlastních výjimek, a to zejména v třídě *Xml\Parser*. Při programování jsem využíval především možnost debugování PHP kódu v prostředí NetBeans.

V průběhu vývoje došlo k několika drobným úpravám návrhu rozšíření. Tyto změny byly provedeny nikoliv z důvodu, že by implementaci dle původního návrhu nebylo možné provést, ale proto, že se ukázalo, že s provedením menších změn v návrhu se používání rozšíření Nette Frameworku pro uživatele výrazně zjednoduší.

Testovací XML dokumenty mi pomohly ověřit správnou funkčnost navržených knihoven a spolu s testy psanými pro framework Nette Tester zvýšily důvěru v provádění změn. Testy vytvořené pro třídu *Xml\DOMHelper* odhalily dva nedostatky, které jsem s pomocí těchto testů opravil. Obdobným způsobem by bylo možné napsat testy také pro metody dalších tříd implementovaného rozšíření.

# Kapitola 7

# Závěr

V rámci diplomové práce jsem provedl analýzu existujících rámců pro tvorbu webových aplikací v jazyce PHP, přičemž jsem se zaměřil na nejčastěji používané rámce. Analýza se týkala frameworků CakePHP, Nette Framework, Symfony a Zend Framework. Při analýze jsem kladl důraz především na systém definice a využití šablon stránek. Na základě analýzy jsem zvolil Nette Framework jako vhodný rámec pro tuto diplomovou práci. Dále jsem se seznámil s jazykem XML včetně možnosti využití jmenných prostorů a během studia technologií jsem prostudoval také možnosti zpracování XML dokumentů v jazyce PHP.

V návrhu rozšíření zvoleného rámce o definici šablon stránek v XML jsem odůvodnil volbu daného rámce a také způsob zpracování XML dokumentů. Následně jsem navrhl způsob instalace rozšíření a jeho využití z pohledu uživatele, ale také funkcionalitu rozšíření tak, aby bylo možné rozšíření na základě návrhu implementovat. V návrhu jsem uvažoval možnost využití Latte maker a vazbu mezi stavem prvku uživatelského rozhraní a daty webové aplikace. Rovněž jsem navrhl ukázkové knihovny komponent a ukázkovou webovou aplikaci. Poté jsem na základě návrhu rozšíření implementoval, přičemž došlo k několika změnám návrhu, a to z důvodu lepší použitelnosti rozšíření. Během vývoje jsem prováděl debugování PHP kódu a následně jsem vytvořil testy pro testovací framework Nette Tester.

Funkčnost a použitelnost implementovaného rozšíření zvoleného rámce demonstruje ukázková webová aplikace. Rozšíření umožňuje definovat šablony stránek v jazyce XML, přičemž lze využít veškeré možnosti, které Nette Framework nabízí. Rozšíření navíc rozpoznává změny syntaxe Latte maker a díky jedné z knihoven je možné v XML šablonách stránek využít veškerá makra. Během dalšího vývoje rozšíření se zaměřím zejména na vlastní převod DOM stromu na řetězec, díky kterému bude možné zajistit lepší formátování výstupu, a podporu vazeb mezi dalšími prvky uživatelského rozhraní (*textarea*, elementem *input* typu *checkbox* atd.) a daty webové aplikace. Rozšíření bude rovněž nabídnuto koncovým uživatelům.

# Literatura


[1] Borek Bernard: *Prezentační vzory z rodiny MVC* [online]. Dostupné z: http://www.zdrojak.cz/clanky/prezentacni-vzory-zrodiny-mvc/, Poslední modifikace: 2009-05-11 [cit. 2013-11-03].

[2] Cake Software Foundation, Inc.: *Cookbook 2.x* [online]. Dostupné z: http://book.cakephp.org/2.0/en/index.html, [cit. 2013-11-03].

[3] Cake Software Foundation, Inc.: *CakePHP Folder Structure* [online]. Dostupné z: http://book.cakephp.org/2.0/en/getting-started/cakephp-folder-structure.html, [cit. 2013-11-04].

[4] Cake Software Foundation, Inc.: *Cookbook 2.x - Helpers* [online]. Dostupné z: http://book.cakephp.org/2.0/en/views/helpers.html, [cit. 2013-11-04].

[5] Cake Software Foundation, Inc.: *Cookbook 2.x - Views* [online]. Dostupné z: http://book.cakephp.org/2.0/en/views.html, [cit. 2013-11-04].

[6] David Grudl: *Nette Framework: MVC & MVP* [online]. Dostupné z: http://www.zdrojak.cz/clanky/nette-framework-mvc-mvp/, Poslední modifikace: 2009-03-24 [cit. 2013-11-02].

[7] Fabien Potencier: *Creating and using Templates - Symfony* [online]. Dostupné z: http://symfony.com/doc/current/book/templating.html, [cit. 2013-11-08].

[8] Fabien Potencier: *The Big Picture - Symfony* [online]. Dostupné z: http://symfony.com/doc/current/quick_tour/the_big_picture.html, [cit. 2013-11-08].

[9] Fabien Potencier: *The Book - Symfony* [online]. Dostupné z: http://symfony.com/doc/current/book/index.html, [cit. 2013-11-08].

[10] Garrett, J. J.: *Ajax: A New Approach to Web Applications* [online]. Dostupné z: http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications, Poslední modifikace: 2005-02-18 [cit. 2013-11-08].

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

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

[13] Hruška, T., Burget, R.: *Internetové aplikace (WAP) IV. část Programování serveru (PHP) (Studijní opora).* Brno, 2007.

[14] Hruška, T., Kroulík, J., Techet, J.: *Internetové aplikace (WAP) III. část XML, XML schémata, XPath, XSLT (Studijní opora)*. Brno, 2007.

[15] Hruška, T., Máčel, L., Kužela, A.: *Internetové aplikace (WAP) V. část AJAX (Studijní opora)*. Brno, 2007.

[16] Jakub Havel: *XML pro web aneb od teorie k praxi, 8.díl - jmenné prostory a XHTML* [online]. Dostupné z: http://www.zive.cz/clanky/xml-pro-web-aneb-od-teorie-k-praxi-8dil---jmenne-prostory-a-xhtml/sc-3-a-110128/default.aspx, Poslední modifikace: 2003-01-24 [cit. 2013-11-11].

[17] Jaroslav Jakoubě: *ORM test PHP frameworků – Yii, Zend Framework* [online]. Dostupné z: http://www.zdrojak.cz/clanky/orm-test-php-frameworku-yii-zend-framework/, Poslední modifikace: 2013-08-21 [cit. 2013-11-08].

[18] Jiří Koutný: *Symfony2, těší mě!* [online]. Dostupné z: http://www.zdrojak.cz/clanky/symfony2-tesi-me/, Poslední modifikace: 2013-06-26 [cit. 2013-11-08].

[19] Kolektiv autorů: *Jak dlouho existuje Nette?* [online]. Dostupné z: http://forum.nette.org/cs/10890-jak-dlouho-existuje-nette, Poslední modifikace: 2012-09-27 [cit. 2013-11-04].

[20] Kolektiv autorů: *Model-view-controller* [online]. Dostupné z: http://cs.wikipedia.org/wiki/Model-view-controller, Poslední modifikace: 2013-04-05 [cit. 2013-11-03].

[21] Kolektiv autorů: *HyperText Markup Language* [online]. Dostupné z: http://cs.wikipedia.org/wiki/HyperText_Markup_Language, Poslední modifikace: 2013-08-11 [cit. 2013-11-08].

[22] Kolektiv autorů: *Hypertext* [online]. Dostupné z: http://cs.wikipedia.org/wiki/Hypertext, Poslední modifikace: 2013-10-08 [cit. 2013-11-08].

[23] Kolektiv autorů: *Zend Framework* [online]. Dostupné z: http://cs.wikipedia.org/wiki/Zend_Framework, Poslední modifikace: 2013-10-10 [cit. 2013-11-08].

[24] Kolektiv autorů: *CakePHP* [online]. Dostupné z: http://en.wikipedia.org/wiki/CakePHP, Poslední modifikace: 2013-10-30 [cit. 2013-11-03].

[25] Kolektiv autorů: *PHP* [online]. Dostupné z: http://cs.wikipedia.org/wiki/PHP, Poslední modifikace: 2013-11-06 [cit. 2013-11-11].

[26] Kolektiv autorů: *Hypertext Transfer Protocol – HTTP/1.1 (RFC2616)* [online]. Dostupné z: http://tools.ietf.org/html/rfc2616, Poslední modifikace: červen 1999 [cit. 2013-11-08].

[27] Kosek, J.: *PHP a XML*. Grada Publishing, a.s., 2009, ISBN: 978-80-247-1116-4.

[28] Lavin, P.: *PHP – objektově orientované*. Grada Publishing, a.s., 2009, ISBN: 1-59327-077-1.

[29] Mozilla Developer Network: *JavaScript Language Resources* [online]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Language_Resources, Poslední modifikace: 2013-07-17 [cit. 2013-11-11].

[30] Nette Foundation: *MVC aplikace & presentery* [online]. Dostupné z: http://doc.nette.org/cs/presenters, [cit. 2013-11-03].

[31] Nette Foundation: *Presentery a šablony* [online]. Dostupné z: http://doc.nette.org/cs/book/presenter, [cit. 2013-11-04].

[32] Nette Foundation: *Stažení a instalace* [online]. Dostupné z: http://doc.nette.org/cs/installation, [cit. 2013-11-04].

[33] Nette Foundation: *Výchozí helpery šablon* [online]. Dostupné z: http://doc.nette.org/cs/default-helpers, [cit. 2013-11-04].

[34] Nette Foundation: *Výchozí Latte makra* [online]. Dostupné z: http://doc.nette.org/cs/default-macros, [cit. 2013-11-04].

[35] Nette Foundation: *Debugování a zpracování chyb* [online]. Dostupné z: http://doc.nette.org/cs/2.1/debugging, [cit. 2014-04-07].

[36] Nette Foundation: *Nette Tester – pohodové testování* [online]. Dostupné z: http://tester.nette.org/, [cit. 2014-04-07].

[37] The PHP Group: *What is PHP?* [online]. Dostupné z: http://www.php.net/, [cit. 2014-03-27].

[38] The PHP Group: *Document Object Model* [online]. Dostupné z: http://php.net/manual/en/book.dom.php, Poslední modifikace: 2013-11-12 [cit. 2013-11-12].

[39] The PHP Group: *SimpleXML* [online]. Dostupné z: http://php.net/manual/en/book.simplexml.php, Poslední modifikace: 2013-11-12 [cit. 2013-11-12].

[40] The PHP Group: *XMLReader* [online]. Dostupné z: http://php.net/manual/en/book.xmlreader.php, Poslední modifikace: 2013-11-12 [cit. 2013-11-12].

[41] W3C: *HTML Current Status* [online]. Dostupné z: http://www.w3.org/standards/techs/html, [cit. 2013-11-08].

[42] W3C: *Extensible Markup Language (XML) 1.0 (Fifth Edition)* [online]. Dostupné z: http://www.w3.org/TR/xml/, [cit. 2013-11-08].

[43] W3C: *HTML & CSS* [online]. Dostupné z: http://www.w3.org/standards/webdesign/htmlcss, [cit. 2013-11-08].

[44] W3C: *W3C DOM4* [online]. Dostupné z: http://www.w3.org/TR/domcore/, [cit. 2013-11-08].

[45] W3C: *XML Path Language (XPath)* [online]. Dostupné z: http://www.w3.org/TR/xpath/, [cit. 2013-11-08].

[46] W3C: *XQuery 1.0: An XML Query Language (Second Edition)* [online]. Dostupné z: http://www.w3.org/TR/xquery/, [cit. 2013-11-08].

[47] W3C: *XSL Transformations (XSLT)* [online]. Dostupné z: http://www.w3.org/TR/xslt, [cit. 2013-11-08].

[48] W3C: *What is HyperText* [online]. Dostupné z: http://www.w3.org/WhatIs.html, [cit. 2013-11-08].

[49] W3C: *Namespaces in XML 1.0 (Third Edition)* [online]. Dostupné z: http://www.w3.org/TR/xml-names/, [cit. 2013-11-11].

[50] W3C: *Namespaces in XML 1.1 (Second Edition)* [online]. Dostupné z: http://www.w3.org/TR/xml-names11/, [cit. 2013-11-11].

[51] W3C: *Document Object Model (DOM)* [online]. Dostupné z: http://www.w3.org/DOM/, Poslední modifikace: 2005-01-19 [cit. 2013-11-08].

[52] W3C: *HTTP - Hypertext Transfer Protocol* [online]. Dostupné z: http://www.w3.org/Protocols/, Poslední modifikace: 2013-09-25 [cit. 2013-11-08].

[53] W3C: *Extensible Markup Language (XML)* [online]. Dostupné z: http://www.w3.org/XML/, Poslední modifikace: 2013-10-29 [cit. 2013-11-08].

[54] W3C Schools: *HTML <!DOCTYPE> Declaration* [online]. Dostupné z: http://www.w3schools.com/tags/tag_doctype.asp, [cit. 2013-11-08].

[55] W3C Schools: *HTML Introduction* [online]. Dostupné z: http://www.w3schools.com/html/html_intro.asp, [cit. 2013-11-08].

[56] W3C Schools: *HTML - XHTML* [online]. Dostupné z: http://www.w3schools.com/html/html_xhtml.asp, [cit. 2013-11-08].

[57] W3C Schools: *XML Namespaces* [online]. Dostupné z: http://www.w3schools.com/xml/xml_namespaces.asp, [cit. 2013-11-11].

[58] W3C Schools: *XML Syntax Rules* [online]. Dostupné z: http://www.w3schools.com/xml/xml_syntax.asp, [cit. 2013-11-11].

[59] Zend Technologies Ltd: *Getting started: A skeleton application - Zend Framework* [online]. Dostupné z: http://framework.zend.com/manual/2.2/en/user-guide/skeleton-application.html, [cit. 2013-11-08].

[60] Zend Technologies Ltd: *Introduction - Zend Framework* [online]. Dostupné z: http://framework.zend.com/manual/1.12/en/zend.view.introduction.html, [cit. 2013-11-08].

[61] Zend Technologies Ltd: *Overview - Zend Framework* [online]. Dostupné z: http://framework.zend.com/manual/2.2/en/ref/overview.html, [cit. 2013-11-08].

[62] Zend Technologies Ltd: *View Scripts - Zend Framework* [online]. Dostupné z: http://framework.zend.com/manual/1.12/en/zend.view.scripts.html, [cit. 2013-11-08].

# Přílohy

# Seznam příloh

# Příloha A

# Obsah přiloženého DVD

Na přiloženém DVD lze nalézt:

- zdrojové kódy rozšířeného Nette Frameworku
- zdrojové kódy textu diplomové práce
- text diplomové práce ve formátu PDF