VYSOKÁ ŠKOLA EKONOMICKÁ V PRAZE

Fakulta informatiky a statistiky

Katedra informačních technologií

BAKALÁŘSKÁ PRÁCE

2012

Jan Tölg

Vysoká škola ekonomická v Praze

Fakulta informatiky a statistiky

Katedra informačních technologií

Studijní program: Aplikovaná informatika

Obor: Informatika

Nette framework

BAKALÁŘSKÁ PRÁCE

Student : Jan Tölg

Vedoucí : Ing. Jarmila Pavlíčková

Oponent : Ing. Martin Lukáš, Ph.D

2012

Prohlášení

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a že jsem uvedl všechny použité prameny a literaturu, ze které jsem čerpal.

V Praze dne 15. prosince 2012 .....

Jan Tölg

Poděkování

Rád bych poděkoval své vedoucí práce Ing. Jarmile Pavlíčkové za vedení práce a za to, že se mě na poslední chvíli ujala, Bc. Františku Tomanovi za cenné připomínky a Mgr. Dagmar Tölgové za korekturu.

Abstrakt

Tato práce se věnuje zajímavému českému PHP frameworku Nette. První část seznámí čtenáře s obecnými pojmy, návrhovými vzory použitými v Nette a popíše základní vlastnosti frameworku. Cílem druhé části, tentokrát praktické, je demonstrace frameworku při psaní reálné webové aplikace. Práce není výukovým materiálem, ale pomáhá osvětlit základní výhody frameworku a případnému zájemci tak nabízí detailnější pohled v ucelené formě.

Klíčová slova

Nette, Framework, MVC, PHP, webová aplikace, návrhové vzory

Abstract

This thesis deals with an interesting Czech PHP framework called Nette. The first part of the thesis introduces the reader to general ideas, design patterns used in Nette and also describes the main properties of the framework. Goal of the second part is focused on real-life application demonstration of the framework while developing a web application. This thesis is not an instructional material, but helps to explain the main advantages of the framework and can thus offer a more detailed description in a comprehensive form to a prospective interested person.

Keywords

Nette, Framework, MVC, PHP, web application, design patterns

Obsah

1 Úvod

1.1 Vymezení tématu práce a důvod výběru tématu

Web, jak jej známe dnes, je výsledkem překotného vývoje a obrovského rozmachu internetu posledních let. K 7. říjnu čítá internet zhruba 8,42 miliard zaindexovaných stránek [Kunder, 2012] a toto číslo samozřejmě roste. Všechny tyto stránky musel někdo vytvořit a je tedy zřejmé, že význam nástrojů pro vývoj webových stránek roste také.

Současné webové technologie jsou výsledkem dlouholetých snah otevřené komunity, která definuje webové standardy, zejména pak rychle se vyvíjející HTML[1] (v roce 2014 bude vydána již pátá verze) [Jackson, 2011] a CSS3[2], které již fungují v drtivé vetšině moderních prohlížečů. [findmebyIP, 2012]

Co se týče serverové části, dlouhá léta vítězí technologie PHP[3] nad ASP[4], Ruby, Javou či dalšími technologiemi, nad kterými se dají stavět dynamické webové stránky, [2012] proto jsem jako téma bakalářské práce záměrně zvolil nový český framework (označovaný také jako FW), který je postaven právě nad PHP. Popularita PHP je dána především jeho jednoduchostí.

V této době na trhu existuje přes 50 konkurenčních aktivně vyvíjených PHP frameworků – CakePHP, Zend, Symfony, Prado, CodeIgniter [Superdit, 2009]. Mnozí programátoři si během své praxe napsali nějaké vlastní frameworkové řešení, nebo alespoň řešení, které framework vzdáleně připomínalo [Morgan, 2012]. Proč tedy existuje tolik ostatních frameworků?

PHP je jazyk, který programátora moc nesvazuje a nevnucuje mu, jakým způsobem má v PHP programovat (objektově, či procedurálně) [Leseberg, 2009]. Různé frameworky mají tudíž svoje vlastní cesty, kterými řeší konkrétní problémy. Nezřídka se také stává, že si programátoři nad stávající FW píšou své vlastní nadstavby. Pokud se takových nadstaveb shromáždí více a ideově nezapadají do konceptu používaného FW, může vzniknout nový framework.

[1] HyperText Markup Language
[2] Cascading Style Sheets
[3] PHP: HyberText Preprocessor
[4] Active Server Pages

Rozdíly mezi frameworky nenajdeme jen v přístupu k řešení problémů, ale např. i ve velikosti, podporované verzi PHP, podpoře AJAXu[5] a XML[6] nebo rychlosti zpracování požadavků. Nette ve verzi 0.7 bylo velice rychlé a prakticky předčilo konkurenční frameworky [Daněk, 2008]. K Nette 2 se mi nepodařilo vyhledat relevantní informace a srovnávat rychlost aktuální verze není náplní této práce.

Důvodem výběru tématu je má dlouholetá zkušenost s Nette, ke kterému jsem se dostal ještě před vydáním první plné verze. Od jeho vzniku v roce 2004 [Grudl, 2011] prošel FW bouřlivým vývojem a jak již bylo zmíněno v předchozím odstavci, nyní se nachází v aktuální verzi s pořadovým číslem 2.

Ačkoli od počátku vývoje Nette byla dokumentace jeho největší slabinou [Daněk, 2008], nyní je již dokumentace obsáhlá. A přestože je dnes pravidelně rozšiřována komunitou pohybující se okolo Nette od jeho počátků, dodnes existuje pouze v elektronické podobě. Nic také nenasvědčuje tomu, že by se situace okolo tištěné dokumentace měla zlepšit. Dalším pozitivem je utlumení bouřlivého vývoje před verzí 2. Do té doby se Nette potýkalo s problémy se zpětnou kompatibilitou, zejména u šablon.

Z těchto důvodů je ideální doba, aby se s frameworkem a jeho přednostmi seznámilo více lidí v přehledné a ucelené formě.

Čtenářům tato práce umožní rychlé pochopení základních znaků frameworku a usnadní první kroky v Nette Frameworku.

Teoretická část práce se zabývá specifikami a možnostmi Nette. Má přiblížit jeho přínosy a přístupy k programování, ke kterým vývojáře vede samotná filozofie frameworku Nette. Součástí budou i fragmenty kódu, které pomáhají dokreslit teoretický výklad. Vzhledem k rozsáhlosti samotného frameworku zúžím jeho popis pouze na části, které budou využity v praktickém příkladu.

Druhá část práce se zaměří na práci se samotným frameworkem. Předvedu zde ukázky a srovnání kódu stejné aplikace v čistém PHP a v Nette. Výstupy obou zapsaných zdrojových kódů budou shodné.

[5] Asynchronous JavaScript and XML
[6] Extensible Markup Language

Na závěr setřídím poznatky ohledně Nette a zjistím, zda vývoj v Nette je pro vývojáře opravdu efektivnější a příjemnější.

1.2 Cíle práce

Cílem práce je seznámení čtenáře s rychle se rozvíjejícím PHP frameworkem Nette, s jeho koncepty a filozofií. Práce má ukázat jeho široké možnosti, např. zvýšení efektivity práce při jeho užití, nebo možnosti propojení s databázovou vrstvou Dibi a další přínosy, které z toho plynou.

1.3 Předpoklady a omezení práce

Aby byl čtenář schopen plně porozumět textu, je třeba disponovat alespoň elementární znalostí objektově orientovaného programování, jazyka PHP, základy HTML / CSS, AJAXu, jQuery[7] a základní znalostí databází.

Omezením práce je stále bouřlivý vývoj IT technologií a samotného Nette. Od února, kdy vyšla druhá verze frameworku [Grudl, 2012] se k 1. říjnu objevilo již 6 podverzí [Grudl, 2012]. Naštěstí se nyní jedná pouze o rozšiřování stávající funkcionality.

1.4 Struktura práce

Struktura práce je rozdělena do dvou velkých částí.

První, teoretická část, která čítá 2 hlavní kapitoly (kapitola 2 a 3), uvádí čtenáře do dané problematiky, seznámí ho s důležitými definicemi potřebnými pro pochopení dalšího textu. Dále pak obsahuje teoretický popis frameworku, použité návrhové vzory a popis jeho hlavních součástí.

Druhá část, která obsahuje pouze kapitolu 4, je částí praktickou. Je rozdělena na dva menší celky, kdy první z nich uvádí teoretický popis Nette frameworku do praxe při vývoji rezervační aplikace. Další celek popisuje vývoj shodného rezervačního systému, ale tentokrát v PHP. Na konci dochází k porovnání obou řešení a vyvození závěru.

[7] http://jquery.com/

1.5 Typografické konvence

Nadpisy jsou označovány standardní desetinnou tečkovou notací a to až do úrovně 3. Zdrojové kódy jsou uvedeny v šedém podbarvení a psány neproporciálním písmem. Stejným písmem jsou psány i části kódu, které jsou součástí souvislého textu.

1.6 Rešerše prací na podobné téma

Prací ohledně Nette lze nalézt na internetu několik. Z níže uvedených jsem nečerpal, uvádím je jen jako obecný přehled zpracovaných prací na stejné téma. Jedná se především o texty, které na rozdíl od této práce kladou důraz na vývoj konkrétní aplikace spíše než na demonstraci síly frameworku a jeho širšího uplatnění:

- **Vývoj hudebního komunitního portálu v prostředí Nette - Framework** - Práce je rozdělena na dvě části, teoretickou a praktickou, s důrazem na část praktickou. Krátce se take věnuje problematice webových komunitních portálů. Jak jsem již zmínil na začátku kapitoly, tato práce se liší především v odlišném přístupu k tématu, kdy jejím hlavním cílem je naprogramování vzorové aplikace [Plzák, 2011].
- **Přístup k databázím v PHP frameworcích Zend a Nette** - Práce dokumentuje frameworky Nette a Zend a nabízí detailní pohled na databázové vrstvy v obou frameworcích. Dále se zabývá návrhovými vzory od Martina Fowlera. Součástí jsou i praktické ukázky [Moravec, 2009].
- **Vývoj internetového obchodu s využitím Nette Framework** – Tato práce se zabývá především vývojem e-shopu se zaměřením na sportovní výživu a fitness [Hájek, 2011].
- **Analýza a návrh internetového obchodu pro knihkupectví** - Stejný cíl jako v práci předchozí, její součástí je i analýza podmínek, ve kterých bude obchod vznikat. Dalším cílem je se v implementaci a optimalizaci pro vyhledávače těmto podmínkám přizpůsobit [Hypš, 2012].
- **Podpora prodeje osobní elektroniky** - Oproti předchozím pracem, se tato o Nette framework jen otírá. Samotný eshop je realizován v Zend Framework [Pramen, 2009].

Při tvorbě návrhových vzorů a architektury jsem vycházel především z knihy Martina Fowlera - Patterns of Enterprise Application Architecture a z knihy Design Patterns od "Gang of Four".

Při zpracování části věnující se přímo Nette jsem použil především dokumentaci, oficiální fórum a blogy, které se Nette věnují. V podobě tištěné literatury k Nette, jak jsem uvedl na stránce 2, ještě neexistuje žádný materiál.

2 PHP Frameworky

2.1 Definice

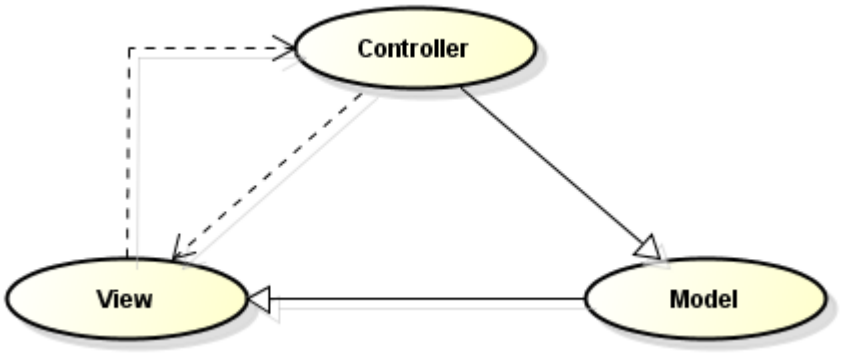
Framework je sada tříd, knihoven a nástrojů, která vývojářům umožňuje soustředit se na samotný projekt, místo neustálého psaní opakujících se řádků kódu [Janssen]. Takovým příkladem mohou být např. rutinní operace, které se v rámci webu vyskytují prakticky neustále jako je přihlašování, odhlašování, stránkování výpisů, tvorba a zpracování formulářů, lokalizace a další. Navíc ve většině případů přispívá k lepší čitelnosti kódu, protože méně napsaného kódu zpravidla provádí více činností. Obvykle poskytují určitou logickou strukturu, ve které je aplikace psána a vede vývojáře k určitému stylu psaní kódu. Některé frameworky více, některé méně.

Framework je tedy nástroj, který zásadním způsobem člověku napomáhá při psaní aplikací a zrychluje jejich vývoj.

2.2 MVC

Definici MVC v roce 1979 naformuloval norský profesor Trygve Reenskaug.

MVC, neboli Model-View-Controller je softwarová architektura, která rozděluje zdrojový kód do tří na sobě nezávislých vrstev a napomáhá tak kontrole velkých a komplexních projektů. Změna v jedné vrstvě má tudíž minimální dopad na vrstvy ostatní a proto na jednom projektu může pracovat více vývojářů [Reenskaug, 1979]



Obrázek 1: Schéma MVC

Obrázek 1 zobrazuje schéma MVC, ze kterého jsou zřejmé tyto vztahy:

Uživatel směruje své požadavky na Controller, ten je zpracuje a na jejich základě se dotáže Modelu, který Controlleru vrátí požadovaná data. Controller data předá View, který se postará o převedení dat do podoby vhodné pro prezentaci uživateli. Dále je možné vytvořit požadavek z View na Controller nějakou uživatelskou interakcí (např. klikem na odkaz). Celý cyklus se pak odehrává znovu.

Souhrnně by se tyto tři vrstvy daly popsat takto:

- **Model** – objekt reprezentující nějaká data, nebo aplikační logiku, např. databázovou tabulku
- **View** (pohled) – je určité zobrazení aktuálního stavu modelu, neboli vrstva která obstarává výstup
- **Controller** – prostředník mezi view a modelem, řídící vrstva, která zpracovává požadavky, mění stavy modelu

Proč ovšem udržovat v kódu několik na sobě nezávislých vrstev? Martin Fowler uvádí tyto důvody:

- Na rozdíl od vizuálních komponent se ty nevizuální mnohem lépe testují. Tím, že se oddělí model a prezentační vrstva, dá se kód lépe testovat bez nutnosti spoléhat na GUI[8] testovací nástroje.
- Dle situace mohou uživatelé chtít zobrazit konkrétní data různými způsoby. Oddělením prezentační a view vrstvy je možné vytvářet různá rozhraní, která

[8] Graphical User Interface

ovšem používají naprosto stejný model. Nejvíce patrné je to v případě potřeby přístupu k datům jak z příkazového řádku, tak i z prohlížeče, nebo nějakého mobilního zařízení.

- Z pohledu vývoje jsou view a model v zásadě dvě velice odlišné struktury. Když vývojář píše view, musí přemýšlet nad různými UI mechanismy, ergonomičností layoutu a přípravou správného rozhraní pro koncového uživatele. Kdežto u modelu se přemýšlí, jakým způsobem data spolu souvisí, jak jsou propojena, jaká jsou omezení jejich provázání a jak spolu mají interagovat. Obě vrstvy mohou používat i odlišné knihovny a programátoři mohou být specializováni jen na jednu z vrstev.

Naproti tomu Nette používá svou vlastní modifikaci této architektury, která nese název MVP. Liší se hlavně v tom, že Presenter zde vystupuje pouze jako "lepidlo" mezi Modelem a View, kdežto Controller u původního MVC principu ještě k tomu řídí i některé událost v uživatelském rozhraní [Reenskaug, 1979].

V souhrnu se dá říci, ze velkým přínosem MVC (resp. MVP) architektury je možnost rozdělení práce mezi více programátorů, aniž by si vzájemně zasahovali do kódu. Práce je tak nejen efektivnější, ale hlavně se udržuje čistý a logicky oddělený zdrojový kód, ke kterému není problém se v průběhu času vrátit a provádět v něm případné změny bez nutnosti do zdrojového kódu znovu pronikat a studovat ho.

2.3 Výhody a nevýhody frameworků

Mezi největší výhody patří bezesporu vysoká znovupoužitelnost kódu (díky komponentovému přístupu) a čistota kódu. Každý framework více či méně vede programátora k určitému přístupu při psaní aplikací a tím se kód stává přehlednějším nejen pro samotného vývojáře, ale i pro ostatní uživatele, kteří by byli nuceni v kódu cokoli měnit. Obvykle frameworky používají návrhové a architektonické vzory, u PHP frameworků je to velmi často architektonický vzor MVC, který dělí aplikace do vrstev a tudíž zde vzniká možnost rozdělit práci mezi více vývojářů, aniž by si vzájemně zasahovali do zdrojového kódu.

Navíc frameworky řeší neustále se opakující problémy a činnosti, které vývojáře zdržují.

Tento přístup vystihuje citace Martina Michálka:

"Rozemelte pšenici, přidejte kvasnice, sůl a vodu. Hotové těsto pak vložte do předehřátě trouby. Představte si, že takhle by vypadaly vaše první kroky pokaždé, když pomyslíte na chléb." [Michálek, 2008]

Usnadnění rutinních operací je věcí první, druhou a neméně důležitou, je bezpečnost.

Již v základu bývají ve frameworcích implementovány bezpečnostní mechanismy pro různé druhy webových útoků, jako je například Cross-site scripting nebo Session hijacking.

V neposlední řadě pak frameworky programátory vedou k čistšímu návrhu aplikací a ke srozumitelnějšímu kódu, což je zejména u PHP důležité, vzhledem k tomu, že jazyk PHP dává programátorovi velkou volnost a tudíž podporuje vznik špatných programovacích návyků [Lengstorf, 2012].

Frameworky ale mohou být v určitých situacích špatným řešením. Bývá to velmi individuální, ale každý framework vývojáře více či méně svazuje. Pokud se jedná o vývoj vysoce nestandardního projektu, bude možná dobrou volbou od frameworku upustit úplně.

Dalším negativem je fakt, že výkon frameworku je vždy nižší, než výkon aplikace napsané bez jeho použití. Jak ale již bylo napsáno v úvodu, i tyto hodnoty se mezi frameworky velice liší a je možné najít frameworková řešení, jejichž výkon plně dostačuje.

Užití frameworku ovšem není jednorázová věc, protože čas, který je při programování ve frameworku ušetřen, je kompenzován časem nutným k osvojení frameworku. Výrazná úspora přichází až při opakovaném používání.

3 Nette

Nette, aktuálně ve verzi 2.0.6., je open source událostmi řízený framework postavený nad PHP 5 a aktuálně se dá provozovat na PHP 5.2., PHP 5.3., a nově také na PHP 5.4. Slouží k vytváření moderních webových aplikací s důrazem na znovupoužitelnost kódu a podporou AJAXu.

S frameworkem Nette se pojí několik principů, ke kterým FW programátora vede:

KISS – Keep It Simple, Stupid – ve volném překladu znamenající „Zachovej to jednoduché, troubo“ je obecný návrhový vzor, který říká, že je lepší udržovat systémy jednodušší a tudíž méně náročné na údržbu, než systémy komplexnější [Rouse, 2005]

DRY – Don't repeat yourself – v překladu „Neopakuj se“, princip v tomto kontextu vývojáře vede ke znovuvyužívání již dříve napsaného kódu a snaze kód neduplikovat [Cunningham & Cunningham, Inc., 2011].

AJAX – Asynchronous JavaScript and XML – jedná se o souhrnné označení technologií, které umožňují měnit obsah bez nutnosti znovunačtení stránky [Garrett, 2005]. Využívá tyto konkrétní technologie:

- XML / xhtml / CSS
- DOM
- XMLHttpRequest
- Javascript

Nette je také pravidelně školen a odehrávají se okolo něho “Poslední soboty”, což jsou neformální setkání Nette vývojářů [Grudl, 2009].

3.1 Vývoj

Duchovním otcem projektu je David Grudl, který v roce 2004 položil základní stavební kámen. Poprvé byl framework představen v roce 2007 na konferenci PHP Frameworky [Grudl, 2007].

„Nette Framework prošel dlouhým vývojem, na jehož počátku byla (dnes si troufám říci) revoluční myšlenka: nechť je odkaz totéž, co zavolání funkce.“ [Grudl, 2009]

Jak dále David Grudl vysvětluje, jedná se o princip, jehož cílem je oprostit se od uvažování v rovině URL. Skrze URL se tedy opravdu dají volat funkce a předávat jim parametry. Slouží na to klíčová slova "handle" a "action", které se dají před název funkce, kterou chceme volat - function `actionArticle($id_article){}`. Funkce se pak jednoduše zavolá skrze url `/article?id_article=1`. Dále je toto téma rozvedeno v kapitole *Životní cyklus projektu*.

V současnosti se o další rozvoj stará komunita vzniknuvší okolo nadace Nette Foundation a připravuje se verze 2.1.

Nette je od počátku uvolněn pod licencemi BSD a GNU GPL.

3.2 Návrhové vzory a jejich použití v rámci Nette

Návrhové vzory v obecném smyslu zná asi každý – již stavitelé používali návrhové vzory, podle kterých se dá poznat, do kterého období stavba patří (např. gotická katedrála).

V softwarovém pojetí dnes neexistuje obecně platná definice spojení „návrhový vzor“, ale "Gang of four" ho popsaly takto:

„Každý návrhový vzor popisuje problém, který se ve vývojovém prostředí objevuje neustále dokola a popisuje jádro řešení problému. V takovém případě můžete toto řešení použít třeba milionkrát bez toho, aniž byste problém museli řešit úplně stejným způsobem vícekrát.” [Gang of four, 2002]

Jedná se tedy o nějaký doporučený a efektivní postup, jak se vypořádat s řešením konkrétního problému, nebo více souvisejících problémů. Dále se pak na návrhový vzor dá nahlížet jako na rozdělení problému na menší a relativně nezávislé části, které se pak dají řešit samostatně. [Fowler, 2002]

Zde popisované vzory se dají často použít i samostatně. Následující popisy však ukazují, jak jsou vzory využívány přímo ve frameworku Nette.

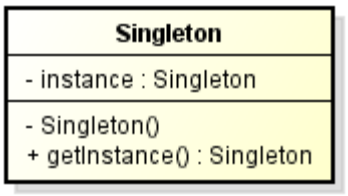
Návrhové vzory se dělí do třech skupin:

- Creational Patterns (tvořivé vzory)
- Structural Patterns (vzory strukturální)
- Behavioral Patterns (vzory chování)

3.2.1 Singleton

Singleton patří do kategorie vytvářejících vzorů. Cílem návrhového vzoru singleton (česky jedináček) je existence maximálně jedné vytvořené instance třídy (např. pro zapouzdření http požadavku). Třída singleton je tedy zodpovědná za vytvoření své vlastní instance, za ochranu proti vytvoření instance další a za zajištění globálního přístupu ke třídě. Po vytvoření je instance dostupná v rámci celé aplikace. Její implementace se v každém jazyce trochu liší. Nejčastějším řešením unikátnosti instance bývá deklarace proměnné, která udržuje referenci na již vytvořený singleton objekt [OODesign.com].

Následující diagram v obrázku 2 princip singletonu graficky popisuje.



Obrázek 2: Diagram singletonu

V Nette se Singleton používá ve třídě Nette\Session a Nette\Environment, ve výpisu 1 je uvedena velice zjednodušená ukázka implementace ve třídě Nette\Session.

Výpis 1: Session.php

```
<?php
  class Session {

    /** @var bool  has been session started? */
    private static $started;

    public function __construct(IRequest $request, IResponse $response)
    {
      $this->request = $request;
      $this->response = $response;
    }

    /**
    * Starts and initializes session data.
    * @throws Nette\InvalidStateException
    * @return void
    */
    public function start()
    {
      if (self::$started) {
            return;
      }

      // zde se spouští instance třídy, nastavují se proměnné, atd.
```

```
      self::$started = TRUE;
    }

  }
?>
```

V této konkrétní implementaci je uvedena statická proměnná `$started` v definici třídy a díky ní můžeme rozpoznat, zda už instance Session je vytvořena. Singleton pak můžeme zavolat jednoduše skrze `$singleton = Session::start();`

V tu chvíli tedy nastávají dva případy:

a) v případě, že instance neexistuje, vytvoří se a `$started` se nastaví na TRUE

b) pokud již instance existuje, vrátí sama na sebe referenci

3.2.2 Dependency Injection

Návrhový vzor Dependency Injection (dále DI) ideově vychází z obecnějšího vzoru IoC [sqdw, 2008], kterému se taky jinak říká obrácené řízení, což se dá vystihnout např. Hollywoodským principem „Nevolejte nám, my se ozveme.“ [Mead, 2008]

Dependency Injection“, nebo také jinak „vkládání závislostí“ nicméně tento okruh IoC zužuje. O co se tedy jedná? Jde o odebrání zodpovědnosti třídě za získání objektu, který potřebuje pro svůj chod. Výsledkem je pak srozumitelnější kód bez skrytých závislostí. Kód se pak stává transparentnějším a nově příchozí vývojář je schopen na první pohled rozpoznat, které závislosti konkrétní třída potřebuje pro svůj chod [Grudl, 2012].

V dřívějších dobách byly v PHP často používány globální proměnné. Jednalo se o nastavení direktivy "register_globals on" v nastavení samotného PHP i použití globálních proměnných naskrz třídami.

V prvním popisovaném případě byly všechny proměnné - ať už se jednalo o proměnné, které šly skrze GET, či POST- převáděny na globální proměnné. Proměnná `$_GET['promenna']` byla proto převedena na `$promenna`. Je na první pohled patrné, že zde vzniká bezpečnostní díra, díky které se dají přímo z URL podstrčit hodnoty proměnných v případě, že se ve zdrojovém kódu nepoužívá inicializování proměnných.

Druhým uváděným příkladem bylo použití globálních proměnných kdekoli v kódu. Časté bylo např. použití globálního připojení k databázi. David Grudl popisuje bezpečnostní riziko na příkladu platební brány - stáhli jste si třídu, která je používána pro odesílání plateb skrze číslo platební karty a chcete ji kódem ve výpise 2 otestovat.

Výpis 2: Ukázka platební brány

```
$pgw = new PaymentGateway('4478 1238 5789 5786', 05, 2014, 323);
$pgw->pay(100);
```

Nyní Vám bylo strženo 100Kč z karty a vy netušíte, jak se to mohlo stát. Třída Payment Gateway si z globálního prostoru vytáhla připojení k databázi a sama platbu provedla.

Řešením je již při deklaraci třídy připojení požadovat, což ukazuje výpis 3.

Výpis 3: Deklarace objektu podle DI

```
class PaymentGateway($card_number, $expire_month, $expire_year, $CCV,
Connection $connection){
// ...
}
```

Nette DI využívá skrze třídu Nette\DI\Container, což je implementace základního DI kontejneru a zajišťuje, že instance předávaného objektu je vytvořena pouze jednou (např. instance třídy starající se o připojení k databázi).

3.2.3 Observer pattern

Návrhový vzor Observer, neboli pozorovatel se používá ve chvílích, kdy je třeba o nastalé události jednoho objektu informovat jeden nebo více dalších objektů. Závislým objektům je oznámena změna a jsou automaticky aktualizovány.

Motivací ke vzniku tohoto návrhového vzoru byla potřeba udržení konzistence mezi několika třídami. Kdyby byly třídy pevně svázány, byla by omezena jejich znovupoužitelnost.

Observer se používá při řešení těchto problémů:

- při změně v jednom objektu je potřeba provést automaticky změny i v dalších objektech
- když je třeba, aby objekt byl schopen oznamovat změny ostatním objektům bez nutnosti vědět, jaké objekty přesně aktualizuje

- v případě složitější abstrakce, která používá dva aspekty, přičemž jeden je závislý na druhém, oddělením těchto dvou aspektů je dosaženo variability a lepší znovupoužitelnosti

3.2.4 Factory method pattern

Creational pattern factory method se používá ve chvíli, kdy potřebujeme, aby třída vytvářela nějaký objekt a v době psaní rozhraní ještě nevíme, jaký objekt to bude. Nadefinuje se tedy třída, česky zvaná tovární metoda, do které se při implementaci rozhraní píše, který objekt má třída vytvořit a vrátit.

Factory method se používá v těchto případech:

- třída neví, jaké objekty má vytvářet
- uzavírá logiku vytváření objektů do vlastní třídy, při potenciální změně vytvářených objektů se pak změny provedou jen v jednom místě ve zdrojovém kódu

Výpis 4 ukazuje implementaci návrhového vzoru factory method.

Výpis 4: Ukázka factory method

```
interface ICarFactory
{
  function createCar($car_type);
}

class CarFactory implements ICarFactory
{
  public function createCar($car_type)
  {
    return new Car($car_type);
  }
}

class Car
{
  private $car_type;

  function __construct($car_type) {
    $this->car_type = $car_type;
  }

  public function getCarType()
  {
    return $this->car_type;
  }
}


$carFactory = new CarFactory(); // vytvoří instance CarFactory
```

```
$car = $carFactory->createCar('BMW'); // CarFactory vytvoří objekt Car a vrátí

echo $car->getCarType(); // vypíše typ vytvořeného auta
```

V Nette se factory method používá zvláště pro vytváření formulářů v šabloně. Instance objektu je vytvořena až když je v šabloně požadováno vykreslení.

3.2.5 Composite

Tento návrhový vzor se používá, když je potřeba seskupit třídy do určité stromové struktury tak, aby práce s objekty byla jednotná. Motivací vzniku byla nutnost seskupovat objekty do logických struktur, kdy mohou vznikat jednoduché a složené objekty. Jednoduchý objekt je takový, který neobsahuje referenci na žádné další objekty. Naproti tomu složený objekt obsahuje referenci na další objekty, které se také dále mohou větvit. Základem celého návrhového vzoru je abstraktní třída, která definuje operace specifické pro samotnou třídu (např. třída pro vykreslení bude mít metodu `render()`), a dále operace potřebné pro správu vlastních potomků.¨

V Nette jsou tímto návrhovým vzorem řešeny komponenty (např. komponenta řešící definici a vykreslení formulářů - Nette\Application\UI\Form), které si uživatel může psát i sám. Třída, která má být připojena do hierarchie stromu komponent musí implementovat rozhraní Nette\IComponentContainer a je potomkem komponenty Nette\Application\Control. Pokud se jedná o vykreslitelnou komponentu, musí také nutně implementovat metodu `render()`.

3.2.6 Iterator

Iterator, patřící do skupiny behaviorálních patternů, poskytuje rozhraní pro sekvenční přístup k prvkům objektu, aniž by bylo třeba znát vnitřní strukturu objektu. Cílem tohoto vzoru je odebrání zodpovědnosti objektu za procházení seznamu prvků a delegování zodpovědnosti do objektu Iterator.

Výhody návrhového vzoru iterator.

- Jednotné procházení různých datových struktur.
- Možnost procházet datové struktury různými způsoby (např. na základě určitých kritérií)

- Není nutno znát vnitřní strukturu procházeného objektu

Iterator se samozřejmě velmi často používá v samotném PHP, ale v Nette je využíván například v šablonách, kdy v cyklu foreach vzniká proměnná $iterator (objekt typu Nette\Iterators\CachingIterator), která poskytuje užitečné informace o právě probíhajícím cyklu (zda se jedná o sudý, lichý, první či poslední průchod a další)

3.3 Architektura MVP

Jak jsem uvedl výše, Nette má svoji implementaci Model-View-Controller architektury známou jako Model-View-Presenter.

3.3.1 Routování

Pro pochopení dalšího textu je třeba si ujasnit, co je routování a jak přesně funguje.

Jedná se o překlad mezi URL a akcemi presenteru. Díky faktu, že způsob překladu se dá jednoduchým způsobem konfigurovat, tak je možno docílit dobře čitelných a optimalizovaných URL adres. Přínos těchto URL adres je patrný i z pohledu SEO, kdy vyhledávače preferují URL adresy vystihující jejich obsah. Stejných výsledků se dá dosáhnout pomocí známého Apache souboru .htaccess, který má ovšem v porovnání s routami omezené možnosti a jehož použití je nepoměrně složitější.

Výpis 5: HTTP požadavek

```
http://www.domain.com/article/edit/10
```

Výpis 6: Routa

```
$route = new Route('<presenter>/<action>[/<id>]', 'Basic:default');
```

Nette se při vytvoření požadavku (výpis 5) podívá do definovaných rout (výpis 6), aby zjistil, jak přesně má odkaz přeložit. Z předchozího příkladu je zřejmé, že první výraz za doménou je název presenteru, druhý název metody, která se vyvolá a třetí označuje nepovinnou proměnnou id (nepovinnou kvůli zápisu v hranatých závorkách). Konkrétně u tohoto požadavku se zavolá metoda `actionEdit($id){}` ve třídě ArticlePresenter.

Routy v Nette mají ještě jednu důležitou vlastnost a tou je obousměrnost. Obousměrností se rozumí, že se nepřekládá jen z URL do akce presenteru, ale Nette skrze routy dokáže automaticky tvořit odchozí odkazy ve tvaru, který je routy definován. Odkazy tudíž v šablonách nejsou uváděny přímo, ale zadávají se makrem `{link}`.

Definovaný odkaz v šabloně, který si Nette samo přeloží:

```
<a href="{link Article:detail, 10}">Přečtěte si článek</a>
```

Odkaz, který bude vytvořen, bude mít stejnou podobu, jak již bylo uvedeno v předchozím příkladě:

```
http://www.domain.com/article/detail/10
```

3.3.2 Presenter a jeho životní cyklus

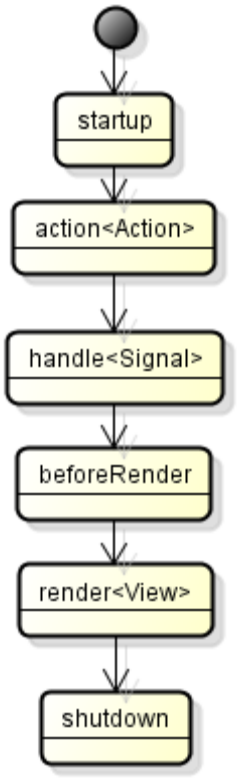
Presenterem se chápe třída, která je potomkem `Nette\Application\UI\Presenter` a tudíž má pevně definovaný životní cyklus.

Když přijde HTTP požadavek, je tedy přeložen routou. Poté co routa vyhodnotí tvar URL adresy, zavolá metodu, neboli akci konkrétního presenteru. Zpravidla tedy presenter žádá model o data a následně view o vykreslení. Odpovědi presenteru ale mohou být rozličné - např. JSON[9], XML soubor či přesměrování.

Pro programátora je zcela zásadní vědět, v jakém sledu se volají metody při vytvoření požadavku na presenter a tudíž co si kde programátor v kódu může dovolit.

Následující obrázek 3 přehledně ukazuje cyklus.

[9] JavaScript Object Notation



Obrázek 3: Životní cyklus presenteru

- **Startup** – nejčastěji se používá pro ověřování přihlášení a nastavování layoutu, který se konkrétnímu uživateli vykreslí
- **action<Action>** – metody využívané pro různé akce, které potřebují po provedení přesměrovat – např. zápis do databáze, smazání záznamu. Bez přesměrování by se nám například mohlo stát, že uživatel aktualizuje stránku a právě uložený záznam se v databázi zduplikuje. Dále se zde v deklaraci metody definuje, které parametry se mají při http požadavku presenterem očekávat.
- **handle<Signal>** - jedná se o metody, které zpracovávají signály a díky tomu jsou více než vhodné pro zpracování AJAXových požadavků.
- **beforeRender** – je vhodná pro registraci Latte filtrů, případně pro předání dat globálně do všech view v konkrétním presenteru
- **render<View>** - používá se pro předávání dat do šablony
- **shutdown** – spustí se při skončení životního cyklu

Jednoduchý presenter zobrazuje výpis 7 (příklad je záměrně koncipován tak, aby korespondoval s předchozími příklady):

Výpis 7: ArticlePresenter

```
class ArticlePresenter extends Presenter

{
	public function actionDetail($id){

	}
}
```

3.3.3 Model

Model v Nette se nijak neliší od základní MVC definice. Stará se o reprezentaci dat, pro ostatní třídy udržuje jednotné rozhraní a navíc neví o existenci presenteru ani view (a pokud ano, tak zásadní problém vznikl již při návrhu aplikace).

Příklad jednoduchého je modelu ve výpisu 8:

Výpis 8: ArticleModel

```
class ArticleModel extends Nette\Object{

	private $db = 'article';

	public function getItem($id)
	{
		return dibi::fetch("SELECT id, title, text FROM $this->db WHERE
id = %i", $id);
	}

}
```

Pozn.: Dědičnost od třídy Nette\Object v modelu patří v Nette k best practices. V opačném případě bychom byli ochuzeni o komfort, který Nette\Object nabízí, čímž může být například vyhazování výjimek při přístupu k nedeklarovaným členům třídy.

3.3.4 View

Taktéž jako u modelu se implementace v Nette neliší. Základním úkolem view je prezentovat data uživateli. Nette využívá vlastní šablonovací systém Latte, kterému je věnována vlastní kapitola.

3.3.5 Propojení všech částí dohromady

Jednotlivé vrstvy byly teoreticky rozebrány a teď je třeba se podívat, jak by se v praxi zobrazil detail onoho článku, což zobrazují výpisy 8, 9 a 10. Komentáře jsou přímo uvnitř zdrojového kódu.

Výpis 9: Presenter

```
class ArticlePresenter extends Presenter
```

```
{
	public function actionEdit($id){

		// vytvoří instanci modelu
		$model = new ArcticleModel();

		// vytáhne z modelu požadovaná data a předá je view
		$this->template->article = $model->getItem($id);

	}
}
```

Výpis 10: View

```
<h1>{$article->title}</h1><br />

<p>{$article->text}</p>
```

3.4 Doporučená adresářová struktura

Každý framework má svou danou vnitřní adresářovou strukturu. U MVC frameworků jsou zpravidla na první pohled odděleny vrstvy kontroleru, modelu a pohledu.

V čem se ale mohou lišit, je možnost s adresářovou strukturou manipulovat. Nette ani v tomto případě člověka nenutí doporučenou adresářovou strukturu používat. Vše je plně konfigurovatelné skrze soubor bootstrap.php. Pokud je tedy člověk zvyklý na jiný typ adresářové struktury, není problém ji během pár chvil překonfigurovat.

Po stažení základního balíku Nette frameworku se kostra Nette nalézá v adresáři sandbox. Doporučenou adresářovou strukturu uvádí výpis 11.

Výpis 11: Doporučená adresářová struktura

```
sandbox/
├── app/                    ← adresář s aplikací
│   ├── config/             ← konfigurační soubory
│   │   └── config.neon     ← hlavní konfigurační soubor
│   │
│   ├── model/              ← modelová vrstva a její třídy
│   ├── presenters/         ← třídy presenterů
│   │   └── HomepagePresenter.php  ← třída presenteru Homepage
│   │
│   │   templates/          ← adresář se šablonami
│   │   ├── @layout.latte ← šablona společného layoutu
│   │   └── Homepage/       ← šablony presenteru Homepage
│   │       └── default.latte  ← šablona akce default
│   │
│   └── bootstrap.php       ← zaváděcí soubor aplikace
│
├── libs/                   ← adresář na knihovny (např. třetích stran)
│   ├── Nette/              ← oblíbený framework
```

```
│   └── ...
│
├── log/                    ← obsahuje logy, error logy atd.
├── temp/                   ← pro dočasné soubory, cache, ...
└── www/                    ← veřejný adresář, document root projektu
    ├── .htaccess           ← pravidla pro mod_rewrite
    ├── index.php           ← který spouští aplikaci
    └── images/             ← další adresáře, třeba pro obrázky
```

Ovšem na co je třeba dávat pozor, jsou adresáře log a temp, které musí mít nastavena všechna přístupová práva pro zápis.

V České republice je častým problémem sdílený hosting, který Vám root určí, takže jedinými způsoby jak aplikaci spustit, je právě výše zmíněná rekonfigurace nebo přesměrování skrze .htaccess.

3.5 Komponenty

Výraz komponenta je obecně známá v reálném světě jako nějaká součástka nebo díl. Nejinak tomu je i u softwaru. V Nette se komponentou rozumí část kódu, určitá konstrukce, která poskytuje určitou požadovanou funkcionalitu a stává se tak základním kamenem pro znovupoužitelnost kódu, jak již bylo řečeno v kapitole o PHP frameworcích. Nezřídka se také stává, že komponenty jsou distribuovány mimo oficiální distribuci frameworku a prakticky tak rozšiřují možnosti frameworku [Rouse, 2005].

Nette napomáhá především při tvorbě vykreslitelné komponenty. Příkladem takové komponenty může být například kalendář, který se pak může kdekoli v šabloně vykreslit na základě rozličných předaných parametrů.

3.5.1 Factory

Factory, pro niž se v komunitě zažil výraz "továrnička", slouží k vytvoření instance nadefinované komponenty. Výhodou je, že se vytvoří opravdu až ve chvíli, kdy je v šabloně vyžádáno její vykreslení. Tím, že komponenty jsou vyčleněné mimo normální aplikaci, nabývá zdrojový kód přehlednosti. Její vytvoření probíhá skrze další klíčové slovo `createComponent`.

3.5.2 Vytvoření a použití vykreslitelné komponenty

- Třída, která má zastávat funkci vykreslitelné komponenta v Nette dědí od třídy Nette\Application\UI\Control.

- Implementuje vlastní metodu `render()` z Nette\Application\UI\PresenterComponent, která je předkem Nette\Application\UI\Control
- Uvnitř metody `render()` se specifikuje cesta k Latte šabloně a předávají se do ní potřebná data
- V požadovaném presenteru se skrze továrničku vytvoří instance komponenty
- Skrze `{control názevKomponenty}` se v šabloně komponenta vykreslí

Následující výpisy 12, 13 a 14 postupně zobrazují proces deklarace komponenty, její připojení do stromu komponent a její vykreslení.

Výpis 12: MyComponent

```
class MyComponent extends Nette\Application\Control
{
    /** @var bool Bude nám uchovávat stav rozhodnutí */
    private $decision = FALSE;

    /**
     * Vykreslení komponenty
     */
    public function render()
    {
        $template = parent::createTemplate(); //Vytvoříme šablonu
        $template->setFile(__DIR__ . '/DialogControl.phtml'); //Soubor se
šablonou
        //Abychom měli decision přístupné i v šabloně
        $template->decision = $this->decision;

        $template->render();
    }

    /**
     * Interakce uživatele
     * @param bool $decision
     */
    public function handleDecide($decision)
    {
        //Akceptujeme jen true nebo false
        if ($decision != 'true' && $decision != 'false') {
            throw new InvalidArgumentException;
        }
        //Nastav novou hodnotu decision
        $this->decision = $decision;
    }
}
```

Výpis 13: metoda createComponentMyComponent v Presenteru

```
protected function createComponentMyComponent
 ($name)
{
      $instance = new MyComponent(); // vytvoření instance vl. komponenty
      return $instance; // vrátí instanci vytvořené komponenty
}
```

Výpis 14: Vykreslení komponenty v šabloně

```
{control myComponent}
```

3.6 Formuláře

Žádná dynamická aplikace se v dnešní době neobejde bez zpracovávání nějakých uživatelských vstupů. S tvorbou a zpracováním formulářů jsou spojeny rutinní a velmi zdlouhavé činnosti. Příkladem se dají uvést tyto:

- Validace - jak klientská, tak serverová
- Bezpečnost proti útokům XSS[10] a CSRF[11]
- Automatické odmazání případných mezer na začátku a konci odeslaného řetězce

3.6.1 Prvky formuláře

Názvy metod, které vytvářejí formulářové prvky jsou prakticky shodné s html tagy. Názvy metod vždy začínají klíčovým slovem `add`. Tabulka 1 ukazuje, jak se formulářové elementy přidávají v Nette.

Tabulka 1: Zápis formulářových elementů

Html zápis	**Nette zápis**
<input type="text" />	addText($name, $label)
<select><option /></select>	addSelect($name, $label, array $items)
<input type="checkbox" />	addCheckbox($name, $caption)
<input type="radio" /> <input type="radio" />	addRadioList($name, $label, array $items)
<input type="submit" />	addSubmit($name, $label)
<input type="password" />	addPassword($name, $label)
<textarea></textarea>	addTextArea($name, $label)

[10] Cross-site scripting
[11] Cross-site request forgery

<input type="file" />	addUpload($name, $label)
<input type="image" />	addImage($name, $src, $alt)
<input type="button" />	addButton($name, $caption)
<input type="hidden" />	addHidden($name)

3.7 Validace

Validace probíhá jak na straně klienta (skrze JavaScript) tak i na serveru v případě, že by uživatel měl vypnutý JavaScript. Validační pravidla se definují při definici jednotlivých položek formuláře metodou `addRule()`. Na výběr je připravena široká paleta nejpoužívanějších validací. Nejčastější případy uvádí tabulka 2.

Tabulka 2: Zápis validací

Nette zápis	**Význam**
Form::FILLED	je hodnota vyplněna?
Form::EMAIL	odpovídá hodnota emailu?
Form::URL	odpovídá hodnota URL adrese?
Form::MAX_LENGTH	je hodnota delší než maximální délka?
Form::NUMERIC	je hodnota číslo?

3.7.1 Vytvoření formuláře

Nejjednodušším způsobem, jak vytvořit formulář je skrze továrničku, která byla popsána v předchozí kapitole. V šabloně se pak vykresluje buď skrze renderer Nette\Forms\Rendering\DefaultFormRenderer nebo je zde možnost ručního vykreslení. Pokročilí programátoři mají dokonce možnost napsat si vlastní renderer [Grudl, 2009].

Následující příklad (výpis 15) ukazuje, jak by vypadala kompletní definice nějakého průměrného zabezpečeného registračního formuláře. Objevují se zde různé druhy definic formulářových prvků a rozličné druhy validace. Následující dva příklady postihují dva možné způsoby vykreslení formuláře (výpis 16 a 17). V ručním vykreslení

neuvádím žádné zbytečné formátovací html tagy, aby neodváděly pozornost od vlastní podstaty ručního vykreslení.

Výpis 15: Definice formuláře

```
protected function createComponentRegisterForm()
{
	$form = new AppForm;


	$form->addText('login', 'Login')->setRequired('Vyplňte prosím
login');
	$form->addPassword('heslo', 'Heslo')->setRequired('Zadejte prosím
heslo');
	$form->addPassword('heslo_znovu', 'Heslo znovu')
		->addRule(Form::EQUAL, "Hesla se musí shodovat !",
$form["heslo"]);

	$form->addText('email', 'Email')
		->addRule(Form::EMAIL, 'Uveďte prosím správný email.');

	$form->addSelect('pohlavi', 'Pohlaví', array('muž', 'žena'))
		->setPrompt('---')
		->setRequired('Vyplňte prosím email');

	$form->addText('rok_narozeni', 'Rok narození')
		  ->addRule(Form::INTEGER, 'Rok narození musí být číslo')
		  ->addRule(Form::RANGE, 'Toto není správný rok narození',
array(1900, 2012));

	$form->addSelect('narodnost', 'Národnost', array('slovenská',
'česká'))
		->setPrompt('---')
		->setRequired('Vyplňte prosím národnost');

	$form->addSubmit('odeslat', 'Registrovat se');

	$form->onSuccess[] = callback($this, 'registerFormSubmitted');


	return $form;
}
```

Výpis 16: Vykreslení formuláře skrze Nette\Forms\Rendering\DefaultFormRenderer

```
{control registerForm}
```

Výpis 17: Ruční vykreslení formuláře

```
{form registerForm}

	{label login /} {input login}<br />
	{label heslo /} {input heslo}<br />
	{label heslo_znovu /} {input heslo_znovu}<br />
	{label email /} {input email}<br />
```

```
      {label pohlavi /} {input pohlavi}<br />
      {label rok_narozeni /} {input rok_narozeni /}<br />
      {label narodnost /} {input narodnost}<br />
      {label odeslat /} {input odeslat}

{/form}
```

3.8 Šablonovací systém Latte

Nette používá svůj vlastní šablonovací systém, který nese název Latte. Jeho syntaxe vychází ze známého šablonovacího systému Smarty – jedná se především o zápis výrazů ve složených závorkách {}. Mezi jeho přednosti patří především rychlost, ochrana před Cross site scriptingem, automatické escapování nebo použití různých helperů, napomáhající transformaci textu při výstupu. Další podobnost se Smarty je možno nalézt v názvech maker. Šablony zpravidla bývají uloženy v adresáři templates/ a mají příponu .latte (nebo také ještě .phtml, ale to jen z důvodu zpětné kompatibility)

3.8.1 Makra

Makry se v Latte rozumí výrazy uzavřené ve složených závorkách {}. Jsou to funkce, skrze které se dají vypisovat proměnné, iterovat nad nimi, nebo zapisovat podmíněné výrazy. Předdefinovaných maker je velká spousta, ale v tabulce 3 je uveden pouze výtah těch nejpoužívanějších.

Tabulka 3: Latte zápis a jeho význam

Latte zápis	**Význam**
{$promenna}	Vypíše proměnnou s escape sekvencemi
{!$promenna}	Vypíše proměnnou bez escape sekvencí
{var $novaPromenna = 10}	Definuje proměnnou a uloží do ní hodnotu
{if $podminka} {elseif} {else} {/if}	Podmínečný výraz, funkcionalita shodná s PHP
{foreach $pole as $polozka} {/foreach}	Procházení jednotlivých položek v poli
{control komponenta}	Vykreslí komponentu
{form formular} {/form}	Započne ruční vykreslení formuláře

{dump $promenna}	Vypíše obsah proměnné do Debug Baru
{link Presenter:akce}	Vytvoří odkaz na akci v požadovaném presenteru

Pokud by ovšem programátoru nestačila předdefinovaná makra, může si je sám dopsat [Grudl].

3.8.2 Helpery

Častým problémem bývá, že data, která skrze model a následně presenter do view putují, je potřeba pro prezentační účely ještě trochu poupravit. Příkladem může být předané datum ve tvaru YYYY-MM-DD, což je v našich končinách tvar poměrně neobvyklý a pro obyčejného uživatele je třeba ho prezentovat ve tvaru MM.DD.YYYY či MM/DD/YYYY.

Helpery tedy napomáhají transformaci vstupu skrze nasměrování výstupu jednoho proudu do vstupu proudu dalšího atd. Využívají zaběhnutý *nixový zápis skrze rouru[12] [Davisson, 2002]. Stejně jako v *nixu je možno tyto proudy řetězit.

Pro kodéry jsou tyto nástroje nezbytností, protože pomáhají rychle řešit neustále se opakující problémy. Příkladem uvádím použití helperu ve výpise 18, který vypíše zformátované datum ve tvaru 1.10.2012.

Výpis 18: Výpis helperu v Latte

```
{var $datum = “2012-10-01”}
{$datum|date:’j.n.Y’}
```

Tabulka 4 uvádí příklady nejpoužívanějších helperů.

Tabulka 4: Příklady helperů

Helper	**Význam**
truncate (length, append = '...')	zkrátí délku textu, zachová celá slova a ve výchozím nastavení za slovo doplní tři tečky

[12] Český ekvivalent slova "pipeline". V kódu je označován znakem "|".

trim	odstraní počáteční a koncové bílé znaky
lower	převede text na malá písmena
upper	převede text na velká písmena
length	vrátí délku řetězce

3.8.3 Komentáře

V Latte je možný dvojí zápis komentářů, standardní HTML komentáře skrze `<!-- komentář -->` fungují, ale výhodnější je používat Latte zápis `{* komentář *}`, protože funguje jak pro Latte, tak pro normální HTML tagy.

3.9 Snippety

Vzhledem k postupnému posunu webu k větší dynamičnosti má Nette připraveny nástroje usnadňující použití AJAXu. Jsou označovány jako snippety, neboli ústřižky.

Ústřižek obalí část kódu, která má být dynamicky načítána, případně měněna, a to vše se děje pomocí AJAXu. O zachytávání, zpracování ajaxových požadavků (které jsou realizovány signály) a překreslování snippetu se pak stará presenter.

Velkou výhodou je pak, že se přenáší jen potřebná data a překreslí se samotný ústřižek a ne celá stránka. Příklad snippetu uvádí výpis 19.

Výpis 19: Zápis snippetu v Latte

```
{snippet}
      Počet stupňů v Praze: {$praha->pocet_stupnu}
{/snippet}
```

3.10 Chyby a jejich řešení

Známá programátorská poučka nám říká, že v každém programu je alespoň jedna chyba [Kosek, 1999]. Nette má pro ladění chyb připravené dva důležité nástroje:

3.10.1 Nette\Diagnostics\Debugger

Nette\Diagnostics\Debugger (nebo také Laděnka, což je již zažitý název používaný komunitou) transformuje standardní výpis chyb v PHP do atraktivní podoby. Nejen, že vypíše chybu, ale navíc ukazuje, kde přesně v kódu chyba vznikla se širším kontextem, jaké parametry byly předány, a podává i další důležité informace. Vzhledem k tomu, že

třídy v Nette dědí vlastnosti od Nette\Object, která mimo jiné dokáže vyhazovat při přístupu k nedeklarovaným proměnným výjimku, je schopna Laděnka zachytit i takovéto chyby, které by se ve standardním chybovém hlášení vůbec neobjevily.

3.10.2 Debugger Bar

Debugger Bar je důležitý pomocník zobrazující údaje o běhu aplikace.

Ve standardním nastavení má debugger bar tyto grafické komponenty

- Délka zpracování skriptu
- Velikost paměti, kterou je třeba pro běh skriptu
- Informace o routování
- Údaje o přihlášeném uživateli a jeho rolích
- Výpis SQL[13] příkazů a délka jejich zpracování

Mimo standardní funkcionalitu je možné do Debugger Baru přidat své další funkce v podobě doplňků, které např. mohou informovat o validnosti stránky, či vypisovat ToDo úkoly poznamenané ve zdrojovém kódu.

3.11 Databázová abstraktní vrstva DIBI

DIBI je abstraktní databázová vrstva, která Nette provází již od počátku vývoje, ač není jeho přímou součástí a je tedy nutno ji stáhnout samostatně[14].

Známých databázových systémů máme pro využití na webu hned několik – MSSQL, PostgreSQL, MySQL, Oracle nebo SQLite. S těmito systémy můžete komunikovat buď přímo skrze SQL, nebo skrze tzv. Database Abstraction Layer, neboli česky databázovou abstraktní vrstvu.

Jaký je tedy důvod existence databázové vrstvy, když s každým databázovým systémem se dá komunikovat přímo? Důvodem je vytvoření rozhraní mezi konkrétní databází a aplikací, která je nad ní provozována. Poskytuje tak programátorovi jednotné vývojové API[15] bez nutnosti znát konkrétní databázové systémy do hloubky. Vývojáři je tak

[13] Structured Query Language
[14] http://dibiphp.com/
[15] Application Programming Interface

nabídnuta možnost za běhu přejít na jiný databázový systém (ať už z důvodu rychlosti, nebo dalších výhod konkrétního databázového systému).

Databázový layer Dibi tuto definici splňuje a přidává další nezbytnou funkcionalitu ulehčující vývojáři práci, jak ukazují výpisy 20, 21, 22, 23 a 24.

Výpis 20: Dibi zápis

```
$array = (
      ‘nazev_sloupce’ => ‘hodnota’,
      ‘nazev_sloupce2’ => ‘hodnota2’,
);
Dibi::query(“INSERT INTO [tabulka]”, $array);
```

Výpis 21: Vzniklý SQL dotaz

```
INSERT INTO [tabulka] (`nazev_sloupce`, `nazev_sloupce2`) VALUES
(‘hodnota’, ‘hodnota2’)
```

Výpis 22: Automatické escapování/slashování proměnných a automatické formátování typů

```
Dibi::query(‘SELECT * FROM [tabulka] WHERE nazev = %s’, $vyraz);
```

Výpis 23: Podmíněný sql příkaz

```
Dibi::query(‘SELECT * FROM [tabulka] %if’, isset($vyraz), ‘WHERE nazev =
%s’, $vyraz);
```

Výpis 24: Přidání pomocné funkcionality pro běžně používané úkony

```
$pole = $vysledek->fetchAssoc('id'); // výsledek SQL dotazu převede do
asociativního pole s klíčem ‘id’
$pary = $vysledek->fetchPairs('id', 'nazev'); // výsledek převede do
asociativního pole id => nazev
```

3.12 Výhody a nevýhody Nette

3.12.1 Výhody

Komunita – okolo Nette se nalézá velká komunita lidí, kteří spolu tvoří největší fórum o frameworku v Čechách [Grudl, 2009]. Pokud je v dnešní době ještě něco nezdokumentováno, s velkou pravděpodobností to bude k nalezení ve fóru. Dále lidé z komunity tvoří doplňky, které mohou v některých směrech usnadnit některé rutinně prováděné operace, a nakonec tvoří i sekci Planette, což jsou školící videa, návody a obecné materiály týkající se vývoje.

Strmá učící křivka – proniknutí do FW pro středně pokročilého PHP programátora není zas tak velkým oříškem. Od verze 2 je k dispozici mnohem lepší dokumentace, než

byla v případě verze 1 a quickstart na Nette webu ukáže nejdůležitější aspekty a základní použití frameworku.

Automatické načítání tříd a cachování – součástí Nette je Robot Leader, což je třída zajišťující automatické načítání tříd ve frameworku. Při prvním načtení se načtou třídy do cache, takže poté je přístup k nim mnohem rychlejší. Tato třída velice napomáhá vývojářskému komfortu, kdy programátor není nucen starat se o ruční načítání použitých tříd.

Použití Nette tříd samostatně - pokud se z nějakého důvodu vývojáři nehodí použít celý framework, může použít minifikovanou verzi frameworku a z ní načíst pouze některé třídy, např. Nette\Forms\Form [Grudl] nebo Nette\Diagnostics\Debugger (neboli Laděnku).

Rychlost - Nette patří mezi jeden z nejrychlejších PHP/webových framework [Daněk, 2008].

3.12.2 Nevýhody

Nerozšířenost a menší uživatelská základna – ačkoliv Nette existuje 8 let, anglická dokumentace a fórum je novinkou, proto tento framework neměl možnost se za hranicemi dostat do širšího povědomí

Nejistá budoucnost – ač za Nette stojí početná komunita, největším tahounem je stále jeho tvůrce David Grudl. Těžko tedy říci, jak by vývoj pokračoval, kdyby projekt opustil.

Rychlost – není náhodou, že rychlost je uvedena jak ve výhodách tak nevýhodách. Ačkoli se Nette řadí k těm rychlejším frameworkům, stále je mnohem pomalejší než samotné PHP.

Malá podpora syntax highlight Latte šablon v IDE - nyní komunita naprogramovala formou modulů syntax highlight pouze pro NetBeans a pro méně známý editor Sublime Text 2. Na syntax higlight v IDE jako je Eclipse, PHPEdit či Aptana Studio si bude třeba ještě nějakou dobu počkat.

4 Praktická část – aplikace

Na následujících stránkách popíši návrh a implementaci jednoduché demonstrativní aplikace jak v Nette, tak v čistém PHP. Bude se jednat o moderně vypadající a fungující web s použitím ajaxu. V závěru srovnám obě řešení z pohledu efektivity, čistoty kódu, náročnosti a dalších kritérií. Příklad je záměrně vybrán tak, aby pokryl důležitou základní funkcionalitu Nette.

4.1 Popis aplikace a použité technologie

Jedná se o jednoduchou demonstrativní aplikaci pro rezervaci tenisových kurtů. Hráči tenisu se na stránce mohou registrovat a tím je jim zpřístupněna možnost si po přihlášení rezervovat kurt na konkrétní datum a čas. V systému jsou dvě uživatelské úrovně, user a admin. User je uživatelský účet, který má možnost spravovat pouze své rezervace. Admin má ještě navíc ta privilegia, že může editovat obsah jednotlivých stránek (jednoduchý CMS[16] systém), registrované uživatele, může přidávat a odebírat kurty a rušit všechny rezervace.

Cílem je naprogramovat moderně vypadající a fungující web podle současných trendů - tzn. validní web s využitím AJAXu a javascriptového frameworku jQuery. Na konec si nastíníme známou modelovou situaci, kdy si klient po dokončení projektu vzpomene na nějaké dodatečné změny, které musí být nutně implementovány v co nejkratším čase do stávajícího kódu. Při popisu obou srovnávaných technologií budu využívat pouze zajímavé fragmenty kódu, které budou následně okomentovány.

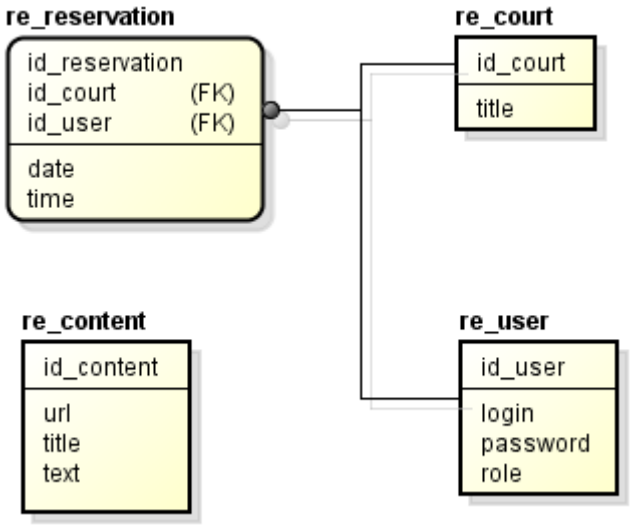
Pro vývoj webu byly použity následující technologie:

- MySQL 5.5.27.
- PHP 5.4.6
- Nette 2.0.7.
- jQuery 1.7.1

[16] Content Management System

Jako základ grafického layoutu byl použit Twitter Bootstrap, což je front-endový framework s předdefinovanými CSS a JavaScriptovými třídami, které usnadňují řešení neustále se opakujících problémů při vývoji klientské části webu.

Schéma databáze, které zobrazuje obrázek 4, vypadá následovně:



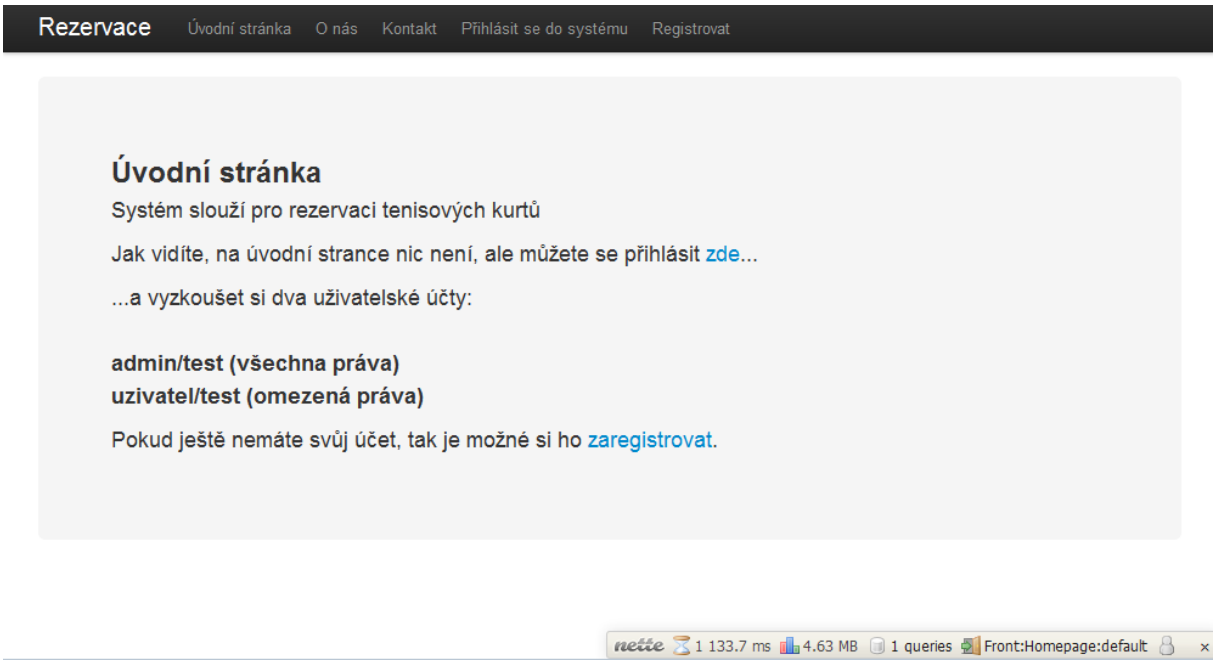
Obrázek 4: Databázové schéma

Srdcem celého projektu je tabulka re_reservation, která udržuje data o rezervovaných termínech a časech jednotlivých kurtů. Protože kurtů může být více, je svázána cizím klíčem s tabulkou re_court. Je potřeba vědět i o uživateli, který přihlášení na kurt provedl, proto je re_reservation skrze cizí klíč propojena i s tabulkou re_user.

Tabulka používá prefixy, aby v případě více projektů v jedné databázi bylo rozpoznatelné, které tabulky se týkají našich rezervací.

4.1.1 Layout aplikace

Protože není třeba procházet každou obrazovku aplikace, nastíním jen ty stěžejní, postupně je zobrazují výpisy 5, 6, 7, 8 a 9.



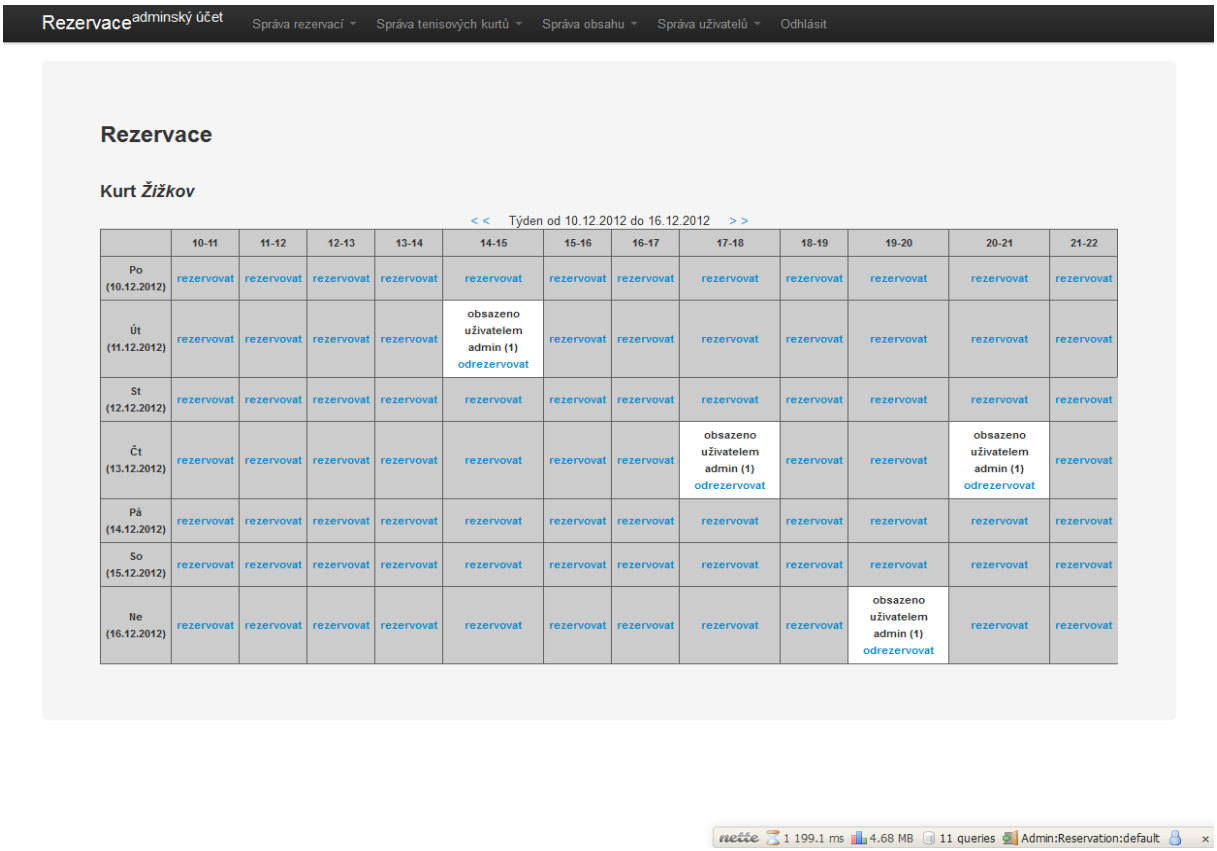
Obrázek 5: Úvodní obrazovka

Úvodní stránka (výpis 5) obsahuje vstupní info a vedou z ní odkazy na informativní stránky "o nás" a "kontakt". Dále je zde možnost založit si uživatelský účet a přihlásit se.

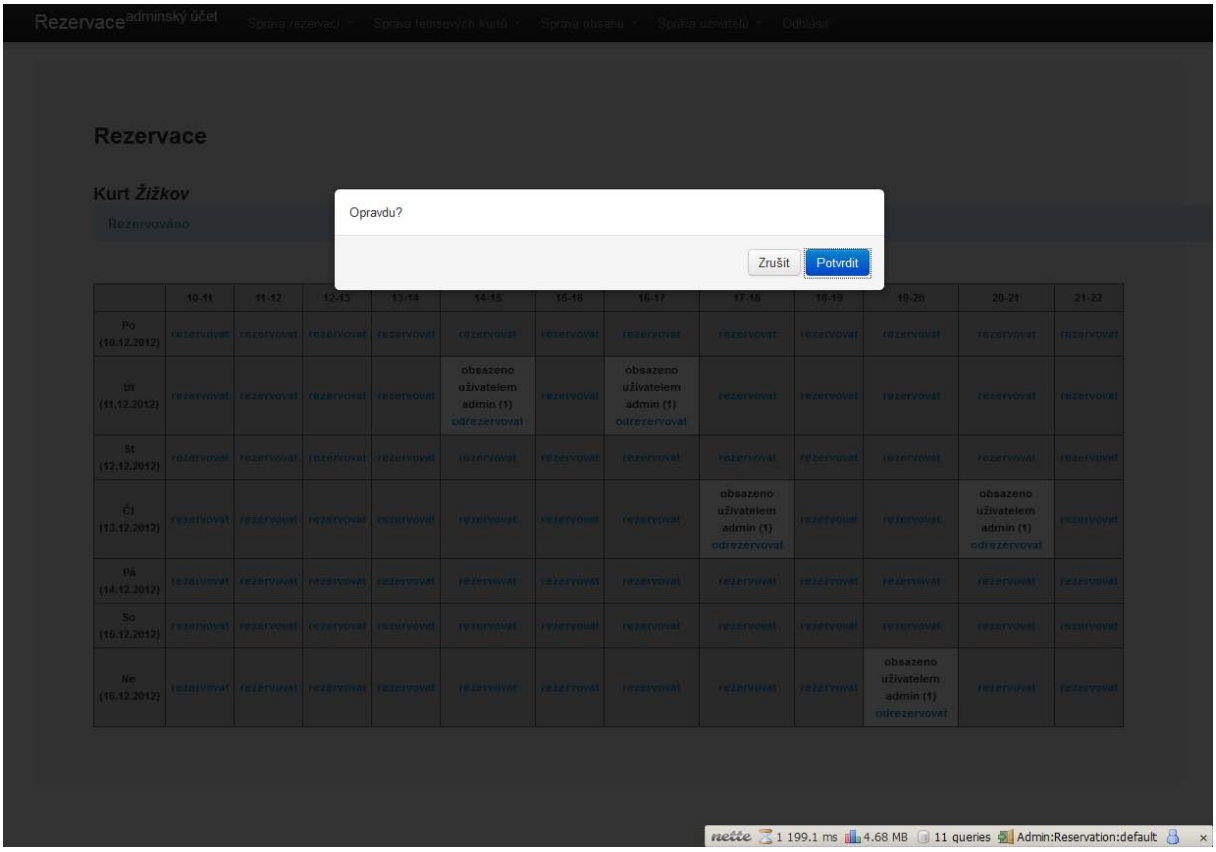
Záměrně je zobrazen i Nette\Debugger v pravém dolním rohu, který sloužil k ladění webu.

Obrázek 6: Formulář pro přidání/editaci uživatele

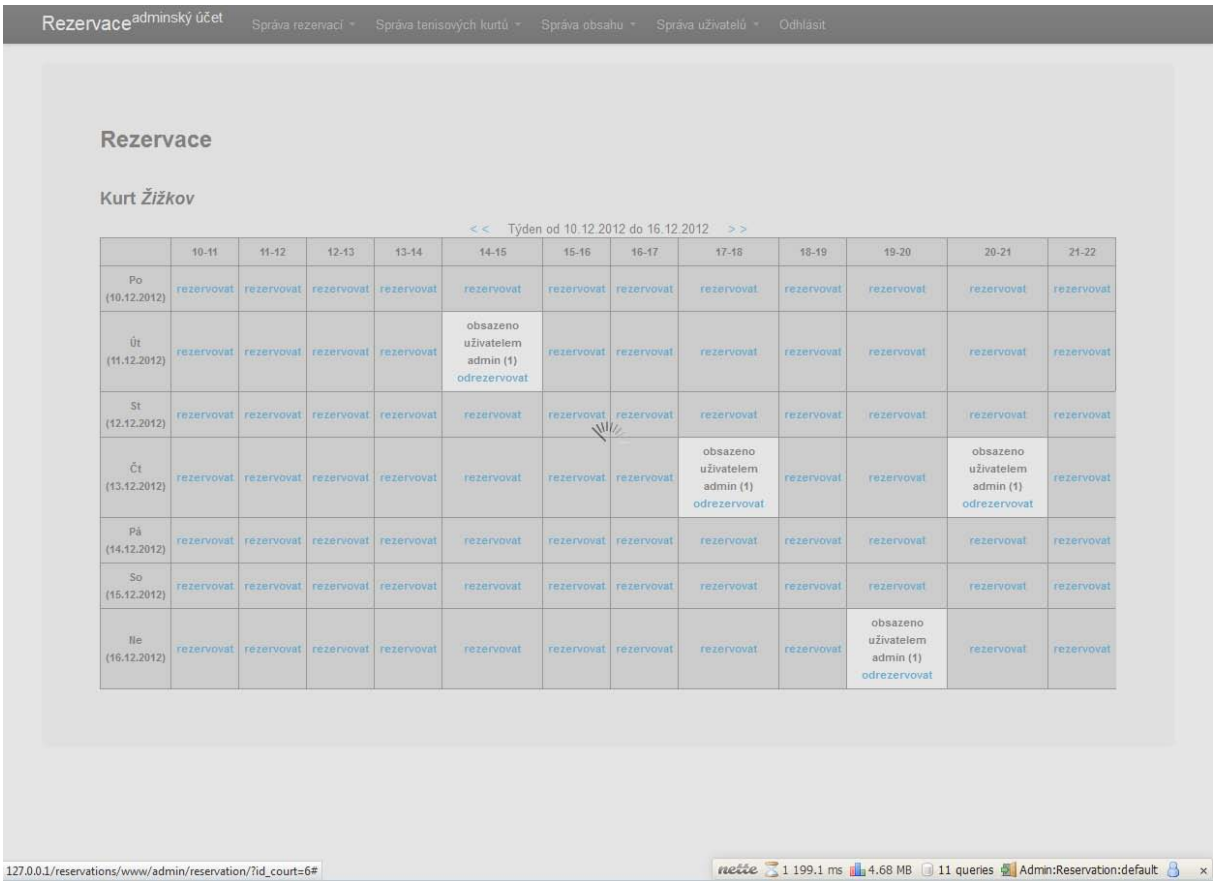
Standardní editační formulář podobný všem formulářům na webu zobrazuje obrázek 6.



Obrázek 7: Rezervační obrazovka



Obrázek 8: Potvrzovací dialog

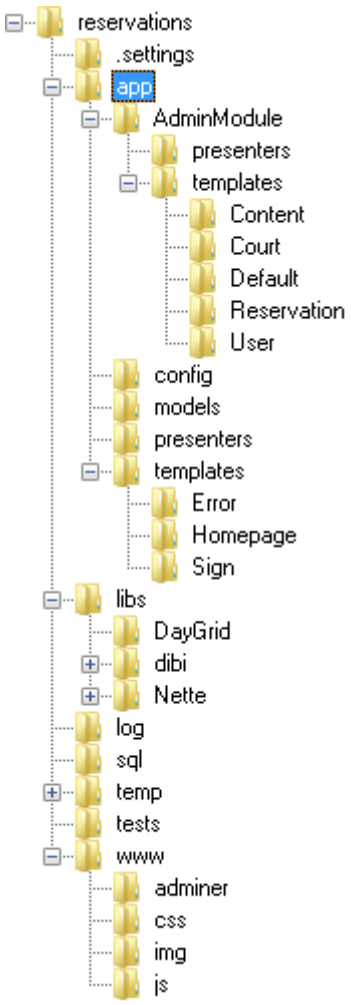


Obrázek 9: Překrytí overlayem při zpracovávání ajaxového požadavku

Samotná rezervační tabulka (výpis 7) obsahuje týdenní přehled rezervací s možností listovat vpřed a zpět, přičemž standardně se zobrazuje právě probíhající týden. V řádcích jsou zobrazeny konkrétní dny, ve sloupcích pak konkrétní hodiny. Rezervace probíhá klikem na odkaz "rezervovat" a následným potvrzením (výpis 8). Obrazovky ukazují celý proces, počínaje klikem na rezervační odkaz, potvrzením a následně overlayem s animovaným obrázkem, tzv. loaderem (výpis 9), který běhá dokola a dává uživateli vědět, že daná akce právě probíhá. Celá akce je řešena ajaxovým požadavkem, který po dokončení překreslí pouze danou tabulku.

4.2 Aplikace v Nette

4.2.1 Adresářová struktura



Obrázek 10: Adresářová struktura projektu v Nette

Adresářová struktura (výpis 10) je shodná s tou, kterou doporučuje dokumentace Nette. Je však rozšířená o další modul - AdminModule, který obsahuje presentery a šablony, které se využívají v případě přihlášeného uživatele. Dále je v adresáři libs nová komponenta starající se o samotné rezervování.

4.2.2 Index a boostrap

Výpis 25: index.php

```
<?php

// absolute filesystem path to this web root
define('WWW_DIR', dirname(__FILE__));

// absolute filesystem path to the application root
define('APP_DIR', WWW_DIR . '/../app');
```

```
// absolute filesystem path to the libraries
define('LIBS_DIR', WWW_DIR . '/../libs');

// uncomment this line if you must temporarily take down your site for
maintenance
// require APP_DIR . '/templates/maintenance.phtml';

// load bootstrap file
require APP_DIR . '/bootstrap.php';
```

Jak již popisují komentáře ve výpisu 25, index.php specifikuje konstanty WWW_DIR, APP_DIR, LIBS_DIR, které se dále využívají v bootstrapu. Z prvního pohledu je též zřejmé, že změnou hodnot těchto konstant je možno měnit základní doporučenou adresářovou strukturu.

Výpis 26: bootstrap.php

```
<?php

// Load Nette Framework
require LIBS_DIR . '/Nette/loader.php';


// Configure application
$configurator = new Configurator;

// Enable Nette Debugger for error visualisation & logging
//$configurator->setProductionMode($configurator::AUTO);
$configurator->enableDebugger(dirname(__FILE__) . '/../log');
Debugger::enable(Debugger::DEVELOPMENT);

// Enable RobotLoader - this will load all classes automatically
$configurator->setTempDirectory(dirname(__FILE__) . '/../temp');
$configurator->createRobotLoader()
      ->addDirectory(APP_DIR)
      ->addDirectory(LIBS_DIR)
      ->register();

// Create Dependency Injection container from config.neon file
$configurator->addConfig(dirname(__FILE__) . '/config/config.neon');
$container = $configurator->createContainer();

dibi::connect( $container->params['database'] );
dibi::getSubstitutes()->pre = 're_'; // add prefix to DIBI

// Setup router
$container->router[] = new Route('index.php', array(
            'module' => 'Front',
        'presenter' => 'Default',
        'action' => 'default',
), Route::ONE_WAY);

$container->router[] = new Route('content[/<url>]', array(
            'module' => 'Front',
        'presenter' => 'Homepage',
        'action' => 'content',
));
```

```
$container->router[] = new Route('admin/<presenter>/<action>', array(
            'module' => 'Admin',
        'presenter' => 'Default',
        'action' => 'default',
));
$container->router[] = new Route('<presenter>/<action>', array(
            'module' => 'Front',
        'presenter' => 'Homepage',
        'action' => 'default',
));


// Configure and run the application!
$container->application->run();
```

Bootstrap (výpis 26) načítá důležité třídy Nette frameworku, vytváří instanci konfigurační třídy Configurator.

Configurator se tedy stará o tyto věci:

- Logování chyb skrze třídu Nette\Debugger do adresáře log, příp. o výpis chyb přímo na obrazovku,
- Vytváří RobotLoader, který se stará o automatické načítání tříd. Při vytváření se mu předávají cesty k adresářům, které má RobotLoader automaticky procházet.
- Nastavuje cestu k temp adresáři, který je používán pro dočasné soubory, které vytváří Nette
- Specifikuje routovací pravidla, nazvanou jako routovací tabulka
- Načítá konfigurační soubor config.neon

Ve chvíli kdy už je spuštěn RobotLoader, Nette je schopno si samo dohledat třídu Dibi v adresáři libs a proto je možno rovnou spustit statickou funkci dibi::connect s parametry z konfiguračního souboru config.neon, která nás připojí k databázi.

Posledním krokem v bootstrapu je spuštění samotné aplikace.

4.2.3 Modely

V každé aplikaci se jeden model zpravidla stará o jednu entitu. Rezervace používají dohromady pět modelů: Authenticator, ContentModel, CourtModel, ReservationModel a UserModel. Popíšeme si dva z nich.

Výpis 27: Authenticator.php

```
<?php
```

```
class Authenticator extends Object implements IAuthenticator

{
	/** @var TableSelection */
	private $users;

	public function __construct(TableSelection $users)
	{
		$this->users = $users;
	}

	/**
	 * Performs an authentication
	 * @param  array
	 * @return Identity
	 * @throws AuthenticationException
	 */
	public function authenticate(array $credentials)
	{
		list($login, $password) = $credentials;

		$row = $this->users->where('login', $login)->fetch();

		if (!$row)
			throw new AuthenticationException("Už. jméno '$login'
nebylo nalezeno", self::IDENTITY_NOT_FOUND);

		if ($row->password !== $this->calculateHash($password))
			throw new AuthenticationException("Špatné heslo",
self::INVALID_CREDENTIAL);

		unset($row->password);

		return new Identity($row->id_user, $row->role, $row-
>toArray());
	}

	/**
	 * Computes salted password hash.
	 * @param  string
	 * @return string
	 */
	public function calculateHash($password)
	{
		return sha1($password);
	}
}
```

Authenticator (výpis 27), který je distribuován v rámci Nette, se stará o autentizace uživatelů. Název tabulky s uživatelskými účty je definován v souboru config.neon a v Nette je do Authenticatoru automaticky předán. Authenticator se pokusí uživatele přihlásit s odeslanými údaji a v tu chvíli mohou nastat tři stavy:

- Uživatel vůbec neexistuje, je vyhozena výjimka.
- Uživatel existuje, ale zadal špatné heslo, je vyhozena výjimka.

- Uživatel úspěšně prošel autentizačním procesem.

Ve třetím případě vrací metoda authenticate instanci třídy Identity, do které je vloženo ID uživatele, jeho role v systému a nakonec všechny údaje, které jsou k uživateli dostupné v databázi.

Výpis 28: ReservationModel.php

```
<?php
class ReservationModel extends Object{

	private $db = ':pre:reservation';

	public function getReservations($id_court)
	{
		return dibi::fetchAll("SELECT id_user, time, date FROM $this->db WHERE id_court = %i", $id_court);
	}

	public function addReservation($date, $time, $id_court, $id_user)
	{
		// protection
		if(dibi::fetchSingle("SELECT id_reservation FROM $this->db WHERE date = %d AND time = %i AND id_court = %i", $date, $time, $id_court))
			return false;
		else
			return dibi::query("
				INSERT INTO $this->db
				SET date = %d, time = %i, id_court = %i, id_user = %i", $date, $time, $id_court, $id_user);
	}

	public function deleteReservation($date, $time, $id_court, $id_user = null)
	{
		return dibi::query("
			DELETE FROM $this->db
			WHERE date = %d AND time = %i AND id_court = %i", $date, $time, $id_court);
	}

	public function getUserInReservation($date, $time, $id_court)
	{
		return dibi::fetchSingle("SELECT id_user FROM $this->db
			WHERE date = %d AND time = %i AND id_court = %i", $date, $time, $id_court);

	}
```

Na začátku třídy ReservationModel (výpis 28) je definován název tabulky, ve které jsou rezervace uloženy. Zde se využívá klíčové slovo "pre", které jsme již dříve definovali v bootstrapu a které Dibi přeloží do řetězce "re_". Bude se tedy pracovat s tabulkou, která má název "re_reservation". Další zásadní věcí je použití dibi modifikátorů - `%d` a `%i`. Než totiž databázi předáme nějaká data, jejichž původ může být dokonce až z uživatelských

formulářů, je třeba je ošetřit. Specifikováním modifikátorů dibi říkáme, jakou hodnotu má při předání očekávat. V případě %i dibi bude očekávat integer, v případě %d bude očekávat datum. Nejen, že tato vlastnost je zásadní z pohledu bezpečnosti, ale druhotným efektem je, že v případě předané hodnoty se špatným typem nekončí SQL dotaz chybou.

První metoda getReservations vrací všechny rezervace na konkrétním kurtu skrze metodu fetchAll. Ta vrací pole objektů DibiRow, které umožňuje k objektům přistupovat buď jako k asociativním polím nebo jako k hodnotám objektů.

Druhá metoda addReservation přidává rezervaci kurtu, v konkrétním datu a čase a zaznamenává uživatele, který rezervaci vykonává. Součástí metody je i ochrana před neoprávněnou rezervací již dříve rezervovaného termínu (např. pokud uživatel přijde na způsob jak vyvolat ajaxový dotaz s vlastními hodnotami proměnných).

Třetí metoda `deleteReservation()` maže rezervaci. O ověření, zda je mazání oprávněné, se tentokrát stará presenter.

Poslední metoda `getUserInReservation()` vrací ID uživatele v konkrétní rezervaci a používá se zejména při ověření, zda daný uživatel má právo rezervaci zrušit. O ochranu se i tentokrát stará presenter. Stejně jako ve druhé metodě se i zde používá funkce fetchSingle, která vrací pouze jedinou vybranou hodnotu, což je velice pohodlné a přehledné.

4.2.4 Presentery

Protože web má dvě sekce - pro přihlášené a pro nepřihlášené, jsou tedy i presentery rozděleny do samostatných modulů. Již z adresářové struktury projektu je vidět, že presentery pro nepřihlášené uživatele jsou v adresáři "presenters/" a pro přihlášené jsou umístěny v adresáři "AdminModule/presenters/".

4.2.4.1 Nepřihlášený uživatel

Základním kamenem je presenter BasePresenter, od kterého všechny následující presentery dědí jeho vlastnosti. Metody a proměnné z BasePresenteru jsou tedy dostupné pro všechny potomky. V rezervacích používá pouze jednu metodu, a to metodu pro odhlášení (výpis 29).

Výpis 29: Metoda pro odhlášení uživatele

```
public function actionLogout()
{
	$this->getUser()->logout();
	$this->flashMessage('Byl jste odhlášen');
	$this->redirect(':Front:Homepage:default');
}
```

Důvodem, proč je umístěna přímo v BasePresenteru, je budoucí možnost modulového rozšíření. Odhlášení každého typu uživatele probíhá stejně, takže není třeba zapisovat metodu pro odlogování do každého modulu zvlášť. Toto je praktická ukázka přístupu DRY (Don't repeat yourself).

O ostatní podstránky se pak dále stará HomepagePresenter, který vytváří formuláře pro přihlášení, registraci a načítá statické stránky. Jako ukázky lze uvést dvě metody, týkající se přihlášení. První vytváří samotný přihlašovací formulář a druhá zpracovává jeho odeslání.

Výpis 30: Definice přihlašovacího formuláře

```
protected function createComponentSignInForm()
{
	$form = new AppForm;
	$form->addText('username', 'Už. jméno:')
		->setRequired('Vyplňte prosím už. jméno');

	$form->addPassword('password', 'Heslo:')
		->setRequired('Vyplňte prosím heslo');

	$form->addSubmit('send', 'Přihlásit')->getControlPrototype()-
>setClass('btn btn-primary');

	$form->onSuccess[] = callback($this, 'signInFormSubmitted');
	return $form;
}
```

Metoda `createComponentSignInForm()` (ve výpise 30) vytváří instanci třídy AppForm, přidává jednotlivé položky formuláře a rovnou i jejich validace, které se při vykreslení projeví jak na klientské, tak serverové straně. Tudíž uživateli vypnutí javascriptu nepomůže v jeho případném úmyslu obejít validaci. V šabloně se formulář vykreslí skrze makro `{control signInForm}`.

Výpis 31: Metoda pro zpracování odeslaného formuláře

```
public function signInFormSubmitted($form)
{
	try {
		$values = $form->getValues();
		$this->getUser()->setExpiration('+ 60 minutes', TRUE);

		$this->getUser()->login($values->username, $values->password);
		$this->redirect(':Admin:Default:');
```

```
	} catch (AuthenticationException $e) {
		$form->addError($e->getMessage());
	}
}
```

Metoda starající se o přihlášení uživatele (výpis 31) dostává jako parametr instanci formuláře AppForm. Získá z ní data a pokusí se uživatele přihlásit. Pokud autentizace byla úspěšná, přihlášeného uživatele přesměruje do adminského modulu.

Dále metoda využívá autentizační výjimku, kterou Nette vyhazuje v případě neúspěšného přihlášení. Metoda je schopna ji zachytit a uvést tak uživateli důvod neúspěšného přihlášení. Konkrétní případ vyhazování výjimek je nastíněn v předchozí kapitole o modelech.

4.2.4.2 Přihlášený uživatel

Úspěšně přihlášený uživatel je tedy přesměrován do adminského modulu. Stejně jako u hlavního modulu je zde hlavní presenter, dědící od BasePresenteru, který je rodičem všech dalších presenterů v adminském modulu. Jeho název je DefaultPresenter a hned jeho první metoda `startup()` (výpis 32) se stará o pár zásadních věcí.

Výpis 32: Metoda startup()

```
public function startup()
{
	parent::startup();

	$user = $this->getUser();

	// allow only authenticated users
	if (!$user->isLoggedIn()){
		$this->flashMessage('Zalogujte se');
		$this->redirect(':Front:Homepage:default');
	}

	$model_court = new CourtModel();
	$this->template->courts_menu = $model_court->getCourtsMenu();

	if($user->isInRole('admin'))
		$this->setLayout('adminlayout');
	else
		$this->setLayout('userlayout');


	// prava
	$this->isUserAllowed();
}
```

Metoda nejprve volá metodu `startup()` svých předků, což je důležité z bezpečnostního hlediska. Kdyby náhodou byla v BasePresenteru řešena autentizace/autorizace

uživatele, DefaultPresenter o tom sám o sobě nic neví a může tak povolit nepřihlášenému uživateli pohybovat se neoprávněně v administračním rozhraní. Nicméně v této rezervační aplikaci je autorizace řešena až v DefaultPresenteru. Další příkaz se snaží dostat informace o uživateli. Pokud není přihlášen, je ihned přesměrován do hlavního modulu pro nepřihlášené (podobný případ může nastat, pokud přihlášení již uživateli vypršelo). Pokud je ale uživatel v pořádku přihlášen, zjišťujeme, zda je v roli admina, či obyčejného uživatele. Podle toho je mu pak nabídnut layout s jeho dostupnými možnostmi. Adminu se načítá adminlayout a uživateli userlayout. Jako poslední se volá metoda `isUserAllowed()`, která kontroluje, zda uživatel má k určité akci oprávnění (výpis 33).

Výpis 33: Metoda pro autorizaci uživatele

```
public function isUserAllowed()
{
	// forbidden sections
	$not_allowed = array(
		'Admin:Content',
		'Admin:Court',
		'Admin:User'
	);

	if(!$this->getUser()->isInRole('admin')	AND	(in_array($this-
>getName(), $not_allowed)))
		$this->accessDenied();
}
```

Z kódu je zřejmé, že běžný přihlášený uživatel není oprávněn vstoupit do sekcí Content, Court a User a při pokusu o přístup do jedné z této sekcí je ihned přesměrován metodou `accessDenied()` s odpovídající chybovou hláškou.

Poslední a patrně nejzajímavější věcí je řešení komponenty samotných rezervací (výpis 34).

Výpis 34: Deklarace proměnných ve třídě DayGrid

```
<?php
class DayGrid extends Control
{
	/** @persistent */
	public $year;

	/** @persistent */
	public $week;

	private $user;
	private $id_court;
...
}
```

Jak již bylo v teoretické části napsáno, komponentou může být pouze potomek třídy Nette\Control. Na začátku třídy deklarujeme čtyři základní proměnné. První dvě jsou ovšem velice zajímavé. Anotace `/** @persistent */` totiž Nette říká, že proměnné budou zachovávat své hodnoty a budou se přenášet ve všech odkazech. V ajaxových dotazech pak tedy není nutnost na tyto dvě proměnné vůbec myslet, budou se přenášet automaticky. Zásadní je to ve chvíli, kdy uživatel potřebuje rezervovat termín, jehož datum je až následující týden. Přeskočí tedy aktuální týden a zarezervuje si termín. Bez persistentních parametrů `$year` a `$week` by ho překreslení rezervační tabulky přesunulo zpět do právě probíhajícího týdne. Není tedy nutnost na tyto parametry myslet ani při invalidaci snippetů.

Výpis 35: Konstruktor třídy DayGrid

```
public function __construct(User $user, $id_court, IContainer $parent =
NULL, $name = NULL)
{
      parent::__construct($parent, $name);

      $this->year = $this->getParam('year', date("Y"));
      $this->week = $this->getParam('week', date("W"));

      $this->user = $user;
      $this->id_court = $id_court;
}
```

Vzhledem k tomu, že tato komponenta nemá žádnou přímou návaznost na presenter, ve kterém se zrovna uživatel nachází, není možné se žádným způsobem dostat k informacím o uživateli, který právě s komponentou pracuje. Proto je reference na uživatela předána při vytváření komponenty (výpis 35). Využívá se zde již popsaný princip vkládání závislostí. Dále metody `$this->getParam();` získávají z metody GET proměnné year a week, díky kterým se udrží pohled na týden, s nímž uživatel v danou chvíli pracuje. Druhý parametr metody je pak hodnota, která se předá v případě, že proměnná v GETu neexistuje - využívá se tedy při prvním načtení komponenty, kdy ještě year ani week nejsou nastaveny.

Výpis 36: Metoda render() ve třídě DayGrid

```
public function render()
{

      $template = parent::createTemplate();
      $template->setFile(dirname(__FILE__) . '/DayGrid.latte');

      $template->year = $this->year;
```

```
	$template->headings = array('Po', 'Út', 'St', 'Čt', 'Pá', 'So',
'Ne');

	$template->times = array(
		10 => '10-11',
		11 => '11-12',
		12 => '12-13',
		13 => '13-14',
		14 => '14-15',
		15 => '15-16',
		16 => '16-17',
		17 => '17-18',
		18 => '18-19',
		19 => '19-20',
		20 => '20-21',
		21 => '21-22');

	$template->day = $this->WeekToDate($this->week, $this->year);


	$model = new ReservationModel();

	$reservations = $this->getEvents($model->getReservations($this-
>id_court));
	$template->events = $reservations[0];
	$template->events_users = $reservations[1];

	$model_court = new CourtModel();
	$template->court = $model_court->getItem($this->id_court);

	$template->render();
}
```

Protože se jedná o vykreslitelnou komponentu, je zapotřebí, aby existovala metoda `render()` (výpis 36). Ta se stará o nastavení souboru šablony a předává do šablony důležitá data potřebná k vykreslení komponenty. Dále se stará o získání konkrétních rezervací a uživatelů, kteří si termíny rezervovali a získává též informace o konkrétním kurtu, přičemž momentálně se do šablony propisuje jen jméno načteného kurtu.

Výpis 37: Metoda handleNextWeek ve třídě DayGrid

```
public function handleNextWeek()
{
	$this->week++;

	if ($this->week == 53) {
		$this->week = 1;

		$this->year++;
	}

	$this->invalidateControl('calendar');
}
```

Metoda `handleNextWeek()` (výpis 37) se stará o přepnutí aktivního týdne. Je opravdu jednoduchá. Navýší týden a pokud číslo týdne přesáhne 52, je jasné, že se jedná o nový

rok a rok je inkrementován. Na konec je celá tabulka invalidována a tudíž překreslena. Obdobně funguje i tabulka `handlePrevWeek()`, starající se o přepnutí týdne zpět. Obě metody jsou volány skrze AJAX.

Výpis 38: Metoda handleAdd ve třídě DayGrid

```
public function handleAdd($date, $time)
{
	$model = new ReservationModel();
	if($model->addReservation($date, $time, $this->id_court, $this->user-
>getId())){
		$this->flashMessage('Rezervováno');

		$this->invalidateControl('calendar');
		$this->invalidateControl('flashMessage');
	}
	else {
		$this->flashMessage('Snažíte se rezervovat již rezervovaný
termín!');

		$this->invalidateControl('flashMessage');
	}
}
```

Výpis 38 zobrazuje metodu starající se o přidávání rezervací. V úvodu je načten ReservationModel, o kterém již byla řeč v kapitole Modely. Pokud je termín úspěšně rezervován, metoda překreslí tabulku a flashMessage se zprávou o úspěšné rezervaci. Pokud je již termín rezervován, je vyhozena chybová hláška.

Výpis 39: Metoda handleDelete ve třídě DayGrid

```
public function handleDelete($date, $time)
{
	$model = new ReservationModel();

	// only admin is allowed to delete OR user his own reservation
	if($this->user->isInRole('admin') OR $this->user->getId() == $model-
>getUserInReservation($date, $time, $this->id_court)){

		$model->deleteReservation($date, $time, $this->id_court);

		$this->flashMessage('Odrezervováno');

		$this->invalidateControl('calendar');
		$this->invalidateControl('flashMessage');
	}else{
		$this->flashMessage('Nedostatečná práva!');

		$this->invalidateControl('flashMessage');
	}
}
```

Poslední zajímavá metoda `handleDelete()` (výpis 39) se používá pro mazání termínů. Zde je specifikována podmínka, že jakýkoli termín může smazat jen admin a uživatel smí

smazat pouze svůj vlastní termín. V obou stavech již následuje klasické překreslení buď celé tabulky, nebo tabulky a informativní zprávy.

Aby komponenta mohla být komponentou, musí splňovat i druhou podmínku a to tu, že musí být připojena do stromu komponent. O to se stará již zmiňovaná metoda `createComponent()` (zde konkrétně použitá jako `createComponentDayGrid()`) ve třídě ReservationPresenter, jak zobrazuje výpis 40.

Výpis 40: Metoda createComponentDayGrid ve třídě ReservationPresenter

```
protected function createComponentDayGrid()
{
      return new DayGrid($this->getUser(), $this->getParam('id_court'));
}
```

Jaký je důvod řešení rezervací komponentou? V úvodu jsem nastínil známou situaci, kdy si klient po dokončení systému uvědomí, že potřebuje, aby systém rezervací fungoval jinak. Původně totiž vlastnil pouze kurt na Žižkově, a proto mu stačila jen jedna rezervační tabulka. V průběhu vývoje však koupil ještě kurt na Smíchově a proto systém potřeboval rozšířit. Mezi zásadní požadavek pak patřil fakt, že by rád v budoucnu nabízel možnost rezervací různých kurtů pouze na jediné stránce. Jako nejlepší alternativou se pak ukázalo předělání rezervačního systému do vlastní komponenty, která může být na jedné stránce použita nesčetněkrát. Odlišovat se bude jen předanými parametry. Jedna se bude starat o rezervaci Smíchova, druhá Žižkova a v budoucnu pravděpodobně ještě přibudou další komponenty, starající se o nové kurty.

4.2.5 Šablony

V rezervacích se používají šablony dvojího typu, prvním je layout, což je šablona, která obsahuje shodné prvky pro více podstránek. Že se jedná o šablonu s layoutem poznáme tak, že v názvu je před prvním písmenkem použit zavináč - např. @layout.latte.

Druhým typem jsou pak parciální šablony, které se do layoutu vkládájí.

Protože se všechny šablony nesou v podobném duchu, nastíním jen několik z nich.

Výpis 41: Layout templates/@layout.latte

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Rezervace{ifset #include} | {include #title}{/ifset}</title>
  </head>
```

```
  <body>
      {snippet flashMessage}
            {foreach $flashes as $flash}
            <div class="container">
                  <div class="alert alert-info">
                        <strong>{$flash->message}</strong>
                  </div>
            </div>
            {/foreach}
      {/snippet}

      <div class="container">
            <div class="hero-unit">
                  {include #content}
            </div>
      </div>
```

Layout (výpis 41) je pro potřeby textu trochu ořezán. Presenter načítá layout a do něj automaticky vkládá parciální šablonu, která má specifikované dva bloky, blok "title" a blok "content". Přes Latte makro `{include}` pak Nette volí, kam se každý z těchto bloků má propsat. Za elementem body následuje výpis flashMessages. A protože jsou uzavřeny v makru `{snippet}{/snippet}`, jsou překreslitelné i skrze ajaxový požadavek.

Výpis 42: Šablona AdminModule/templates/User/default.latte

```
{block title}
      Správa uživatelů - prohlížet
{/block}

{block content}

<h2>{include #content}</h2>

{snippet list}
      {if $users}
      <table class="list">
            <tr>
                  <th>Login</th><th>Role</th><th     class="action"></th><th
class="action"></th>
            </tr>
            {foreach $users as $user}
                  <tr>
                        <td>{$user->login}</td>
                        <td>{$user->role}</td>
                        <td><a        href="{link        edit,        $user-
>id_user}">editovat</a></td>
                        <td><a    href="{link    delete!,    $user->id_user}"
class="confirm">smazat</a></td>
                  </tr>
            {/foreach}
      </table>
      {else}
            Nebyly nalezeny žádné záznamy.
      {/if}
{/snippet}
```

Šablona default.latte (výpis 42) slouží pro demonstraci několika věcí. Za prvé ukazuje definici bloků, které se propisují do layoutu a za druhé proměnnou `$users`, která je předána presenterem a představuje výsledek, jenž dodal UserModel. Obsahuje položky typu DibiRow, nad kterými je Latte schopno iterovat a vypisovat tak položky postupně. Třetí a zásadní věcí je pak použití makra `{link Presenter:action}`. Nette díky tomuto makru automaticky vytvoří odkaz, který vede na konkrétní akci v určitém presenteru. Výsledek použití obou maker `{link}` v této šabloně není shodný, první vytváří odkaz na akci, druhý na signál (Nette rozpozná, že chceme signál díky použitému vykřičníku). Komunikace na úrovni signálu probíhá pod prahem view (šablony se vůbec nevykreslí) a je tudíž vhodná pro volání ajaxových požadavků. Aby bylo možné ajaxem celou tabulku překreslit, je znovu celý blok obalen do makra `{snippet}{/snippet}`.

Výpis 43: Šablona templates/Homepage/signIn.latte

```
{block title}
      Přihlásit se do systému
{/block}


{block content}

<h2>{include #title}</h2>

{if !$user->loggedIn}
      {control signInForm}
{else}
      Již jste přihlášen<br />
      <a href="{link logout}">Logout</a>
{/if}
```

Tato šablona se stará především o vykreslení komponenty přihlašovacího formuláře skrze makro `{control signInForm}`.

Na závěr opět uvádím nejzajímavější šablonu, která vykresluje samotnou rezervační tabulku (výpis 44).

Výpis 44: libs/DayGrid/DayGrid.latte

```
<h3>Kurt <i>{$court->title}</i></h3>


{snippet calendar}

<a href="{link prevWeek!}" class="ajax">&lt; &lt;</a>

Týden od {$day[0]|date:'j.n.Y'} do {$day[6]|date:'j.n.Y'}

<a href="{link nextWeek!}" class="ajax">&gt; &gt;</a>
```

```
<table cellpadding="0" cellspacing="0" class="calendar">

    <tr class="calendar-row">
      <th></th>
            {foreach $times as $time}
                  <th>{$time}</th>
            {/foreach}
      </tr>

      {foreach $headings as $heading}
            {var $date = $day[$iterator->counter-1]}

            <tr>
                  <th>
                        {$heading}<br />
                        ({$date|date:'j.n.Y'})
                  </th>
                  {foreach $times as $time => $item}
                        {if  $needle  =  array_search($date  .  '/'  .  $time,
$events)}
                              <th class="blocked">
                                    {if $user->isInRole('admin')}
                                          obsazeno                uživatelem
{$events_users[$needle-1]['login']} ({$events_users[$needle-1]['id_user']})
                                          <a   href="{link   delete!,   $date,
$time}" class="confirm">odrezervovat</a>
                                    {elseif           $events_users[$needle-
1]['id_user'] == $user->id}
                                          obsazeno vámi
                                          <a   href="{link   delete!,   $date,
$time}" class="confirm">odrezervovat</a>
                                    {else}
                                          obsazeno
                                    {/if}
                              </th>
                        {else}
                              <th><a   href="{link   add!,   $date,   $time}"
class="confirm">rezervovat</a></th>
                        {/if}
                  {/foreach}
            </tr>
      {/foreach}




</table>
{/snippet}
```

V podstatě se jedná o celé vykreslení rezervační tabulky. Kód je krátký a jednoduchý, protože aplikační logika je již připravena presenterem a předána do šablony. První řádek vypisuje pouze název kurtu. Dále se vypíše první a poslední den týdne a šipky, které vyvolají ajaxový požadavek posunující pohled o jeden týden zpět či vpřed.

Iterace skrze pole `$times` postupně postaví hlavičku sloupce, která obsahuje konkrétní hodiny, ve kterých lze kurt rezervovat.

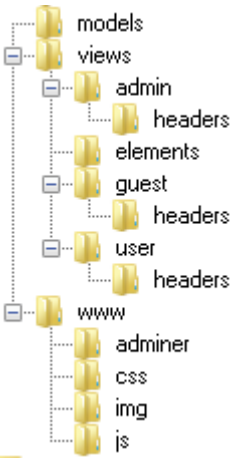
Dále iterujeme skrze pole `$headings`, které obsahuje označení dnů. V každém řádku pak probíhá ještě samostatné procházení prvku z pole $times, čímž jsme schopni nakonec do každého řádku získat časy rezervací pro konkrétní den. V této iteraci se ještě ověřuje, zda je již termín rezervován (informace o rezervaci je držena v poli `$events` pod klíčem, který je sestaven z data a konkrétního času - např. 2012-12-12/10) a podle toho nabízí dostupné možnosti.

- Nejste admin a termín je rezervován Vámi, tudíž je Vám nabídnuta možnost termín odrezervovat.
- Nejste admin a termín je rezervován jiným uživatelem, pak nemůžete termín ani rezervovat, ani odrezervovat.
- Jste admin a máte tedy možnost rušit veškeré rezervace a vytvářet rezervace nové.

4.3 Aplikace v PHP

Prvním zádrhelem, který mě potkal, byl problém, jak vlastně vývoj aplikace v PHP uchopit. Vzhledem k obrovské volnosti, které PHP poskytuje, jsem měl nutkání psát aplikaci v Nette stylu.

4.3.1 Adresářová struktura



Obrázek 11: Adresářová struktura projektu v PHP

Adresářová struktura (obrázek 11) je o poznání jednodušší, než v Nette, což ale nutně neznamená přehlednější. Web má pouze tři složky, adresář www, models a adresář views, který obsahuje šablony s aplikační logikou. Adresáře "admin/", "guest/" a "user/" obsahují šablony pro jednotlivé uživatelské účty a v každém je ještě adresář "headers/",

kde jsou umístěny soubory se stejným názvem jako šablona, které se ale volají ještě před tím, než je jakýkoli výstup odeslán do prohlížeče. Po vykonání akce je zde možnost přesměrovat stránku, a proto uživatel nemůže stejnou akci provést dvakrát, např. při stisknutí obnovovacího tlačítka. Soubory "headers" mají ještě druhou důležitou funkci, a to částečné oddělení aplikační logiky od samotných pohledů.

4.3.2 Index.php

Výpis 45: Soubor index.php

```
<?php
session_start();



// handle logout
if($_GET['page'] == 'logout'){
	unset($_SESSION['user']);
	$_SESSION['flashMessage'] = 'Odhlášen';
	$_GET['page'] = 'default';
}



// set roles
if(!isset($_SESSION['user'])) // guest menu
	$role = 'guest';
elseif($_SESSION['user']['role'] == 'user') // user menu
	$role = 'user';
elseif($_SESSION['user']['role'] == 'admin') // user menu
	$role = 'admin';


// define constants
define('views', dirname(__FILE__).'/../views/' . $role . '/');
define('headers', dirname(__FILE__).'/../views/' . $role . '/headers/');
define('elements', dirname(__FILE__).'/../views/elements/');
define('models', dirname(__FILE__).'/../models/');


// if homepage is not set
if(!isset($_GET['page']))
	$_GET['page'] = 'default';


// include headers
if(file_exists(headers . $_GET['page'] . '.php'))
	include(headers . $_GET['page'] . '.php');


include("connect.php");


?>

<!DOCTYPE html>
<html lang="en">
```

```
      <head>
      <!--
            html code
            ...
      -->


<?php
// flash messages
if(isset($_SESSION['flashMessage'])){
?>
      <div class="container">
            <div class="alert alert-info">
                  <strong><?php echo $_SESSION['flashMessage']; ?></strong>
            </div>
      </div>
<?php
unset($_SESSION['flashMessage']);
}
?>



<?php
include(views . $_GET['page'] . ".php");
// ...
```

První řádek v souboru index.php (výpis 45) spouští sessions, které jsou potřebné pro udržení informací o uživateli.

Následující podmínka se stará o odhlášení uživatele v případě, že je vytvořen požadavek skrze GET proměnnou action s hodnotou "logout". Dále se vyhodnocuje, o jaký typ uživatele se jedná, a podle toho se deklaruje proměnná `$role`, podle které se odvíjí použití konkrétního view a jeho hlavičky.

U následující řádků, které definují konstanty views, headers, elements, models je důležité použití funkce `dirname()`, které specifikují, z jakých adresářů se mají soubory načítat. V případě neošetření by do našeho kódu mohl útočník includovat svůj vlastní PHP skript a získat například kontrolu nad aplikací.

Poslední řádky již vkládají hlavičky, soubor connect.php, potřebný pro připojení k databázi, a samotný soubor šablony umístěný v adresáři views. Také načítají případné flashMessages, informující o provedených operacích.

Z kódu je dále zřejmé, že index.php se stará o veškerou obsluhu požadavků, které přicházejí z URL.

4.3.3 Modely

Stejně jako v případě Nette každý model odpovídá jediné entitě. Rozdíl je ovšem v tom, že tentokrát se nevyužívá abstraktní vrstva dibi, proto je práce s databází o něco složitější a méně přehledná. Pro porovnání uvádím kód z ReservationModel, tentokrát bez použití dibi.

Výpis 46: Model ReservationModel

```
public function getReservations($id_court)
{
	return mysql_query("SELECT id_user, time, date FROM $this->db WHERE
id_court = '" . mysql_real_escape_string($id_court) . "'");
}

public function addReservation($date, $time, $id_court, $id_user)
{
	$date = mysql_real_escape_string($date);
	$time = mysql_real_escape_string($time);
	$id_court = mysql_real_escape_string($id_court);
	$id_user = mysql_real_escape_string($id_user);


	// ochrana
	$protect = mysql_query("SELECT id_reservation FROM $this->db WHERE
date = '$date' AND time = '$time' AND id_court = '$id_court'");

	if(mysql_num_rows($protect) != 0)
		return false;
	else
		return mysql_query("
			INSERT INTO $this->db
			SET date = '$date', time = '$time', id_court =
'$id_court', id_user = '$id_user'");
}

public function deleteReservation($date, $time, $id_court, $id_user = null)
{
	$date = mysql_real_escape_string($date);
	$time = mysql_real_escape_string($time);
	$id_court = mysql_real_escape_string($id_court);
	$id_user = mysql_real_escape_string($id_user);

	return mysql_query("
		DELETE FROM $this->db
		WHERE date = '$date' AND time = '$time' AND id_court =
'$id_court'");
}

public function getUserInReservation($date, $time, $id_court)
{
	$date = mysql_real_escape_string($date);
	$time = mysql_real_escape_string($time);
	$id_court = mysql_real_escape_string($id_court);

	$result = mysql_query("SELECT id_user FROM $this->db
		WHERE date = '$date' AND time = '$time' AND id_court =
'$id_court'");
```

```
      return mysql_result($result, 0, 'id_user');
}
```

Uvedený kód ve výpise 46 nepotřebuje obsáhlý komentář. Nutností je ale uvést funkci mysql_real_escape_string, která se stará o ošetření vstupních proměnných. Pokud by proměnné zůstaly neošetřeny, je možné provést útok skrze SQL injection. Druhou možností ochrany před SQL injection je použití direktivy magic_quotes_gpc, která se zapíná přímo na serveru. V tuto chvíli ale ještě nevíme, zda projekt bude provozován na serveru, kde je tato direktiva zapnutá. V opačném případě bychom totiž aplikaci vystavili vysokému riziku.

4.3.4 Hlavičky a šablony s aplikační logikou

Výpis 47: Soubor views/guest/sign-in.php

```
<h2>Přihlásit se do systému</h2>

<?php if(!isset($_SESSION['user'])){ ?>
<form action="?page=sign-in" method="post" class="sign-in"/>
      <table>
            <tr>
                  <td>Už. jméno:</td><td><input type="text" class="login"
name="login" /></td>
            </tr>
            <tr>
                  <td>Heslo:</td><td><input type="password" name="password"
class="password" /></td>
            </tr>
            <tr>
                  <td></td><td><input     type="submit"     name="submitted"
value="Přihlásit" class="btn btn-primary" /></td>
            </tr>
      </table>
</form>
<?php
      } else
            echo 'Již jste přihlášen';
?>
```

Jako protipól je uvedena šablona sign-in.php (výpis 47). Zatímco u Nette stačilo použít makro `{control signInForm}` a formulář se automaticky vykreslil, zde je nutné celý formulář napsat ručně. To se týká i klientské i serverové validace. Navíc jsme ochuzeni o možnost použít svůj vlastní titulek, protože nelze použít makro `{block}`, ve kterém bychom titulek definovali.

Výpis 48: Soubor views/guest/headers/sign-in.php

```
if(isset($_POST['submitted'])){
      /*
```

```
        * Validace
        */
        if(!$_POST['login'])
             $flashMessage = 'Vyplňte login';
        elseif(!$_POST['password'])
             $flashMessage = 'Vyplňte heslo';
        else{
             // validace prosla
             include_once(models . "UserModel.php");
             $model = new UserModel();

             try{
                  $user          =          $model->signMe($_POST['login'],
$_POST['password']);

                  $_SESSION['user'] = $user;
                  $_SESSION['flashMessage'] = 'Byl jste přihlášen';


                  header('location: index.php');
                  exit;

             } catch(exception $e){
                  $_SESSION['flashMessage'] = 'Chyba: ' . $e->getMessage();
             }
        }
}
```

V hlavičce sign-in.php (výpis 48) se odehrává serverová validace a neprovede žádnou operaci, dokud data nejsou korektně vyplněna. Dále je zde vidět, že je třeba explicitně vkládat soubor UserModel.php, který obsahuje třídu UserModel, díky které se můžeme dostat k uživatelům. V případě úspěchu se nastaví zpráva do flashMessage, uložené v session, vyvolá se požadavek na přesměrování a zbytek skriptu se ukončí, protože již není třeba ho vykonávat.

Výpis 49: Klientská validace formuláře views/guest/sign-in.php

```
/*
 * Validation function
 */
function validate(el, e){
      if(el.val()==''){
            e.preventDefault()
            return false;
      }
      else
            return true;
      }
}

$(document).ready(function(){
      /*
       * Login validation
       */
      $('form.sign-in').submit(function(e){
            if(!validate($('.login'), e))
```

```
                alert('Vyplňte login');
            else if(!validate($('.password'), e))
                alert('Vyplňte heslo');
        });
});
```

Pro potřeby rezervační aplikace jsem napsal jednoduchou jQuery funkci `validate()` (výpis 49), která zjišťuje, zda bylo formulářové políčko vyplněno. V žádném případě ale však nedosahuje širokých možností validace, které nabízí Nette již v základu.

Výpis 50: Soubor views/elements/reservation.php

```
include_once(elements . 'ClassDayGrid.php');
include_once(models . 'ReservationModel.php');
include_once(models . 'UserModel.php');
include_once(models . 'CourtModel.php');


$court = new DayGrid();
$template = $court->template;
?>


<h3>Kurt  <i><?php  echo  mysql_result($template['court'],  0,  'title');
?></i></h3>



<a href="?page=court&amp;id=<?php echo $_GET['id']; ?>&amp;week=<?php echo
($template['week']-1); ?>&amp;year=<?php echo ($template['year']); ?>">&lt;
&lt;</a>


Týden  od  <?php  echo  $template['day'][0]  ?>  do  <?php  echo
$template['day'][6] ?>

<a href="?page=court&amp;id=<?php echo $_GET['id']; ?>&amp;week=<?php echo
($template['week']+1); ?>&amp;year=<?php echo ($template['year']); ?>">&gt;
&gt;</a>


<table cellpadding="0" cellspacing="0" class="calendar">
    <tr class="calendar-row">
      <th></th>
      <?php
       foreach($template['times'] as $time)
                echo '<th>' . $time . '</th>';
      ?>

      </tr>

      <?php foreach($template['headings'] as $key => $heading){ ?>

            <?php $date = $template['day'][$key]; ?>

            <tr>
                  <th>
                        <?php echo $heading; ?><br />
                        <?php echo $date; ?>
```

```
                </th>
                <?php foreach($template['times'] as $time => $item){ ?>
                      <?php  if($needle  =  array_search($date  .  '/'  .
$time, $template['events'])){ ?>

                            <th class="blocked">
                                  <?php
                                  if($_SESSION['user']['role']          ==
'admin'){
                                        echo  'obsazeno  uživatelem  '  .
$template['events_users'][$needle]['login']      .      '      ('      .
$template['events_users'][$needle-1]['id_user'] . ')';
                                  ?>

                                        <a
href="?page=court&amp;action=delete&amp;date=<?php       echo       $date;
?>&amp;time=<?php    echo    $time;    ?>&amp;id=<?php    echo    $_GET['id'];
?>&amp;week=<?php    echo    $template['week'];    ?>&amp;year=<?php    echo
($template['year']); ?>">
                                              odrezervovat
                                        </a>

                                  <?php
                                  }
elseif($template['events_users'][$needle-1]['id_user']                    ==
$_SESSION['user']['id_user']) {
                                        echo 'obsazeno vámi'; ?>

                                        <a
href="?page=court&amp;action=delete&amp;date=<?php       echo       $date;
?>&amp;time=<?php    echo    $time;    ?>&amp;id=<?php    echo    $_GET['id'];
?>&amp;week=<?php    echo    $template['week'];    ?>&amp;year=<?php    echo
($template['year']); ?>">
                                              odrezervovat
                                        </a>

                                  <?php
                                  } else {
                                        echo 'obsazeno';
                                  }
                                  ?>
                            </th>
                      <?php } else { ?>
                            <th>
                                  <a
href="?page=court&amp;action=add&amp;date=<?php         echo         $date;
?>&amp;time=<?php    echo    $time;    ?>&amp;id=<?php    echo    $_GET['id'];
?>&amp;week=<?php    echo    $template['week'];    ?>&amp;year=<?php    echo
($template['year']); ?>">
                                        rezervovat
                                  </a>
                            </th>
                      <?php } ?>
                <?php } ?>
          </tr>
     <?php } ?>




</table>
```

Vzhledem ke složitosti rezervační části jsem zvolil opět Nette způsob, aplikační logiku zpracovává třída DayGrid (výpis 50), která má velice podobnou stavbu jako v případě verze v Nette. Zásadním pojícím prvkem je předávání pole `$template` obsahující všechny informace, které šablona následně vypisuje. Nicméně toto řešení se s Nette rozchází v několika aspektech:

- Neexistence persistentních parametrů - proměnné, které udržují aktuální pohled, `$year` a `$week` je třeba uvádět v každém odkazu. Při tvorbě nového odkazu je velice snadné na ně zapomenout.
- Nepoužívá ajax – výsledkem je větší datová náročnost, protože dochází k obnovení celé stránky. Druhotným je pak menší uživatelský komfort, kdy v případě menšího monitoru se při obnovení obrazovky vytratí scrollovací pozice.
- Nepřehlednost – kód je na první pohled podstatně méně přehledný než v případě Nette verze.
- Je nutné includovat všechny používané třídy, dále pak vytvořit instanci třídy DayGrid, která se stará o logiku celé rezervační části aplikace.
- Tato část systému není řešena komponentově, a proto v případě nového klientského požadavku nastává problém, jak vyřešit kolizi parametrů.

4.4 Rozdíly plynoucí ze srovnání

Na úvod kapitoly bych rád zmínil, že jsem v PHP programoval déle než v Nette a díky tomu si troufám tvrdit, že srovnání je objektivnější, než kdybych technologie znal jen okrajově. Aplikace v Nette byla naprogramována doslova jedním dechem a během pár hodin jsem měl hotovou, bezpečnou aplikaci, připravenou pro nasazení na produkci. Horší to bylo ovšem u PHP, u kterého jsem nebyl občas schopen identifikovat vzniklé chyby. Nette totiž veškeré chyby hlásí a má připraveny předprogramované metody pro detailní výpis proměnných jak v presenterech, modelech, tak i v šabloně. Při přímém srovnání rezervační aplikace tedy PHP zcela propadlo. Nastíním důvody, které mě k tomuto tvrzení vedou:

- PHP objekty nedědí ze třídy Nette\Object, čímž se vývojář ochuzuje o použití Laděnky. Nette/Object je navíc striktnější a vede vývojáře k čistšímu psaní (např. vyhodí výjimku při přístupu k nedeklarovanému atributu třídy)

- Config.neon udržuje nastavení konzistentně na jednom místě a např. při připojení k databázi je využíváno dvojího nastavení, pro místní a pro produkční server. Nette automaticky pozná, na kterém serveru se aplikace spouští a podle toho vybere odpovídající nastavení.
- V Nette vede striktní oddělení vrstev k mnohem větší čistotě kódu. Ač u tohoto demonstračního projektu nebyl rozdíl příliš markantní, bylo cítit, že v případě budoucího rozšiřování by údržba zdrojového kódu v PHP byla mnohem namáhavější a hlavně nákladnější.
- Abstraktní vrstva dibi poskytuje opravdu velký uživatelský komfort, ať už se jedná o automatické escapování vstupních proměnných nebo o sadu předdefinovaných metod postihujících základní akce potřebné pro běžnou práci s databází. Vzhledem k možnosti podmíněných dotazů je v dibi přítomna i metoda dibi::test, která na obrazovku vypíše znění aktuálně volaného SQL dotazu. Zde tedy doporučuji použít dibi vrstvu i ve spojení se samotným PHP. Třída není přímo svázaná s Nette a je tedy možno ji použít i samostatně.
- Ruční načítání tříd v PHP je po určitém čase opravdu namáhavé a je velmi pravděpodobné, že vývojář při psaní na vložení souboru s deklarací třídy zapomene.
- Jak již bylo uvedeno v teoretické části, Nette veškeré vstupy a výstupy automaticky ošetřuje. Nečeká Vás tedy nepřijemné překvapení při zapomenutí zavolání metody `htmlspecialchars()` či `addslashes()` na vstupní/výstupní proměnné. Ošetření vstupních proměnných jde však ještě dále a Nette ještě všechny vstupy ošetří funkcí `trim()`, která odstraní bílé znaky na začátku a na konci vstupního řetězce.
- Nette poskytuje velký věcí v Nette, jsou formuláře předdefinovány v presenteru, a v šabloně pak automaticky vykresleny. Součástí je i velice pohodlná klientská i serverová validace. Toto všechno je v PHP nutno vypisovat ručně.
- Při pohledu na velikost uživatelské podpory jsou si Nette a PHP rovnocenné. Jak pro PHP a Nette existuje velká komunita, která je schopna v krátké době vývojáři pomoci vyřešit vzniklý problém.
- Tento poslední bod výčtu sice nevychází z přímého srovnání, ale je třeba ho zmínit. Pokud se Nette učí člověk jako první framework, nutně si musí osvojit

zcela nový přístup k programování webových aplikací, což vnímám jako zásadní nevýhodu .

5 Závěr

První část nastínila teoretické aspekty frameworku a čtenáři přiblížila zásadní výhody, které Nette framework přináší. V druhé části bylo cílem otestovat framework v reálném nasazení a srovnat jeho používání s vývojem v čistém PHP. Zadání aplikace bylo opravdu typové a při vývoji bylo znát, že Nette je opravdu naprogramováno zkušeným vývojářem se záměrem co nejvíce usnadnit vývoj webových aplikací. Citelně se poté výhody projevily zvláště v oblastech formulářů, jejich validací a v práci s databází. Ačkoli se Nette tváří, že přidává jen pár drobností, usnadňujících programátorovi vývoj, v globálním měřítku se jedná o obrovskou úsporu času. Člověk totiž není neomylný a s tím Nette počítá. Proto nabízí mocné nástroje sloužící pro ladění kódu.

Nette je dle mého názoru velice zajímavou volbu pro programování menších a středních aplikací. Napomáhá udržovat čistotu kódu, která je důležitá pro budoucí rozšiřitelnost a zpětné pochopení zdrojového zápisu. Troufám si tvrdit, že by byl dobrou volbou i v případě velké aplikace, protože Nette dnes používají i dlouho fungující weby jako root.cz, idnes.cz, CSFD.cz a dokonce je na něm postavena i webová stránka našeho pana prezidenta, Václava Klause.

5.1 Dosažené cíle

Všech stanovených cílů bylo dosaženo. V teoretické části byl čtenář uveden do obecné problematiky frameworků a následně byla v kapitole 3 představena jeho základní funkcionalita, postupy a principy. Praktická část přímo navazovala na část teoretickou, zde popsané postupy byly uvedeny v praxi a výsledky srovnány s vývojem shodného projektu naprogramovaného v čistém PHP. Přínosem práce je shrnutí teorie a praktická ukázka aplikace se zdrojovými kódy. Neméně významným přínosem pak bylo zjištění, že Nette framework opravdu urychluje a zpříjemňuje vývoj bezpečných webových aplikací.

6 Terminologický slovník

Termín	Význam	Zdroj
JavaScript	Multiplatformní, objektově orientovaný skriptovací jazyk využíván zejména pro web.	autor
XML	Značkovací jazyk určený pro výměnu dokumentů a dat mezi aplikacemi.	autor
AJAX	Obecné označení pro technologie, které dokáží měnit obsah bez nutnosti znovu načíst stránku.	autor
jQuery	Knihovna, postavená nad JavaScriptem, usnadňující např. obsluhu událostí nad elementy nebo práci s AJAXem	autor
SQL	Standardizovaný dotazovací jazyk, používaný v databázích pro práci s daty.	autor
API	Sada předdefinovaných tříd a funkcí nějaké konkrétní knihovny, které může programátor využívat.	autor
CSRF	Útok, který nevědomky provádí sám přihlášený uživatel v aplikaci. Principem je nalákání uživatele na útočníkovu stránku, která obsahuje např. tag <img>, který místo URL adresy obrázku obsahuje adresu, která vyvolá zamýšlený požadavek - např. smazání článku.	autor
XSS	Typ útoku, kdy útočník při nedostatečném ošetření výstupů, podstrčí do aplikace svůj vlastní JavaScriptový kód, který je následně vykonán.	autor

Session Hijacking	Útok, jehož cílem je odcizení session ID přihlášeného uživatele.	autor
JSON	Formát pro výměnu dat, používaný zejména při AJAXových požadavcích. Jeho výhodou oproti XML je jeho menší velikost.	autor

7 Citovaná literatura

Cunningham & Cunningham, Inc. 2011. Cunningham & Cunningham, Inc. *Dont Repeat Yourself.* [Online] 13. Prosinec 2011. [Citace: 19. Listopad 2012.] http://c2.com/cgi/wiki?DontRepeatYourself.

Daněk, Petr. 2008. Velký test PHP frameworků: Zend, Nette, PHP a RoR. *Root.cz.* [Online] 11. Září 2008. [Citace: 19. Listopad 2012.] http://www.root.cz/clanky/velky-test-php-frameworku-zend-nette-php-a-ror/.

Davisson, Gordon. 2002. OS X's BSD/unix command-line. *WESTWIND computing.* [Online] 2002. [Citace: 19. Listopad 2012.] http://www.westwind.com/reference/os-x/commandline/pipes.html.

findmebyIP. 2012. HTML5 & CSS3 Support, Web Design Tools & Support - FindMeByIP - CSS3 & HTML5 Browser Support. *findmebyIP.* [Online] 2012. [Citace: 19. Listopad 2012.] http://fmbip.com/litmus.

Fowler, Martin. 2002. *Patterns of Enterprise Application Architecture.* místo neznámé : Addison Wesley, 2002. ISBN: 0-321-12742-0.

Gang of four. 2002. *Design Patterns.* Indianapolis : Addison-Wesley, 2002. ISBN: 0-201-63361-2.

Garrett, Jesse James. 2005. adaptive path. *Ajax: A New Approach to Web Applications.* [Online] 18. Únor 2005. [Citace: 19. Listopad 2012.] http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications.

Grudl, David. 2009. David Grudl – rozhovor. *PĚSTUJEME WEB.* [Online] 23. 12 2009. [Citace: 19. Listopad 2012.] http://www.pestujemeweb.cz/clanky-rozhovory/text-27-david-grudl-rozhovor.html.

—. 2011. David Grudl: Marketing Nette dělají spokojení uživatelé. *Zdroják.cz.* [Online] 26. Květen 2011. [Citace: 19. Listopad 2012.] http://www.zdrojak.cz/clanky/david-grudl-marketing-nette-delaji-spokojeni-uzivatele/.

—. 2007. David Grudl: Nette Framework poprvé na konferenci PHP frameworky 2007. *YouTube.* [Online] 2007. [Citace: 19. Listopad 2012.]

—. Formuláře samostatně. *Nette Framework.* [Online] [Citace: 19. Listopad 2012.] http://dev.nette.org/cs/forms/standalone.

—. 2012. Nette Framework 2.0.6 released! *Nette Framework.* [Online] Říjen 2012. [Citace: 19. Listopad 2012.] http://forum.nette.org/en/1087-nette-framework-2-0-6-released.

—. 2009. Nette Framework: MVC & MVP. *Zdroják.cz.* [Online] 24. Březen 2009. [Citace: 19. Listopad 2012.] http://www.zdrojak.cz/clanky/nette-framework-mvc--mvp/.

—. 2009. Nette Framework: Neprůstřelné formuláře III. *Zdroják.cz.* [Online] 16. Červen 2009. [Citace: 19. Listopad 2012.] http://www.zdrojak.cz/clanky/nette-framework-neprustrelne-formulare-iii/.

—. 2009. phpFashion. *Poslední sobota - sraz Nette Framework.* [Online] 23. 2 2009. [Citace: 19. Listopad 2012.] http://phpfashion.com/posledni-sobota-sraz-nette-framework.

—. 2012. PHPFashion. *Co je Dependency Injection?* [Online] 12. Březen 2012. [Citace: 19. Listopad 2012.] http://phpfashion.com/co-je-dependency-injection.

—. Vlastní Latte makra. *Nette Framework.* [Online] [Citace: 19. Listopad 2012.] http://pla.nette.org/cs/vlastni-latte-makra.

—. 2012. Vyšel Nette Framework 2.0 final. *Zdroják.cz.* [Online] 3. Únor 2012. [Citace: 19. Listopad 2012.] http://www.zdrojak.cz/zpravicky/vysel-nette-framework-2-0-final/.

Hájek, Martin. 2011. Vývoj internetového obchodu s využitím Nette Framework. *Theses.cz.* [Online] 10. Listopad 2011. [Citace: 19. Listopad 2012.] http://theses.cz/id/cq44vl/.

Hypš, Bc. Otakar. 2012. Analýza a návrh internetového obchodu pro knihkupectví. *Masarykova Univerzita.* [Online] 2. Leden 2012. [Citace: 19. Listopad 2012.] http://is.muni.cz/th/325347/fi_b/.

Jackson, Joab. 2011. W3C: HTML5 will be finished in 2014 - Computerworld. *COMPUTERWORLD.* [Online] 14. Únor 2011. [Citace: 19. Listopad 2012.] http://www.computerworld.com/s/article/9209322/W3C_HTML5_will_be_finished_in_2014.

Janssen, Cory. What is Software Framework? - Definition from Techopedia. *Techopedia.* [Online] [Citace: 19. Listopad 2012.] http://www.techopedia.com/definition/14384/software-framework.

Kosek, Jíří. 1999. Předmluva knihy. *téměř VŠE O WWW.kosek.cz.* [Online] 26. Leden 1999. [Citace: 19. Listopad 2012.] http://www.kosek.cz/php/predmluva.html.

Kunder, Maurice de. 2012. WorldWideWebSize.com | The size of the World Wide Web (The Internet). *worldwidewebsize.com.* [Online] 2012. [Citace: 19. Listopad 2012.] http://www.worldwidewebsize.com/.

Lengstorf, Jason. 2012. Why You're a Bad PHP Programmer. *nettuts+.* [Online] 28. Únor 2012. [Citace: 19. Listopad 2012.] http://net.tutsplus.com/tutorials/php/why-youre-a-bad-php-programmer/.

Leseberg, Ernie. 2009. So why so many PHP frameworks? *Ernie Leseberg.* [Online] 3. Únor 2009. [Citace: 19. Listopad 2012.] http://ernieleseberg.com/so-why-so-many-php-frameworks/.

Mead, Matthew T. 2008. Matthew T. Mead. *Hollywood Principle – Don't Call Us, We'll Call You!* [Online] 8. Listopad 2008. [Citace: 19. Listopad 2012.] http://matthewtmead.com/blog/?p=267.

Michálek, Martin. 2008. Vzhůru dolů. *Proč i kodéři potřebují svůj framework?* [Online] 13. Červenec 2008. [Citace: 19. Listopad 2012.] http://www.vzhurudolu.cz/archiv/2005-2008/www.vzhurudolu.cz/82/proc-i-koderi-potrebuji-svuj-framework.html.

Moravec, Petr. 2009. Přístup k databázím v PHP frameworcích Zend a Nette. *NU.ŠL.* [Online] 2009. [Citace: 19. Listopad 2012.] http://invenio.nusl.cz/record/10682?ln=cs.

Morgan, Jeremy. 2012. The Programmers Before Us Were Better. *Jeremy Morgan.* [Online] 27. Září 2012. [Citace: 19. Listopad 2012.]

http://www.jeremymorgan.com/blog/programming/the-programmers-before-us-were-better/.

OODesign.com. Singleton Pattern. *OODesign.com.* [Online] [Citace: 19. Listopad 2012.] http://www.oodesign.com/singleton-pattern.html.

Plzák, Pavel. 2011. Vývoj hudebního komunitního portálu v prostředí Nette Framework. *Theses.cz.* [Online] 28. Duben 2011. [Citace: 19. Listopad 2012.] http://theses.cz/id/oxbnul/.

Pramen, Jiří. 2009. Podpora prodeje osobní elektroniky. *Archív diplomových prací ČVUT.* [Online] 12. Červen 2009. [Citace: 19. Listopad 2012.] https://dip.felk.cvut.cz/browse/pdfcache/brhelj1_2009bach.pdf.

Reenskaug, Trygve. 1979. Trygve/MVC. *Hjemmesider ved Institutt for informatikk.* [Online] 1979. [Citace: 19. Listopad 2012.] http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html.

Rouse, Margaret. 2005. SearchCIO-Midmarket. *KISS Principle (Keep It Simple, Stupid).* [Online] Září 2005. [Citace: 19. Listopad 2012.] http://searchcio-midmarket.techtarget.com/definition/KISS-Principle.

—. 2005. What is component? *WhatIs.com.* [Online] Srpen 2005. [Citace: 19. Listopad 2012.] http://whatis.techtarget.com/definition/component.

Rychlý a pohodlný vývoj webových aplikací v PHP. *Nette Framework.* [Online] http://nette.org/cs/.

sqdw. 2008. Inversion of Control, Dependency Injection. *Rising sun.* [Online] 15. Duben 2008. [Citace: 19. Listopad 2012.] https://sqdw.signaly.cz/0804/inversion-of-control-dependency.

Superdit. 2009. Big List of PHP Framework. *Superdit.* [Online] 22. Červenec 2009. [Citace: 19. Listopad 2012.] Big List of PHP Framework.

2012. Usage Statistics and Market Share of Server-side Programming Languages for Websites, December 2012. *W3Techs.* [Online] Prosinec 2012. [Citace: 19. Listopad 2012.] http://w3techs.com/technologies/overview/programming_language/all.

8 Seznam obrázků

9 Seznam výpisů

10 Seznam tabulek

11 Přílohy

Kompletní zdrojové kódy k oboum projektům z praktické části jsou dostupné na githubu pod těmito URL adresami:

- https://github.com/jantolg/reservations
- https://github.com/jantolg/reservationsPHP