Vysoká škola ekonomická v Praze

Fakulta informatiky a statistiky

Katedra informačních technologií

Studijní program: Aplikovaná informatika

Obor: Informační systémy a technologie

Srovnání vývoje webových aplikací v Nette Frameworku (PHP) a Node.JS

Diplomová práce

Student: Bc. Michal Kočárek

Vedoucí: Ing. Rudolf Pecinovský, CSc.

Oponent: Ing. Petr Mazánek

2013

Prohlášení

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

V Praze dne 26. června 2013.

Bc. Michal Kočárek

Poděkování

Děkuji panu Ing. Rudolfu Pecinovskému, CSc. za vstřícnost a ochotu, s jakou mi vycházel vstříc při psaní této práce, a předání hodnotných informací, které mi pomohly se zorientovat v problematice.

Rovněž děkuji mé rodině a přátelům za duševní podporu, která mi velice pomáhala během psaní diplomové práce.

Abstrakt

Práce se zabývá srovnáním platforem pro vývoj webových aplikací, konkrétně frameworku Nette pro platformu PHP a platformy Node.JS. Cílem práce je poskytnout komplexní obraz o rozdílech mezi zkoumanými platformami. Srovnání rozšiřuje předchozí diplomové práce, které porovnávaly frameworky Grails pro platformu Java, Zend pro platformu PHP a ASP.NET MVC. Tímto vzniká srovnání pěti platforem pro vývoj webových aplikací.

Součástí práce jsou webové aplikace, které jsem vyvinul nad frameworkem Nette v PHP a nad platformou Node.JS. Aplikace slouží zejména jako zdroj informací pro srovnání. Navíc mohou posloužit začínajícímu vývojáři pro seznámení se s danými platformami nebo jako studijní materiál pro zkoumání rozdílů mezi nimi. Aplikace vycházejí ze stejných uživatelských požadavků, jako aplikace vytvořené v předchozích diplomových pracích. Je tak možné srovnávat aplikace pro všech pět platforem.

Jednotlivé platformy jsou porovnávány ve více oblastech. První oblast se zaměřuje na fundamentální odlišnosti. Druhá oblast je zaměřená teoreticky, na architekturu platforem. Platformy jsou hodnoceny dle míry implementace architektonických vzorů definovaných Martinem Fowlerem. Další oblast se zabývá vlastním používáním platforem při vývoji webových aplikací. Oblast zahrnuje jak objektivní kritéria, tak osobní hodnocení nabyté právě z vývoje ukázkových aplikací. Poslední oblast srovnává dostupnou podporou daných platforem.

Výsledky práce poskytují komplexní srovnání pěti odlišných platforem dle stejných kritérií. Přínosem je také existence ukázkových aplikací, vyvinutých dle stejných požadavků. Případný zájemce si tak díky nim může vytvořit vlastní obrázek vycházející z odlišné implementace stejné funkcionality v různých platformách.

Klíčová slova

Nette Framework, Node.JS, srovnání, framework, PHP, webová aplikace

Abstract

The thesis is concerned with comparison of web application development platforms, namely Nette Framework on PHP platform and platform Node.JS. The objective is to provide complex picture of differences between the frameworks. Previous theses compared Grails framework on Java platform, Zend Framework on PHP platform and ASP.NET MVC platform. This thesis builds upon them thus provides comparison of five different platforms for web application development.

As a part of the thesis, I developed two web applications. One is in Nette Framework in PHP and one is in Node.JS. Their purpose is to be main source of information for the comparison. Furthermore, beginning developer learning new platform can use them as a study material. They can also help to examine platform differences. Applications I developed meet same user requirements as the applications contained in previous theses. Therefore it is possible to compare all five applications on five platforms with each other.

The platforms are compared in several areas. First area focuses on fundamental platform differences. Second area is more theoretical. It concerns the platform architecture. Platforms are judged by amount of the support for Martin Fowler's architectonical patterns. Next area concerns platforms' usage during the web application development. There are both objective criteria as well as subjective evaluation based on the application development included. The last area compares available support of the platforms.

Thesis provides complex comparison of five different platforms by same criteria. Developed web applications, based on same requirements, are also a benefit. Potential reader can form her own judgement based on the differences in implementation of same functionality she founds on different platforms.

Keywords

Nette Framework, Node.JS, comparison, framework, PHP, web application

Obsah

1 Úvod ... 7
1.1 Cíl práce ... 7
1.2 Očekávané přínosy práce ... 8
1.3 Okruh čtenářů ... 8
1.4 Předpoklady a omezení práce ... 9
1.5 Použité konvence ... 9
2 Vymezení základních pojmů ... 10
2.1 Framework ... 10
2.1.1 Frameworky pro rychlý vývoj aplikací ... 10
2.2 Mapování URL ... 11
2.3 Přesměrování a podstrčení stránky ... 11
3 Výběr frameworků ... 13
3.1 Důvody pro výběr frameworku Nette ... 13
3.2 Důvody pro výběr platformy Node.JS ... 13
4 Rešerše ... 14
4.1 Framework Nette ... 14
4.2 Platforma Node.JS ... 14
5 Popis jednotlivých platforem ... 16
5.1 Framework Nette ... 16
5.2 Platforma Node.JS ... 16
6 Definice kritérií pro srovnání ... 18
7 Požadavky na ukázkovou aplikaci ... 19
7.1 Účel aplikace ... 19
7.2 Formální požadavky ... 19
7.3 Popis aplikace internetový obchod ... 20
7.4 Omezení aplikace ... 20
8 Návrh architektury ... 21
8.1 Úpravy modelu ... 21
8.2 Výběr komponent pro Node.JS ... 22
9 Vlastní vývoj ukázkové aplikace ... 23

9.1 Pravidla a doporučení pro vývoj aplikací 23
9.1.1 Framework Nette 23
9.1.2 Platforma Node.JS 23
9.2 Volba IDE nástroje 24
9.2.1 Framework Nette 24
9.2.2 Platforma Node.JS 25
9.3 Instalace a zprovoznění 26
9.3.1 Instalace databáze 26
9.3.2 Framework Nette 27
9.3.3 Platforma Node.JS 27
9.4 Vytvoření projektu 27
9.4.1 Framework Nette 27
9.4.2 Platforma Node.JS 28
9.5 Adresářová struktura 29
9.5.1 Framework Nette 29
9.5.2 Platforma Node.JS 30
9.6 Nastavení aplikace 31
9.6.1 Framework Nette 31
9.6.2 Platforma Node.JS 33
9.7 Implementace modelu 33
9.7.1 Framework Nette 34
9.7.2 Platforma Node.JS 36
9.8 Implementace řadičů (controller) 37
9.8.1 Framework Nette 37
9.8.2 Platforma Node.JS 39
9.9 Implementace pohledů (view) 40
9.9.1 Framework Nette 40
9.9.2 Platforma Node.JS 41

10 Výsledky srovnání frameworku 42
10.1 Základní srovnání 42
10.2 Architektura frameworku 43
10.2.1 Vzory doménové logiky 44
10.2.2 Vzory datových zdrojů 44
10.2.3 Objektově-relační vzory chování 45
10.2.4 Objektově-relační strukturální vzory 45
10.2.5 Vzory pro objektově-relační mapování dat 46
10.2.6 Vzory pro webové prezentace 46

10.2.7 Vzory pro souběžné zpracování 47
10.3 Práce s frameworkem 48
10.3.1 Práce s modelem 49
10.3.2 Práce s řadiči 52
10.3.3 Práce s pohledy 56
10.3.4 Testování 58
10.3.5 Celkové osobní hodnocení 60
10.4 Podpora frameworku 62
11 Závěr 65
Příloha A: Ukázková aplikace 67
Terminologický slovník 68
Seznam literatury 70
Seznam obrázků a tabulek 73
Seznam obrázků 73
Seznam tabulek 73

1 Úvod

S dlouhodobým rozmachem internetu se zvyšuje i komplexita webových prezentací. Statické weby v podstatě již neexistují, byly nahrazeny dynamickými webovými aplikacemi, které jsou si často podobné jako vejce vejci. S rostoucími požadavky a tlakem na snižující se náklady vznikají typová řešení a nástroje usnadňující jejich tvorbu. Do této sekce řadíme také platformy pro tvorbu webových aplikací, hlavně webové frameworky.

Jak se mění technologie a trendy, mění se i webové frameworky a webové platformy. Nové vznikají a stávající podléhají změnám. Současným trendem u webových frameworků je koncept usnadňující rychlý vývoj a rychlé nasazení aplikace. Tento koncept je u různých frameworků podporován více či méně a ne vždy je zcela zřejmé, jak moc daný framework opravdu usnadní vývojáři tvorbu aplikace.

S rozvojem technologií se rozšiřuje i paleta dostupných platforem. Na jedné straně stojí velké a silné platformy jako .NET a Java, na druhé straně jednoduché a nové platformy jako Ruby on Rails nebo Node.JS. Může být zajímavé zjistit, jak velké spektrum funkcionality tento široký vějíř platforem sdílí.

Právě porovnáním platforem pro vývoj webových aplikací se zabývá tato práce. Navazuje na práci Františka Odehnala, který porovnával webové platformy Groovy nad Javou s Zend Frameworkem nad PHP, a na práci Petra Loudy, který ke srovnání přidal platformu ASP.NET MVC. K této trojici přidává tato práce srovnání s frameworkem Nette nad PHP a s platformou Node.JS a tím rozšiřuje výsledné srovnání na pět platforem.

1.1 Cíl práce

Vycházím z diplomové práce zabývající se srovnáním platforem Grails a Zend Framework (1) a práce rozšiřující toto porovnání o framework ASP.NET MVC (2).

Pro tuto diplomovou práci jsou definovány následující cíle:

1. Popsat základní architekturu a styl vývoje ve frameworku Nette na platformě PHP a na platformě Node.JS.
2. Demonstrovat styl vývoje ve frameworku Nette na platformě PHP a na platformě Node.JS vytvořením ukázkových aplikací ekvivalentních k aplikacím v (1) a (2) dle podobných požadavků.
3. Dle kritérií v (1) porovnat vlastnosti frameworku Nette na platformě PHP a platformy Node.JS s frameworky rozebíranými v (1) a (2).

Aby byla zajištěna co největší objektivita porovnání, návrh tvořených aplikací vychází u všech pěti verzí ze stejných požadavků.

Pro hodnocení byla použita u všech předchozích frameworků stejná kritéria (1). Mnou porovnávané frameworky jsou hodnoceny rovněž dle těchto hledisek.

1.2 Očekávané přínosy práce

Práce doplňuje srovnání frameworků Grails, Zend a ASP.NET MVC o další dva, a to Nette frameworkem a Node.JS. Takto vzniká ucelený přehled pěti frameworků pro vývoj webových aplikací na čtyřech platformách. Tento přehled může sloužit začínajícímu vývojáři jako orientační bod pro rozhodnutí, v jakém frameworku a platformě se nadále vzdělávat. Ukázkové aplikace mohou posloužit návrhářům a vývojářům k rozhodnutí, jaký framework a platformu použít pro jimi požadovanou funkcionalitu.

Dalším přínosem jsou naprogramované ukázkové aplikace, které až na minimální odchylky vycházejí ze stejných uživatelských požadavků (1). Dle těchto aplikací si může čtenář utvořit na jednotlivé frameworky vlastní náhled skrze odlišnosti v implementaci stejné funkcionality. Práce tak řeší problém nedostatku kvalitních příkladů zdrojových kódů komplexních webových aplikací dostupných na internetu, které by pro srovnání mohly sloužit.

1.3 Okruh čtenářů

Práce je určena programátorům, kteří se rozhodují, ve kterém frameworku se chtějí dále vzdělávat. Zejména pro české programátory může být zajímavé srovnání frameworku Zend nad platformou PHP s v Česku rozšířeným frameworkem Nette nad toutéž platformou. Například studenti často znají pouze jeden z těchto frameworků a nemají dostatečný rozhled pro kvalitní rozhodnutí o tom, kterou funkcionalitu plní ten který framework lépe.

Návrhářům a programátorům, kteří začínají nový projekt, může tato práce usnadnit jejich rozhodnutí pro konkrétní framework.

1.4 Předpoklady a omezení práce

Práce předpokládá od čtenářů základní znalosti ohledně vývoje webových aplikací, základní zkušenosti s programováním a základy objektově orientovaného návrhu. Čtenářům může pomoci, mají-li alespoň základní znalosti programovacích jazyků popisovaných platforem.

V práci používám kromě českých termínů i termíny anglické, a to tehdy, používá-li anglické termíny i komunita kolem daného frameworku nebo nenašel-li jsem pro termín vhodný český ekvivalent. Cílem je usnadnit čtenářům orientaci v dané problematice ušetřením používání nových a ne zcela známých termínů, byť to může být na úkor českého jazyka.

1.5 Použité konvence

V textu práce jsou použité následující grafické konvence:

- Některé názvy tříd, identifikátory, části zdrojových kódů a jiných příkazů v textu jsou vysázeny `proporcionálním písmem`.
- Ukázky zdrojových kódů jsou rovněž vysázeny neproporcionálním písmem na žlutém pozadí:

```
<?php echo "Hello World"; ?>
```

- V kapitolách porovnávajících oba frameworky je často na prvním řádku odstavce název **Nette** frameworku nebo **Node.JS** platformy vysázen tučně.
- Dodatky, zejména ve srovnávacích tabulkách, jsou zmíněny v poznámkách pod čarou.[1]

[1] Často jsou takto uvedené i URL, ze kterých byly dané informace čerpány. A to tehdy, pokud by v daném místě neměla citace výrazný smysl.

2 Vymezení základních pojmů

Tato kapitola si klade za cíl vymezit základní pojmy, které se používají v rámci celé práce. Méně podstatné pojmy lze nalézt v terminologickém slovníku na konci práce.

František Odehnal a Petr Louda ve svých pracích (1) (2) shrnuli některé pojmy velice důkladně. Následující podkapitoly proto jen stručně připomínají obsah těchto částí a jsou doplněny o některé další informace.

2.1 Framework

Existuje více pohledů na to, co je a co není framework. Omezím tedy svoji definici na uvedení definic jiných. Mark Clifton uvádí zajímavý popis funkčnosti frameworku (3):

- „it makes it easier to work with complex technologies,
- it ties together a bunch of discrete objects/components into something more useful,
- it forces the team (or just me) to implement code in a way that promotes consistent coding, fewer bugs, and more flexible applications,
- everyone can easily test and debug the code, even code that they didn't write.“

V překladu to znamená, že frameworky nám usnadňují práci s komplexními technologiemi, spojují více nezávislých objektů a tím vytvářejí něco s větší přidanou hodnotou. Frameworky nutí jedince implementovat kód v tak, aby upřednostňoval konzistentní způsob zápisu, byl méně náchylný k chybám a umožňoval vytvářet více flexibilní aplikace, které jsou snadněji odladitelné (3).

Frameworky je možné členit do skupin dle podporované funkcionality, například frameworky pro tvorbu webových aplikací. (2)

Frameworky pro tvorbu webových aplikací „by měly podporovat jednoduché vytvoření a zprovoznění autorizace a autentizace uživatelů, perzistenci dat, „management sezení“, caching, bezpečnost a administrativní rozhraní.“ (2)

2.1.1 Frameworky pro rychlý vývoj aplikací

Frameworky pro rychlý vývoj aplikací se svým charakterem odlišují od běžných frameworků. Dnešním trendem je upřednostňovaný MVC/MVP model, podpora pro mapování URL, cachování, JavaScriptové knihovny a určitou formu automatického testování. (1)

U rychle vyvíjených aplikací se počítá s menší složitostí aplikace. Často zde chybí platforma široké palety funkcionality, jako u enterprise aplikací, naopak se jde cestou co nejtenčí codebase.

2.2 Mapování URL

Mapování URL je v současnosti u webových frameworků oblíbená a hodně používaná metoda. Mapování URL funguje jako obousměrný překlad mezi URL adresou a logikou, která má zpracování vyřídit. (1)

URL mapování je výhodné pro SEO a umožňuje tvorbu tzv. „pěkných adres“, které jsou přívětivé pro uživatele namísto pro systém. Například místo adresy `http://www.dovolena.cz/zajezd.php?id=249423` je vygenerována adresa `http://www.dovolena.cz/zajezdy-do-chorvatska/2013-last-minute/makarska-hotel-riviera-23-zari-7dni`.

Takovéto adresy jsou nejen přívětivé pro vyhledávače, ale rovněž pro uživatele. A to už stojí za úvahu na ně myslet. Jsou-li adresy dobře koncipovány, může je uživatel chápat i jako složkovou strukturu, kdy se na úroveň o jedno výše dostane prostým umazáním části za posledním lomítkem.

2.3 Přesměrování a podstrčení stránky

Přesměrování stránky nastává v okamžiku, kdy je činnost controlleru ukončena a prohlížeči je odeslán návratový HTTP kód 3xx, často s odkazem vedoucím na jinou akci jiného controlleru. (1)

Výše uvedená technika se často používá jako prevence proti vícenásobnému odeslání formulářů. Pokud totiž po odeslání formuláře nepřesměrujeme uživatele, a ten obnoví aktuální stránku (nebo se na ní dostane tlačítkem zpět v prohlížeči), formulář se odešle znova, což není žádoucí. Z toho důvodu se uživatel přesměruje na další stránku, která zobrazuje vždy stejné údaje, závislé pouze na GET parametrech.

Podstrčení stránky je na rozdíl od přesměrování technikou existující ryze uvnitř frameworku. V zásadě se jedná o to, že akce controlleru v určité fázi přeruší své provádění a interně předá provádění jiné akci (jiného) controlleru, který požadavek vyřídí. (1)

Může se tak stát, že na URL, která by dle mapování měla být vyřízena nějakým controllerem, odpovídá někdo úplně jiný. Příkladem může být zobrazení přihlašovacího dialogu při neautorizovaném přístupu na jakoukoliv URL uvnitř klientské sekce.

3 Výběr frameworků

Kapitola rozebírá důvody výběr a zpracování těchto konkrétních frameworků v rámci srovnání.

Téma webových aplikací je v poslední době velmi široké, a pro jejich vývoj existuje mnoho platforem, programovacích jazyků a frameworků. Existuje již srovnání s vyspělými a na funkcionalitu bohatými platformami Java a .NET, stejně tak jako srovnání s dnes již 7 let starým a na funckionalitu rovněž bohatým Zend Frameworkem.

3.1 Důvody pro výběr frameworku Nette

Nette Framework má v české PHP komunitě velkou váhu. Existuje kolem něj početná komunita vývojářů, jsou pořádány diskuze i školení. (4) Framework používají některé z velkých firem v České republice. (5)

Framework je často srovnáván právě se Zend Frameworkem. Tato srovnání jsou ale kusá, týkají pouze dílčích funkcionalit a neposkytují komplexní pohled na srovnání těchto frameworků. (6)

Ze své zkušenosti znám mladé PHP vývojáře, kteří začínají programovat právě nad Nette frameworkem. V historii jsem měl možnost pracovat jak se Zend Frameworkem, tak s Nette. Uvítal jsem tedy příležitost provést komplexní porovnání těchto dvou frameworků.

3.2 Důvody pro výběr platformy Node.JS

Vlastnosti Node.JS platformy, jako asynchronní přístup ke vstupně-výstupním operacím a její snadná škálovatelnost, jsou velice zajímavé z hlediska jejího výběru pro vývoj aplikací na straně serveru (7). Platforma byla zveřejněna v roce 2009 a od té doby zažívá vzestup. Platforma je podporována a využívána významnými firmami, jako jsou Microsoft, eBay, LinkedIN, Yahoo a další. (8)

Z mého pohledu je zajímavé srovnat stabilní a silné platformy jako Java nebo .NET, nad kterými staví architektury Grails nebo ASP.NET MVC s vysoce modulární a rapidně se vyvíjející platformou, do které vkládá komunita kolem ní velká očekávání.

Z výše uvedeného důvodu jsem se rozhodl právě pro srovnání s Node.JS.

4 Rešerše

Téma webových frameworků je velice široké a hojně řešené. Tato kapitola se zaměřuje na přiblížení současného stavu probírané oblasti. Jejím cílem je zasadit tuto práci do kontextu existujících prací. Téma omezuji na práce z České republiky. Rovněž se zaměřuji na práce vydané v posledních 3 letech, tedy od roku 2011 včetně. Důvodem pro výběr tří let je to, že framework Nette prošel v těchto letech zásadními změnami, stejně jako platforma Node.JS. Práce starší než je tato doba považuji za zastaralé. Dalším důvodem pro tento limit je fakt, že práce, ze které vycházím (1) je také z roku 2011.

4.1 Framework Nette

Prací týkajících se Nette frameworku je k dispozici relativně hodně. Vybírám ty nejrelevantnější, které se týkají popisu funkcionality nebo jejího porovnání.

Jan Tölg se ve své bakalářské práci z roku 2013 zabývá Nette frameworkem a popisuje jeho vlastnosti (9). Práce je rozdělená na dvě části: teoretickou část popisující vlastnosti frameworku a dále praktickou část, ve které demonstruje vlastnosti frameworku na vytvořené aplikaci. Část popisující vlastnosti Nette frameworku je rozhodně přínosná, ovšem nehodí se pro hodnocení, neboť Tölg nejde tolik do hloubky z hlediska hodnocení architektury.

Petr Olišar ve své bakalářské práci z roku 2012 porovnává Zend framework a Nette framework (10). Porovnání se zaměřuje pouze na rozdíly ve funkcionalitě a základní architektonické rozdíly a více se zaměřuje na metriky výsledných aplikací, jako například rychlost běhu skriptu, apod.

Dalším vhodným úvodem do Nette frameworku je bezesporu seriál na serveru Zdroják, a to Začínáme s Nette framework (11). Článek hovoří nejen o frameworku jako takovém, ale rozebírá i koncepty Dependency Injection, což je rozhodně zajímavé téma nejen pro tento framework.

4.2 Platforma Node.JS

Prací týkajících se platformy Node.JS se vyskytuje výrazně méně, než prací týkajcích se Nette frameworku.

Oto Dočkal v roce 2013 ve své diplomové práci popisuje tvorbu CRM systému nad platformou Node.JS a frameworkem Express (12), což je řešení, které zčásti používám v této práci.

Bohužel nepoužívá MySQL databázi a pro přístup k ukládání dat zvolil MongoDB. Práce je vhodná akorát pro získání základního přehledu o Node.JS.

Z vhodných knih bych doporučil Node.js: Up and Running od vydavatelství O'Reilly (7). Kniha poskytuje základní vhled do architektury.

Zajímavé informace lze najít i v knize Professional Node.js: Building Javascript Based Scalable Software od vydavatelství Wronx (13). Kniha obsahuje zajímavé informace k vývoji dobře škálovatelných aplikací.

5 Popis jednotlivých platforem

Tato kapitola se zabývá popisem jednotlivých platforem pro vývoj webových aplikací.

5.1 Framework Nette

Nette framework vznikl v roce 2004. Od roku 2008 byl zpřístupněn jako open-source pod licencí vycházející z BSD licence. Hlavním vývojářem frameworku je David Grudl. Framework v současnosti patří organizaci Nette Foundation. (14)

Framework se snaží v maximální možné míře ulehčit vývoj webových aplikací. Přístup convention over configuration (15) urychluje tvorbu kódu a odebírá nutnost psaní zbytečných konstruktů.

Framework je vyvíjen nad platformou PHP. Současné verze frameworku využívají nejnovější vlastnosti jazyka dostupné od verze PHP 5.3, jako jsou jmenné prostory a late static binding a anonymní funkce.

Nette se prezentuje jako framework s mj. následujícími vlastnostmi: (4)

- vyzrálý objektový návrh,
- vysoká míra zabezpečení,
- existence kvalitních ladících nástrojů,
- dobrá rozšiřitelnost.

Poslední dostupná verze Nette byla uvolněna 6. března 2013, a to verze 2.0.10.

Framework poskytuje pro tvorbu pohledů vlastní šablonovací jazyk Latte, který je velice podobný jazyky Smarty.

Framework je navržen tak, aby byla umožněna jeho snadná konfigurovatelnost a rozšiřitelnost.

5.2 Platforma Node.JS

Téma běhu JavaScriptového prostředí na serveru se diskutuje už od roku 1996, kdy firma Netscape uveřejnila jazyk pro skriptování na straně serveru na bázi JavaScriptu. (16) Řešení se však příliš nerozšířilo. (17) Změnu přinesla až platforma Node.JS, která byla zveřejněna v roce 2009. (18)

Platforma Node.JS je navržena jako tenká platforma se základním množstvím funkcionality. Její návrh je od počátku kompletně asynchronně orientovaný. Veškeré vstupně/výstupní operace, jako například práce se soubory, databázová spojení, neblokují hlavní vlákno, ale pokračují až tehdy, když získají od protistrany nějaká data. Tento přístup umožňuje velice dobré škálování webových serverů (19).

Platforma Node.JS sama o sobě neposkytuje žádnou MVC funkcionalitu. Tato funkcionalita, stejně jako middlewarem pro vyřizování HTTP požadavků nebo připojení k databázi musí být doplněny externími komponentami.

Způsob organizace komponent zajišťuje systém na řízení závislostí NPM, který umožňuje z centrálního repositáře získávat komponenty, na kterých je daná aplikace závislá. (13)

6 Definice kritérií pro srovnání

Práce vychází z prací Odehnala (1) a Loudy (2). Stejně tak z těchto prací vychází i způsob hodnocení jednotlivých frameworků. Kritéria pro srovnání frameworků jsou uvedena v kapitole 3: Definice kritérií pro srovnání Odehnalovy práce.

Kritéria jsou členěna do základních čtyř skupin:

- základní porovnání,
- architektura frameworku,
- práce s frameworkem,
- podpora frameworku.

Tyto skupiny se zaměřují převážně na hodnocení architektury a domnívám se, že jsou stále platné. Případným námětem na rozšíření hodnocení by mohlo být rozšíření o hodnocení na náročnost jednotlivých zdrojů, jako například dobu běhu skriptu, paměťovou náročnost, aj. nebo rozšíření do hledisek bezpečnosti, možnosti integrace s webovými službami, atp. V této práci se ale držím původních kritérií.

Strukturu srovnání, jak je vidno v kapitole 10, jsem převzal rovněž z Odehnalovy práce. Výsledky hodnocení pro framework Grails a framework Zend jsou převzaty z téže práce. Výsledky pro ASP.NET MVC jsou převzaty z Loudovy práce.

Hodnotami, které byly aktualizovány, jsou hodnoty v kapitole 10.4 Podpora frameworku. Čísla v tabulce se totiž rychle mění, a ke všemu není směrodatné porovnávat historické a současné hodnoty. Všechny hodnoty jsou tedy aktualizované k současnosti.

Z důvodu úspory místa bylo nutné komentáře k hodnocení jednotlivých frameworků vyřadit z hodnotících tabulek. Díky tomu ale vzniká ucelený přehled srovnání pro všechny frameworky. Čtenář tak může získat rychlý přehled o srovnání frameworků. Pro podrobnější komentáře k převzatému hodnocení je však třeba nahlédnout do příslušných diplomových prací.

7 Požadavky na ukázkovou aplikaci

Kapitola definuje požadavky na ukázkovou aplikaci, které jsou shodné jak pro aplikaci vyvíjenou ve frameworku Nette tak pro aplikaci vyvíjenou ve frameworku Node.JS.

Požadavky na aplikaci byly definovány v (1), kapitola 6 Požadavky na ukázkovou aplikaci. Jelikož se návrh aplikace drží Odehnalových požadavků, omezím se v této kapitole na popsání odchylek od těchto požadavků.

7.1 Účel aplikace

Jak je napsáno v původních požadavcích, účelem vytvoření ukázkové aplikace není vytvoření plně funkční aplikace, ale dostatečně funkčního prototypu, který spolehlivě poslouží k demonstraci použité architektury.

Tvorba aplikace se zaměřuje na front-end, tedy veřejně přístupnou část. Administrační rozhraní není v požadavcích na aplikaci nijak řešeno a nebylo vyvíjeno.

7.2 Formální požadavky

Odehnal ve své práci definuje požadavky na tvorbu aplikace elektronického obchodu. Tato aplikace by měla splňovat následující kritéria (1):

- velikost modelu 15 – 25 doménových objektů (zhruba),
- základní autentizaci (přihlašování),
- mapování URL,
- stránkovací výpis (paginator).

V rámci vývoje se držím návrhu databázové struktury Odehnalovy aplikace pro Zend Framework včetně jmenných konvencí. Cílem je dodržet co nejvíce stejných konvencí napříč aplikacemi, abych usnadnil případným zájemcům jejich porovnávání.

Dalším požadavkem je kritérium stejného designu. Zde jsem rovněž zvolil jako základ Odehnalovu verzi aplikace pro Zend Framework. Designové komponenty, které nepodléhají srovnání, byly převzaty. Jmenovitě se jedná o:

- obrázky,
- klientské JavaScripty,

- CSS styly.

Šablony vzhledu budou přepsány pro konkrétní framework, ale s důrazem na maximální zachování původního HTML kódu, přestože tento kód nemusí být dle dnešních kritérií hodnocen jako čistě napsaný.

7.3 Popis aplikace internetový obchod

Aplikace internetového obchodu se chová dle popisu z (1), kapitoly 6.3 Popis aplikace internetový obchod.

7.4 Omezení aplikace

Jelikož se jedná o ukázkový prototyp, jehož účelem je srovnání, a ne hotovou aplikaci, nejsou v aplikaci dokončeny následující části (1):

- administrační rozhraní,
- ověření zvolené kombinace platby a dopravy,
- zobrazení košíku na základě URL.

Z důvodu zachování maximálního množství klientského kódu stejného se může jevit výsledný HTML kód nově vytvořených aplikací jako ne zcela kvalitní. Pokud bych některé konstrukce psal znova, rozhodně bych je navrhl jinak. Cílem ovšem není poskytnout kvalitní HTML kód, ale porovnat architekturu frameworků. Z tohoto důvodu jsem se držel HTML struktury použité ve vzorové aplikaci Zend Frameworku.

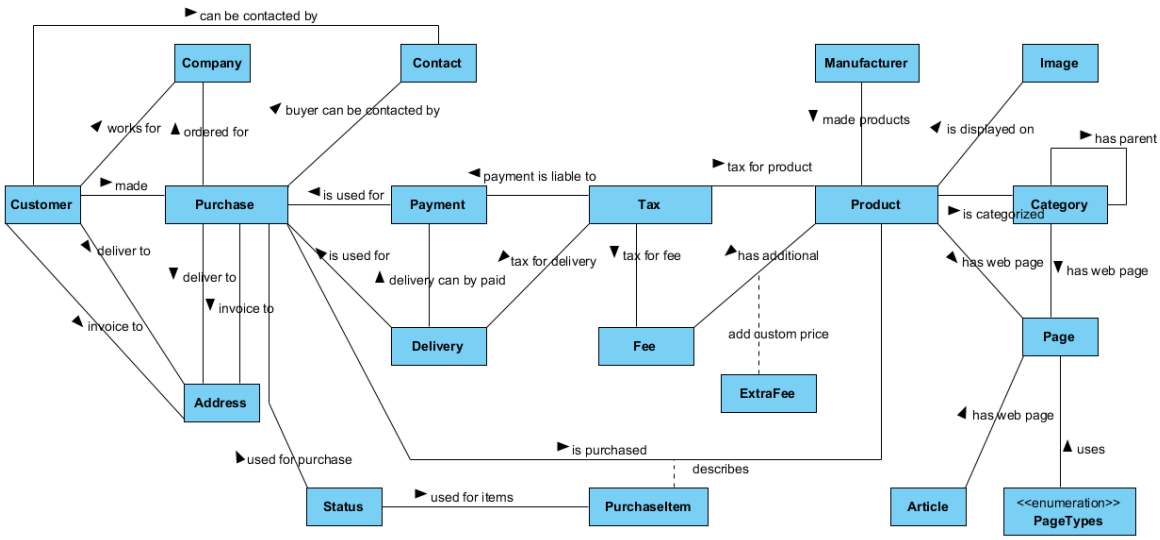
8 Návrh architektury

Návrh architektury aplikace vychází z návrhu architektury aplikace elektronického obchodu, popsaného v Odehnalově práci, kapitola 7 – Návrh architektury ukázkové aplikace (1). Mým cílem bylo držet se původního návrhu v maximální možné míře z toho důvodu, aby bylo možné porovnat výsledné aplikace co nejobjektivněji.

Uživatelské rozhraní aplikace je převzato z ukázkové aplikace naprogramované pro framework Zend (1). Každá z nově vytvořených aplikací používá jiný šablonovací systém, tedy přestože aplikace produkují obdobný HTML kód, jejich pozadí je odlišné. Do aplikací byly převzaty i CSS styly a ilustrační obrázky.

Pro obě nově vytvořené aplikace byl využit jazyk LESS pro tvorbu stylů. Využití jazyka LESS ovšem nijak neovlivňuje porovnání výsledných webových aplikací, neboť jeho použití by bylo možné i v kterékoliv jiné z porovnávaných aplikací. Důvodem použití je má zkušenost s tímto způsobem zápisu stylů a tedy značné zjednodušení jejich zápisu.

8.1 Úpravy modelu



Obrázek 1

Kompletní upravený diagram tříd pro Nette i Node.JS (zdroj: František Odehnal, upraveno autorem)

V rámci vývoje ukázkových aplikací jsem se rozhodl odstranit entitu Cart, která měla zajišťovat udržení informací o rozpracovaných objednávkách. Tyto informace jsou v obou aplikacích drženy mimo databázi, a to v session. Pokládám toto řešení za efektivnější. Pokud

by se totiž rapidně zvýšilo množství uživatelů v e-shopu, mohlo by to vést k přetížení databáze a zároveň plnění nerealizovanými objednávkami.

Dalším rozdílem je úprava mapperu pro entitu Category, která je, na rozdíl od Odehnalovy verze, schopna vracet akorát své děti. Jelikož oba frameworky podporují cachování, domnívám se, že je lepší pro účely vykreslování menu zjednodušit model a poté implementovat cachování na patřičném místě.

8.2 Výběr komponent pro Node.JS

U Node.JS je důležité zmínit, že samotná platforma toho moc neumí. Je tedy nutné ji doplnit dalšími komponentami, které poskytnou potřebnou funkcionalitu.

Analýzou balíčků v NPM repositáři jsem se rozhodl vybrat ty nejpoužívanější a z nich sestavil komponenty na míru pro ukázkovou aplikaci.

Z těch nejdůležitějších komponent uvádím:

- Connect middleware, zpracovávající HTTP požadavky
- Express.js framework pro webové aplikace
- Locomotive.js MVC framework rozšiřující funkcionalitu Express.js
- Jade šablonovací engine pro využití v pohledech
- Sequelize ORM nástroj pro tvorbu entit a připojování k databázi
- Q knihovna pro řízení tzv. „promises“
- Nodeunit pro unit testing

9 Vlastní vývoj ukázkové aplikace

Kapitola se věnuje vlastnímu vývoji ukázkových aplikací. Aplikace v souladu s požadavky definovanými v této práci v kapitole 7. Texty jsou doplněny ukázkami zdrojových kódů, případně ilustračních obrázků.

Kompletní zdrojové kódy aplikací jsou součástí přílohy.

9.1 Pravidla a doporučení pro vývoj aplikací

Kapitola popisuje pravidla a doporučení pro vývoj aplikací v jednotlivých frameworcích.

9.1.1 Framework Nette

Framework Nette nedefinuje explicitně žádné „best practices“, které by bylo vhodné dodržovat při vývoji. (4)

Přesto jsou zde ale standardy, které Nette preferuje:

- používání PHPDoc notace pro dokumentaci (20),
- využití Dependency Injection pro kompozici objektů (21),
- dodržování principů KISS a DRY.

Speciálně princip Dependency Injection mi přijde velice zajímavý. Celý způsob konfigurace objektů logiky skrz Neon soubory je založen na vhodném skládání a předávání objektů dle tohoto principu. Celé Nette je dle tohoto principu navrženo. Důsledkem je například to, že na rozdíl od Zend Frameworku v celém Nette není skoro k nalezení statická třída.

9.1.2 Platforma Node.JS

Pro vývoj nad platformou Node.JS se mi nepodařilo najít žádná konkrétní prověřená doporučení. Přesto je ale několik obecných doporučení, které je vhodné dodržovat.

Dokument JS: The Right Way (22) může sloužit jako rozcestník odkazující na různá doporučení a standardy, které je vhodné používat.

Google JavaScript Style Guide popisuje základní standardy pro psaní JavaScriptu (23). Mimo jiné například upozorňuje na způsob zápisu komentářů pomocí JSDoc notace.

Návrhové vzory implementované v JavaScriptu jsou popsány v dokumentu od Addy Osmaniho. (24)

9.2 Volba IDE nástroje

Kapitola popisuje má osobní doporučení pro volbu IDE nástrojů pro vývoj jednotlivých aplikací.

9.2.1 Framework Nette

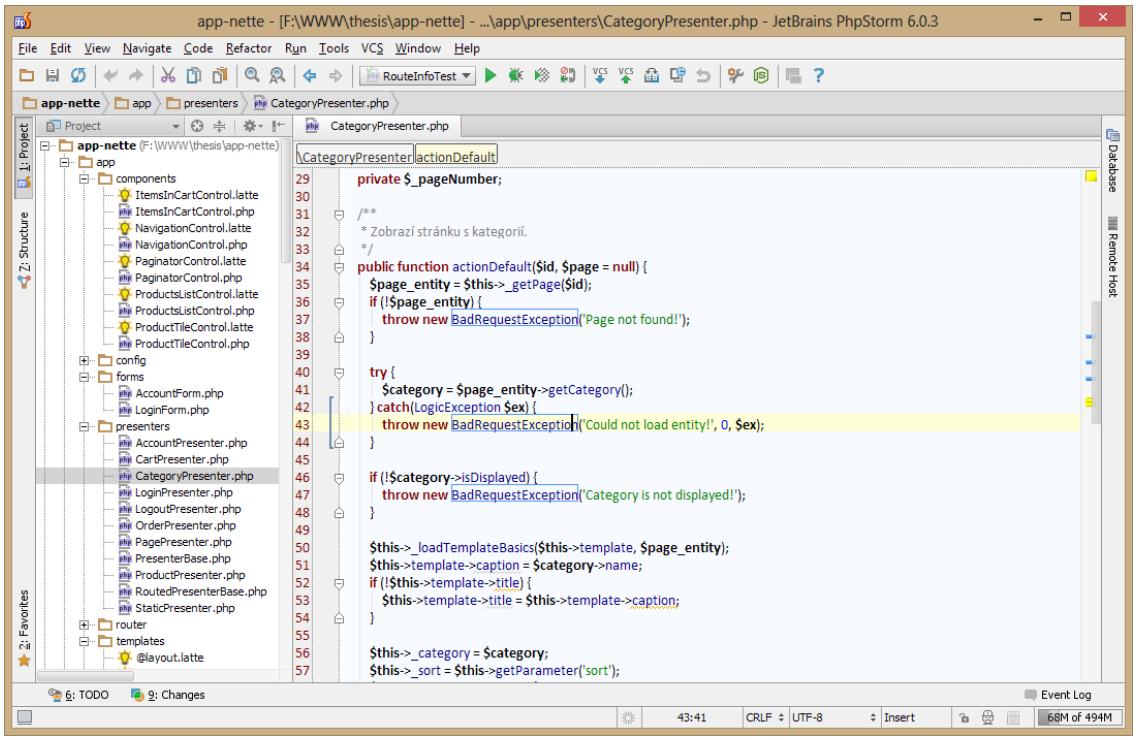
Přestože PHP editorů existuje celá řada (25), žádný z editorů plně nepodporuje integraci s frameworkem Nette. Třídy frameworku jsou velice dobře zdokumentované, a kvalitní PHP editory by si měly poradit s napovídáním a kontrolou kódu aplikační logiky a presenterů. Největším problémem je absence zvýrazňování, napovídání a kontroly pro šablonovací jazyk Latte. Tento jazyk je však svojí syntaxí velmi podobný šablonovacímu jazyku Smarty. Je vhodné si nastavit editor tak, aby zvýrazňoval `.latte` šablony alespoň dle Smarty konvencí.

Pro podporu Nette frameworku a šablonovacího systému Latte jsou v současnosti vyvíjeny dva pluginy:

- plugin pro NetBeans IDE (26),
- plugin pro PhpStorm IDE (27).

Já osobně jsem si zvykl na placené PhpStorm IDE od firmy JetBrains, přestože nepodporuje v současnosti plně Nette framework. Důvodem pro moji volbu je velice kvalitní podpora psaní zdrojového kódu, refaktoringu, testování, deploymentu a všech ostatních aspektů, které při vývoji potřebuji.

V rámci osobního hodnocení v kapitole 10.3 budu hodnotit framework z hlediska jeho používání právě v PhpStorm IDE.



Obrázek 2

Editace projektu v PhpStorm IDE (zdroj: autor)

9.2.2 Platforma Node.JS

Vývoj aplikací v Node.JS je v současnosti podporován následujícími editory (28):

- placené JetBrains WebStorm/PhpStorm IDE,
- placené Komodo IDE (plná podpora pro ladění Node.js od verze 8),
- Nodeclipse – plugin pro Node.JS do Eclipse IDE,
- Cloud9,
- vim,
- Sublime Text.

Pro vývoj v PHP již používám editor JetBrains PhpStorm, jak uvádím v kapitole 9.2.1. Z toho důvodu jsem se rozhodl ten samý editor využít i pro editaci Node.JS projektu.

Rozdíl mezi JetBrains PhpStorm a WebStorm IDE je v tom, které z pluginů jsou defaultně předinstalovány. Je tedy možné do PhpStorm IDE doinstalovat plugin pro Node.JS (což byl můj případ) nebo zakoupit WebStorm IDE (do kterého lze opět doinstalovat plugin pro PHP).

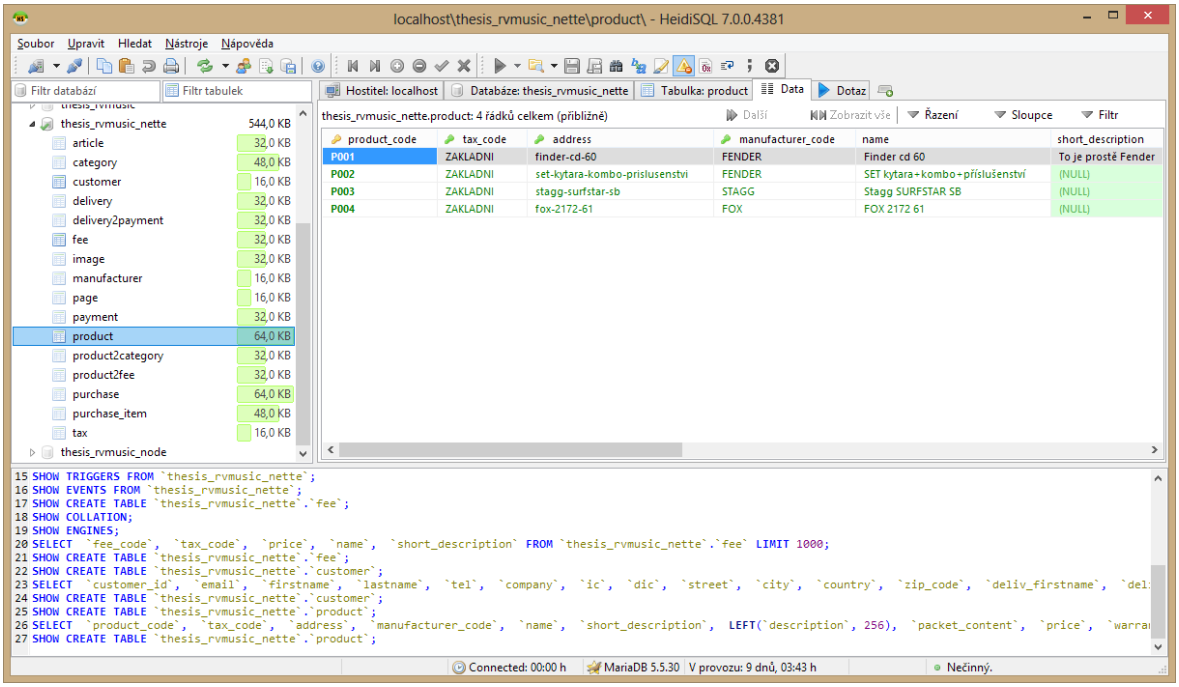
9.3 Instalace a zprovoznění

Kapitola popisuje rozběhnutí základního prostředí pro oba frameworky.

Aplikace pro oba frameworky byly vyvíjeny na platformě Windows 8 64-bit. Domnívám se, že aplikace by měly být bez problémů funkční na jakékoliv jiné platformě Windows, s případnými úpravami na jiných platformách.

9.3.1 Instalace databáze

Obě aplikace vyžadují ke svému provozu MySQL databázi. Na vývojové platformě mám nainstalovanou kompatibilní alternativu k MySQL, a to MariaDB[2]. Databáze je k dispozici v instalačním balíčku společně s kvalitním GUI pro správu databáze, programem HeidiSQL.



Obrázek 3

Program HeidiSQL pro správu MySQL-kompatibilní databáze (zdroj: autor)

Máme-li nainstalovaný MySQL-kompatibilní server, vytvoříme v něm dvě databáze, pro Nette aplikaci a Node aplikaci, které poté naplníme vzorovými daty.

Vzorová data pro databázi jsou k dispozici v příloze s programy.

[2] https://mariadb.org/

9.3.2 Framework Nette

Nette framework vyžaduje ke svému běhu webový server a PHP platformu. Přestože PHP obsahuje od verze 5.4 vestavěný webový server, tento je určen pouze pro vývoj a ne pro produkční nasazení. (29) Z toho důvodu používám webový server Apache, na kterém je PHP nainstalováno jako FastCGI modul.

Uživatelům, kteří s PHP začínají, bych doporučil instalaci některého z WAMP serverů[3], který jim zajistí funkcionalitu jak webového serveru, tak platformy PHP.

9.3.3 Platforma Node.JS

Platforma Node.JS vyžaduje ke svému běhu akorát instalaci Node.JS. Tato je dostupná na stránkách Node.JS[4] ke stažení. Platforma obsahuje vestavěný webový server, není tedy třeba žádný jiný instalovat.

Po nainstalování Node.JS je dát o jeho existenci vědět PhpStorm IDE v nastavení projektu.

9.4 Vytvoření projektu

Kapitola popisuje, jakým způsobem se pro obě aplikace vytvoří základní projekt, nad kterým se poté dále pracuje.

9.4.1 Framework Nette

Nette umožňuje několik způsobů vytvoření základního projektu:

1. preferovaný způsob stažení skeletu aplikace pomocí Composeru[5],
2. ruční vytvoření adresářové struktury a vytvoření potřebných souborů.

Já jsem se rozhodl pro první způsob, tedy stažený skeletu aplikace pomocí Composeru. Pro tento úkol je nutné stáhnout do počítače archiv `composer.phar`, což je ve zkratce správce balíčků.

Poté stačí spustit následující příkaz (po úpravě cest k souborům a adresářům):

---

[3] Viz https://en.wikipedia.org/wiki/WAMP

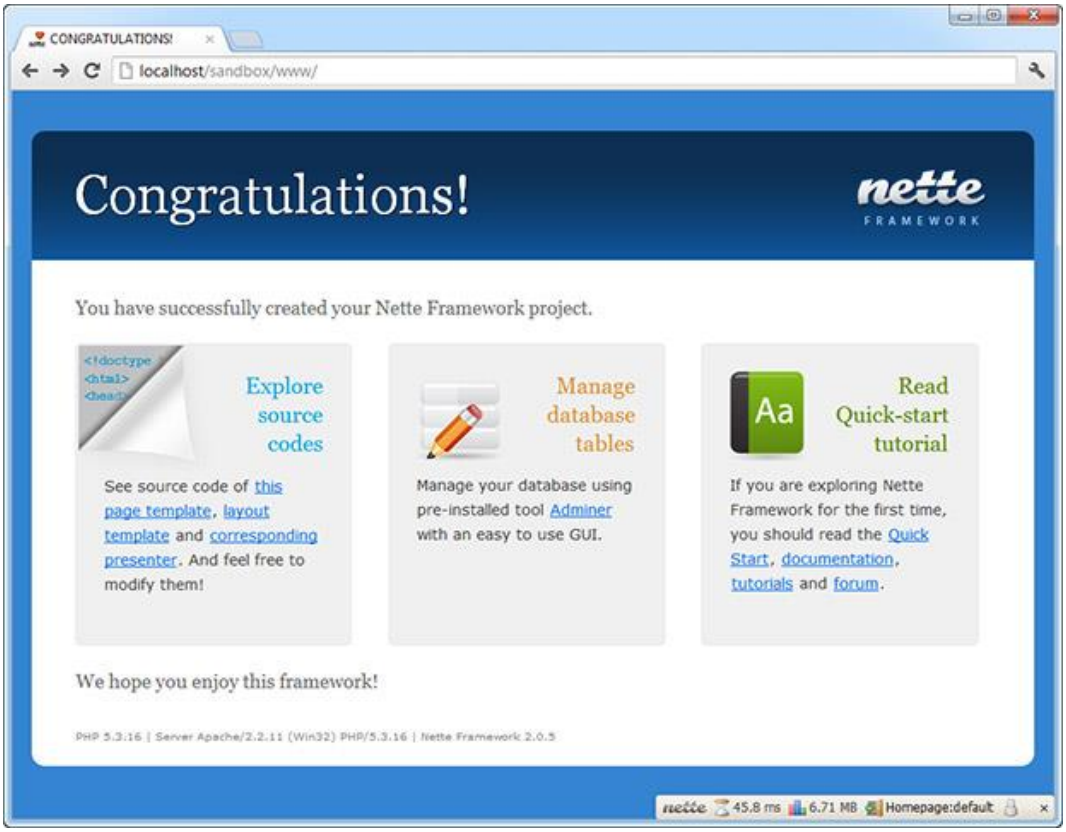
[4] Viz http://nodejs.org/

[5] Viz http://doc.nette.org/cs/book/start

Výpis 1: Vytvoření počátečního projektu Nette (zdroj: nette.org)

```
php composer.phar create-project nette/sandbox [cílová.složka] dev-release-2.0.x
```

Příkaz po svém ukončení vytvoří v cílové složce skelet Nette projektu. Pokud si poté otevřeme v prohlížeči adresu dotyčném složky na lokálním webovém serveru, dostaneme zhruba tuto stránku:



Obrázek 4

Počáteční aplikace Nette po spuštění v prohlížeči (zdroj: nette.org)

9.4.2 Platforma Node.JS

Prvním úkolem pro úspěšné vytvoření projektu bude instalace Locomotive, které nám pomůže vytvořit skelet projektu. Do skeletu poté doplníme závislosti na ostatních komponentách.

Nejjednodušším způsobem je vytvořit v cílovém adresáři soubor `package.json` s tímto obsahem:

Výpis 2: Obsah soouboru package.json (zdroj: nette.org)

```
{
    "name": "app-nodejs",
```

```
    "description": "An example e-shop application",
    "author": "Michal Kočárek <michal.kocarek@brainbox.cz>",
    "version": "1.0.0",
    "private": true,
    "dependencies": {
          "locomotive": "0.3.x"
    }
}
```

Poté ve stejném adresáři spustíme následující příkaz:

Výpis 3: Instalace Locomotive (zdroj: nette.org)

```
npm update
```

Pokud se podaří, měl by se ve složce objevit adresář `node_modules`, obsahující adresář locomotive. Dále budeme pokračovat podle manuálu na stránkách frameworku[6] a vytvoříme prvotní skelet aplikace.

9.5 Adresářová struktura

Kapitola popisuje typické uspořádání adresářové struktury pro každý z frameworků.

9.5.1 Framework Nette

Nette aplikace vytvoří ve svém adresáři několik složek (např. `app`, `libs`, `temp` a `www`), které slouží různým účelům. Složka `www` je kořenovým adresářem webu (na Apachi známá např. pod direktivou `DocumentRoot`). Umístění ostatních složek mimo kořen webu zvyšuje zabezpečení, protože se nemůže stát, že by citlivé soubory byly přístupné z webového serveru přes pouhou URL.

Výpis 4: Struktura Nette aplikace (zdroj: nette.org)

```
www/                    ← kořenový adresář webového serveru
└── sandbox/
    ├── app/            ← místo pro naši aplikaci
    │   ├── config/     ← konfigurační soubory
    │   ├── model/      ← třídy modelové vrstvy
    │   ├── presenters/ ← třídy presenterů
    │   ├── router/     ← továrnička na routy
```

[6] viz http://locomotivejs.org/guide/get-started/

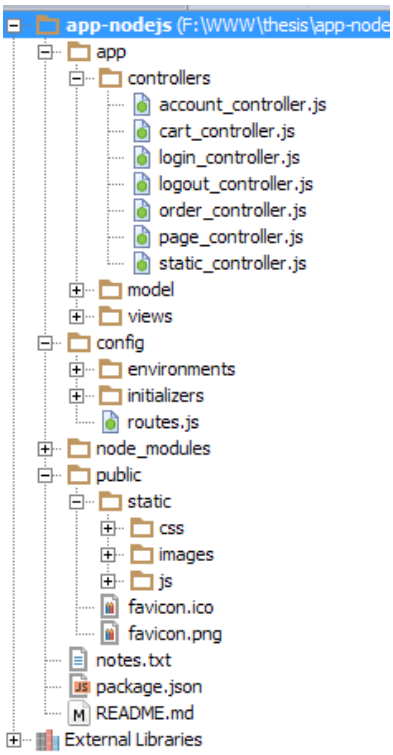
```
│   ├── templates/    ← šablony
│   └── bootstrap.php ← zaváděcí soubor
│
├── libs/             ← knihovny, které bude naše aplikace využívat
│   └── Nette/        ← jako třeba váš oblíbený framework :-)
│
├── log/              ← zde se v produkčním módu logují chybová hlášení
├── temp/             ← místo pro dočasné soubory cache nebo session
├── test/             ← unit testy naší aplikace
└── www/              ← složka veřejně přístupná z webu
```

U Nette je zajímavé, že strukturu aplikace lze libovolně upravovat. Každý z adresářů se může jmenovat jinak a dokonce může být i jinde umístěn. Změna cesty je pouze otázkou úpravy konfigurační direktivy buď v souborech `.neon` nebo v souboru `www/index.php`.

Nette oplývá elegantní vlastností pro autoloading tříd. Vlastnost se nazývá RobotLoader. Instance této třídy proskenuje předem definované adresáře a zapamatuje si, kde se nachází definice kterých tříd a rozhraní. Při příštím požadavku se načte správný soubor. Mapování názvů tříd na soubory se kešuje, aby se nemusely skenovat všechny soubory při každém novém požadavku. V tomto se Nette liší od Zend Frameworku, které rovněž poskytuje logiku pro autoloading, ale třídy musí pečlivě dodržovat jmenné konvence a uspořádání ve složkách, aby je Zend našel.

9.5.2 Platforma Node.JS

Aplikace vytvoří složku `app`, `config`, `node_modules` a `public`. Adresář public je nastaven jako kořen webového serveru po spuštění. Umístění ostatních složek mimo kořenový adresář webu zvyšuje bezpečnost, protože útočník se nemá jak dostat k potenciálně nebezpečným souborům pouhým zadáním URL adresy.



Obrázek 5

Znázornění struktury složek webové aplikace nad Node.JS (zdroj: autor)

V porovnání s Nette je zde struktura pevně dána a lze ji upravovat jen velice těžko. Stejně tak s konvencemi pro názvy souborů je třeba se smířit.

9.6 Nastavení aplikace

Tato kapitola popisuje, jakým způsobem se provádí základní nastavení aplikací pro jednotlivé frameworky.

9.6.1 Framework Nette

Stěžejním konfiguračním bodem v Nette je soubor `config.neon`[7]. Souborem je možné ovlivňovat tato nastavení:

- konfigurační direktivy PHP,
- vlastní proměnné, které jsou přístupné v aplikaci,
- definici služeb, které jsou přístupné v aplikaci,

[7] viz http://doc.nette.org/cs/configuring

- úpravu či záměnu existujících služeb poskytovaných frameworkem,
- ovlivnění výše uvedených nastavení pro jednotlivá běhová prostředí.

Při spuštění Nette proskenuje konfigurační soubor a vygeneruje dle něj definici PHP třídy systémového kontejneru (s názvem `SystemContainer`), kterou poté načte. Třída je přístupná z presenterů pod property `$this->context`, a obsahuje odkazy na všechny proměnné a služby, které jsme si v souboru definovali.

Tento koncept je velice silný, a ještě ke všemu usnadňuje vývoj. Jelikož Nette má properties, které vrací `SystemContainer` uveden název této třídy, IDE po prvním načtení konfiguračního souboru proskenuje i vygenerovanou definici třídy kontejneru, která je rovněž správně zdokumentována, a poté nabízí kvalitní napovídání pro veškeré služby.

Výpis 5: Ukázka části config.neon souboru (zdroj: autor)

```
common:
    parameters: # proměnné dostupné ve skriptu
           defaultRoute: 'Static:default'
           foo: 1234
           # … zkráceno

    php: # konfigurační direktivy pro PHP
           date.timezone: Europe/Prague

    nette: # konfigurace Nette
           application:
                  errorPresenter: Error

           database: # definice připojení k databázi
                  dsn: 'mysql:host=localhost;dbname=thesis_rvmusic_nette'
                  user: 'user'
                  password: '******'

           session:
                  autoStart: true
                  expiration: 14 days
                  save_path: "%tempDir%/sessions"

    services: # definice vlastních služeb dostupných ve skriptu
           imagePathProvider: RvMusic\ImagePathProvider('static/images')
           database: @nette.database.default
           authenticator: RvMusic\Authenticator(@mappers::getMapper('Customer'))
           routerFactory: RouterFactory
           mappers: RvMusic\ MapperService(@database, 'RvMusic\Entities')
           templateHelpers: RvMusic\Templating\Helpers
production < common:
development < common:
```

Ve výpisu kódu konfiguračního souboru je nejzajímavější část `services`. V zásadě každý z řádků znamená, že pro danou službu bude vytvořena instance dané třídy. Třídy jsou vytvářeny až na vyžádání, takže definice málo používaných služeb nezatěžují výrazně aplikaci.

Notace se zavináčem umožňuje odkazovat na jiné služby, je tedy možné předat konstruktoru jedné služby jinou instanci. Notace s procenty umí předávat proměnné. Tak je možné zase předat parametry ze začátku konfiguračního souboru službám.

Myslím si, že Nette Framework má konfiguraci vyřešenou velice dobře. Veškerá byrokracie spojená s parsováním `.neon` souboru a generováním systémového kontejneru je prováděna pouze při změně konfiguračního souboru. Poté se už načítá PHP soubor.

9.6.2 Platforma Node.JS

V rámci platformy Node.JS a frameworku Locomotive je možné rozlišovat nastavení pro jednotlivá běhová prostředí (development, production). Dle těchto prostředí je poté při konfiguraci volán soubor `config/environments/development.js` nebo `config/environments/production.js`.

Konfigurační soubory jsou standardní JavaScriptové soubory.

Zarážející pro mne na první pohled bylo, v jakém kontextu se soubory spouští, jaký objekt je nastaven na hodnotu `this`, atp. Toto je věc, která nebyla v dokumentaci řečená a musel jsem na ni přijít sám.

Zajímavým konceptem konfigurace jsou soubory v adresáři `config/initializers`. Adresář je proskenován a všechny soubory v něm jsou postupně načteny. Je tak tedy možné oddělit konfigurace jednotlivých komponent od sebe.

9.7 Implementace modelu

Tato kapitola popisuje implementaci tříd doménové logiky v jednotlivých aplikacích.

9.7.1 Framework Nette

Pro účely ukázkové aplikace bylo třeba navrhnout modelové třídy dle zadání. Nette Framework neobsahuje funkcionalitu, která by usnadnila tvorbu modelových tříd, ani funkcionalitu, která by usnadnila mapování entit modelu na entity v databázi.

Z tohoto důvodu bylo nutné vytvořit vlastní řešení pro mapování modelu.

Nette Framework nabízí kvalitní objekt typu Active Record pro práci s databází. Je vytvořit logiku, která data z databáze přemapuje na entity modelu.

Výsledkem návrhu byla kombinace tříd:

- `MapperService` – drží si připojení k databázi, je schopen vytvořit třídu libovolného mapperu
- `BaseMapper` – abstraktní třída mapperu, ze které budou dědit její konkrétní instance; poskytne podporu pro usnadnění zápisu mapování a společnou funkcionalitu pro získávání a ukládání entit
- `BaseEntity` – abstraktní třída entity modelu, drží si vazbu na svůj mapper; poskytuje potomkům společnou funkcionalitu pro definici vlastností entity.

BaseEntity

Primárním cílem při návrhu třídy bylo usnadnění práce s potomky, konkrétními entitami. Z tohoto důvodu byla ve třídě naprogramována „magie“, která pomocí getterů a setterů zajišťuje přístup k properties dané třídy. Properties jsou definovány pouze pomocí PHPDoc notace a logika třídy BaseEntity je zodpovědná za zařízení čtení a zápisu těchto properties.

Výpis 6: Výpis PHP třídy entity výrobce (zdroj: autor)

```
namespace RvMusic\Entities;
use RvMusic\Entities\BaseEntity;

/**
 * Instance této třídy zastupuje jednu entitu výrobce.
 *
 * @property string $manufacturerCode
 * @property string $name
 */
class Manufacturer extends BaseEntity {

}
```

Na příkladu výše je možné vidět plně funkční třídu výrobce. Přestože properties jsou definovány jen pomocí PHPDocu, přístup k nim je možný pomocí standardní notace `$manufacturer->name`.

Třída entity získává při svém vzniku instanci mapperu, který ji vytvořil. Mapper je zodpovědný za dotazy do databáze a samotná entita vůbec s databází nekomunikuje.

BaseMapper

BaseMapper je abstraktní třída usnadňující svým potomkům mapování tabulky v databázi na entitu. Pro toto usnadnění využívá třída PHPDoc notaci, podobně jako třída BaseEntity.

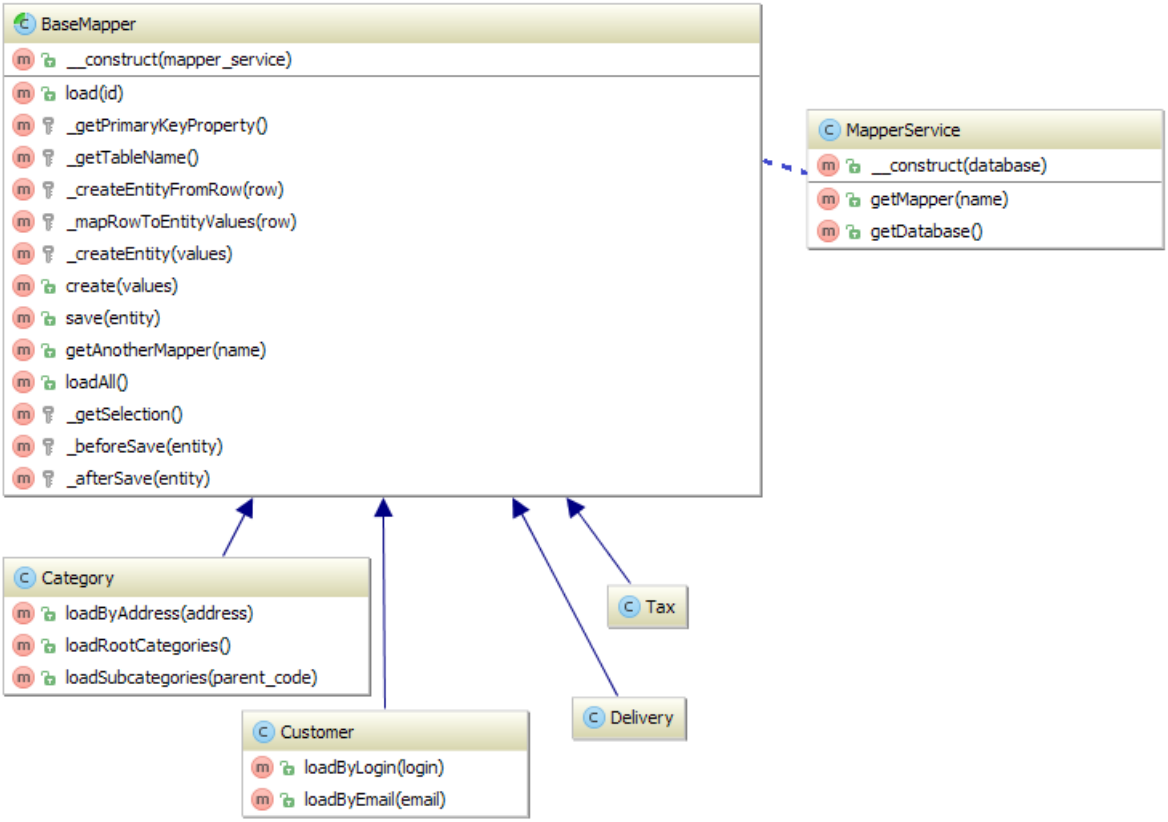
Výpis 7: Výpis PHP třídy mappery výrobce (zdroj: autor)

```
namespace RvMusic\DatabaseMappers;

/**
 * Instance této třídy zastupuje mapper pro třídu Page z databáze.
 *
 * @entity-classname \RvMusic\Entities\Manufacturer
 * @mapped-table manufacturer
 * @mapped-primary manufacturer_code
 * @map-column manufacturer_code => manufacturerCode
 * @map-column name
 */
class Manufacturer extends BaseMapper {

}
```

Jak je vidět v příkladu výše, PHPDoc notace popisuje kompletní informace o mapování tabulky na třídu entity. Tyto informace třídě stačí k tomu, aby byla schopna

- vrátit entitu dle primárního klíče,
- uložit entitu,
- vytvořit novou entitu.

Pro další funkcionalitu je nutné do mapperu doprogramovat metody, které mají přístup k širšímu spektru funkcionality, včetně přístupu k databázi.



Obrázek 6

Diagram vybraných tříd mapperu znázorňující základ datového modelu (zdroj: nette.org)

Obrázek výše zobrazuje model tříd Mapperů.

9.7.2 Platforma Node.JS

Komponenta Sequelize poskytuje docela slušné zázemí pro definici modelů. Přesto bylo nutné před jejím využitím trochu hledat, abych našel řešení, které umožňuje definovat každý z modelů v jiném souboru.

Sequelize mimo jiné umí dle definovaného modelu aktualizovat strukturu databáze. Tento přístup jsem se ale nerozhodl použít, a naopak jsem dle databáze ručně vytvářel jednotlivé modely.

Po definici třídy `Provider.js`, zastupující poskytovatele všech modelů jsem definoval jednotlivé entity.

Výpis 8: Ukázka entity výrobce v Node.JS (zdroj: autor)

```
/** @type {Provider} */
var orm = require('../Provider.js');
Seq = orm.Seq();
```

```
module.exports = {
    options: {
          tableName: 'manufacturer'
    },
    model: {
          manufacturer_code: {type: Seq.STRING, primaryKey: true},
          name: Seq.STRING
    },
    relations: {

    }
};
```

Po definici modelu Sequelize vytvoří třídu modelu pro danou tabulku, která má informace o tom, jak její tabulka vypadá, a je schopna vracet entity pro jednotlivé řádky.

Entitě je možné v definici doplnit metody, které budou dostupné pro danou instanci. Dále je možné jí doplnit metody, které budou dostupné pouze na třídě modelu.

9.8 Implementace řadičů (controller)

Kapitola popisuje způsob implementace řadičů v jednotlivých frameworcích.

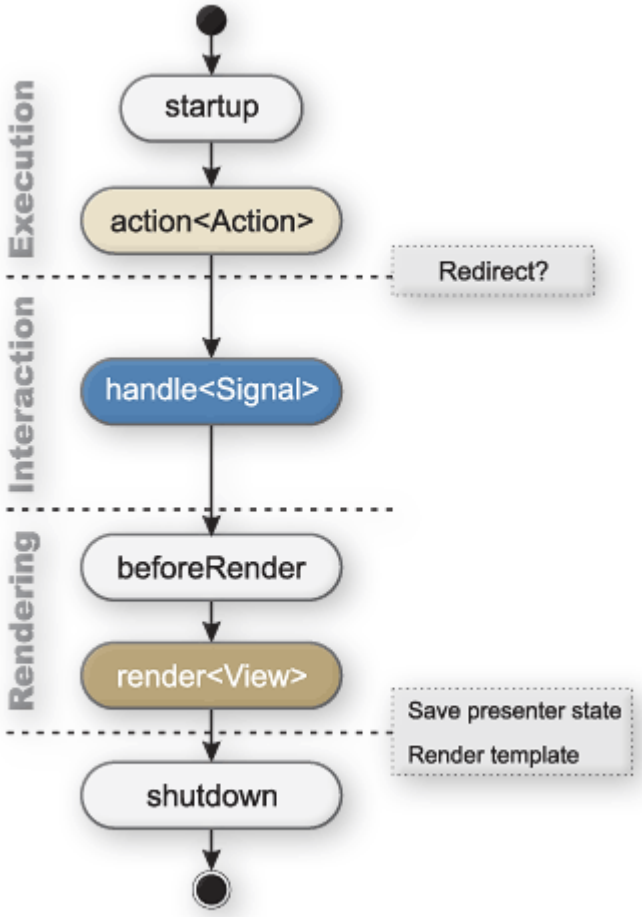
9.8.1 Framework Nette

Nette Framework namísto controllerů v původním slova smyslu využívá presentery[8]. Rozdílem je to, že presenter je z hlediska odpovědnosti tenčí, a spíše předává informace mezi modelem a pohledem, zatímco controller může obsahovat i část aplikační logiky.

Presentery se definují jako potomci třídy `Nette\Application\UI\Presenter`. Samozřejmě je možné využívat dědičnost, a je to i vhodné – například pro vytvoření obecného presenteru pro danou aplikaci, který pomůže naplnit layout společnými informacemi.

Presenter podporuje princip Convention over Configuration (15). Tedy vhodně zvolené názvy metod určují podporované akce, případně zachycené signály a možnou logiku pro vykreslené pohledy. Podrobný náhled na životní cyklus presenteru je vidět na obrázku níže.

[8] Viz http://doc.nette.org/cs/presenters



Obrázek 7
Životní cyklus presenteru (zdroj: nette.org)

Metoda `action…()` je hlavním prvkem presenteru, zodpovědná za zpracování požadavku.

Část logiky pro vykreslení pohledů je skryta ve třídách řadiče. Konkrétně se jedná o přípravu dat pro jejich vykreslení v pohledech před tím, než jsou předány šabloně.

Mapování URL

Nette Framework poskytuje solidní základ pro mapování URL. Lze-li přímo z adresy odvodit název presenteru, případně akce, definují se jednotlivé cesty pseudoregulérními výrazy.

Výpis 9: Ukázka definice několika cest (zdroj: autor)

```
new Route('<presenter>/<action>[/<id>]', 'Static:default');
new Route('index.php', ':Static:default', Route::ONE_WAY);
```

Na ukázce kódu výše vidíme, jakým způsobem se definuje obecná URL pro libovolný presenter, akci a volitelné ID. Například adresa `example.com/order/view/1` se poté přeloží na volání `OrderPresenter->actionView(1)`.

Framework poskytuje metodu presenteru `link()`, která je schopná vytvořit URL odkaz pro zadaný presenter, akci a parametry. Díky této funkcionalitě vlastně nemusíme nikde v aplikaci využívat absolutní URL, ale vystačíme si s orientací podle názvů modulů, presenterů a akcí.

Nette dbá na kanonizaci URL. Pokud je tedy možné se na jeden zdroj dostat více URL, Nette přesměruje uživatele z ostatních URL na tu primární.

Vlastní URL mapper

Ukázková aplikace definuje požadavek, aby všechny výrobky, kategorie a stránky měly URL uloženou v databázové tabulce. To, která entita se nakonec zobrazí, se rozhoduje dle konkrétního URL řetězce.

Abychom byli schopni dosáhnout této funkcionality, musíme v Nette naprogramovat vlastní Router.

Router je třída, která je zodpovědná za dvě věci:

- překlad URL adresy na presenter, akci a parametry,
- překlad presenteru, akce a parametrů na URL adresu.

9.8.2 Platforma Node.JS

V Locomotive jsou controllery standardními objekty, jejichž metody se musí jmenovat dle daného schématu. Locomotive uznává teorii MVC oproti MVP, tedy širší vrstvu logiky v controlleru.

Pro zadaný controller je vyhledána akce, a ta je poté zavolána.

Mapování URL

Funkcionalita mapování URL mě bohužel zklamala. Nenašel jsem žádný příkaz pro variabilní mapování pro libovolné množství controllerů tak, jak to má například Nette. Rovněž jsem nenašel zdokumentovanou možnost implementace vlastního mapovače.

Nakonec jsem se musel spokojit s taxativně vymezeným seznamem mapy URL na controllery a akce.

9.9 Implementace pohledů (view)

Kapitola popisuje způsob implementace pohledů v jednotlivých frameworcích.

9.9.1 Framework Nette

Pohledy jsou vyvolávány presenterem, který během svého workflow určí, který pohled se má zavolat. Pohledu jsou poté předány proměnné z presenteru, je zpracován `.latte` soubor s pohledem a vygenerován výstup.

Latte šablony

Latte soubory používají syntaxi obdobnou Smarty šablonovacímu jazyku.[9] Stejně, jako Smarty, i Nette transpiluje šablony před prováděním do PHP kódu. Šablony obsahují již v základní verzi dostatečné množství maker a filtrů pro to, aby umožnily velice kvalitní výpis dat.

Další výhodou šablonovacího systému je důraz na bezpečnost. Fakt, že při standardním volání výpisu jsou proměnné escapovány (a to v závislosti na kontextu, do kterého jsou vypisovány), výrazně zvyšuje bezpečnost výsledných skriptů.

Výpis 10: Ukázka části config.neon souboru (zdroj: autor)

```
{$promenna}  ← vypíše escapovaný obsah proměnné
{!$promenna} ← vypíše obsah proměnný tak, jak je
```

Samozřejmostí v Latte šablonách je i sdílení společného souboru pro vzhled, tzv. layoutu. Soubor s layoutem je hledán automaticky, ale je možné v šabloně nastavit svůj vlastní.

Komponenty

Nette umožňuje definování komponent, což jsou vykreslitelné entity s vlastní logikou a vlastní šablonou. Komponenty usnadňují sdílení společného kódu napříč jednotlivými šablonami. Rovněž umožňují logiku, která se týká ryze výpisu, přesunout právě blíž k danému místu, kde se s logikou pracuje.

[9] Viz http://doc.nette.org/cs/templating

Strukturování pohledů v ukázkové aplikaci

V ukázkové aplikaci jsem založil jeden soubor s layoutem, který sdílí všechny šablony pohledů.

Pro zjednodušení logiky jsem si definoval následující komponenty:

- komponenta vykreslující počet položek v košíku v pravém horním menu,
- komponenta pro navigaci (používaná v horním menu, levém menu i spodním menu),
- komponenta stránkovače (používaná ve výpisu kategorie),
- komponenta výpisu kategorie (zobrazuje více komponent dlaždice produktu),
- komponenta dlaždice produktu (zobrazuje se ve výpisu kategorie).

9.9.2 Platforma Node.JS

Pro Node.JS existuje mnoho šablonovacích jazyků. Já jsem se rozhodl v ukázkové aplikaci pro jazyk Jade. Jednak z důvodu jeho velké rozšířenosti, dále kvůli nestandardní syntaxi, která slibovala zkrácení doby psaní, a naposled kvůli jeho podpoře v IDE.

Co do základních funkcí podporuje Jade prakticky to samé, co v Latte. Je zde možnost vytvářet bloky, vnořovat je, základní příkazy pro kontrolu toku. Přesto ale působí Jade oproti intuitivnímu Nette místy těžkopádně. Zejména z důvodů rozšiřitelnosti – dobrých návodů, jak přidat vlastní helpery nebo makra, je na internetu poskrovnu. Já jsem nakonec vyřešil volání maker pomocí standardních JavaScriptových funkcí.

Syntaxe Jade pro HTML nezapisuje špičaté závorky. Vlastně celý koncept zápisu tagu se mění. Jade vytváří HTML taky až na při renderování.

Výpis 11: Ukázka mixinu pro šablonu Jade (zdroj: autor)

```
mixin row_input_password(name, label)
    tr
          td.fieldKey: mixin label(name, label)
          td.fieldValue: input(type="password", name=name, id=label)
```

Způsob psaní v Jadu se mi dnes už zdá mírně jednoduší. Jeho osvojení mi ale trvalo nějakou chvíli.

10 Výsledky srovnání frameworku

Kapitola obsahuje srovnání frameworků vycházejících z práce F. Odehnala (1), doplněné o výsledky z práce P. Loudy (2) a o mnou porovnávané frameworky. Při porovnání jsem se držel původních kritérií z kapitoly Definice kritérií pro srovnání (1).

Struktura tabulek pro srovnání byla převzata, stejně jako bylo převzato hodnocení těchto frameworků, pokud není uvedeno jinak. Případné komentáře k srovnávaným frameworkům **Nette** a **Node.JS** uvádím pod jednotlivými tabulkami. Případného zájemce o komentáře k hodnocení ostatních frameworků odkazuji na diplomové práce, ve kterých bylo toto hodnocení formulováno.

10.1 Základní srovnání

V tabulce níže jsou výsledky základního srovnání, jehož kritéria byla definována v kapitole 3.1 Odehnalovy práce (1).

Tabulka 1: *Výsledky základního srovnání (zdroj: Grails, Zend (1); ASP.NET MVC (2); Nette, Node.JS autor)*

Kritérium	Grails	Zend	ASP.NET MVC	Nette	Node.JS
Použitý jazyk/platforma	Groovy/Java	PHP	C#/.NET	PHP	JavaScript/Node.JS
Způsob řešení frameworku	pluginy	vše v jednom	vše v jednom	vše v jednom	komponenty
Způsob konfigurace	.groovy soubory	.ini soubory, .php soubory	.xml soubory, C# soubory	.neon soubory, .php soubory	package.json, .js soubory
Podpora i18n	ano	ano	ano	ano[10]	komponenty[11]

[10] http://doc.nette.org/cs/localization

[11] https://npmjs.org/search?q=i18n

Kritérium	Grails	Zend	ASP.NET MVC	Nette	Node.JS
Podpora ACL	plugin	ano	ne	ano[12]	komponenty[13]
Podpora cache	plugin	ano	ano	ano[14]	komponenty[15]

Nette Framework poskytuje solidní základ, na němž lze vyvinout webovou aplikaci, a to bez nutnosti získávání dalších komponent či pluginů. Konfigurace probíhá skrz .neon soubory a způsob konfigurace podporuje čistý objektový návrh.

Cache v Nette frameworku je snadno dostupná jak v logice aplikace, tak v šablonách. Při návrhu lze tedy cachovat data jak v modelu, tak jednotlivé komponenty v šabloně, tak i celé výsledné stránky.

Samotné **Node.JS** poskytuje vývojáři akorát základní funkcionalitu. Šíře této funkcionality je dle mého názoru srovnatelná s funkcionalitou PHP ve verzích 3 či 4. Funkcionalitu je třeba rozšířit komponentami pracujícími na vyšší úrovni. Komponenty se stahují skrz NPM repositář, a jejich instalace je vymyšlena velice elegantně včetně řešení závislostí. Toto řešení přenáší na vývojáře zodpovědnost v rozhodnutí, které komponenty do vyvíjeného projektu zařadit, a které ne.

10.2 Architektura frameworku

Kapitola se zaměřuje na architekturu frameworku. Kritéria srovnání se týkají implementace architektonických vzorů Martina Fowlera (30), což bylo popsáno v kapitolách 2.4 a 3.2 Odehnalovy práce (1).

U hodnocení Node.JS platformy zahrnuji do srovnání i komponenty použité při vývoji ukázkové aplikace (viz kapitola **Chyba! Nenalezen zdroj odkazů.**). Tento fakt je vždy u konkrétního srovnání zmíněn.

---

[12] http://doc.nette.org/cs/security

[13] https://npmjs.org/search?q=acl

[14] http://doc.nette.org/cs/caching

[15] https://npmjs.org/search?q=cache

10.2.1 Vzory doménové logiky

Tabulka níže obsahuje srovnání podle architektonických vzorů doménové logiky. Kritéria jsou definována v kapitole 3.2.1 Odehnalovy práce (1).

Tabulka 2: *Výsledky srovnání podle architektonických vzorů doménové logiky (zdroj: Grails, Zend (1); ASP.NET MVC (2); Nette, Node.JS autor)*

Kritérium	Grails	Zend	ASP.NET MVC	Nette	Node.JS
Způsob řešení doménové logiky	Domain Model	Domain Model / Transaction Script	Domain Model	Domain Model	Domain Model / Transaction Script
Podpora Service Layer (servisní vrstvy)	ano	ne	ne	ne	ne

10.2.2 Vzory datových zdrojů

Tabulka níže obsahuje srovnání podle implementace vzorů datových zdrojů. Kritéria jsou definována v kapitole 3.2.2 Odehnalovy práce (1).

Tabulka 3: *Výsledky srovnání podle implementace vzorů datových zdrojů (zdroj: Grails, Zend (1); ASP.NET MVC (2); Nette, Node.JS autor)*

Kritérium	Grails	Zend	ASP.NET MVC	Nette	Node.JS
Použité vzory datových zdrojů	Data Mapper	Table Data Gateway, Row Data Gateway	Data Mapper	Table Data Gateway, Row Data Gateway, Active Record	Table Data Gateway, Row Data Gateway, Active Record

U **Node.JS** je zajištění podpory datových zdrojů řešeno komponentou Sequelize.

10.2.3 Objektově-relační vzory chování

Tabulka níže obsahuje srovnání podle implementace objektově-relačních vzorů chování. Kritéria jsou definována v kapitole 3.2.3 Odehnalovy práce (1).

Tabulka 4: *Výsledky srovnání podle implementace objektově-relačních vzorů chování (zdroj: Grails, Zend (1); ASP.NET MVC (2); Nette, Node.JS autor)*

Kritérium	Grails	Zend	ASP.NET MVC	Nette	Node.JS
Implementace Unit of Work	ano	ne	ano	ne	ne
Implementace Identity Map	ano	ne	ano	ne	ne
Implementace Lazy Load	ano	ne	ano	ano	ne

Nette usnadňuje implementaci lazy-loaded objektů pomocí třídy `Nette\DI\Container`.

Přestože **Node.JS** samo o sobě lazy load chování nepodporuje, je možné ho jednoduše simulovat přímo vlastnostmi jazyka JavaScript, a to pomocí uzávěr.

10.2.4 Objektově-relační strukturální vzory

Tabulka níže obsahuje srovnání podle implementace objektově-relačních strukturálních vzorů. Kritéria jsou definována v kapitole 3.2.4 Odehnalovy práce (1).

Tabulka 5: *Výsledky srovnání podle implementace objektově-relačních vzorů chování (zdroj: Grails, Zend (1); ASP.NET MVC (2); Nette, Node.JS autor)*

Kritérium	Grails	Zend	ASP.NET MVC	Nette	Node.JS
Identity Field	ano	ne	ne	ano	ano
Implementace vzorů pro mapování asociací	Foreign Key Mapping, Association Table Mapping, Dependent Mapping	žádné	Foreign Key Mapping, Association Table Mapping, Dependent Mapping	Foreign Key Mapping	Foreing Key Mapping, Dependent Mapping

Embedded Value	ano	ne	ano	ne	ne
Serialized LOB	ne	ne	ano	ne	ne
Implementované vzory pro řešení dědičnosti	Single Table, Class Table	žádné	Single Table, Class Table, Concrete Table	žádné	žádné

Nette umí načítat relace mezi daty (Foreign Key Mapping) buď přímo reflexí databáze nebo je schopno se na relace doptat poskytovatele, který je mu podstrčen vývojářem.

Pro **Node.JS** zajišťuje podporu objektově-relačních strukturálních vzorů komponenta Sequelize.

10.2.5 Vzory pro objektově-relační mapování dat

Tabulka níže obsahuje srovnání podle implementace vzorů pro objektově-relační mapování dat. Kritéria jsou definována v kapitole 3.2.5 Odehnalovy práce (1).

Tabulka 6: *Výsledky srovnání podle implementace vzorů pro objektově-relační mapování dat (zdroj: Grails, Zend (1); ASP.NET MVC (2); Nette, Node.JS autor)*

Kritérium	Grails	Zend	ASP.NET MVC	Nette	Node.JS
Implementace Metadata Mapping	ano	ne	ano	ne	ne
Implementace Query Object	ano	ne	ano	ne	ne
Implementace Repository	ano	ne	ano	ne	ne

10.2.6 Vzory pro webové prezentace

Tabulka níže obsahuje srovnání podle implementace vzorů pro webové prezentace. Kritéria jsou definována v kapitole 3.2.6 Odehnalovy práce (1).

Tabulka 7: *Výsledky srovnání podle implementace vzorů pro webové prezentace (zdroj: Grails, Zend (1); ASP.NET MVC (2); Nette, Node.JS autor)*

Kritérium	Grails	Zend	ASP.NET MVC	Nette	Node.JS
Způsob implementace MVC	Front Controller	Front Controller	Front Controller	Front Controller	Front Controller
Implementace Template View	ano	ano	ano	ano	ano
Implementace Transform View	ano	ne	ano	ne	ne
Implementace Two Step View	ano	ne	ano	ano	ne

Nette využívá architekturu MVP[16], která se od MVC mírně liší mírou odpovědností presenteru/controlleru. Tendencí u Nette je přesunout většinu logiky z presenterů jinam, a ponechat prezentery co nejtenčí.

Způsob realizace šablon v Nette umožňuje pohlížet na webovou stránku jako na strom komponent, které lze libovolně vkládat do stránky i do sebe. Společně se sdílením šablon pro vzhled, makry a filtry v šablonách poskytuje Nette velice kvalitní základ pro tvorbu pohledu.

V **Node.JS** je implementace MVC řešena komponentami, stejně jako šablonovací jazyk. V ukázkové aplikaci je MVC řešeno frameworkem Locomotive vystaveným nad frameworkem Express. Jako šablonovací systém byl zvolen Jade.

10.2.7 Vzory pro souběžné zpracování

Tabulka níže obsahuje srovnání podle implementace vzorů pro souběžné zpracování. Kritéria jsou definována v kapitole 3.2.7 Odehnalovy práce (1).

[16] http://doc.nette.org/cs/book/presenter

Tabulka 8: *Výsledky srovnání podle implementace vzorů pro souběžné zpracování (zdroj: Grails, Zend (1); ASP.NET MVC (2); Nette, Node.JS autor)*

Kritérium	Grails	Zend	ASP.NET MVC	Nette	Node.JS
Impl. Optimistic Offline Lock	ano	ne	ano	ne	ne
Impl. Pessimistic Offline Lock	ano	ne	ne	ne	ne
Impl. Coarse-Grained Lock	ano	ne	ne	ne	ne
Impl. Implicit Lock	ano	ne	ne	ano	ne

Nette umožňuje získání implicitního zámku a tím zajištění exkluzivity při přístupu k souboru[17].

10.3 Práce s frameworkem

Kapitola obsahuje taková kritéria porovnání, která by měla odrážet úroveň kvality práce s vybranými frameworky. Kapitola je dělena na práci s modelem, řadiči, pohledy a na testování.

Hlavním zdrojem pro hodnocení jsou poznatky získané při vývoji ukázkových aplikací dle kritérií v kapitole 7.

Každá kapitola obsahuje tabulku pro slovní hodnocení jednotlivých kritérií týkajících se dané oblasti. Díky tomu je možné jednoduše porovnat, jak se liší v daném konkrétním bodě funkcionalita všech srovnávaných frameworků.

Na konci každé kapitoly je prostor i pro osobní hodnocení. Toto hodnocení je subjektivní a vychází z mých dojmů z jednotlivých frameworků nabytých při tvorbě ukázkových aplikací.

[17] http://doc.nette.org/cs/atomicity

Hodnocení je porovnáno se známkami udělenými autory předchozích prací (1) (2). Při hodnocení jsem se snažil srovnávat dva mnou hodnocené frameworky nejen mezi sebou, ale i s ostatními frameworky, a to tak, aby poměr známek dával vždy v daném řádku smysl.

Známky v osobním hodnocení jsou udělovány stejně jako v české škole, tedy od stupně 1 (výborně) až po stupeň 5 (nedostatečně).

10.3.1 Práce s modelem

Tabulka níže obsahuje výsledky srovnání práce s modelem. Kritéria tohoto porovnání jsou definována v kapitole 3.3.1 Odehnalovy práce (1).

Tabulka 9: *Výsledky srovnání práce s modelem (zdroj: Grails, Zend (1); ASP.NET MVC (2); Nette, Node.JS autor)*

Kritérium	Grails	Zend	ASP.NET MVC	Nette	Node.JS
Řídící role při vytváření	Diagram tříd	Schéma databáze	Diagram tříd	Schéma databáze	Diagram tříd
Scaffolding	ano	ne	ano	ne	ano
Podpora vztahu 1:N (N:1)	ano	ne	ano	ne	ano
Podpora vztahu 1:1	ano	ne	ano	ne	ano
Podpora vztahu M:N	ano	ne	ano	ne	ne
Podpora dědičnosti tříd	ano	ne	ano	ne	ne

Kritérium	Grails	Zend	ASP.NET MVC	Nette	Node.JS
Způsob výběru záznamů	základní, dynamické, podle kritérií, pomocí HQL	základní, pokročilé, pomocí Zend_Db_Select, dynamické	základní, LINQ	základní, pokročilé pomocí dotazovacího API[18]	základní, pokročilé pomocí dotazovacího API[19]
Validace modelu	ano	ne	ano	ne	ano

Nette framework poskytuje dotazovací API, které usnadňuje výběr záznamů svázaných v databázi pomocí cizích klíčů. Dotaz je možné pokládat přímo komponováním řetězce; komponováním řetězce a předáním proměnných, které framework automaticky ošetří; postupným voláním metod, které zkomponují dotaz za programátora.

Komponenta Sequelize poskytuje pro **Node.JS** výše uvedenou funkcionalitu. Stejně jako u Nette, je možné se dotazovat na záznamy různými způsoby. Hlavním rozdílem je ale to, že komponenta podporuje definici modelu.

Je možné nadefinovat objekty modelu, které framework automaticky mapuje na tabulky v databázi. Framework rovněž poskytuje možnost automatického vytvoření/úpravy struktury tabulek dle definice objektů.

Osobní hodnocení

Framework **Nette** usnadňuje dotazování na data v databázi a jejich úpravu pomocí vzoru Active Record. Tato logika si prošla u Nette dlouhým vývojem. Nejdříve existovala jako oddělená vrstva dibi[20], která byla poté zapracována od verze 2.0 přímo do Nette, ovšem s některými vylepšeními.

Přesto ale ve frameworku není funkcionalita, která by usnadnila tvorbu objektů doménové logiky. Bylo nutné tyto objekty navrhnout a vytvořit ručně. Rozhodl jsem se využít podobného návrhového vzoru, jako Odehnal ve své ukázkové aplikaci pro Zend (1). Díky „magii“ ve frameworku se mi ale povedlo snížit počet vytvářených tříd pro každou třídu modelu ze tří na dvě. Oficiální dokumentace Nette se o žádném ideálním způsobu tvorby modelu nezmiňuje – je tedy necháno čistě na programátorovi, jak s danou problematikou naloží.

---

[18] http://doc.nette.org/cs/database

[19] http://sequelizejs.com/documentation

[20] http://dibiphp.com/cs/

Aplikace nad **Node.JS** často využívají NoSQL databázi MongoDB, případně komponentu Mongoose pro propojení Node.JS a MongoDB. Tato kombinace umí pracovat s entitami modelu lépe, a patrně by pro ni hodnocení v této kapitole dopadlo přínosněji, než za současného návrhu.

Vzorová aplikace ovšem běží nad MySQL, čemuž bylo nutné podřídit i výběr komponent pro komunikaci s databází. Z toho důvodu byl zvolen ORM Sequelize. Na první pohled se zdá, že komponenta poskytuje dostatečnou dokumentaci. Bohužel po hlubším zkoumání se projevil stejný nešvar, jako u jiných komponent v Node.JS, a to nedostatek kvalitní oficiální dokumentace pro komplexní příklady.

Z toho důvodu jsem při tvorbě modelu vycházel z článku (31), který pojednává o správném strukturování modelu v Sequelize nad middlewarem Express.

Osobně se domnívám, že ani Nette, ani Node.JS neposkytuje kvalitní základ pro práci s modelem. V obou frameworcích je nutné, aby programátor ručně napsal definice tříd doménové logiky. V Nette je i nutné ručně provádět mapování mezi databází a třídami modelu. Existuje zde jasný rozdíl mezi Grails a ASP.NET MVC vs. Zend, Nette a Node.JS.

Tabulka 10: *Výsledky osobního hodnocení práce s modelem (zdroj: Grails, Zend (1); ASP.NET MVC (2); Nette, Node.JS autor)*

Kritérium	Grails	Zend	ASP.NET MVC	Nette	Node.JS
Potřebná funkcionalita	1	4	1	4	2
Přehlednost kódu	1	4	1	3	2
Obtížnost naučení	3	4	3	4	3
Integrace s IDE	1	5	2	3	5

Komentáře k hodnocení:

Kritérium	Nette	Node.JS
Potřebná funkcionalita	Nette obsahuje pouze knihovnu pro práci s databází, ne pro tvorbu tříd modelu.	Node.JS umožňuje tvorbu tříd modelů za pomoci komponenty Sequelize.

Kritérium	Nette	Node.JS
Přehlednost kódu	Navrhované řešení vyžaduje 2 třídy pro jednu třídu modelu, což je stále lepší než řešení v Zendu.	Každá třída modelu je v odděleném souboru, kód je v zásadě přehledný.
Obtížnost naučení	Řešení vyžadovalo vlastní návrh logiky pro mapování dat z databáze na třídy modelu.	Řešení vyžadovalo další hledání vhodných řešení na internetu (31) a následně úpravu těchto řešení.
Integrace s IDE	Nástroj IntelliJ PhpStorm nerozlišuje třídy modelu od ostatních PHP tříd.	Nástroj IntelliJ WebStorm nerozlišuje struktury modelu od jiných datových struktur v Node.JS.

10.3.2 Práce s řadiči

Tabulka níže obsahuje výsledky srovnání práce s řadiči (controller). Kritéria tohoto porovnání jsou definována v kapitole 3.3.2 Odehnalovy práce (1).

Tabulka 11: *Výsledky srovnání práce s řadiči (zdroj: Grails, Zend (1); ASP.NET MVC (2); Nette, Node.JS autor)*

Kritérium	Grails	Zend	ASP.NET MVC	Nette	Node.JS
URL mapování	ano	ano	ano	ano	ano
Způsoby mapování	statické, dynamické, bez řadiče (pouze šablona pohledu)	statické, dynamické, pomocí regulárního výrazu, na úrovni domény, řetězité	statické, dynamické	statické, dynamické, pomocí regulárního výrazu	statické
Reverzní vytváření odkazů	ano	ano	ano	ano	ano

Kritérium	Grails	Zend	ASP.NET MVC	Nette	Node.JS
Podpora funkcionality napříč řadiči	filtry, interceptory	pluginy	filtry	služby	komponenty
Použití filtrů/pluginů pro konkrétní řadič	ano	ne	ne	ne	ne
Podpora práce se sezeními	ano	ano	ano	ano	ano
Objekt sezení pro 2 požadavky (flash)	ano	ano	ano	ano	ano
Možnost přesměrování	ano	ano	ano	ano	ano
Možnost podstrčení stránek	ano	ano	ano	ano	ano
Druhy parametrů	session, request, params, flash, servletContext	params, post, query	Session, Request, Response, TempData, RouteData	application, user, session, context, request, …	request, response, app, flash, session, …
Validace vstupních dat	ano	ano	ano	ano	ano
Definice zobrazení chybové stránky	v souboru UrlMappings.groovy	v řadiči	v řadiči, ve web.config	v config.neon	v řadiči

Kritérium	Grails	Zend	ASP.NET MVC	Nette	Node.JS
Podstrčení stránky při výskytu chyby	ne	ano	ano	ano	ano

Osobní hodnocení

Framework **Nette** poskytuje intuitivní práci s řadiči, která je navíc velmi dobře zdokumentována v oficiální dokumentaci (4). Framework nativně podporuje i správu modulů, což umožňuje organizovat řadiče do stromové struktury (například komponenty administrace tvoří jeden modul, fórum další modul, atp.) a zvýšit čistotu kódu aplikace.

Mapování URL se nastavuje globálně pro celou aplikaci, a to v konfiguračním souboru `.neon`. Možnosti nastavení mapování jsou srovnatelné s nastavením v Zend Frameworku. V případě potřeby speciálního způsobu mapování není problém si napsat vlastní třídu na mapování, což je rovněž dobře zdokumentováno. Framework myslí i na takové drobnosti, jako je automatická kanonizace URL. Existuje-li tedy k jedné činnosti více URL, framework automaticky uživatele přesměruje na tu primární.

Framework již ve svém základu obsahuje chybovou stránku, která se zobrazí uživateli na produkční verzi webu. Rovněž ale obsahuje i velmi výřečné chybové hlášení, které se zobrazí vývojáři. Tato velice užitečná drobnost například v Zend Frameworku chybí a velice usnadňuje vývoj aplikace.

Při použití komponent Connect, Express a Locomotive v **Node.JS** získáme výše uvedenou funkcionalitu pro práci s řadiči. Stejně jako v Nette, i zde je možné strukturovat řadiče do jednotlivých modulů.

Při vývoji aplikace jsem se setkal s problémem u mapování URL. Framework vyžaduje explicitní uvedení mapování URL pro všechny controllery, což značně snižuje udržitelnost aplikace.

Framework poskytuje možnost zobrazení chybové stránky při výskytu chyby. Tato funkcionalita ovšem zobrazuje i technické informace o tom, kde chyba nastala, což jsou informace, které by rozhodně neměly být viditelné na produkčním webu. Pro produkční web je tedy nutné zobrazování uživatelsky přívětivé chybové stránky doprogramovat ručně.

Tabulka 12: *Výsledky osobního hodnocení práce s řadiči (zdroj: Grails, Zend (1); ASP.NET MVC (2); Nette, Node.JS autor)*

Kritérium	Grails	Zend	ASP.NET MVC	Nette	Node.JS
Potřebná funkcionalita	2	1	1	1	3
Přehlednost kódu	1	1	1	1	2
Obtížnost naučení	1	3	2	2	4
Integrace s IDE	1	5	1	3	5

Komentáře k hodnocení:

Kritérium	Nette	Node.JS
Potřebná funkcionalita	Nette poskytuje kvalitní platformu pro tvorbu řadičů, mapování URL a další funkcionalitu.	Kombinace Connect, Express a Locomotive zaostává za ostatními frameworky ve funkcionalitě.
Přehlednost kódu	Kód je stručný a přehledný.	Kód je stručný a přehledný, ovšem přístup ke zcela standardní funkcionalitě vyžaduje vyšší znalosti.
Obtížnost naučení	Nette poskytuje jednoduché a zdokumentované řešení pro nejčastější problémy. Rozšíření dle vlastních požadavků je relativně lehké.	Je zdokumentován pouze nejjednodušší způsob použití, který pro aplikaci nestačil. Je nutné studovat zdrojové kódy, blogy a uživatelská fóra pro pochopení funkcionality.
Integrace s IDE	Nástroj IntelliJ PhpStorm nerozlišuje třídy modelu od ostatních PHP tříd. Nette však striktně dodržuje PhpDoc konvence, což značně usnadňuje napovídání v IDE.	Nástroj IntelliJ WebStorm nerozlišuje struktury modelu od jiných datových struktur v Node.JS. Různost způsobu zápisů kódů jednotlivých komponent zamezuje dostatečnému pochopení kódu editorem.

10.3.3 Práce s pohledy

Tabulka níže obsahuje výsledky srovnání práce s pohledy (view). Kritéria tohoto porovnání jsou definována v kapitole 3.3.3 Odehnalovy práce (1).

Tabulka 13: *Výsledky srovnání práce s pohledy (zdroj: Grails, Zend (1); ASP.NET MVC (2); Nette, Node.JS autor)*

Kritérium	Grails	Zend	ASP.NET MVC	Nette	Node.JS
Typ souborů šablon	soubory GSP	soubory PHTML	soubory CSHTML	soubory LATTE	soubory JADE
Jazyk šablony	XML doplněný o dynamické značky	HTML doplněný o dynamické značky	HTML doplněný o dynamické značky	HTML doplněný o dynamické značky	vlastní jazyk
Podpora stránkování	ano	ano	plugin	plugin	plugin
Označení pro definované prvky webové stránky	značka z knihovny značek (Tag Library)	třídy pomocníků (helper)	třídy pomocníků (helper)	pomocníci (helpery, makra)	funkce (mixiny)
Počet definovaných značek/pomocníků	60	52	14	81 [21]	5 [22]
Možnost vytvářet vlastní značky/pomocníky	ano	ano	ano	ano	ano
Vytváření formulářů	pomocí klasických značek	pomocí objektu Zend_Form	pomocí helperů	pomocí kombinace komponenta a helpery	pomocí klasických značek
Podpora proti CSFR	ano	ano	ano	ano	ne

[21] 53 maker (http://doc.nette.org/cs/default-macros) a 28 helperů (http://doc.nette.org/cs/default-helpers)

[22] https://github.com/visionmedia/jade

Kritérium	Grails	Zend	ASP.NET MVC	Nette	Node.JS
Podpora JavaScriptu	prototype, dojo, yui, mootols	dojo, jquery	jquery, jquery ui, jquery validate, knockout, modernizr	jquery	jquery

Osobní hodnocení

Framework **Nette** poskytuje kvalitní základ pro implementaci šablon. Využívá vlastní jazyk Latte, který je velice podobný jazyku Smarty. Šablony jsou, stejně jako u knihovny Smarty, před spuštěním automaticky transpilovány do PHP. Na Latte šablonách vnímám jako veliký přínos jejich orientaci na bezpečnost. Proměnné jsou při výpisu automaticky escapovány tak, aby do výsledného HTML nemohl proniknout kód, který tam nemá co dělat. Chce-li programátor vypsat syrovou verzi výstupu, musí využít delší zápis makra pro výpis proměnné. V tomto přístupu se Nette naprosto liší například od Zend Frameworku, který neposkytuje vývojářům žádné usnadnění pro výpis escapovaných proměnných.

Psaní maker pro Latte vyžaduje ovšem potřebu seznámit se důkladně s frameworkem. Makra jsou totiž zodpovědná při transpilaci za generování PHP kódu, který se provede až při spuštění šablony.

Šablonovací systém Jade pro **Node.JS** je zajímavý svojí syntaxí, která programátora zcela odstiňuje od HTML kódu. Kromě této vlastnosti si ovšem dovolím považovat Jade za velice chudý jazyk pro tvorbu šablon webových aplikací. V základu neobsahuje žádná makra. Způsob rozšíření jazyka není v dokumentaci kvalitně popsán, a programátor je tedy odkázán na zdrojové kódy a poznatky jiných uživatelů.

Tabulka 14: *Výsledky osobního hodnocení práce s pohledy (zdroj: Grails, Zend (1); ASP.NET MVC (2); Nette, Node.JS autor)*

Kritérium	Grails	Zend	ASP.NET MVC	Nette	Node.JS
Potřebná funkcionalita	1	1	1	1	3
Přehlednost kódu	1	3	2	1	1

Obtížnost naučení	2	1	2	1	2
Integrace s IDE	1	5	1	4	2

Komentáře k hodnocení:

Kritérium	Nette	Node.JS
Potřebná funkcionalita	Latte je kvalitní a bezpečný jazyk s pestrou funkcionalitou a dobrou rozšiřitelností.	Jazyk Jade slouží pouze k výpisu šablon. Vše ostatní je programátor nucen řešit v čistém JavaScriptu.
Přehlednost kódu	Kód je stručný a přehledný.	Kód je stručný a přehledný.
Obtížnost naučení	Latte je dobře dokumentováno, použití funkcionality je intuitivní.	Jiná syntaxe pro zápis HTML vyžaduje jistou praxi. Poté se ale využívá docela dobře.
Integrace s IDE	Nástroj IntelliJ PhpStorm nezná jazyk Latte. Trochu pomůže nastavit zvýrazňování pro jazyk Smarty, který je podobný.	Nástroj IntelliJ WebStorm zvýrazňuje jazyk Jade a napovídá v něm HTML a CSS značky. Neumí v něm ovšem zvýrazňovat JavaScript.

10.3.4 Testování

Tabulka níže obsahuje výsledky srovnání práce s testy. Kritéria tohoto porovnání jsou definována v kapitole 3.3.4 Odehnalovy práce (1).

Tabulka 15: *Výsledky srovnání práce s testy (zdroj: Grails, Zend (1); ASP.NET MVC (2); Nette, Node.JS autor)*

Kritérium	Grails	Zend	ASP.NET MVC	Nette	Node.JS
Podpora testování	ano	ano	ano	ano	ano
Úrovně testování	jednotkové, integrační, funkcionální (pomocí pluginu)	nerozlišuje se	jednotkové, integrační (plugin), funkcionální (plugin)	nerozlišuje se	nerozlišuje se

Podpora napodobenin (Mock object)	ano	ano	plugin	ano	ano
Podpora datových setů	plugin	ano	plugin	ano	ne

Osobní hodnocení

Pro testování v **Nette** Frameworku jsem zvolil knihovnu PHPUnit. Stejnou knihovnu využil i F. Odehnal ve své ukázkové aplikaci (1). Funkcionalita testovací knihovny je tedy stejná. Na rozdíl od Zend Frameworku je Nette k testování přívětivější. U aplikací založených na Zendu bývá problémem extrahovat z inicializačních skriptů tu logiku, kterou je nutné provést při inicializaci testovacího prostředí. Dalším problémem je návrh architektury založen do jisté míry na statických třídách, což může při testování způsobovat problémy (32).

Mimo výše uvedené rozdíly ale Nette nijak programátorovi testování neusnadňuje, jako například u frameworku Grails.

Testování pod **Node.JS** je zhruba na stejném vývojovém stupni jako u Zend Frameworku nebo u Nette. Pro demonstrační účely jsem zvolil knihovnu Nodeunit, která zajišťuje běh testů.

Tabulka 16: *Výsledky osobního hodnocení práce s testy (zdroj: Grails, Zend (1); ASP.NET MVC (2); Nette, Node.JS autor)*

Kritérium	Grails	Zend	ASP.NET MVC	Nette	Node.JS
Potřebná funkcionalita	1	1	1	1	1
Přehlednost kódu	1	1	1	1	1
Obtížnost naučení	2	4	3	3	4
Integrace s IDE	4	1	1	1	1

Komentáře k hodnocení:

Kritérium	Nette	Node.JS
Potřebná funkcionalita	Framework PHPUnit poskytuje dostatečnou funkcionalitu.	Framework Nodeunit poskytuje dostatečnou funkcionalitu.
Přehlednost kódu	Kód je stručný a přehledný.	Kód je stručný a přehledný.
Obtížnost naučení	Na rozdíl od Zend Frameworku je integrace PHPUnitu s Nette o něco jednodušší.	Je nutné důkladné pochopení logiky aplikace pro správnou součinnost s testovacím frameworkem.
Integrace s IDE	Nástroj IntelliJ PhpStorm umí spouštět PHPUnit testy, zobrazovat výsledky i míru pokrytí kódu.	Nástroj IntelliJ WebStorm umí spouštět Nodeunit testy, zobrazovat výsledky i míru pokrytí kódu.

10.3.5 Celkové osobní hodnocení

Kapitola obsahuje celkové osobní zhodnocení frameworků.

Metodika hodnocení

Tabulka níže obsahuje výsledky srovnání práce s modelem. Kritéria tohoto porovnání jsou definována v kapitole 3.4 Odehnalovy práce (1).

Známky jsou spočteny aritmetickým průměrem z předchozích hodnocení. Známka pro dané kritérium a framework je tedy aritmetickým průměrem ze všech čtyř předchozích osobních hodnocení (tj. práce s modelem, řadiči, pohledy a testování) pro dané kritérium a framework. Výsledná známka pro framework je poté aritmetickým průměrem známek v celkovém osobním hodnocení pro daný framework.

Výsledky hodnocení

Tabulka 17: *Výsledky celkového osobního hodnocení práce s frameworky (zdroj: Grails, Zend (1); ASP.NET MVC (2); Nette, Node.JS autor)*

Kritérium	Grails	Zend	ASP.NET MVC	Nette	Node.JS
Potřebná funkcionalita	1,25	1,75	1	1,75	2,25

Kritérium	Grails	Zend	ASP.NET MVC	Nette	Node.JS
Přehlednost kódu	1	2	1,25	1,5	1,5
Obtížnost naučení	2	3	2,5	2,5	3,25
Integrace s IDE	1,75	4	1,25	2,75	3,25
Celkové hodnocení	**1,45**	**2,65**	**1,5**	**2,13**	**2,56**

Slovní hodnocení

Framework **Nette** vznikl proto, aby usnadnil vývoj webových aplikací. Jeho snahou bylo vzít si od těch „větších“ to nejlepší a poučit se z předchozích chyb ostatních frameworků. Osobně se domnívám, že se to frameworku docela vydařilo. Ve funkcionalitě, kterou obsahuje, zpravidla plní velice dobře svoji funkci.

Nejvíce bych ocenil funkcionalitu, která se v hodnocení moc nepromítla, a to jsou nástroje pro usnadnění vývoje. Jedná se hlavně o velmi propracovaná chybová hlášení, automatické cachování šablon, generování PHPDoc komentářů pro usnadnění napovídání objektů v systémovém kontextu, důraz na bezpečnost v šablonách a možnost jednoduchého zasílání chybových hlášení z produkčního serveru e-mailem.

Nejen tato funkcionalita poskytuje kvalitní nástroje pro rychlé nasazení webové aplikace menšího rázu. Programátorům v PHP bych doporučil orientovat se oproti Zend Frameworku na tento framework, a případnou chybějící funkcionalitu dokombinovat jinými knihovnami – například ORM mapování.

Platforma **Node.JS** vznikla jako způsob běhu JavaScriptu na straně serveru. Platforma je zajímavá jednoduchostí škálování a svým asynchronním návrhem všech vstupně/výstupních operací. Samotná platforma je svojí funkcionalitou velice tenká. Dovolil bych si ji přirovnat mírou funkcionality k PHP ve verzích 3 či 4. Pro návrh webové aplikace je nutné vhodně zvolit a zkombinovat komponenty zajišťující funkcionalitu na vyšší úrovni.

V ukázkové aplikaci s komponentami Connect, Express, Locomotive, Sequelize, Jade, Q, aj. jsem si vyzkoušel náročnost rozběhnutí webové aplikace nad Node.JS. Přestože některé koncepty architektury se mi jeví jako zajímavé, příště bych ji patrně pro účely rapidního vývoje aplikace již znova nepoužil. Domnívám se, že v současnosti jde o mladou a ještě nevyzrálou

architekturu. Tento fakt se projevuje také nízkou mírou standardizace zápisu zdrojového kódu používaných knihoven. Relativní míra volnosti zápisu v JavaScriptu způsobuje, že IDE často odmítá napovídat informace o používaných datových strukturách. Společně s nízkým množstvím kvalitní dokumentace je člověk často odkázán opravdu pouze na zdrojové kódy.

Osobně se domnívám, že platforma Node.JS potřebuje ještě několik let vývoje k tomu, aby se její zdrojová báze a vývoj ustálily. Tehdy může být zajímavou alternativou k ostatním.

Při srovnání **Zend Frameworku** a **Nette** bych volil jednoznačně Nette, s případným doplněním některých knihoven od Zendu. Propracovanost MVC logiky a výhody popsané výše jsou pro mne jasným argumentem. Zend Framework je ale dost strukturovaný, a některé jeho knihovny mimo MVC koncept považuji za velice kvalitní. Nevidím tedy důvod, proč je nevyužít i v případném projektu postaveném nad Nette.

Ze srovnání **Grails, ASP.NET MVC** vs. **Zend**, **Nette** a **Node.JS** je jasně vidět rozdíl mezi „velkými“ frameworky, vystavenými nad platformami se širokou funkcionalitou, a „malými“ frameworky určenými pro vývoj webových aplikací. (Zend si dovoluji zařadit do druhé kategorie, přestože co do funkcionality daleko převažuje své „malé“ konkurenty, a to právě kvůli platformě, nad kterou je napsán.) V dnešní době, kdy roste nabídka .NET hostingů, bych uvažoval i nad tvorbou webové aplikace nad touto platformou. U projektu s velkou mírou integrace s ostatními systémy bych uvažoval nad jedním z „velkých“ frameworků.

10.4 Podpora frameworku

Tabulka níže obsahuje výsledky srovnání dostupné podpory jednotlivých frameworků. Veškeré údaje jsou aktualizovány ke dni 23. 6. 2013, a to i pro frameworky Grails, Zend a ASP.NET MVC. Při hledání byly použity následující výrazy: „Grails“, „Zend Framework“, „ASP.NET MVC“, „Nette Framework“ a „Node.JS“.

Kritéria pro srovnání definoval F. Odehnal (1).

Tabulka 18: *Výsledky srovnání dostupné podpory frameworku (zdroj: Grails, Zend, ASP.NET MVC, Nette, Node.JS autor)*

Kritérium	Grails	Zend	ASP.NET MVC	Nette	Node.JS
Aktivita komunity	v2.2.3 21. 6. 2013[23]	v2.2.1 12. 6. 2013[24]	v4.0 15. 8. 2012[25]	v2.0.10 6. 3. 2013[26]	v0.10.12 18. 6. 2013[27]
Počet knih na amazon.com	2478	205	416	0	308
Počet příspěvků v Google Groups	628353	91936	49924	48	11797
Počet nalezených záznamů na Google Blogsearch (v 1000)	625	610	6390	10	1270
Počet problémů na stackoverflow.com	15638	10610	46223	21	17861
Kvalita dokumentace	2	3	1	3	4

Srovnání ukazuje, že Nette Framework v současnosti z většiny český framework, který do zahraničí dosud nepronikl. Většina debaty probíhá na fóru webu Nette[28], kde je ke dni 23. 6. 2013 uloženo 12 551 témat s počtem 95 388 příspěvků.

U Node.JS je zajímavý počet dostupných knih na Amazonu, který spolehlivě dorovnává ustálené frameworky jako Zend či ASP.NET MVC. Podle mého názoru to jen kopíruje vlnu zájmu, která v současnosti kolem frameworku probíhá.

---

[23] Zdroj: http://grails.org/download

[24] Zdroj: http://framework.zend.com/downloads/latest

[25] Zdroj: https://en.wikipedia.org/wiki/ASP.NET_MVC_Framework

[26] Zdroj: http://nette.org/en/download

[27] Zdroj: http://blog.nodejs.org/

[28] http://forum.nette.org/cs/

Kvalita dokumentace Nette Frameworku je srovnatelná s kvalitou dokumentace Zend Frameworku. Oba frameworky jsou psány kvalitně okomentovaným zdrojovým kódem, ze kterého je k dispozici vygenerovaná API dokumentace API[29]. Nette obsahuje i dokumentaci pro vývojáře, která popisuje logiku frameworku, obsahuje komplexní příklady a slouží jako základ pro vývoj[30]. V porovnání s oficiální Zend dokumentací[31] se domnívám, že vývojářská dokumentace pro Nette je kvalitnější. Přesto ale nepovažuji dokumentaci za tak kvalitní, aby si zasloužila stejnou známku, jako dokumentace Grails.

Kvalita dokumentace Node.JS je dle mého názoru nejhorší ze všech hodnocených frameworků. Dokumentace základního API Node.JS[32] je kvalitní a pochopitelná. Dokumentace komponent, které jsem používal při vývoji, je ovšem často minimální a neaktualizovaná. Často se dokumentace omezuje pouze na základní příklady použití dané funkcionality, která je ovšem v dané komponentě použita mnohem komplexněji. Přílišná volnost struktury zdrojových kódů způsobuje, že jednotlivé komponenty jsou psány různým stylem, což ztěžuje vývojáři orientaci. Domnívám se, že nemá-li člověk dostatečně hluboké znalosti JavaScriptu, může být pro něj v současnosti Node.JS velkým problémem.

---

[29] http://api.nette.org/

[30] http://doc.nette.org/cs/

[31] http://framework.zend.com/manual/

[32] http://nodejs.org/api/

11 Závěr

Cílem práce bylo rozšířit již tak komplexní srovnání tří frameworků pro vývoj webových aplikací o další dva frameworky. Srovnání rozšiřuje diplomové práce o porovnání frameworku Zend na platformě PHP a Grails na platformě Java od Františka Odehnala a frameworku ASP.NET MVC od Petra Loudy.

Srovnání bylo rozšířeno o framework Nette na platformě PHP a platformu Node.JS. Pro každý z uvedených frameworků byla popsána základní architektura a styl vývoje ve frameworku.

Pro zajištění co největší objektivity porovnání byla pro každý z minulosti porovnávaných frameworků vyvinuta ukázková aplikace vycházejících ze stejných požadavků. Rovněž pro framework Nette a platformu Node.JS byly vyvinuty ukázkové aplikace o stejné funkcionalitě. Tyto aplikace slouží jako zdroj informací pro srovnání. Dále mohou sloužit jako studijní materiál zájemcům o hlubší porovnání frameworků či začínajícím programátorům.

Nette Framework se snaží maximálně ulehčit rapidní vývoj webových aplikací. Dle výsledků porovnání a i dle mého názoru poskytuje lepší komfort vývojáři, než Zend Framework. Tento fakt může být způsoben relativním stářím Zend Frameworku oproti Nette, ale osobně se domnívám, že jde spíš o lepší návrh ve prospěch Nette.

Na rozdíl od Nette Frameworku, Node.JS jde jinou cestou. Jako platforma se snaží být minimalistická a poskytovat pouze základní funkcionalitu. Vývojář si přes správce závislostí stáhne zbývající komponenty. V tomto případě jsem pracoval s komponentami Connect, Express, Q, Sequelize, Jade a Locomotive.

V porovnání všech frameworků vychází Node.JS nejhůře. Po zkušenosti s vývojem aplikace se domnívám, že dosud není dostatečně vyspělé, aby umožnilo opravdu rapidní vývoj. Chybí zde kvalitní dokumentace, zdrojové kódy jednotlivých komponent nejsou konzistentní a často je problém se v nich zorientovat. Přesto se může Node.JS na některá řešení hodit, hlavně díky svým výkonnostním charakteristikám.

Z porovnání je jasně zřejmý rozdíl mezi frameworky Grails nad Javou a ASP.NET MVC, které mají pod sebou robustní platformu se širokou funkcionalitou, a těmi ostatními. Z hlediska porovnání architektury frameworku vítězí výše uvedené frameworky. Rozdíl je vidět i v nutnosti věnovat u ostatních frameworků spoustu času programování doménového modelu.

U tvorby aplikace v Node.JS mi nezřídka chyběla funkcionalita, kterou jsem měl k dispozici v Nette frameworku. Její absenci přikládám za důsledek relativnímu mládí a horečnému vývoji kolem platformy.

Měl-li bych zhodnotit odděleně pouze framework Nette a řešení nad platformou Node.JS, rozhodně bych zvolil Nette, a to z důvodů současné nedospělosti platformy Node.JS.

Pokud mám srovnat všechny platformy, Node.JS zůstává na posledním místě, Nette svoji kvalitou podpory vývoje předčí Zend. Na prvních místech v podpoře vývoje stále zůstávají frameworky Grails a ASP.NET MVC se svojí bohatou funkcionalitou.

Příloha A: Ukázková aplikace

Přiložené CD obsahuje 2 archivy:

- `nette_framework.zip`
- `node_js.zip`

Archivy obsahují zdrojové kódy ukázkových aplikací. Funkcionalita obou aplikací je totožná.

V každém z archivů je soubor `cti_me.txt`, který obsahuje informace potřebné pro spuštění aplikace.

Terminologický slovník

Termín	Význam (zdroj)
ACL	Access Control List – seznam přístupových práv k dané entitě, například informace o tom, který uživatel může zobrazit danou stránku (autor)
Apache	Název software webového serveru (autor)
API	Application Programming Interface – v překladu rozhraní pro programování aplikací. Používá se v termínu vymezení rozhraní aplikace, skrz které ji může ovládat někdo jiný. (autor)
BSD	Berkeley Software Distribution (autor)
Composer	Správce závislostí pro PHP[33] (autor)
Convention over configuration	Konvence namísto nastavování – název pro princip, kdy namísto taxativní konfigurace stačí dodržet určitou konvenci pro dosažení stejného výsledku (například pojmenovat metody unit testů s prefixem `test`) (15)
CRUD	create-read-update-delete – synonymum pro nejběžnější funkce nad databází (autor)
CSS	Cascaded Style Sheet (autor)
Dependency Injection	Technika pro vkládání závislostí mezi jednotlivými komponentami programu tak, aby jedna komponenta mohla používat druhou, aniž by na ni měla v době sestavování programu referenci (21)
DRY	Don't repeat yourself – snaha o minimalizaci redundance kódu (autor)
Express	Framework pro tvorbu webových aplikací pro Node.JS[34] (autor)
FastCGI	Protokol pro interaktivní komunikaci podřízeného procesu s webovým serverem (autor)
HTML	HyperText Markup Language (autor)
i18n	zkratka pro slovo internationzalization, tedy internacionalizaci (autor)
IDE	Integrated Development Environment – na funkcionalitu bohaté vývojové prostředí (autor)
Jade	šablonovací engine pro Node.JS[35]
KISS	Keep it Simple Stupid – snaha navrhovat co nejjednodušší řešení (autor)

[33] Viz http://getcomposer.org/

[34] Viz http://expressjs.com/

[35] Viz https://github.com/visionmedia/jade

Late Static Binding	Způsob přístupu ke statickým proměnným a metodám podporující polymorfismus. (Tj. rodičovská třída přistupuje k překryté metodě/proměnné.) (autor)
Latte	Šablonovací jazyk pro Nette Framework (autor)
LESS	Dynamický jazyk rozšiřující vlastnosti CSS a usnadňující jeho zápis.[36] (autor)
Locomotive	MVC framework pro Node.JS.[37] (autor)
MariaDB	MySQL-kompatibilní alternativa k MySQL databázi. (autor)
middleware	software (zpravidla knihovna, komponenta) poskytující určitou funkcionalitu a rozšiřující tím vlastnosti platformy (autor)
MongoDB	v současnosti jedna z vedoucích NoSQL databází (autor)
Mongoose	ORM framework pro MongoDB[38] (autor)
MVC	Model-View-Controller (autor)
MVP	Model-View-Presenter (autor)
NPM	Node Packaged Modules – systém pro správu závislostí pro Node.JS[39] (autor)
ORM	Objektově-relační mapování (autor)
scaffolding	funkcionalita pro generování CRUD logiky pro operace nad databází (autor)
Sequelize	ORM framework pro Node.JS, operující mj. nad MySQL databází (autor)
Smarty	šablonovací systém pro PHP (autor)
transpiler	překlad zdrojového kódu jednoho jazyka do zdrojového kódu jiného jazyka (autor)
WAMP	Zkratka pro distribuci webového serveru Apache, MySQL a PHP pro Windows (autor)

[36] Viz http://lesscss.org/

[37] Viz http://locomotivejs.org/

[38] Viz http://mongoosejs.com/

[39] Viz https://npmjs.org/

Seznam literatury

1. **Odehnal, František.** *Komplexní srovnání vývoje webových aplikací ve frameworcích Zend (PHP) a Grails (Java-Groovy).* Vysoká škola ekonomická v Praze. Praha : Fakulta informatiky a statistiky, 2011. Diplomová práce.

2. **Louda, Petr.** *Srovnání platformy .NET s dalšími platformami pro vývoj webových aplikací.* Vysoká škola ekonomická v Praze. Praha : autor neznámý, 2012. Diplomová práce.

3. **Clifton, Marc.** What is a Framework? *CodeProject – for those who code.* [Online] 3. listopad 2003. [Citace: 23. červen 2013.] http://www.codeproject.com/Articles/5381/What-Is-A-Framework.

4. **Nette Foundation.** Rychlý a pohodlný vývoj webových aplikací v PHP. *Nette Framework.* [Online] Nette Foundation. [Citace: 23. červen 2013.] http://nette.org/cs/.

5. —. Kdo používá Nette Framework. *Nette Framework.* [Online] 3. duben 2013. [Citace: 23. červen 2013.] http://nette.org/cs/kdo-pouziva-nette-framework.

6. **Daněk, Petr.** Velký test PHP frameworků: Zend, Nette, PHP a RoR. *Root.cz.* [Online] 11. září 2008. [Citace: 23. červen 2013.] http://www.root.cz/clanky/velky-test-php-frameworku-zend-nette-php-a-ror/. ISSN 1212-8309.

7. **Hughes-Croucher, Tom a Wilson, Mike.** *Node: Up and Running: Scalable Server-Side Code with JavaScript.* Sebastopol : O'Reilly Media, 2012. ISBN 978-1449398583.

8. **Joyent, Inc.** Node.JS in the Industry. *Node.JS.* [Online] [Citace: 23. červen 2013.] http://nodejs.org/industry/.

9. **Tölg, Jan.** *Framework Nette.* Fakulta informatiky a statistiky : Vysoká škola ekonomická v Praze, 2013. Bakalářská práce.

10. **Olišar, Petr.** *Porovnání Nette a Zend Frameworku.* Vysoká škola ekonomická. Praha : Fakulta informatiky a statistiky, 2013. Bakalářská práce.

11. **Grudl, David.** Seriál: Začínáme s Nette Framework. *Zdroják.* [Online] 2009. [Citace: 23. červen 2013.] http://www.zdrojak.cz/serialy/zaciname-s-nette-framework/.

12. **Dočkal, Oto.** Webový CRM systém. *(diplomová práce, Vedoucí práce Martin Sysel).* [Online] 2013. [Citace: 23. červen 2013.] http://theses.cz/id/j6bz47/.

13. **Teixeira, Pedro.** *Professional Node.js: Building Javascript Based Scalable Software.* Indianapolis : Wrox, 2012. ISBN 978-1118185469.

14. **Grudl, David.** Nette Framework: zvyšte svoji produktivitu. *Zdroják.* [Online] 10. březen 2003. [Citace: 23. červen 2013.] http://www.zdrojak.cz/clanky/nette-framework-zvyste-svoji-produktivitu/. ISSN 1803-5620 .

15. **Wikipedia contributors.** Convention over Configuration. *Wikipedia, The Free Encyclopedia.* [Online] 17. květen 2013. [Citace: 23. červen 2013.] https://en.wikipedia.org/w/index.php?title=Convention_over_configuration&oldid=555521666.

16. **readwrite author.** Server-Side Javascript: Back With a Vengeance. *readwrite.* [Online] 17. prosinec 2009. http://readwrite.com/2009/12/17/server-side_javascript_back_with_a_vengeance.

17. **Oracle.** Chapter 2: Getting Started. *Server-Side JavaScript Guide.* [Online] [Citace: 23. červen 2013.] http://docs.oracle.com/cd/E19957-01/816-6411-10/getstart.htm.

18. **O'Dell, Jolie.** Why Everyone Is Talking About Node. *Mashable.* [Online] 10. březen 2011. [Citace: 23. červen 2013.] http://mashable.com/2011/03/10/node-js/.

19. **McLaughlin, Brett.** *What Is Node?* Sebastopol : O'Reilly Media, 2011. ISBN 978-1-4493-1005-9.

20. **phpDocumentor 2 authors.** Learn about phpDocumentor. *phpDocumentor 2.* [Online] [Citace: 23. červen 2013.] http://www.phpdoc.org/docs/latest/index.html.

21. **Purchart, Vašek.** Seriál: Jak na Dependency Injection (3 díly) . *Zdroják.* [Online] 2011. [Citace: 23. červen 2013.] http://www.zdrojak.cz/serialy/jak-na-dependency-injection/. ISSN 1803-5620 .

22. **Oliviera, William.** JS: The Right Way. [Online] 23. duben 2013. [Citace: 23. červen 2013.] http://jstherightway.com/.

23. **Whyte, Aaron, a další.** Google JavaScript Style Guide (revision 2.72). *google-styleguide.* [Online] 21. březen 2013. [Citace: 23. červen 2013.] https://google-styleguide.googlecode.com/svn/trunk/javascriptguide.xml.

24. **Osmani, Addy.** Learning JavaScript Design Patterns (volume 1.5.2). *Addy Osmani: Articles for Developers.* [Online] 2012. [Citace: 23. červen 2013.] http://addyosmani.com/resources/essentialjsdesignpatterns/book/.

25. **Wikipedia contributors .** List of PHP editors. *Wikipedia, The Free Encyclopedia.* . [Online] 23. červen 2013. [Citace: 23. červen 2013.] https://en.wikipedia.org/w/index.php?title=List_of_PHP_editors&oldid=561578207.

26. **Brejla, Ondřej.** NetBeans plugin for Nette. *GitHub.* [Online] 3. březen 2013. [Citace: 23. červen 2013.] https://github.com/nette/netbeans-plugin.

27. **Doleček, Jan.** Latte in IntelliJ IDEA / PhpStorm. *GitHub.* [Online] [Citace: 23. červen 2013.] https://github.com/juzna/intellij-latte.

28. **Stack Overflow users.** IDE for Node.js / Javascript? *Stack Overflow.* [Online] 30. květen 2013. [Citace: 23. červen 2013.] http://stackoverflow.com/questions/3919977/ide-for-node-js-javascript.

29. **The PHP Group.** Built-in web server. *PHP.* [Online] 23. červen 2013. [Citace: 23. červen 2013.] http://php.net/manual/en/features.commandline.webserver.php.

30. **Fowler, Martin a další.** *Patterns of Enterprise Application Architecture.* Stafford : Addison-Wesley Professional, 2002. ISBN 978-0321127426.

31. **Guevara, Jeysson.** Express with Sequelize. *GitHub.* [Online] 30. říjen 2012. [Citace: 23. červen 2013.] https://github.com/JeyDotC/articles/blob/master/EXPRESS WITH SEQUELIZE.md.

32. **Hevery, Miško.** Static Methods are Death to Testability . *Google Testing Blog.* [Online] 17. prosinec 2008. [Citace: 23. červen 2013.] http://googletesting.blogspot.cz/2008/12/static-methods-are-death-to-testability.html.

Seznam obrázků a tabulek

Seznam obrázků

Obrázek 1 21
Obrázek 2 25
Obrázek 3 26
Obrázek 4 28
Obrázek 5 31
Obrázek 6 36
Obrázek 7 38

Seznam tabulek

Tabulka 1: Výsledky základního srovnání (zdroj: Grails, Zend (1); ASP.NET MVC (2); Nette, Node.JS autor) 42
Tabulka 2: ..... Výsledky srovnání podle architektonických vzorů doménové logiky (zdroj: Grails, Zend (1); ASP.NET MVC (2); Nette, Node.JS autor) 44
Tabulka 3: ..... Výsledky srovnání podle implementace vzorů datových zdrojů (zdroj: Grails, Zend (1); ASP.NET MVC (2); Nette, Node.JS autor) 44
Tabulka 4: ..... Výsledky srovnání podle implementace objektově-relačních vzorů chování (zdroj: Grails, Zend (1); ASP.NET MVC (2); Nette, Node.JS autor) 45
Tabulka 5: ..... Výsledky srovnání podle implementace objektově-relačních vzorů chování (zdroj: Grails, Zend (1); ASP.NET MVC (2); Nette, Node.JS autor) 45
Tabulka 6: ..... Výsledky srovnání podle implementace vzorů pro objektově-relační mapování dat (zdroj: Grails, Zend (1); ASP.NET MVC (2); Nette, Node.JS autor) 46
Tabulka 7: ..... Výsledky srovnání podle implementace vzorů pro webové prezentace (zdroj: Grails, Zend (1); ASP.NET MVC (2); Nette, Node.JS autor) 47
Tabulka 8: .. Výsledky srovnání podle implementace vzorů pro souběžné zpracování (zdroj: Grails, Zend (1); ASP.NET MVC (2); Nette, Node.JS autor) 48
Tabulka 9: .... Výsledky srovnání práce s modelem (zdroj: Grails, Zend (1); ASP.NET MVC (2); Nette, Node.JS autor) 49
Tabulka 10: Výsledky osobního hodnocení práce s modelem (zdroj: Grails, Zend (1); ASP.NET MVC (2); Nette, Node.JS autor) 51
Tabulka 11: Výsledky srovnání práce s řadiči (zdroj: Grails, Zend (1); ASP.NET MVC (2); Nette, Node.JS autor) 52

Tabulka 12: Výsledky osobního hodnocení práce s řadiči (zdroj: Grails, Zend (1); ASP.NET MVC (2); Nette, Node.JS autor) 55
Tabulka 13: Výsledky srovnání práce s pohledy (zdroj: Grails, Zend (1); ASP.NET MVC (2); Nette, Node.JS autor) 56
Tabulka 14: Výsledky osobního hodnocení práce s pohledy (zdroj: Grails, Zend (1); ASP.NET MVC (2); Nette, Node.JS autor) 57
Tabulka 15: Výsledky srovnání práce s testy (zdroj: Grails, Zend (1); ASP.NET MVC (2); Nette, Node.JS autor) 58
Tabulka 16: Výsledky osobního hodnocení práce s testy (zdroj: Grails, Zend (1); ASP.NET MVC (2); Nette, Node.JS autor) 59
Tabulka 17: Výsledky celkového osobního hodnocení práce s frameworky (zdroj: Grails, Zend (1); ASP.NET MVC (2); Nette, Node.JS autor) 60
Tabulka 18: Výsledky srovnání dostupné podpory frameworku (zdroj: Grails, Zend, ASP.NET MVC, Nette, Node.JS autor) 63