Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informatiky a kvantitativních metod

Generátor kódu pro webové aplikace v Nette

Bakalářská práce

Autor: Radek Brůha
Studijní obor: Aplikovaná informatika – ai3

Vedoucí práce: Ing. Pavel Kříž, Ph. D.

Hradec Králové duben 2015

Prohlášení:

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne 25.4.2015 Radek Brůha

Poděkování:

Děkuji vedoucímu bakalářské práce Ing. Pavlu Kříži, Ph. D. za odborné vedení a cenné rady při zpracovávání této práce.

Anotace

Bakalářská práce se v první části zabývá analýzou existujících řešení pro automatické generování CRUD aplikací ve vybraných PHP frameworcích – Nette, CakePHP a Symfony. Druhá část práce je pak vzhledem k neexistenci tohoto nástroje pro Nette Framework věnována návrhu a implementaci vlastního řešení s využitím moderních technologií – relačního databázového systému MySQL, knihovny Doctrine poskytující objektově relační mapování, knihovny Nette\Database\Table fungující na principu NotORM a responzivního CSS frameworku Twitter Bootstrap. Vysoký důraz je přitom kladen jednak na využívání doporučených řešení typických úloh v Nette Frameworku, důsledné zabezpečení výsledné aplikace a velkou míru přizpůsobitelnosti generované aplikace pomocí sad šablon. Poslední část práce je pak věnována popisu výchozí sady šablon, principu jejich fungování, postupu tvorby vlastní sady šablon a uživatelskému návodu pro práci s vytvořeným
generátorem.

Annotation

Title: Code-generator for Nette Web Applications

The first part of this Bachelor Thesis analyses existing solutions for automatic code generation in three selected PHP frameworks – Nette, CakePHP and Symfony. Because Nette Framework does not contain this tool, the second part of this work is dedicated to design and implementation own solution using modern technologies –relational database system MySQL, object-relational mapping PHP library Doctrine, Nette\Database\Table library based on NotORM principles and responsive CSS framework Twitter Bootstrap. High emphasis is placed on using best practices for solving common tasks in Nette Framework, high security standards and high degree of adaptability provided by using sets of templates. The last part is devoted to description of the default set of templates, especially principles of their work, process of creating own sets of templates and user instructions how to work with established code generator.

Obsah

Seznam obrázků

Seznam tabulek

Seznam zkratek

CRUD – shrnuje čtyři základní operace s daty – Create (vytvoření), Read (čtení), Update (úprava) a Delete (mazání).

ORM – Object-relational mapping, programovací technika zajišťující automatickou konverzi dat mezi relační databází a objektově orientovaným programovacím jazykem oběma směry.

XSS – Cross-Site Scripting, jeden z útoků na webové aplikace, kdy díky neošetřeným vstupům a výstupům může útočník na napadené stránce spouštět svůj kód.

CSRF – Cross-Site Request Forgery, další z útoků na webové aplikace, kdy útočník pomocí závadné stránky, na kterou vstoupí uživatel, skrytě provede útok na jinou webovou aplikaci, na které je uživatel přihlášen.

MVC - Model View Controller, návrhový vzor psaní aplikací, kde je kód aplikace rozdělen na datovou, prezentační a řídí část za účelem odstínění závislostí mezi nimi.

AJAX – Asynchronous JavaScript and XML je metoda, pomocí které lze překreslovat části obsahu webových stránek bez nutnosti jejich celého nového načtení.

1 Úvod

V dnešním informačním světě si už prakticky nedovedeme bez počítačů a internetu představit život. Většina z nás asi jistě ví, že na počítači běží řada aplikací, které musely být někým naprogramovány, a to samé platí i o webových aplikacích na internetu. Málokdo ale již tuší, že podstatná část programátorské práce na menších projektech tvoří běžnou a stále se opakující rutinu jako v jiných oblastech lidské činnosti. Tato programátorská rutina označovaná zkratkou CRUD[1] se skládá ze čtyř základních operací s daty – vytváření, čtení, úprava a jejich mazání.

Bez ohledu na to, kde jsou data a informace uloženy či v jakém programovacím jazyce jsou aplikace napsány, jsou tyto čtyři základní operace stále stejné a potřebné, protože víceméně každá aplikace pracuje nějakým způsobem s daty a informacemi. Jediné, co se mění, jsou konkrétní data, s nimiž pracujeme, a místo, kde je uchováváme.

Z pohledu programátora se jedná většinou o stereotypní věci, které již mnohokrát za svoji kariéru naprogramoval. Ztracený čas by mohl věnovat řešení důležitějších a zajímavějších problémů, které se vyznačují především tím, že k jejich řešení není možné využít nějaký automatický nástroj.

Z toho důvodu postupně začaly vznikat jednoduché generátory kódu, které na základě nějakého zdroje informací, například struktury dat v nějakém relačním databázovém systému, dokázaly vygenerovat kostru aplikace podporující výše zmíněné CRUD aplikace. Ačkoli se jistě najdou odpůrci této metody, tak bezpochyby dokáže programátorům ušetřit čas a případně začátečníkům pomoci pochopit principy práce v daném programovacím jazyce.

Tato bakalářská práce se zabývá CRUD generátory pro skriptovací jazyk PHP, kdy je v první části provedena analýza těchto nástrojů pro vybrané tři PHP frameworky – český Nette Framework a světové CakePHP a Symfony. Analýzou zjišťujeme, že ačkoliv poslední dva zmiňované generátor obsahují, a dokonce jej prosazují jako základ pro rychlé vytvoření aplikace, tak české Nette tento nástroj postrádá, a

[1] Shrnuje čtyři základní operace s daty – Create (vytvoření), Read (čtení), Update (úprava) a Delete (mazání).

proto se bakalářská práce v další kapitole věnuje jeho návrhu a implementaci. Poslední část analýzy je ještě věnována popisu předpokládaných použitých technologií pro tvorbu tohoto generátoru a funkcí, které by měl obsahovat. Funkce vychází především z řešení konkurenčních frameworků.

Hlavní část obsahuje popis návrhu a implementaci vlastního řešení pro Nette Framework. Práce postupně poodkrývá informace o architektuře aplikace a jejich pomocných třídách na ukládání informací o struktuře databáze v relačním databázovém systému MySQL, kde je například vysvětleno, proč je důležité mít detailní informace o referenční integritě nebo primárních klíčích. Následuje popis implementace získání potřebných informací přímo prostřednictvím SQL dotazů na databázi a jejich další úprava do použitelnějšího stavu. Další část práce se zabývá popisem výchozí sady šablon, podle kterých je výsledná aplikace vygenerována za pomoci technologií vysvětlených v analýze.

Předposlední část je věnována podrobnému popisu tvorby vlastních šablon, čímž generátor získává nový rozměr z důvodu maximální možné přizpůsobitelnosti generované aplikace potřebám jeho uživatele díky jednoduchému a robustnímu API.

Poslední část obsahuje jednoduchou uživatelskou příručku na téma, jak ovládat tento vytvořený generátor.

Pro správné pochopení této bakalářské práce se předpokládají alespoň minimální znalosti v oblasti tvorby webu, a to ideálně za pomoci Nette Frameworku.

2 Existující řešení

2.1 Analýza

Návrhu vlastního řešení generátoru pro Nette Framework předcházela analýza řešení od konkurenčních PHP frameworků – CakePHP a Symfony. Primárním účelem analýzy bylo najít silné a slabé stránky jednotlivých frameworků. Silné stránky se poté staly inspirací pro návrh vlastního řešení, slabé se naopak pokouší odstranit či obejít. Oba frameworky umožňují generování aplikace z databáze a Symfony ještě pomocí entit ORM[2] Doctrine. Pro samotný přístup k databázi pak využívají buď vlastní ORM v případě CakePHP, nebo Doctrine u Symfony. Vlastní řešení proto přichází s kombinací obojího – generování aplikace je možné jak pomocí databáze, tak Doctrine entit a pro přístup k datům využívá jak vlastního řešení Nette Frameworku – Nette\Database\Table, tak i ORM Doctrine. Pro ulehčení práce s designem výsledné aplikace a zajištění reprezentativního vzhledu je využit Twitter Bootstrap. Vlastní řešení dále generuje statické texty do překladových souborů pro snadné doplnění více jazykových verzí aplikace a nabízí generování přístupových práv na úrovni databázových tabulek a jejich sloupců. Samotné přihlašování a přidělování přístupových práv je už však ponecháno na doprogramování uživatelům, protože tato část návrhu výsledné webové aplikace je již hodně subjektivní. Součástí generátoru je i jedna sada výchozích šablon, kterou lze libovolně upravovat, díky požadavku na jednoduchou možnost úpravy stávajících šablon či vytváření vlastních pro specifická použití generátoru.
Samotné vlastní řešení je rozděleno do čtyř hlavních částí – Examiner, Builder, Templates a Utils, kde má každá část má své jasně vymezené úlohy, které budou v následujících kapitolách podrobně vysvětleny.

2.2 Nette Framework 2.2

Pro Nette Framework verze 2 a vyšší v současné době neexistuje žádný oficiální nástroj pro generování CRUD aplikací a jeho tvorba se ani nepředpokládá dle

[2] Object-relational mapping – programovací technika zajišťující automatickou konverzi dat mezi relační databází a objektově orientovaným programovacím jazykem oběma směry

dřívějšího vyjádření původního tvůrce Nette Frameworku, Davida Grudla: *„Nette se bez generátorů obejde, protože kombinuje komponentový a MVC přístup. Takže to, co se píše pořád dokola, a co ti ty generátory generují, si můžeš vytvořit jako komponentu. Mně nepřipadá jako správné dělat generátor, protože si říkám, že v okamžiku, kdy potřebuju generovat spoustu podobného kódu, tak na to asi nejdu úplně dobře. Mám ale v plánu do Nette dát komponentu pro administraci. Stáhneš si Nette, vytvoříš projekt, a budeš mít připravenou administraci. Nechci jít cestou generování kódu, ale cestou komponenty, která tohle vyřeší.“* [1]

Žádná oficiální administrační komponenta se bohužel neobjevila ani do konce roku 2014, i přes lehkou snahu uživatelů Nette fóra vytvořit ji svépomocí. [2]

V případě neoficiálních nástrojů se dají objevit jisté pokusy, například jednoduché generování Nette formulářů na základě databázové tabulky pomocí pluginu do vývojového prostředí NetBeans. [3] Toto řešení je sice plně funkční, ale zabývá se pouze vytvářením formulářů a nelze ho tedy ani zdaleka považovat za plnohodnotný nástroj pro generování CRUDu a je taktéž více než tři roky staré, tudíž není kompatibilní se stávající druhou verzí Nette Frameworku. Jediným dohledatelným pokusem o skutečný CRUD generátor je projekt z roku 2011, který dokáže stáhnout Nette Framework a poté zadáním do příkazové řádky názvu databázové tabulky a definicí sloupců (názvu a datového typu, případně výchozí hodnoty) vygenerovat databázový model využívající Nette\Database\Table, presenter a sadu šablon pro přidání, úpravu a mazání jednotlivých položek. [4] Volitelnou součástí pak byla možnost vygenerování databázové tabulky dle těchto informací. Jak lze vidět z videa [5], nástroj byl poměrně intuitivní a výsledná vygenerovaná aplikace například zohledňovala i databázové datové typy, kdy se logické hodnoty zobrazují pomocí přepínače místo textboxu a podobně. V současné době je však oficiální repozitář na GitHubu nedostupný, autor naposledy reagoval před více než dvěma lety a vzhledem k době zveřejnění se předpokládá, že by generátor nebyl kompatibilní se stávající verzí Nette Frameworku. Jediné, co po tomto projektu zbylo, je výše zmíněné ukázkové video. Dále je možné dohledat další dva až tři projekty, ale vzhledem k tomu, že o nich neexistuje nic více než poměrně staré zmínky na Twitteru [6] [7], tak jsou zde kvůli nedostatku informací zmíněny pouze okrajově.

2.3 CakePHP 2.5

CakePHP je populární PHP framework využívaný zhruba 4,5 % vývojářů, kteří se zúčastnili ankety o nejpopulárnější PHP framework na serveru www.sitepoint.com. [8] Vyvíjený je od roku 2005, kdy vznikl jako reakce na popularitu frameworku Ruby on Rails. Jeho vývoj je zastřešen pod Cake Software Foundation a je vydávám pod licencí MIT. V současné době je aktuální verze dva, konkrétně 2.6.2, a je vydána druhá release candidate verze plánované třetí verze frameworku. Framework byl vybrán zejména proto, že se celá jeho myšlenka opírá o scaffolding a generování kódu za účelem rychlého vývoje prototypu aplikací, které nepotřebují žádnou složitou konfiguraci a stačí pouze nastavit připojení k databázi a „péct" (vychází ze samotného názvu, kdy CakePHP znamená dortPHP a pro jeho „upečení" je využita část konzole s názvem „Bake"). Výhodou CakePHP je licence MIT dovolující jeho použití i pro komerční aplikace. Součástí je i řada bezpečnostních mechanismů zabraňujících nejčastějším útokům proti webovým aplikacím jako je například SQL injection (zjednodušeně řečeno využívá neošetřeného vstupu na webu a přepsání SQL dotazu svým vlastním za účelem získání dat, ke kterým nemá být přístup, či rovnou k jejich smazání). Neméně nebezpečným je Cross-Site Scripting (XSS), který je definován následovně: *„Cross-Site Scripting je metoda narušení webových stránek zneužívající neošetřených výstupů. Útočník pak dokáže do stránky podstrčit svůj vlastní kód a tím může stránku pozměnit nebo dokonce získat citlivé údaje o návštěvnících. Proti XSS se lze bránit jen důsledným a korektním ošetřením všech řetězců."* [9] a Cross-Site Request Forgery (CSRF), který lze definovat takto: *„Cross-Site Request Forgery je útok spočívající v tom, že přimějeme uživatele navštívit stránku, která skrytě vykoná útok na webovou aplikaci, kde je uživatel zrovna přihlášen. Lze takto například pozměnit nebo smazat článek, aniž by si toho uživatel všiml. Proti útoku se lze bránit generováním a ověřováním autorizačního tokenu."* [9]

CakePHP aktuálně podporuje dva druhy generování CRUD aplikací. Prvních z nich je tzv. dynamický scaffolding, při kterém nedochází k přímému generování souborů se zdrojovým kódem, který by byl později upravitelný, ale je zase naopak velmi snadno použitelný. Jelikož je ale tento typ scaffoldingu v současné verzi

označen jako zastaralý a v další majoritní verzi nebude ani zahrnut, nebude se jím tato práce podrobněji zabývat. [10]

Druhý typ scaffoldingu nabízí klasické vygenerování souborů se všemi částmi Model View Controller (MVC – návrhový vzor psaní aplikací, kde je kód aplikace rozdělen na datovou, prezentační a řídí část za účelem odstínění závislostí mezi nimi) aplikace, které je po vygenerování možné dále upravovat, tím pádem je přizpůsobovat svým představám.

Samotné generování CRUD aplikace probíhá prostřednictvím příkazového řádku za použití CakePHP konzole, která vyžaduje přítomnost konzolové verze PHP v operačním systému. [11]

Pro generování se tedy v příkazovém řádku přesuneme do předem připraveného projektu založeného na CakePHP frameworku a spustíme samotný generátor zadáním příkazu *.\Console\cake.bat bake*. Jestliže byl generátor na tomto projektu spuštěn poprvé, automaticky se zobrazí průvodce za účelem vytvoření konfigurace připojení k databázi, který nás postupně požádá o hostitele databázového serveru, jméno a heslo k uživatelskému účtu a nakonec o název databáze, se kterou budeme pracovat. Po jejich vyplnění, překontrolování a potvrzení se na pozadí vygeneruje konfigurační soubor se zadanými údaji. Generátor je připraven ke své činnosti.

Generátor využívá jako zdroj dat databázové tabulky. Jeho nejjednodušší použití spočívá ve vygenerování modelů pro všechny nalezené tabulky prostřednictvím příkazu *.\Console\cake.bat bake model all* a obdobně i presenterů pomocí *.\Console\cake.bat bake controller all* a pohledů *.\Console\cake.bat bake view all*.

Nově vygenerovaná aplikace umožňuje standardní přidávání, úpravu a mazání jednotlivých záznamů z databázových tabulek. Aplikace se snaží inteligentně využívat databázových cizích klíčů, a proto jsou v případě jejich použití k provázání tabulek při přidávání a úpravě záznamů místo klasických textboxů zobrazovány selectboxy s možností přímého výběru dat z provázaných tabulek. Generátor taktéž počítá s různými databázovými typy, kdy se například u datového typu pro zadání času objeví hned několik selectboxů pro zadání jednotlivých částí data a času. Nicméně například v případě výčtových datových typů se zobrazuje klasické textové pole místo selectboxů, které by se k nim hodily mnohem lépe. K dispozici je

také pro případ tabulek s hodně záznamy stránkování a možnost řazení záznamů dle jednotlivých sloupců, které ale nepodporuje vícenásobné řazení.

Z pohledu běžného uživatele je generátor poměrně kvalitní a pro rychlou administraci databázových tabulek je rychle použitelný. Z programátorského hlediska je ale naopak nedostatečné standardní ošetřování chyb, kdy se v případě špatně zformulovaného databázového dotazu zobrazí klasická programátorská chyba bez jakéhokoliv ošetření, která samozřejmě není vhodná pro běžného uživatele, a je nutné si tuto logiku přidat do aplikace ručně.

Vzhledem k možnosti úpravy šablon použitých pro generování lze výše zmíněné problémy poměrně snadno vyřešit. Další nevýhoda CakePHP scaffoldingu může být viděna přímo v šablonách – nepoužívají totiž žádný pokročilý šablonovací systém, ale pouze klasické PHP, které může být v případě složitějších šablon nepřehledné, těžší na orientaci a úpravu.

V případě rozhodnutí upravit generované soubory je nejjednodušší přejít do složky *lib/Cake/Console/Templates/default/views*, kde se nachází čtyři základní soubory šablon a ty zkopírovat do složky *app/Console/Templates/[themename]/views*, kde je *themename* unikátní název námi upravené šablony. V případě, že by nestačila pouze změna šablon a byla by požadována i změna dalších součástí generované aplikace, lze ji provést pomocí zkopírování souborů ze složky *lib/Cake/Console/Templates/skel.* Zkopírované soubory lze následně upravit dle vlastních požadavků.

2.4 Symfony 2.5

Symfony je populární PHP framework využívaný zhruba 10,5 % vývojářů, kteří se účastnili ankety o nejpopulárnější PHP framework na serveru www.sitepoint.com [8], který je vyvíjený od roku 2005, tehdy ještě pod názvem Sensio Framework. Podobně jako CakePHP je Symfony inspirován aplikačními frameworky pro jiné jazyky jako Ruby on Rails či Spring pro Javu. O jeho vývoj se stará Sensio Labs (odtud původní název Sensio Framework) a je vydáván pod licencí MIT, která umožňuje mimo jiné i použití pro komerční účely. Symfony je distribuován ve dvou verzích, první je tzv. legacy verze, která obsahuje původní verzi frameworku vyvíjenou od roku 2005 až do roku 2012, která byla v říjnu roku 2012 nahrazena

novou vývojovou větví označovanou jako Symfony2. V současné době je aktuální verze 2.6.4 a současně je k dispozici i verze 2.3, která je označovaná jako long-term support, nabízející prodlouženou podporu a opravování chyb. V případě komerčních aplikací může být oprava chyb důležitější než poslední dostupná verze s veškerými novinkami.

Framework byl vybrán z důvodu velké celosvětové oblíbenosti a snadného generování aplikací pomocí scaffoldingu. Samotný framework používá jako databázovou vrstvu ORM Doctrine, která se stará o komunikaci s databází. Samozřejmostí je ošetřování dotazů sloužící jako prevence proti SQL injection.

Symfony2, podobně jako CakePHP, provádí scaffolding prostřednictvím své konzole. [12] Pro generování se v příkazovém řádku přesuneme do předem připraveného projektu, který je založený na Symfony2 frameworku a pomocí příkazu *php app/console doctrine:generate:entity* vytvoříme Doctrine entitu zadáním jejího jména a zvolením typu formátu, ve kterém chceme uchovávat informace o namapování PHP třídy (Doctrine entity) na databázovou tabulku. Nejjednodušší možností, jak ukládat mapovací informace, je pomocí anotací přímo v souboru se samotnou entitou. Poté je možno rovnou do entity vložit informace o struktuře databázové tabulky nebo ji vytvořit prázdnou s tím, že si tyto informace dopíšeme ručně. [13] Po využití interaktivního průvodce nebo ruční úpravě souboru s entitou je tato entita pomocí příkazu *php app/console doctrine:schema:update –force* přetvořena v databázovou tabulku, kdy se z entity s využitím informací namapovaných pomocí anotací vytvoří SQL příkaz, který se automaticky spustí nad databází. Pokud již máme databázovou tabulku k dispozici, je možné tento krok samozřejmě přeskočit.

Následujícím příkazem *php app/console generate:doctrine:crud* spustíme CRUD generátor, který nás požádá o název entity, ze které budeme generovat. Dále se nás zeptá, zda chceme generovat pouze zobrazení položek nebo i jejich úpravu a mazání. Poslední dotaz se ptá na způsob uložení informací o entitě, kde musí být vybráno to, co bylo vybráno při vytváření entity v předchozím kroku. [14] Pro správnou funkčnost vygenerované aplikace dle tohoto postupu je ještě zapotřebí úprava routovacích informací v souboru *resources/config/routing.yml* přidáním následujících řádků:

table-name:

resource: *"@YourBundle/Controller/"*

type: *annotation*

Nově vygenerovaná aplikace umožňuje přidávání, úpravu a mazání jednotlivých záznamů. V případě provázání tabulek prostřednictvím cizích klíčů výsledná aplikace pro výběr dat z provázaných tabulek používá místo klasických textových polí selectboxy. Vygenerovaná aplikace však postrádá nějaký grafický layout či stránkování záznamů nebo jejich řazení, a proto je skoro nutností vlastnoruční úprava souborů používaných pro generování aplikace.

2.5 Využité technologie

Výsledná aplikace vygenerovaná Nette generátorem pro svoji funkčnost vyžaduje i některé další knihovny spolupracující s Nette Frameworkem. Jedná se zejména o abstraktní databázovou vrstvu reprezentovanou knihovnou Doctrine. Následuje Kdyby\Translation umožňující snadné přeložení výsledné aplikace do více jazyků a Kdyby\Replicator pro vícenásobné filtrace a řazení obsahu. Neméně důležitá je knihovna Nextras\SecuredLinks zajišťující zabezpečení vůči CSRF útokům.

2.5.1 Doctrine2

Doctrine2 je PHP framework starající se o objektově relační mapování. Poskytuje vysokou míru abstrakce databázové vrstvy za použití objektového přístupu a vlastní dotazovací jazyk *DQL* (*Doctrine Query Language*), který je inspirovaný podobným dotazovacím jazykem *HQL* (*Hibernate Query Language*) objektově relačního frameworku Hibernate pro programovací jazyk Java. [15]

Hlavním účelem ORM frameworků obecně je zajištění automatické konverze dat z relační databáze na objekty určitého objektově orientovaného programovacího jazyku a opačně. Mezi další výhody použití ORM lze zařadit odstínění od psaní samotných SQL dotazů, nicméně většinou také nabízí vlastní dotazovací jazyk pro přímou práci s databází. Výhodou tohoto odstínění od psaní SQL je jistá forma databázové nezávislosti, kdy takto napsaný kód bude funkční na jakémkoliv databázovém systému, který vybraný ORM framework podporuje. [16]

Doctrine2 se konkrétně skládá ze třech hlavních částí. První z nich je podpůrná vrstva Common obsahující různé obecné rozhraní a třídy pro práci s kolekcemi, anotacemi, cachováním a podobně. Tato vrstva je využívána dalšími dvěmi vrstvami. Následuje vrstva *DBAL* (*Database Abstraction Layer*), která zajišťuje databázovou abstrakci a tedy nezávislost na konkrétním databázovém systému prostřednictvím definice proprietárního SQL jazyka – *DQL*. Poslední a nejvyšší vrstva *ORM* (*Object Relational Mapper*) se pak stará o samotné mapování objektů do relační databáze, tedy jejich ukládání, a opačný převod (jejich načítání z databáze).

Doctrine2 je postavena na návrhovém vzoru *Data Mapper*, z něhož vyplývá, že samotné objekty (databázové entity) neobsahují žádné operace pro práci se samotnými daty. O tyto operace se stará další objekt, tzv. Mapper, který se v případě Doctrine2 nazývá *EntityManager*. [17]

2.5.1.1 Database Abstraction Layer

Jedna z hlavních součástí ORM Doctrine2 je právě DBAL, která se stará zejména o databázovou abstrakci, tedy nezávislost na konkrétní databázové platformě a definici proprietárního DQL jazyka.

DBAL využívá pro práci s konkrétními databázovými systémy tzv. drivery. V současné době Doctrine2 podporuje většinu používaných databází – MySQL, PostgreSQL, Microsoft SQL Server, Oracle a SQLite. V případě některých databází nabízí dokonce více driverů, například pro MySQL si lze vybrat buď driver využívající standardní PHP knihovnu PDO, nebo driver využívající MySQLi rozšíření. [18]

DBAL dále poskytuje kromě přívětivého API pro práci s databází i *QueryBuilder*, který umožňuje sestavení dotazu prostřednictvím různých metod: [19]

```
$queryBuilder->select('DATE(last_login) as date', 'COUNT(id) AS users')
   ->from('users')->groupBy('DATE(last_login)')->having('users > 10');

$queryBuilder->select('u.id', 'u.name', 'p.number')
->from('users', 'u')->innerJoin('u', 'phonenumbers', 'p', 'u.id = p.user_id');
```

2.5.1.2 Object Relational Mapper

ORM je druhou z hlavních součástí ORM Doctrine2, která se stará o přemapování dat mezi PHP objekty a relační databází. Doctrine2 tyto objekty, do kterých se databázové záznamy přemapovávají, nazývá Entitami. Obecně platí, že jedna databázová tabulka vyžaduje jednu Doctrine2 entitu. Samotná Doctrine2 entita není ničím jiným než obyčejnou PHP třídou doplněnou o metadata, tedy doplňující informace o dané databázové tabulce. Pro zápis metadat podporuje Doctrine2 hned čtyři možnosti – přímý zápis do třídy pomocí dokumentačních komentářů a anotací, XML, YAML či zápis pomocí samotného PHP kódu. Žádná z možností se nedá označit za nejlepší, výběr záleží čistě na uživateli.

Různé způsoby zápisu metadat ilustruje následující příklad. První ukázka využívá dokumentačních komentářů a anotací, druhá XML a třetí YAML. [20]

```
/** @Entity */
class Message {
   /** @Column(type="integer") */
   private $id;
   /** @Column(length=140) */
   private $text;
   /** @Column(type="datetime", name="posted_at") */
   private $postedAt;
}
```

```
<doctrine-mapping>
<entity name="Message">
<field name="id" type="integer" />
<field name="text" length="140" />
<field name="postedAt" column="posted_at" type="datetime" />
</entity>
</doctrine-mapping>
```

Message:

```
 type: entity
 fields:
  id:
   type: integer
  text:
   length: 140
  postedAt:
   type: datetime
   column: posted_at
```

Metadata u jednotlivých entit určují například název databázové tabulky, do které bude konkrétní entita mapována. V případě sloupců pak určují jejich datový typ, možnost, zda mohou zůstat nevyplněny nebo v případě primárního klíče jeho nastavení a případně to, jakým způsobem se bude generovat a podobně.

Doctrine2 si samozřejmě dokáže poradit i s tabulkami provázanými referenční integritou, ale je nutností správně definovat vazby mezi jednotlivými entitami prostřednictvím sady různých vazeb.

Jako příklad využijeme jednoduchou vazbu 1:N následovanou třemi ukázkami. První je pomocí PHP dokumentačních komentářů a anotací, následuje XML a YAML. [21]

```
class User {
  /**
   * @ManyToOne(targetEntity="Address")
   * @JoinColumn(name="address_id", referencedColumnName="id")
   **/
  private $address;
}
/** @Entity **/
class Address { }
```

```
<doctrine-mapping>
<entity name="User">
```

```
<many-to-one field="address" target-entity="Address">
<join-column name="address_id" referenced-column-name="id" />
</many-to-one>
</entity>
</doctrine-mapping>

User:
 type: entity
 manyToOne:
  address:
   targetEntity: Address
   joinColumn:
    name: address_id
    referencedColumnName: id
```

2.5.1.3 Doctrine Query Language

DQL je proprietární dotazovací jazyk odvozený od Hibernate Query Language (HQL) umožňující spouštění příkazů SELECT, UPDATE a DELETE, které jsou přemapovány do nativního SQL jazyka. Příkaz INSERT není v DQL dostupný z důvodu zajištění konzistence objektového modelu.

V jazyce DQL se místo názvů databázových tabulek využívají Doctrine2 entity a místo sloupců se využívají jejich atributy.

Příklady dotazů v jazyce DQL: [22]

```
SELECT u FROM MyProject\Model\User u WHERE u.age > 20;
SELECT u FROM User u JOIN u.address a WHERE a.city = 'Berlin';
SELECT u, count(g.id) FROM Entities\User u JOIN u.groups g GROUP BY u.id;
```

2.5.2 Nette\Database\Table

Nette\Database\Table je založeno na knihovně NotORM od českého autora Jakuba Vrány, který využívá návrhový vzor ActiveRecord, který lze definovat jako „*Objekt, který obaluje řádek v databázové tabulce nebo pohledu, zapouzdřuje přístup k*

databázi, a přidává doménovou logiku.“ [23] Od návrhového vzoru Data Mapper se tedy liší především přidanými CRUD metodami pro práci s daty.

Hlavní myšlenkou Nette\Database\Table je načítání dat pouze z jedné tabulky. V případě, že je tabulka propojena s jinou, jsou pak položeny dva jednoduché dotazy místo jednoho složitého s pomocí JOINu. Spárování dotazů probíhá interně. [24]

Nette\Database\Table obsahuje dvě základní metody pro procházení tabulek spojených referenční integritou. První z nich je *ref*, která slouží pro získání odpovídajícího databázového řádku z druhé tabulky, na který odkazuje řádek první tabulky pomocí cizího klíče. Druhou je pak metoda *related*, která naopak vyhledává všechny záznamy z druhé tabulky, které se odkazují na záznam z první.

Další silnou stránkou je vybírání pouze těch dat, která budou nutně potřeba při dalším zpracování například v šablonách. Prakticky to funguje tak, že při prvním dotazu se vyberou všechny sloupce a do cache se automaticky uloží informace, které sloupce byly později použity, a při dalším dotazu se již automaticky vyberou tyto sloupce.

Následuje jednoduchý příklad použití Nette\Database\Table s *ref* a *related* doplněný o dotazy, které ukázková konstrukce vykoná: [24]

```
$books = $context->table('book');
foreach ($books as $book) {
   echo 'title:      ' . $book->title;
   echo 'written by: ' . $book->author->name;
   echo 'tags: ';
   foreach ($book->related('book_tag') as $bookTag) {
      echo $bookTag->tag->name . ', ';
   }
}

SELECT `id`, `title`, `author_id` FROM `book`;
SELECT `id`, `name` FROM `author` WHERE (`author`.`id` IN (11, 12));
SELECT `book_id`, `tag_id` FROM `book_tag` WHERE (`book_tag`.`book_id` IN (1, 4, 2, 3));
```

SELECT `id`, `name` FROM `tag` WHERE (`tag`.`id` IN (21, 22, 23));

2.5.3 Kdyby\ Replicator

Kdyby\Replicator je jednoduchá PHP knihovna, která přidává do Nette Frameworku podporu pro replikování formulářů přes nový formulářový prvek *addDynamic*. Umožňuje obalit určitou část formuláře do dynamického kontejneru a pomocí tlačítek se speciální funkcí vytvářet kopie této části formuláře a jejich mazání. Obsahuje i jednoduché možnosti nastavení jako například, že v případě nevyplněné poslední kopie nelze přidat další a podobně. Tyto vlastnosti z ní dělají ideální komponentu například pro tvorbu rozhraní pro řazení tabulek dle více možných kritérií a podobně. Nicméně i zde jsou jistá omezení, kdy autor varuje například před nevhodností použití tohoto doplňku pro odesílání souborů. [25]

2.5.4 Kdyby\Translation

Kdyby\Translation je knihovna přinášející robustní systém na lokalizaci webových aplikací do Nette Frameworku. Prakticky se jedná o implementaci knihovny Symfony\Translation doplněnou o věci, na které jsou uživatelé Nette zvyklí – přidání možnosti uchovávat překladové soubory prostřednictvím souborů ve formátu NEON. [26]

Kdyby\Translation tedy podporuje všechno, co Symfony\Translation. Z těch nepoužívanějších věcí jmenujme třeba podporu placeholderů, tedy míst v překladu, kam se vloží nějaký jiný text (typicky jména uživatele). Další podporovanou věcí jsou možnosti pluralizace, definice více překladů u jazyků, u nichž se liší překlad v závislosti na počtu něčeho v překladu (například: jedno jablko, dvě jablka). [27]

Kdyby\Translation dále nabízí hned tři možnosti výběru správného jazyka. Preferovaný způsob je uvedení zkratky jazyka v URL adrese. Následuje *Accept Language Header*, což je jedna z hlaviček, které prohlížeč automaticky odesílá při požadavku na webový server. Tato hlavička obsahuje zkratky preferovaných jazyků. Poslední možností je manuální zvolení jazyka a jeho uložení do session, kdy

se uživateli web automaticky zobrazí v jeho zvoleném jazyce nehledě na to, jaký jazyk je uložený v URL adrese webu a jaký jazyk preferuje jeho prohlížeč. [26]

2.5.5 Nextras\ SecuredLinks

Nextras\SecuredLinks je jednoduché rozšíření, které přidává obranu proti Cross-Site Request Forgery (CSRF) u webových odkazů. Princip rozšíření je velmi jednoduchý a vychází z principu generování bezpečnostního tokenu, kdy je pro každou metodu, která je označena anotací *@secured*, vygenerován bezpečnostní token, který se uloží do session uživatele a akce se provede v pouze případě, že tento token souhlasí. Z důvodu možné krádeže tohoto bezpečnostního tokenu navíc zohledňuje i parametry dané metody, kdy jiný parametr znamená jiný bezpečnostní token. Případné odcizení tokenu by umožňovalo například smazání pouze jednoho konkrétního článku místo možnosti smazání všech. [28]

3 Vlastní řešení

Ačkoliv původní vývojář Nette Frameworku David Grudl nevidí v generátoru kódu přínos, tak ostatní populární PHP frameworky to vidí jinak – buď je mají zabudované přímo v sobě jako CakePHP, nebo jsou dostupné pomocí rozšíření jako v případě Symfony2. Minimálně z tohoto důvodu se zdá tvorba generátoru pro Nette Framework krokem vpřed. Následující řádky práce se proto zabývají popisem návrhu architektury a implementace vlastního řešení pro Nette Framework.

3.1 Pomocné třídy a úložiště informací o databázi

Jmenný prostor *Utils* obsahuje všechny pomocné třídy pro připojení a práci s databází, pro samotné ukládání informací o databázi a jejich tabulkách, které jsou využívány při samotném generování aplikace, ve snadno použitelné podobě.

3.1.1 Databázová tabulka

Třída *Utils\Object\Table* slouží pro uchování informací o jedné databázové tabulce. Nese v sobě informace o originálním názvu tabulky a ošetřeném názvu od

speciálních znaků a mezer. Tento ošetřený název je využíván pro názvy generovaných souborů a tříd. Třída dále obsahuje informace o komentáři, sloupcích a stavu tabulky, který signalizuje, zda může být tabulka zpracovatelná generátorem nebo zda obsahuje nějaké nastavení, které znemožňuje její automatické zpracování (například chybějící primární klíč či jeho definice na více než jednom sloupci).

3.1.2 Databázový sloupec

Třída *Utils**Object**Column* slouží pro uchování informací o jednom sloupci konkrétní databázové tabulky. Nese v sobě informace o jeho názvu, databázovém datovém typu, zda může, či nemůže nabývat hodnoty NULL, klíčích a indexech, které na něm jsou definovány, výchozí hodnotě, komentáři a případné další informace navíc (například je-li sloupec definován jako auto-increment či má-li nastaveno, že se do něj při aktualizaci záznamu automaticky uloží aktuální datum a čas).

3.1.3 Datový typ databázového sloupce

Třída *Utils**Object**Type* slouží pro uchování informací o databázovém datovém typu jednoho konkrétního databázového sloupce. Nese v sobě informace o jeho názvu, velikosti a případné další informace navíc (v případě výčtového databázového typu ENUM nebo SET obsahuje informaci o položkách, které v sobě uchovávají).

3.1.4 Primární klíč

Třída *Utils**Object**Key**PrimaryKey* slouží pro uchování informace o definování konkrétního sloupce jako primárního klíče tabulky. Informace o primárním klíči je nezbytná, protože slouží k jednoznačné identifikaci záznamu tabulky, a je použita pro rozlišení záznamů v případě jejich editace či mazání, proto tabulka, u které nebyl zjištěn primární klíč, nemůže být generátorem automaticky zpracována.

3.1.5 Unikátní klíč

Třída *Utils**Object**Key**UniqueKey* slouží pro uchování informace, zda je nebo není konkrétní sloupec definován jako unikátní klíč tabulky. Informace o unikátním klíči je vhodná z toho důvodu, že stejně jako primární klíč slouží k jednoznačné identifikaci záznamu tabulky. Unikátní klíč by tedy mohl suplovat chybějící primární klíč, ovšem kvůli omezení Doctrine2 jej není možné smysluplně využít. Omezení spočívá v požadavku existence primárního klíče nad databázovou tabulkou.

3.1.6 Index

Třída *Utils**Object**Key**IndexKey* slouží pro uchování informace, zda je nebo není nad konkrétním sloupcem definován databázový index. Informace o indexu momentálně není nikde při generování použita, ale z důvodu komplexní analýzy databáze je informace o něm uchovávána.

3.1.7 Referenční integrita (cizí klíče)

Třída *Utils**Object**Key**ForeignKey* slouží pro uchování informace, zda je nebo není konkrétní sloupec definován jako součást databázové referenční integrity, tedy cizího klíče odkazujícího na jinou tabulku. Informace o cizím klíči je velmi důležitá pro inteligentní generování aplikace, protože v případě existence cizího klíče vygenerovaná aplikace místo textového pole pro zadání hodnoty automaticky nabízí údaje z odkazované tabulky buď prostřednictvím výběrového pole, nebo přímým otevřením dané tabulky za účelem snadné a intuitivní úpravy a vkládání záznamů. Bohužel MySQL obsahuje i databázový engine MyISAM, který nepodporuje cizí klíče, ale i s tímto generátor počítá. V případě dodržení názvů tabulek a sloupců s cizím klíčem dle specifických konvencí dokáže generátor rozpoznat i tyto vazby mezi tabulkami.

3.1.8 Operace se souborovým systémem

Generátor využívá pro práci se souborovým systémem standardní třídu Nette Frameworku dostupnou pod názvem *Nette**Utils**FileSystem*, která nabízí základní metody pro práci se soubory a adresáři, jako je například jejich vytváření,

kopírování, mazání a podobně. Nicméně nenabízí některé další potřebné metody (pro čtení souboru aj.), a proto existuje třída *Utils\File*, která tuto chybějící funkcionalitu zajišťuje pomocí dvou metod.

První z nich je metoda *read* s parametrem přijímajícím cestu k souboru. V případě existence daného souboru je jeho obsah přečten a vrácen. Neexistuje-li, nebo pokud došlo k nějakému problému při čtení (například z důvodu chybějících práv pro čtení), je vrácena výjimka.

Druhou je metoda *getClassesFromPHPFile*, jež slouží pro zjištění názvů tříd definovaných v souborech s PHP kódem, a která přijímá jako parametr cestu k souboru a vrací seznam nalezených tříd v podobě pole.

3.1.9 Operace s příkazovou řádkou

Pomocí třídy *Utils\CLI* ovládá generátor vstupy a výstupy z příkazové řádky operačního systému. První metoda *read* slouží ke čtení vstupu z konzole. Jako první parametr přijímá informaci o tom, zda chceme, pokud je to možné, převést výstup na číslo. Druhý parametr umožňuje zadat pole povolených vstupů, pomocí kterého je možné definovat pouze určitou množinu povolených vstupních informací. Jako poslední parametr je k dispozici nepovinný callback, který se vykoná v případě, že je omezena množina povolených vstupů a uživatel zadal něco, co do ní nepatří. Druhá metoda *write* se naopak stará o výpis do konzole. Jako první parametr přijímá text, který bude vypsán. Druhý parametr určuje velikost odsazení od levého okraje, třetí pak určuje, zda má být na začátku řádku zobrazena šipka. Čtvrtý parametr definuje, zda má za textem následovat odřádkování, a pátý, zda má být odřádkováno ještě před samotným výpisem textu.

3.1.10 Hlavní spouštěcí třída Nette generátoru

Hlavní třída generátoru *Generator* se stará zejména o výpis informací do konzole a získávání údajů od uživatele. Kromě toho také kontroluje správnou konfiguraci Nette Frameworku, například konfiguraci databázového připojení nebo správnost zaregistrování potřebných služeb a podobně. Dále se také stará o načtení celého Nette Frameworku a dalších knihoven jako například Doctrine2. V souvislosti s Doctrine2 pak obsahuje i metodu *buildFromEntities*, která se stará o

vygenerování databázových tabulek z Doctrine2 entit v případě jejich zvolení jako zdrojového formátu dat. Metoda si ze zadané informace o názvu modulu zjistí relativní umístění Doctrine2 entit, ze kterých mají být vygenerovány databázové tabulky, a načte ty, které nemají v názvu znak ~, určující, že se jedná pouze o zálohu souborů entit. Pro samotné zpracování Doctrine2 entit a jejich převedení na SQL kód, který se postará o vytvoření tabulek v databázi, je použit originální nástroj Doctrine2 *SchemaTool* a jeho metoda *createSchema*. [29] Bohužel tento nástroj nedokáže vygenerovat výsledný SQL kód tak, aby přeskočil již existující databázové tabulky. Dojde tak k vyhození výjimky a nevygenerování žádné tabulky. Protože popsané chování není žádoucí, dochází v případě selhání generování ke zjištění názvu tabulky, která se již v databázi nachází, a metoda je rekurzivně volána s přidaným parametrem již existujících tabulek. Tabulky jsou dále využity k ignoraci entit spojených s těmito tabulkami. Proces vytváření tabulek se tedy opakuje, dokud nejsou zjištěny všechny entity, které již v databázi mají odpovídající tabulku. Poté již nic nebrání úspěšnému dokončení generování tabulek ze zbylých entit.

3.1.11 Konstanty

Třída *Utils**Contants* slouží pro definici konstant (slovně pojmenovaná čísla pro snadnější zapamatování a použití v kódu) napříč celým generátorem. Obsahuje tři sady konstant: první pro zdrojová data obsahující MySQL databázi s InnoDB enginem, který podporuje referenční integritu pomocí cizích klíčů, MySQL databázi s MyISAM enginem bez podpory cizích klíčů a nakonec Doctrine2 entity. Druhá sada konstant je určena pro typ modelů komunikující s databází obsahující na výběr z Nette\Database\Table, která je standardní součástí Nette Frameworku 2.2 nebo Doctrine2. Poslední sadou je pak určeno, jakým způsobem budou zpracovávány vztahy mezi tabulkami definované referenční integritou – buď pomocí výběrových polí, nebo pomocí odkazu na provázanou tabulku, která se otevře v novém okně ve speciálním režimu výběru, který zajistí, že se po výběru konkrétního záznamu uloží do daného textového pole primární klíč vybraného záznamu. Následuje ještě jedna sada konstant, která definuje „správnost“ tabulky,

respektive zda je možné zpracovat ji automaticky generátorem nebo zda obsahuje nějakou chybu (chybějící primární klíč a podobně).

3.2 Průzkum databáze

Jmenný prostor Examiner obsahuje rozhraní *IExaminer* a jeho konkrétní implementace dle typu databázového systému, pro který je generátor použit. Jedinou originálním implementací je *MysqlExaminer*, který je implementací pro databázový systém MySQL. Tyto konkrétní implementace pak obsahují veškerou logiku potřebnou pro získání informací o celé databázi, jejich tabulkách, sloupcích, klíčích a indexech.

3.2.1 Rozhraní pro průzkum databáze

Rozhraní *Examiner\IExaminer* definuje všechny metody, které musí konkrétní implementace *IExamineru* poskytovat. Prakticky se jedná o „návod“, jakou funkčnost musí další programátor, který by do generátoru chtěl přidat podporu dalšího databázového systému, naprogramovat, aby byl generátor schopný jej využít. Obsahuje jednak definici konstruktoru, který jako parametr přijímá databázové připojení ve formě PDO a objekt s veškerými nastaveními generátoru.
„Rozšíření PHP Data Objects alias PDO definuje lehké a konzistentní rozhraní pro práci s databázemi. Každý databázový ovladač implementující PDO rozhraní může vyzdvihnout specifické funkce databáze prostřednictvím rozšiřujících funkcí. Samotné PDO neumožňuje vykonávat žádné databázové operace, vždy je nutné využít konkrétní databázový PDO ovladač pro vámi zvolený typ databáze. PDO avšak poskytuje pouze přístupovou abstrakční vrstvu, což znamená, že bez ohledu na typ databáze, kterou používáte, můžete používat stejné funkce pro zasílání příkazů databázím a načítání dat z nich. PDO tedy neposkytuje úplnou databázovou abstrakci jako je například odstínění od psaní SQL dotazů či emulaci chybějících vlastností. Pokud vyžadujete podobnou funkcionalitu, měli byste využít plnohodnotnou abstrakční vrstvu.“ [30]
Obsahuje jedinou metodu *getTables*, vracející pole kompletně zanalyzovaných databázových tabulek, které jsou jako jediné využívány generátorem při

generování výsledné aplikace. Jaké informace o databázi (včetně konkrétního způsobu) budou získány, není nijak definováno, záleží na uvážení programátora, jenž by případnou podporu další databáze implementoval.

3.2.2 Implementace průzkumu MySQL databáze

Třída *MysqlExaminer* nacházející se ve jmenném prostoru *Examiner* je jedinou oficiální implementací výše zmíněného rozhraní *IExaminer* pro průzkum databáze, která obsahuje postupy vedoucí k získání předepsaných informací o samotné databázi a jejich tabulkách nad databázovým systémem MySQL. Při vytváření instance je třídě předáno v konstruktoru databázové připojení pomocí výše zmíněného abstraktního databázového rozhraní PDO a objekt s dosavadními nastaveními generátoru. Nad instancí je poté volána metoda *getTables* vracející pole objektů typu *\Bruha\Generator\Utils\Object\Table* obsahující kompletní informace o všech zanalyzovaných databázových tabulkách. Uvnitř metody je nejdříve zavolán SQL kód *SHOW TABLE STATUS;* který v prostředí MySQL zobrazí základní informace o všech databázových tabulkách. Z těchto informací je převzat pouze název tabulky a její komentář. Pro každou takto zjištěnou tabulku je dále volána metoda *getColumns*, která se stará o hlubší analýzu sloupců tabulky, která vrací sloupce jako pole objektů *typu \Bruha\Generator\Utils\Object\Column.* Metoda přijímá jako jediný parametr dříve zjištěný název databázové tabulky a volá SQL kód *SHOW FULL COLUMNS FROM názevTabulky;* který zobrazí rozšířené informace o všech sloupcích databázové tabulky, mezi které patří zejména název daného sloupce, jeho datový typ v surovém formátu, informace, zda může nabývat hodnotu NULL, informace, zda je na něm definován nějaký klíč či index, výchozí hodnota, komentář a jakékoliv další informace navíc.

Field	Type	Collation	Null	Key	Default	Extra	Privileges	Comment
id	int(10) unsigned	*NULL*	NO	PRI	*NULL*	auto_increment	select,insert,update,references	ID zboží
category_id	int(10) unsigned	*NULL*	NO	MUL	*NULL*		select,insert,update,references	Kategorie zboží
name	varchar(50)	utf8_czech_ci	NO	UNI	*NULL*		select,insert,update,references	Název zboží
description	text	utf8_czech_ci	YES		*NULL*		select,insert,update,references	Popis zboží

Obrázek 1 Ukázka výpisu SQL kódu SHOW FULL COLUMNS FROM items;

Protože práce se surovým datovým typem by byla nepohodlná, je dále zpracován metodou *getColumnType*, která jako parametr přijímá surový datový typ a vrací

objekt typu *\Bruha\Generator\Utils\Object\Type*. Zde dochází k přeměně surového datového typu na tři části – název datového typu, jeho velikost a doplňující informace ke specifickým datovým typům. V případě, že se v surovém datovém typu nachází závorky, pak se datovým typem stává text před nimi a obsah závorek určuje jeho délku. V opačném případě je datový typ název celého surového datové typu. Dále se může v surovém datovém typu vyskytovat klíčové slovo *unsigned*, které značí, že jde o kladné číslo nebo *zerofill*, což značí, že čísla různých délek budou zarovnána k pravému okraji pomocí doplnění nul zleva podle nevětšího z nich. Rozpoznání *zerofill* je důležité zejména proto, že automaticky znamená i vlastnost *unsigned*. [31]

V případě, že je výsledným datovým typem *TINYINT* s délkou jedna, dochází k jeho přejmenování na *BOOLEAN*, který je právě v MySQL databázi vnitřně interpretován jako *TINYINT* s délkou jedna. [31]

Posledním speciálním typem jsou tzv. výčtové datové typy, do kterých patří *ENUM* a *SET*. V tomto případě doplňující informace obsahuje pole jejich možných hodnot a délka těchto datový typů je rovna počtu možných hodnot.

Dále následuje volání metody *getColumnKeys*, která přijímá jako parametry název databázové tabulky a název jejího sloupce, protože informace o klíčích a indexech získané pomocí příkazu *SHOW FULL COLUMNS;* nejsou dostatečně podrobné pro další zpracování. Metoda vrací pole zjištěných indexů a klíčů nad daným sloupcem jakožto objektů čtyř možných typů – *PrimaryKey* pro primární klíč, *UniqueKey* pro unikátní klíč, *IndexKey* pro normální index a *ForeignKey* pro tzv. cizí klíče, které zajišťují referenční integritu. Metoda volá SQL kód *SHOW INDEX FROM názevTabulky WHERE Column_name = názevSloupce;*, který zobrazí všechny klíče a indexy definované nad sloupcem tabulky. Pokud je sloupec primární klíč, pak se jeden z jeho klíčů musí nutně jmenovat *PRIMARY*. [32]

V případě, že se jmenuje jinak než *PRIMARY* a obsahuje informaci, že je unikátní, jedná se o unikátní klíč, v opačném případě se jedná o obyčejný index.

Pro zjištění poslední skupiny klíčů, tedy cizích klíčů, které mohou být v MySQL definovány pouze nad indexem či unikátním klíčem zejména z důvodu rychlosti, je využita metoda *getColumnForeignKeys*, která přijímá taktéž jako parametry název

databázové tabulky a název jejího sloupce a vrací pole objektů typu *ForeignKey*. [33]

V případě využití MySQL enginu InnoDB, který referenční integritu za pomoci cizích klíčů podporuje, je vykonán SQL dotaz *SELECT REFERENCED_TABLE_NAME AS 'table', REFERENCED_COLUMN_NAME AS 'column' FROM information_schema.KEY_COLUMN_USAGE WHERE TABLE_SCHEMA = DATABASE() AND REFERENCED_TABLE_NAME IS NOT NULL AND TABLE_NAME = názevTabulky AND COLUMN_NAME = názevSloupce;*, který vypíše informace o všech cizích klíčích definovaných nad daným sloupcem. Menší problém nastává v případě enginu MyISAM, který je stále poměrně dost rozšířený, ale nepodporuje referenční integritu pomocí cizích klíčů. U verze MySQL 4.1 se počítalo s přidáním podpory referenční integrity pro MyISAM, ale ani v aktuální verzi 5.7 nebyla přidána. [34] [35]

V tomto případě je využito konvenčního pojmenování sloupců pro nahrazení definovaných cizích klíčů, přičemž je využita přímo konvence Nette Frameworku. [36] Primární klíč každé tabulky se jmenuje id a sloupec, kterým se odkazuje jako cizím klíčem na jinou tabulku, se jmenuje *názevOdkazovanéTabulky_id*. V případě dodržení těchto konvencí dokáže generátor správně detekovat referenční integritu stejně, jako kdyby byly definovány cizí klíče. V případě, že sloupec obsahuje znak podtržítka a část názvu sloupce před podtržítkem odpovídá názvu nějaké databázové tabulky, je daný sloupec vyhodnocen jako cizí klíč.

Poslední volanou metodou je *getColumnForeignKeyValue*, která přijímá jako parametr objekt typu *ForeignKey* a vrací název sloupce z odkazované tabulky. Cílem této metody je zajistit zobrazení lepšího údaje, než nic neříkajícího cizího klíče. V případě, že se nějaký sloupec odkazované tabulky jmenuje *title* nebo *name*, je zobrazen jeho obsah místo cizího klíče. V případě, že není takový sloupec dostupný, vezme se první sloupec datového typu varchar nebo char, který by taktéž mohl obsahovat relevantnější data. V případě dalšího neúspěchu se vezme druhý sloupec tabulky a v případě, že neexistuje, tak první.

Jako poslední věc se v tabulce nastavuje příznak *state*, který určuje, zda je tabulka schopná automatického zpracování. V tomto případě se jedná o to, zda obsahuje

primární klíč. Pokud obsahuje primární klíč, je vše v pořádku. V opačném případě je označena jako chybná.

3.3 Generování aplikace

Jmenný prostor *Builder* obsahuje jedinou třídu *Builder*, která implementuje logiku generování aplikace pomocí sad šablon, které jsou podrobně rozebrané v následující kapitole 3. 5. Tvorba vlastních šablon.

3.3.1 Implementace generování aplikace

Třída *Builder* je instanciována konstruktorem s jediným parametrem obsahujícím objekt s kompletním nastavením generátoru od uživatele a volá metodu *build*, která je zodpovědná za kompletní vygenerování aplikace s využitím zadaných dat uživatelem, samotnou databází a použitou šablonou. Uvnitř metody nejprve dochází k vyhledání všech spustitelných šablon ve formátu *Latte*, které mají koncovku *.latte*, a následnému odstranění této koncovky. Dále dochází k odstranění slova *Module*, a to v případě, že uživatel zvolil negenerování do modulu. Může dojít i k nahrazení slova *Module* reálným názvem modulu v případě, že uživatel vybral generování do modulu. Dále je na řadě rozhodnutí, které modely budou zpracovány. Pokud uživatel zvolil Nette\Database\Table modely, pak budou všechny šablony, které mají kdekoli v cestě název *D2*, vynechány a naopak ty, které tam mají *NDBT*, budou zpracovány. V případě zvolení generování Doctrine2 modelů budou naopak vynechány šablony mající v cestě *NDBT* a spuštěny budou ty, které tam mají *D2*. Před samotným překopírováním spuštěných šablon jsou prefixy *NDBT* a *D2* z názvu výsledného souboru odstraněny. Speciálním případem jsou pak šablony, které ve své cestě obsahují slovo *Table* – toto slovo bude nahrazeno reálným názvem tabulky a daná šablona bude zpracována pro každou databázovou tabulku zvlášť. Poslední pravidlo pak říká, že v případě, má-li šablona pouze jednu koncovku, a to *.latte*, tak bude pouze spuštěna, ale nebude nikam překopírována.

Po zpracování všech *Latte* šablon dochází k překopírování všech zbývajících souborů jedna ku jedné do kořenového adresáře generované aplikace.

3.4 Výchozí šablony

Nette generátor obsahuje pouze jednu oficiální a zároveň i výchozí sadu šablon s názvem *default* umístěnou v adresáři *src/Templates/default*. Výchozí šablona generuje aplikaci umožňující spravovat obsah všech databázových tabulek za použití moderních technik, jako je AJAX[3]. Dále je připravena na vícejazyčné rozhraní prostřednictvím použití překladových souborů pomocí rozšíření *Kdyby\Translation* a podobně. [37]

Zároveň se aplikace snaží dodržovat základní principy Nette Frameworku, ať již v podobně adresářové struktury či doporučených postupů, jak v Nette programovat čistě. [38]

3.4.1 Výkonné jádro aplikace

Jádro aplikace se nachází v adresáři *app*, který obsahuje všechny námi naprogramované části aplikace. Základem je adresář *Module*, který se v případě generování do modulu přejmenuje na skutečný název modulu zadaný uživatelem Nette generátoru, nebo se z cesty odstraní. Odstranění nastává v případě, pokud nebylo zvoleno generování do modulu. V další úrovni se pak nachází trojice adresářů pro modely, presentery a šablony – *models*, *presenters*, *templates*. Adresář *models* obsahuje dvě sady modelů, jednu pro Nette\Database\Table modely a druhou pro Doctrine2 modely, ke které ještě patří adresář *Entities*, do níž budou vygenerovány Doctrine2 entity získané reverzním inženýrstvím z databáze a šablony pro vygenerování základních oprávnění generované aplikace.

Šablona *D2BaseRepository.php.latte* obsahuje abstraktní třídu modelu založenou na *Doctrine2*, zatímco šablona *D2TableRepository.php.latte* obsahuje již konkrétní model pro každou databázovou tabulku. Podobně je to v případě Nette\Database\Table modelů až na to, že místo prefixu *D2* používají prefix *NDBT*. Poslední soubor *Permissions.php.latte* slouží pro vygenerování základních přístupových oprávnění (zobrazení, přidání, úpravu a smazání) pro každou tabulku jako celek a také pro každý sloupec databázové tabulky zvlášť.

---

[3]Asynchronous JavaScript and XML alias AJAX je metoda, pomocí které lze překreslovat části obsahu webových stránek bez nutnosti jejich celého nového načtení.

Šablona *BasePresenter.php.latte* obsahuje abstraktní presenter, který slouží jako předek všem presenterům pro konkrétní databázové tabulky. Stará se mimo jiné zejména o kontrolu oprávnění pro zobrazení, přidání, úpravu a smazání položek jednotlivých tabulek a zobrazování správně seřazených a vyfiltrovaných položek. Dále definuje formulář pro filtrování, řazení a ovládání počtu zobrazených položek tabulky. Za tímto účelem využívá rozšíření *Kdyby\Replicator*, pro dynamické přidávání položek formuláře, které zajišťují filtraci a řazení dle prakticky neomezeného množství pravidel. [39]

Definuje také zpracování formuláře pro přidání a úpravu stávajícího záznamu, kdy volá jednotlivé modelové funkce. Obsahuje také základní ošetření běžných chybových stavů týkajících se zejména vkládání duplicitního údaje do sloupce označeného unikátním klíčem a snahou vložit NULL hodnotu do sloupce, který NULL hodnoty neumožňuje. V případě jiné chyby zobrazuje její znění generované samotným databázovým systémem.

Poslední věcí, o kterou se stará, je vyřizování požadavků na smazání položky tabulky, která obsahuje základní ošetření běžných chybových stavů týkajících se zejména nemožnosti smazat záznam kvůli porušení referenční integrity (cizího klíče). V případě jiné chyby zobrazuje její znění generované samotným databázovým systémem.

Šablona *TablePresenter.php.latte*, která je díky svému prefixu *Table* zpracovávána pro každou databázovou tabulku zvlášť, pak zajišťuje zejména přístup k modelu, pomocí kterého pracuje s databází, a k definici pole sloupců dané tabulky spolu s jejich názvem či komentářem, pokud je k dispozici. Dále definuje formulář pro přidání a úpravu jednotlivých položek dané tabulky. První část se stará o vygenerování správného formulářového typu dle databázového typu – například textbox pro krátké řetězcové a číselné typy, texteareu pro dlouhé textové typy a selectboxy pro datové typy EMUM, SET a BOOLean. Poté jsou přidány speciální HTML5 datové typy jako *number* pro čísla a *date*, *datetime* a *time* pro zadávání časových údajů. Následuje přidání výchozích hodnot, pokud je daný sloupec má nastaveny, a přidání atributu *placeholder* zobrazujícího název daného formulářového prvku. Poslední část zahrnuje přidání nejrůznějších validačních pravidel na ověření správnosti zadaných hodnot. Prvním z nich je validační

pravidlo zajišťující povinnost vyplnit daný formulářový prvek, které je aplikováno v případě, že daný sloupec neumožňuje vložní NULL hodnoty a zároveň nemá definovanou výchozí hodnotu a nejedená se o primární klíč s definovaným autoincrementem či časový datový typ s nastavením automatické změny času v případě změny položky. V opačném případě je místo toho přidána podmínka, že všechny následující validační pravidla jsou platná pouze za předpokladu, že uživatel explicitně zadá vlastní hodnotu. Následuje sada validačních pravidel pro číselné datové typy. V případě celočíselných datových typů je přidáno jednak validační pravidlo kontrolující vložení celého čísla a také validační pravidlo rozsahu, které zajišťuje, aby uživatel nemohl překročit maximální velikost daného datového typu. V případě desetinných datových typů je pak přidáno pouze pravidlo zajištující kontrolu, zda se jedná o libovolné číslo. Jako poslední se přidává validační pravidlo maximální délky pro textové datové typy. V případě dynamických datových typů je maximální délka určena z databáze, v případě statických textových datových typů pak z maximálního počtu znaků, které je daný textový datový typ schopen pojmout. Dochází zde taky k odlišení, zda uživatel zvolil zobrazení cizích klíčů pomocí výběru z tabulky či pomocí selectboxu. V případě zvolení první hodnoty je cizí klíč zobrazen jako textbox pouze pro čtení, pod ním se nachází odkaz pro otevření dané tabulky, na kterou odkazuje. Pokud však uživatel vybral selectbox, pak se místo textboxu zobrazí speciální selectbox s vyhledáváním, který taktéž umožňuje snadné najití položky.

3.4.1.1 Výpis obsahu databázové tabulky

Struktura výpisu položek jednotlivých databázových tabulek je umístěna v šabloně *app/Module/templates/Table/list.latte.latte*, přičemž ve vygenerované aplikaci je název adresáře *Table* nahrazen reálným názvem dané databázové tabulky. Nejdříve je definován blok *title*, který obsahuje buď ošetřený název databázové tabulky, nebo její komentář (pokud je dostupný). Ten je poté použit jako hlavní nadpis stránky a v HTML tagu *title*, který se zobrazuje v titulku okna internetového prohlížeče. Druhá část šablony obsahuje výpis formuláře umožňujícího filtrování a řazení dle zadaných údajů spolu se změnou počtu zobrazovaných položek databázové tabulky na jedné stránce. Hlavní část šablony samozřejmě patří

samotnému výpisu položek databázové tabulky, kde jsou jako záhlaví sloupců tabulky použity názvy sloupců databázové tabulky nebo jejich komentáře (pokud existují). Poslední sloupec vedle záhlaví pak zobrazuje tlačítko pro přidání nové položky. Tělo tabulky pak zobrazuje jednotlivé položky a poslední sloupec vždy zobrazuje dvě tlačítka – jedno pro úpravu dané položky a druhé pro její smazání. V případě, že je na daném sloupci databázové tabulky definován cizí klíč odkazující na primární klíč jiné tabulky, pak je v tabulce místo něj zobrazen jiný sloupec, který byl v odkazované tabulce automaticky vybrán jako nejlepší možný. Patička tabulky pak zobrazuje přehledné stránkování, v němž je vždy zobrazeno tlačítko pro přechod na první a poslední stránku spolu se výrazněným tlačítkem aktuální stránky, na které se nacházíme. Pro přechod mezi stránkami je z pravé i levé strany od aktuální stránky zobrazeno nejvýše pět stran pro přechod na jinou stranu. Po kliknutí na tlačítka přidání nové položky nebo úpravy té stávající, dojde k přesměrování na šablonu zmíněnou níže. Při kliknutí na tlačítko pro smazání položky se zobrazí javascriptové potvrzení smazání dané položky. Potvrdíme-li ho, dojde ke smazání položky z databázové tabulky. Samozřejmostí je podpora oprávnění jednak na možnost zobrazení samotné tabulky, tak možnosti přidávání, úpravy a mazání jejich položek. Nicméně výchozí sada šablon jde ještě dál a umožňuje nastavit možnosti zobrazení, přidání, úpravy a mazání na úrovni jednotlivých sloupců databázových tabulek. Pro každou skupinu uživatelů výsledné aplikace je tedy možné určit, že bude schopná přidat novou položku, ale upravit bude možné pouze některou část této položky.

3.4.1.2 Výpis formuláře pro změnu položky databázové tabulky

Struktura výpisu formuláře pro přidávání a úpravu položek databázových tabulek je umístěna v šabloně *app/Module/templates/Table/change.latte.latte*, přičemž ve vygenerované aplikaci je název adresáře *Table* nahrazen reálným názvem dané databázové tabulky. Nejdříve je definován blok *title*, který obsahuje buď ošetřený název databázové tabulky, nebo její komentář (pokud je dostupný) spolu s informací, zda jde o přidání nové položky nebo úpravu stávající. Dále je blok *title* použit jako hlavní nadpis stránky a v HTML tagu *title*, který se zobrazuje v titulku okna internetového prohlížeče. Zbylá část šablony se stará o samotný výpis

formuláře. Jako první probíhá rozhodnutí, zda přidávání a úpravu sloupců s cizími klíči budeme realizovat pomocí otevření celé tabulky v novém okně nebo pomocí selectboxu s vyhledávám na základě výběru uživatele generátoru. Podle toho vykreslíme správný formulář. Nejdříve je vykreslen popisek daného prvku formuláře, poté prvek samotný. V případě, že se jedná o prvek sloužící k výběru data s časem, samotného data či času, je k němu přidáno tlačítko pro její interaktivní výběr pomocí kalendáře. V případě, že uživatel generátoru zvolil úpravu hodnot sloupců s cizím klíčem pomocí otevření tabulky v novém okně, tak je ještě pod formulářovým prvkem umístěn odkaz sloužící právě pro otevření této tabulky. Poslední část šablony pak definuje blok *jsControl*, který obsahuje inicializaci javascriptového kódu pro zobrazí pokročilého selectboxu s vyhledáváním místo klasického selectboxu, který je naopak využit v případě, že uživatel generátoru zvolil výběr hodnot sloupců s cizím klíčem pomocí selectboxu s vyhledáváním.

3.4.1.3 Výpis celkového grafického layoutu aplikace

Struktura výpisu celkového grafického layoutu aplikace je umístěna v šabloně *app/Module/templates/@layout.latte.latte*. Grafický layout je vytvořen tak, aby splňoval pravidla nejnovější specifikace formátu HTML5. V horní části je umístěna hlavička HTML5 obsahující různé meta informace o stránce a přilinkování souborů s kaskádovými styly. Grafický layout je vytvořen pomocí předpřipravených kaskádových stylů Twitter Bootstrap, které dodávají aplikaci moderní vzhled a responzivní[4] funkcionalitu. V horní části je definován pruh nesoucí název vygenerované aplikace. Po levé straně je pruh, který zobrazuje seznam všech databázových tabulek, z nichž byla aplikace vygenerována, a slouží pro snadné přepínání mezi nimi. Pravá a zároveň největší část obsahuje buď výpis položek databázových tabulek, nebo formulář pro jejich přidávání či úpravu. Nad touto částí je pak ještě zobrazen pruh zpráv, kde jsou zobrazovány různé informace z aplikace, například že položka byla úspěšně upravena. V případě neúspěchu se

[4] Jedná se o způsob stylování HTML dokumentů takovým způsobem, aby výsledná stránka byla dobře použitelná na různých zařízeních – například na počítačích, tabletech a mobilech.

zde zobrazuje přesně znění chyby, ke které došlo, a podobně. Poslední část šablony pak obsahuje přilinkování javascriptových souborů pro zajištění správné funkčnosti stránky. Jedná se především opět o soubory Twitter Bootstrapu a Nette Frameworku, tedy validaci formulářů Nette Frameworku, zajištění „AJAXifikace“ aplikace, zobrazování kalendářů pro výběr dat a nahrazení standardního selectboxu pokročilým s vyhledáváním. [40]

3.4.1.4 Konfigurace

Struktura konfigurace Nette Frameworku a výsledné vygenerované aplikace je umístěna v šabloně *app/configconfig.neon.latte* a *app/config/config.local.neon.latte*. Obsah souboru *config.local.neon* je vymazán, protože ho výchozí sada šablon nijak nevyužívá. Veškerá konfigurace tedy probíhá prostřednictvím souboru *config.neon*, do kterého jsou pouze doplněny modely jednotlivých databázových tabulek v podobě zaregistrovaných služeb a služba zajišťující vytvoření oprávnění jednotlivých uživatelů.

3.4.1.5 Překladové soubory

Struktura výchozích překladových souborů je umístěna v šabloně *app/lang/generator.en_US.neon.latte*. Výsledná vygenerovaná aplikace využívá překladových souborů z důvodu snadného přidání další jazykové mutace aplikace a možnosti snadno, rychle a na jednom místě upravit jakýkoliv text aplikace. Struktura vygenerovaného překladového souboru obsahuje nejdříve překlady presenterů, kde jsou definovány názvy popisků jednotlivých formulářových prvků. Následují překlady šablon, kde se naopak zase jedná o názvy jednotlivých sloupců zobrazovaných v záhlaví tabulky spolu s názvem celé tabulky a překladem názvu formuláře pro přidávání či úpravu položky databázové tabulky. Poslední část pak obsahuje překlady informací společných pro celou aplikaci. Jedná se zejména o překlady různých chybových hlášek při přidávání, úpravě a mazání položek databázových tabulek, chybové hlášky formulářových validačních pravidel a popisky tlačítek zobrazovaných ve filtrovacím a řadícím formuláři s možností definice počtu zobrazovaných položek databázové tabulky na jedné stránce.

3.4.1.6 Routování a uživatelsky přívětivé URL

Struktura routeru pro překládání URL adres na správné moduly, presentery a jejich akce je umístěna v šabloně *app/router/RouterFactory.php.latte*. Jako výchozí stránka aplikace je zobrazován výpis položek první zpracované databázové tabulky. URL adresy jsou sestavovány ve tvaru *názevModulu/názevTabulky/názevAkce*, kde je název modulu nahrazen reálným názvem modulu, nebo vynechán v případě, že uživatel zvolil možnost generování aplikace bez modulu. Název tabulky je nahrazen reálným názvem databázové tabulky a název akce je *list* (v případě zobrazení položek databázové tabulky) a *change* (přidávání nové nebo úpravy stávající položky databázové tabulky). Jelikož je akce zobrazení položek databázové tabulky výchozí, tak se slovo list na konci URL adresy nezobrazuje.

3.4.2 Statické soubory

Statické soubory (nejsou zpracovávány prostřednictvím Latte enginu) jsou umístěny v adresáři *www*.

3.4.2.1 Kaskádové styly

Soubory kaskádových stylů se nachází v adresáři www/css a obsahují kaskádové styly pro grafický layout aplikace.

3.4.2.2 Obrázky

Soubory obrázků se nachází v adresáři *www/images* a obsahují faviconu (obrázek, který je zobrazován vedle titulku stránky v internetovém prohlížeči) a GIFovou animaci v podobně otáčející se šipky, která signalizuje zpracovávání AJAXového požadavku na pozadí aplikace.

3.4.2.3 Javascript

Soubory javascriptu se nachází v adresáři *www/js* a slouží pro zajištění správné funkčnosti stránky. Jedná se především o validaci formulářů, zajištění „AJAXifikace“ aplikace, zobrazování kalendářů pro výběr dat a nahrazení standardního selectboxu pokročilým s vyhledáváním.

3.5 Tvorba vlastních šablon

Generátor nabízí jednak možnost úpravy výchozí sady šablon nebo rovnou vytvoření nové vlastní sady šablon. Možností, kde lze toto využít, je velké množství. Například výchozí sada šablon generuje přístupová práva na úrovni databázových tabulek a sloupců, ale někdo by mohl požadovat i přístupová práva na úrovni řádků. Jednou možností je použít výchozí sadu šablon a do aplikace ručně dopsat danou funkcionalitu. Mnohem lepší řešení z hlediska opětovného použití v budoucnu je ale zkopírování výchozí sady šablon, jejich upravení a použití. Navíc o takto vytvořené šablony se můžeme podělit i s komunitou, která je pak bude moci také využívat pro svou potřebu.

3.5.1 Vytvoření kořenového adresáře a zavaděče šablon

Každá sada šablon se skládá z kořenového adresáře pojmenovaného podle názvu sady šablon. Dále obsahuje libovolné množství podadresářů a souborů.

```
Root                    ← Root directory for project
├── Blog                ← Root directory for templace named Blog
├── Game                ← Root directory for templace named Game
├── Personal            ← Root directory for templace named Personal
└── TemplateLoader.php  ← Template loader for Nette generator
```

Obrázek 2 Adresářová struktura pro vytvoření tří sad šablon a jejich zavaděče

Aby byla nově vytvořená sada šablon dostupná v generátoru, je nutné pro ni vytvořit zavaděč. Zavaděč je soubor pojmenovaný *TemplateLoader.php.* Je umístěný vedle kořenového adresáře dané sady šablon a obsahuje třídu pojmenovanou stejně jako název sady šablon doplněnou o koncovku *Template*.

```php
<?php // TemplateLoader.php
class BlogTemplate {}
class GameTemplate {}
class PersonalTemplate {}
```

Obrázek 3 Obsah souboru TemplateLoader.php, zavaděče sady šablon

3.5.2 Tvorba podadresářů a souborů

Každá sada šablon se skládá z různých adresářů a souborů, se kterými je zacházeno různě podle jejich názvu v závislosti na volbě uživatele generátoru.

3.5.2.1 Zacházení s podadresáři

Všechny podadresáře sady šablon jsou automaticky vytvořeny v kořenovém adresáři aplikace, kromě níže uvedených výjimek.

Podadresáře začínající na Module

Pokud uživatel zvolil generování aplikace do modulu, pak se řetězec Module nahradí reálným názvem modulu zadaným uživatelem.

Tabulka 1 Příklady zpracování adresářů Module při generování do modulu

T1[5]	/app/**Module**/presenters
A1[6]	/app/**ExampleModule**/presenters
T2	/app/**Module**/models/**Module**/repositories
A2	/app/**ExampleModule**/models/**ExampleModule**/repositories

Pokud uživatel zvolil, že nechce výslednou aplikaci generovat do modulu, pak je podadresář začínající na řetězec Module z adresářové struktury vynechán.

Tabulka 2 Příklady zpracování adresářů Module při generování bez modulu

T1	/app/**Module**/presenters
A1	/app/presenters
T2	/app/**Module**/models/**Module**/repositories
A2	/app/models/repositories

Podadresáře začínající na NDBT

Pokud uživatel zvolil generování aplikace pomocí Nette\Database\Table modelů, pak se řetězec NDBT odstraní z názvu podadresáře před jeho vytvořením v adresáři aplikace.

Tabulka 3 Příklady zpracování adresářů NDBT při generování Nette\Database\Table modelů

T1	/app/models/**NDBT**/repositories
A1	/app/models/repositories

[5] Zkratka Tx značí relativní cestu v rámci šablon Nette generátoru.
[6] Zkratka Ax značí relativní cestu v rámci výsledné aplikace vygenerované Nette generátorem.

T2	/app/**NDBTmodels**/repositories
A2	/app/models/repositories

Pokud uživatel zvolil, že chce výslednou aplikaci generovat pomocí Doctrine2 modelů, je tento podadresář včetně veškerého svého obsahu automaticky přeskočen.

Tabulka 4 Příklady zpracování adresářů NDBT při generování Doctrine2 modelů

T1	/app/models/**NDBT**/repositories
A1	Přeskočen
T2	/app/**NDBTmodels**/repositories
A2	Přeskočen

Podadresáře začínající na D2

Pokud uživatel zvolil generování aplikace pomocí Doctrine2 modelů, pak se řetězec D2 odstraní z názvu podadresáře ještě před jeho vytvořením v adresáři aplikace.

Tabulka 5 Příklady zpracování adresářů D2 při generování Nette\Database\Table modelů

T1	/app/models/**D2**/repositories
A1	/app/models/repositories
T2	/app/**D2models**/repositories
A2	/app/models/repositories

Pokud uživatel zvolil, že chce výslednou aplikaci generovat pomocí Nette\Database\Table modelů, pak je tento podadresář včetně veškerého jeho obsahu automaticky přeskočen.

Tabulka 6 Příklady zpracování adresářů D2 při generování Doctrine2 modelů

T1	/app/models/**D2**/repositories
A1	Přeskočen
T2	/app/**D2models**/repositories
A2	Přeskočen

Podadresáře začínající na Table

V tomto případě dojde k nahrazení řetězce Table reálným názvem právě zpracovávané databáze tabulky. Bude vytvořeno tolik kopií, kolik je celkem databázových tabulek.

Tabulka 7 Příklady zpracování adresářů Table pomocí vytváření jeho kopií

T1	/app/templates/**Table**
A1	/app/templates/**ExampleOne**
A1	/app/templates/**ExampleTwo**
A1	/app/templates/**ExampleThree**

3.5.2.2 Zacházení se soubory

Zjednodušeně řečeno se práce se soubory dělí na dvě základní kategorie – zpracované prostřednictvím Latte enginu a nezpracované.

Soubory s alespoň dvěma koncovkami končící na .latte

Soubory, které mají alespoň dvě koncovky a poslední z nich je *.latte,* jsou zpracovány Latte enginem. Před jejich zkopírováním do cílové destinace jim je odstraněna poslední *.latte* koncovka. Latte engine samozřejmě dovoluje i generování jiných Latte šablon. V tomto případě stačí v šabloně generátoru využít dvojitou Latte syntaxi a :n makra, která budou zpracovávána generátorem, a jednoduchou syntaxi a :l makra, která budou zpracována výslednou vygenerovanou aplikací, přičemž :l makra budou před překopírováním souborů do cílové destinace nahrazena :n makry.

Tabulka 8 Příklady zpracování souborů se dvěma koncovkami a poslední .latte

T1	/app/presenters/**BasePresenter.php.latte**
A1	/app/presenters/**BasePresenter.php**
T2	/app/models/**BaseRepository.php.latte**
A2	/app/models/**BaseRepository.php**
T3	/app/templates/**@layout.latte.latte**
A3	/app/templates/**@layout.latte**

T4	/app/config/**config.neon.latte**
A4	/app/config/**config.neon**

Soubory s jednou koncovkou a to .latte

Soubory, které mají jedinou koncovku, a to .latte jsou pouze zpracovány prostřednictvím Latte enginu a nejsou nikam překopírovány. Výchozí sada šablon využívá tuto funkcionalitu například ke generování Doctrine2 entit, protože Latte engine umožňuje spuštění jakéhokoliv PHP kódu, takže je v Latte šabloně možné vygenerovat jakékoliv jiné soubory, ale samotný generátor Doctrine2 entit ve vygenerované aplikace nemusí být umístěn.

Tabulka 9 Příklady zpracování souborů s jednou koncovkou .latte

T1	/app/models/**D2EntityBuilder.latte**
A1	Přeskočeno

Soubory začínající na Module

Pokud uživatel zvolil generování aplikace do modulu, pak se řetězec Module nahradí reálným názvem modulu zadaným uživatelem.

Tabulka 10 Příklady zpracování souborů začínajících na Module při generování do modulu

T1	/app/presenters/**ModuleBasePresenter.php.latte**
A1	/app/presenters/**ExampleModuleBasePresenter.php**

Pokud uživatel zvolil, že nechce výslednou aplikaci generovat do modulu, pak je ze souboru začínajícího na řetězec Module tato část názvu odstraněna.

Tabulka 11 Příklady zpracování souborů začínajících na Module při generování bez modulu

T1	/app/presenters/**ModuleBasePresenter.php.latte**
A1	/app/presenters/**BasePresenter.php**

Soubory začínající na NDBT

Pokud uživatel zvolil generování aplikace pomocí Nette\Database\Table modelů, pak se řetězec NDBT odstraní z názvu souboru ještě před jeho vytvořením v adresáři aplikace.

Tabulka 12 Příklady zpracování adresářů začínajících na NDBT při generování Nette\Database\Table modelů

T1	/app/models/**NDBTBaseRepository.php.latte**
A1	/app/models/**BaseRepository.php**

Pokud uživatel zvolil, že chce výslednou aplikaci generovat pomocí Doctrine2 modelů, je tento soubor automaticky přeskočen.

Tabulka 13 Příklady zpracování adresářů začínajících na NDBT při generování Doctrine2 modelů

T1	/app/models/**NDBTBaseRepository.php.latte**
A1	Přeskočen

Soubory začínající na D2

Pokud uživatel zvolil generování aplikace pomocí Doctrine2 modelů, pak se řetězec D2 odstraní z názvu souboru před jeho vytvořením v adresáři aplikace.

Tabulka 14 Příklady zpracování adresářů začínajících na D2 při generování Doctrine2 modelů

T1	/app/models/**D2BaseRepository.php.latte**
A1	/app/models/**BaseRepository.php**

Pokud uživatel zvolil, že chce výslednou aplikaci generovat pomocí Nette\Database\Table modelů, je tento soubor automaticky přeskočen.

Tabulka 15 Příklady zpracování adresářů začínajících na D2 při generování Doctrine2 modelů

T1	/app/models/**D2BaseRepository.php.latte**
A1	Přeskočen

Soubory začínající na Table

V tomto případě dojde k nahrazení řetězce Table reálným názvem právě zpracovávané databáze tabulky. Následně bude vytvořeno tolik kopií, kolik je celkem databázových tabulek.

Tabulka 16 Příklady zpracování souborů začínajících na Table pomocí vytváření jeho kopií

T1	/app/presenters/**TablePresenter.php.latte**
A1	/app/presenters/**ExampleOnePresenter.php**
A1	/app/presenters/**ExampleTwoPresenter.php**
A1	/app/presenters/**ExampleThreePresenter.php**

3.5.3 Tvorba pokročilé adresářové struktury

Všechna předchozí zmíněná pravidla pro zpracovávání adresářů a souborů je možné libovolně kombinovat za předpokladu dodržení jejich posloupnosti. První musí být vždy uvedeno klíčové slovo *Module*, poté buď *NDBT*, nebo *D2*, jako poslední *Table*. Jednoduchým kombinováním předešlých pravidel je možné dosáhnout libovolné struktury generované aplikace, autor sady šablon není prakticky ničím omezován.

Tabulka 17 Příklad pokročilé adresářové struktury a zpracovávání jednotlivých adresářů a souborů při generování bez modulu a pomocí Nette\Database\Table modelů

T1	/app/**Module**/presenters/**BasePresenter.php.latte**
A1	/app/presenters/**BasePresenter.php**
T2	/app/**Module**/presenters/**TablePresenter.php.latte**
A2	/app/presenters/**ExampleOnePresenter.php**
A2	/app/presenters/**ExampleTwoPresenter.php**
A2	/app/presenters/**ExampleThreePresenter.php**
T3	/app/**Module**/templates/**@layout.lattel.latte**
A3	/app/templates/**@layout.latte**
T4	/app/**Module**/templates/**Table/view.latte.latte**
A4	/app/templates/**ExampleOne/view.latte**
A4	/app/templates/**ExampleTwo/view.latte**
A4	/app/templates/**ExampleThree/view.latte**

T5	/app/**Module**/models/**NDBTBaseRepository.php.latte**
A5	/app/models/**BaseRepository.php**
T6	/app/**Module**/models/**NDBTTableRepository.php.latte**
A6	/app/models/**ExampleOneRepository.php**
A6	/app/models/**ExampleTwoRepository.php**
A6	/app/models/**ExampleThreeRepository.php**
T7	/app/**Module**/models/**D2BaseRepository.php.latte**
A7	Přeskočen
T8	/app/**Module**/models/**D2TableRepository.php.latte**
A8	Přeskočen

3.5.4 Parametry předávané šablonám

Všechny šablony zmíněné v kapitolách výše automaticky dostávají identickou sadu parametrů obsahující zejména informace o databázové struktuře a nastavení generátoru provedených uživatelem.

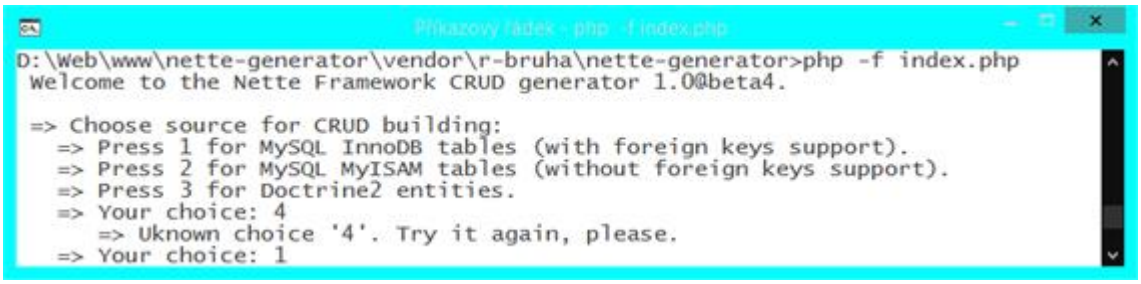
V proměnné *netteRoot* je k dispozici absolutní cesta ke kořenovému adresáři nově generované aplikace. Pole *netteConfig* obsahuje dekódovaný hlavní konfigurační soubor Nette Frameworku – *config.neon*. Objekt *netteDatabase* obsahuje informace o databázovém připojení – adresu databázového serveru, název databáze a přihlašovací jméno s heslem. Následuje konstanta *source* určující zdroj dat, ze kterého probíhá generování, konstanta *target* určující typ modelů, které bude vygenerovaná aplikace využívat a konstanta *foreignKeys* určující způsob práce s cizími klíči. Dále se předává parametr *module* obsahující název modulu, do kterého se bude aplikace generovat a *template*, který obsahuje absolutní cestu ke zvolené sadě šablon. Poslední dva předávané parametry se týkají samotné databáze – pole objektů typu *Bruha\Generator\Utils\Object\Table* – *tables*, které obsahuje veškeré dostupné informace o databázových tabulkách a podílí se tak největší měrou na generování výsledné aplikace, a objekt stejného typu pojmenovaný *table*, který obsahuje aktuálně zpracovávanou tabulku v případě šablon, které jsou duplikovány pro každou databázovou tabulku zvlášť.

3.6 Návod na použití generátoru

Následující postup na použití Nette generátoru využívá nástroj pro správu závislostí PHP knihoven, *Composer*. Jeho instalace a základy používání však nejsou součástí tohoto textu. Pokud jej instalujete a využíváte poprvé, můžete použít například tento návod. [41]

Jako první krok musíme vytvořit nový projekt založený na Nette Frameworku za použití nástroje *Composer*. Proto tedy otevřeme příkazový řádek a přesuneme se do adresáře, kde chceme mít projekt umístěn. Poté spustíme následující příkaz: *composer create-project nette/sandbox my-project 2.2*, který nám zajistí vytvoření adresáře s projektem a automatické stažení Nette Frameworku a dalších potřebných knihoven. Poté se přesuneme do adresáře nově vytvořeného projektu a nainstalujeme Nette generátor příkazem *composer require r-bruha/nette-generator @dev*, který si automaticky stáhne většinu dalších knihoven, které potřebuje pro svou funkci. Jako poslední věc nainstalujeme Kdyby\Replicator pomocí příkazu *composer require kdyby/forms-replicator* 1.2.*@dev. Nyní je instalace Nette generátoru dokončena, můžeme přejít k jeho spuštění pomocí příkazového řádku. Přesuneme se do složky *vendor/r-bruha/nette-generator* a pomocí příkazu *php -f index.php* generátor spustíme.

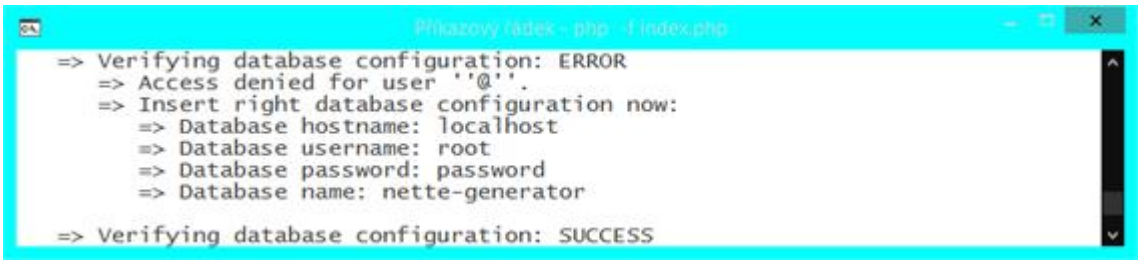
V případě správného nainstalování všech závislostí se po zadání předešlého příkazu objeví uvítací zpráva Nette generátoru a první dotaz na výběr zdrojových dat.



Obrázek 4 Výběr zdrojových dat pro Nette generátor

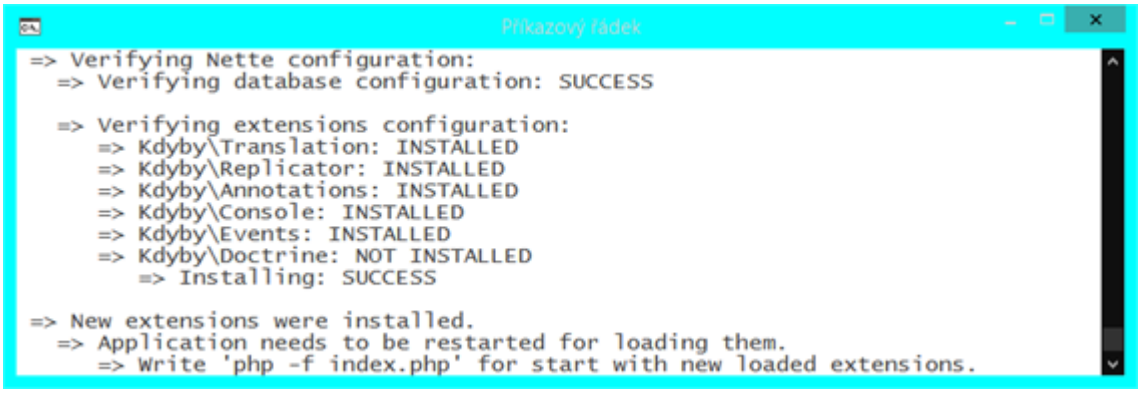
Výběr zdrojových dat provedeme napsáním čísla zvolených zdrojových dat a stisknutím klávesy Enter. V případě zadání nevalidní hodnoty budeme znovu dotázáni na vložení správné hodnoty. Následuje ověření funkčního databázového připojení. V případě, že databázové připojení není nakonfigurováno správně, spustí

se průvodce jeho vytvoření, kde postupně zadáme hostitele databáze, přihlašovací jméno a heslo a název databáze, se kterou se budeme pracovat. V případě dobře nakonfigurované databáze již před spuštěním generátoru je tento krok přeskočen.



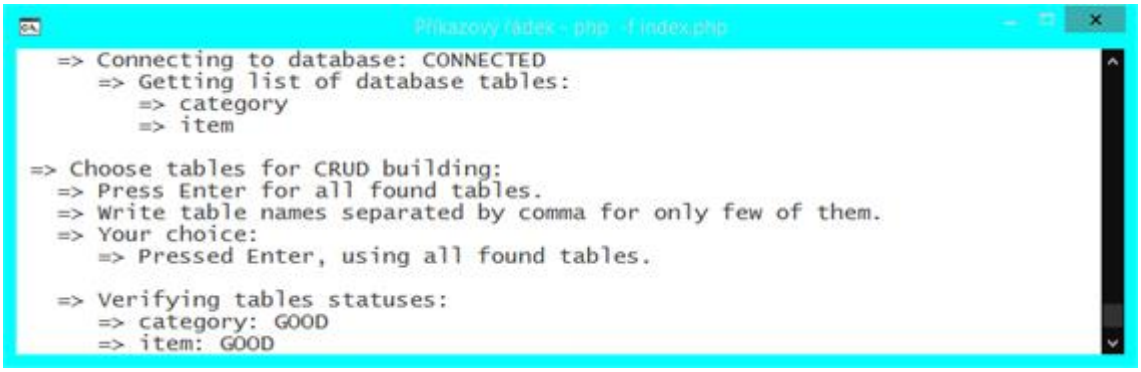
Obrázek 5 Konfigurace databázového připojení v případě jeho nefunkčnosti

Následuje ověření správně nakonfigurovaných služeb pro Nette Framework. Pokud nebudou požadované služby korektně zaregistrované, tak budou zaregistrovány. Bohužel po zaregistrování služeb je vyžadováno znovuspuštění Nette generátoru, na což budete upozorněni. V případě správně nainstalovaných služeb není nutné nic řešit.



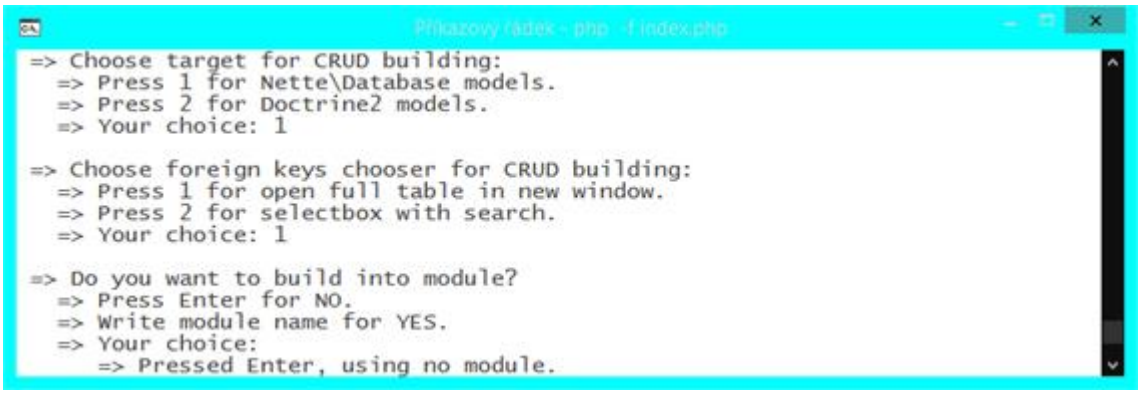
Obrázek 6 Kontrola správné konfigurace rozšíření a jejich případná automatická instalace

V dalším kroku proběhne připojení k databázi a zjištění všech dostupných tabulek. Následuje dotaz: „Které ze zjištěných tabulek se mají použít pro generování výsledné aplikace?“ V případě stisknutí klávesy Enter se použijí všechny. Prostým vyjmenováním jednotlivých tabulek oddělenými čárkou lze vybrat pouze nějaké. Jako poslední část tohoto kroku probíhá testování vybraných databázových tabulek, zda obsahují primární klíč. Pokud klíč neobsahují, nemohou být použity pro generování aplikace a budou proto automaticky přeskočeny.



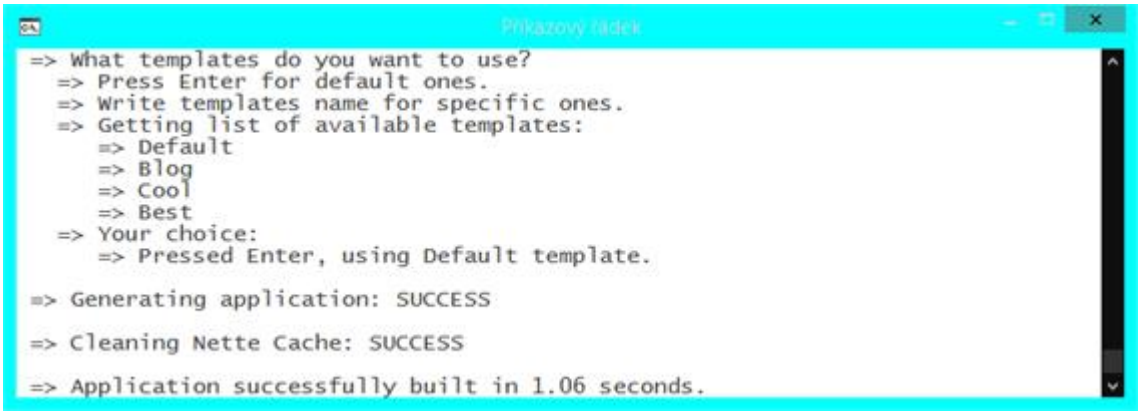
Obrázek 7 Výpis dostupných databázových tabulek spolu s výběrem těch, které budou použity

Následuje trojice dotazů. První z nich se ptá na typ modelů, které pracují s databází. Na výběr jsou Nette\Database\Table modely nebo modely za použití ORM Doctrine2. Druhý se ptá, jak bude implementován výběr dat pro sloupce, které odkazují prostřednictvím cizích klíčů na jinou tabulku. Na výběr je buď otevření dané tabulky v novém okně, nebo pomocí seletboxů s vyhledáváním. Třetí pak nabízí možnost generování aplikace do modulu. V případě, že chceme generovat aplikaci bez modulu, stačí stisknout klávesu Enter.



Obrázek 8 Výběr typu generovaných modelů, způsobu práce s cizími klíči a modulu

Poslední dotaz se týká sady šablon použité pro generování aplikace. Samotný generátor obsahuje pouze jednu výchozí sadu šablon nazvanou *Default*, ale je možné vytvořit si i vlastní sady šablon a přepínat mezi nimi. Generátor nejdříve zjistí veškeré dostupné sady šablon, které vypíše, a poté zadáním názvu vybrané šablony zvolenou šablonu vybereme. Pro výběr výchozí sady šablon stačí stisknout klávesu Enter.



Obrázek 9 Výběr šablon a úspěšné vygenerování aplikace

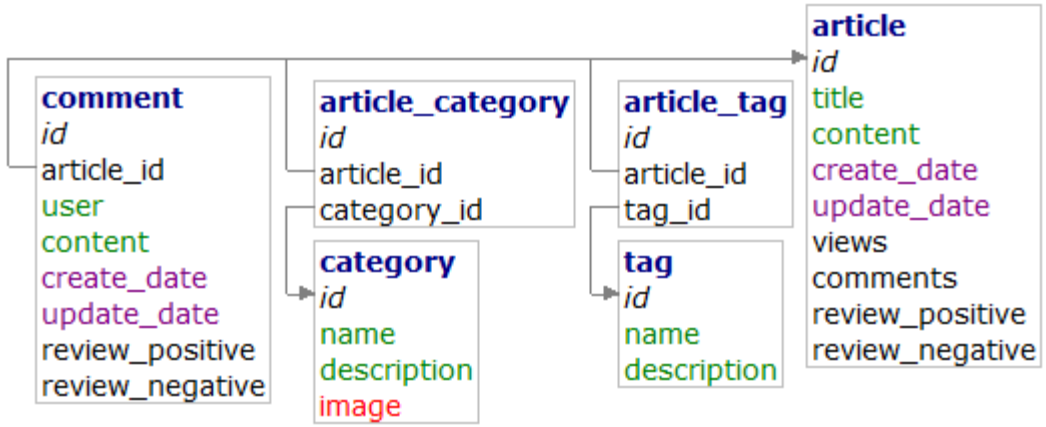
Poté již následuje samotné generování aplikace a vymazání cache Nette Frameworku pro zajištění správného načtení všech nově vygenerovaných součástí aplikace a ukončení samotného generátoru.

4 Shrnutí výsledků

Za účelem testování správné funkčnosti Nette generátoru (tzn. vygenerování aplikace neobsahující žádné závažné chyby) bylo vytvořeno jednoduché schéma v MySQL databázi sloužící jako příklad jednoduchého blogu. Schéma bylo použito i pro testování funkčnosti definic přístupových práv jednotlivým uživatelům na úrovni databázových tabulek a jejich sloupců.

Jednoduchý blog umožňuje publikovat příspěvky, které mohou být kladně nebo záporně hodnoceny. Každý příspěvek je možné zařadit do jedné nebo více kategorií a opatřit jej dalšími tagy pro snadné vyhledávání. Poslední možností je jednoduchá diskuze pod každým příspěvkem, kdy každý komentář může být taktéž kladně nebo záporně ohodnocen dalšími návštěvníky blogu.

Celé databázové schéma je jednak provázané referenční integritou pomocí cizích klíčů, ale zároveň dodržuje i konvenční pojmenování pro možnost přímého testování obou módů generování aplikace – pomocí referenční integrity nebo konvenčního pojmenování.

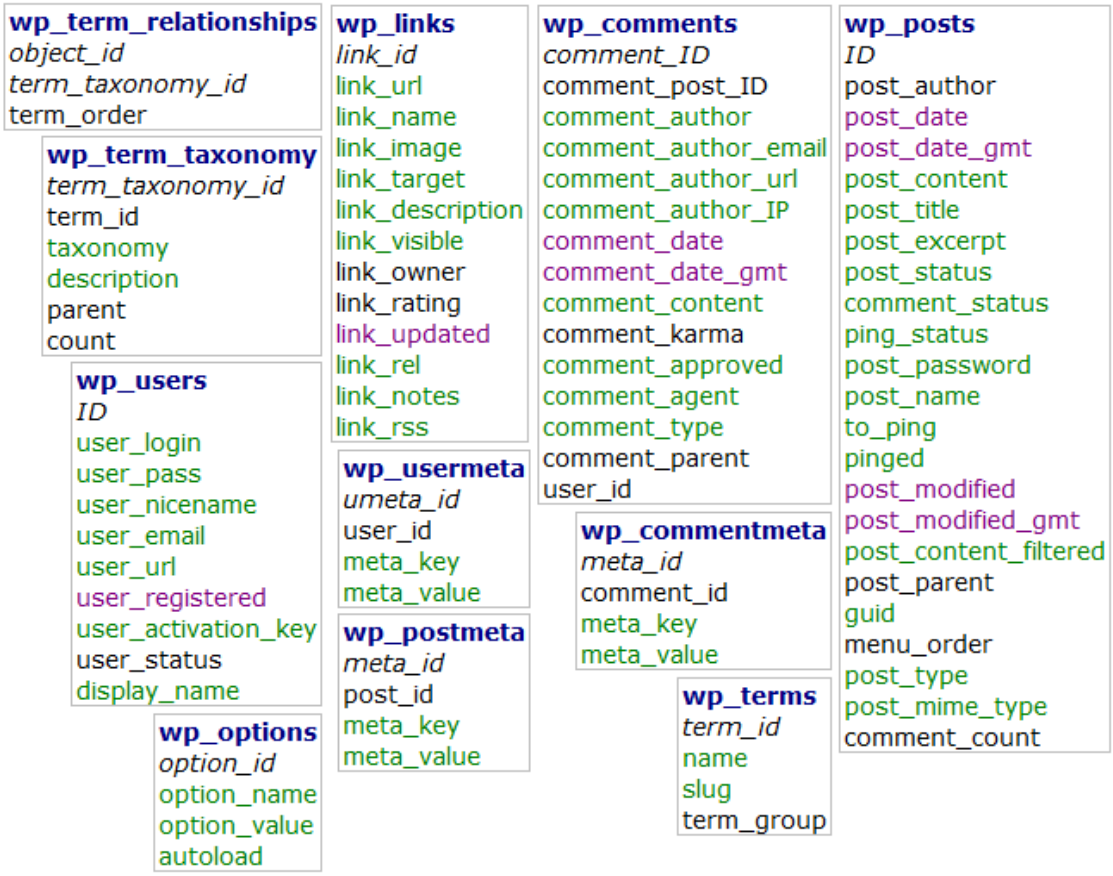


Obrázek 10 Testovací schéma MySQL databáze pro jednoduchý blog

Nette generátor zvládne pro toto schéma vygenerovat bezchybně fungující a jednoduše ovladatelnou aplikaci vhodnou pro administraci tohoto blogovacího systému. Nicméně jsou zde i jistá omezení, například přiřazování kategorií a tagů jednotlivým článkům je realizováno pomocí formuláře umístěného zvlášť (odpovídajícího tabulkám *article_category* a *article_tag*), což znamená, že nejdříve je nutné vytvořit článek, a pak mu přiřadit kategorie a tagy. Vhodnějším řešením

by byla možnost přímo při vytváření článku nabídnout selectboxy pro zvolení jeho kategorií a článků. Nicméně detekování podobných situací není úplně jednoduché a mohlo by docházet k chybám, proto není zahrnuto ve výchozí sadě šablon. Nicméně řešení situace je poměrně jednoduché – po vygenerování aplikace lze tuto funkcionalitu jednoduše dopsat do vygenerovaných souborů, konkrétně do souboru *ArticlePresenter.php*, který obsahuje definici formuláře pro přidávání a úpravu příspěvku. Následující úprava by neměla pro člověka, alespoň trochu znalého práce s Nette Frameworkem, představovat žádný problém, protože stačí na jeho konec přidat dva nové formulářové prvky, konkrétně selectboxy. První z nich bude jako možnosti obsahovat všechny dostupné kategorie a druhý tagy. Šablona se pak již sama postará o jejich správné vykreslení na stránce.

Doplňkové testování Nette generátororu pak probíhalo na databázových schématech používaných oblíbenými redakčními systémy – Wordpress a Joomla.



Obrázek 11 Databázové schéma používané redakčním systémem Wordpress

Databázová schémata obou redakčních systémů se vyznačují především tím, že jejich tabulky nejsou provázány pomocí cizích klíčů. Bohužel také nedodržují konvenční pojmenování sloupců definované Nette Frameworkem. Dále některé z jejich tabulek neobsahují definici primárního klíče, čímž se daná tabulka stává pro Nette generátor automaticky nezpracovatelnou. Absence cizích klíčů pak znamená nemožnost interaktivního provázání tabulek při přidávání a úpravě jednotlivých údajů. Nicméně je potřeba mít na paměti, že toto není chyba samotného Nette generátoru, protože bez konvenčního pojmenování, či ještě lépe referenční integrity pomocí cizích klíčů, není možné interaktivní provázání tabulek identifikovat.

Ve všech třech testovacích případech splnil Nette generátor očekávání a vygeneroval funkční aplikaci, jejíž funkcionalita se liší zejména na základě existence či neexistence referenční integrity a primárních klíčů v jednotlivých databázových tabulkách.

5 Závěry a doporučení

Řádkový CRUD generátor pro PHP framework Nette, kterému předcházela analýza řešení použitých u konkurenčních PHP frameworků CakePHP a Symfony, splnil cíle vytyčené v úvodu práce. Výsledný generátor pro Nette Framework využil dobře zpracovaných řešení určitých úloh u konkurenčních frameworků a zároveň odstranil většinu jejich nedostatků z hlediska jejich okamžité použitelnosti.

Výsledkem práce je generátor CRUD aplikací pro Nette Framework založený na konkrétních tabulkách v prostředí relační databáze. Výstupem generátoru je dobře navržená aplikace dodržující základní standardy a doporučené postupy práce s Nette Frameworkem. Výsledný kód zároveň nevyužívá veškerý potenciál Nette Frameworku zejména kvůli složitosti, která nebyla žádoucí z důvodu plánovaného využití generátoru i uživateli, kteří by výslednou aplikaci mohli využít pro pochopení práce s Nette Frameworkem, nebo jako inspiraci pro svůj další rozvoj.

Vygenerovaná aplikace obsahuje základní definici přístupových oprávnění na úrovni jednotlivých záznamů. Samotný proces přihlašování a přidělování oprávnění není její součástí, protože tato část je velmi subjektivní a těžko obecně zpracovatelná. Součástí výsledné aplikace je také moderně vypadající responzivní grafický design za použití Twitter Bootstrap CSS frameworku, který zajišťuje, že je aplikace ihned po vygenerování připravena pro produkční nasazení.

Samotný generátor využívá plného potenciálu šablon a předpokládá spolupráci komunity při tvorbě vlastních sad šablon či úpravě výchozí sady šablon pro generování aplikace. V případě dostatečného zájmu komunity je počítáno s možností vytváření různých šablon i pro velmi specifické případy použití, které nebyly při tvorbě generátoru brány v potaz, bez jakéhokoliv zásahu do samotného generátoru díky důrazu na vysokou možnost upravitelnosti jednotlivých šablon, který dal vzniku uživatelsky přívětivému a zároveň velice robustnímu API pro tvorbu vlastních šablon.

6 Seznam použité literatury

[1] MALÝ, Martin. David Grudl: Marketing Nette dělají spokojení uživatelé. *Zdroják* [online]. 26. 5.2011 [cit. 2014-11-08]. Dostupné z: http://www.zdrojak.cz/clanky/david-grudl-marketing-nette-delaji-spokojeni-uzivatele

[2] KNYTTL. *Vytvoření „Nette administrace" - Nette Framework forum* [online]. 2011 [cit. 2015-03-07]. Dostupné z: http://forum.nette.org/cs/7392-vytvoreni-nette-administrace

[3] JAAVEHI. NetBeans utilita – Tvorba formulářů z DB. *Nette Framework forum* [online]. 7.10.2011 [cit. 2014-11-08]. Dostupné z: http://forum.nette.org/cs/5079-netbeans-utilita-tvorba-formularu-z-db

[4] DÍTĚ, Mikuláš. Nette Scaffold – generování modelu s CRUD a náležitými presentery. *Nette Framework forum* [online]. 18.11.2011 [cit. 2014-11-08]. Dostupné z: http://forum.nette.org/cs/6430-nette-scaffold-generovani-modelu-s-crud-a-nalezitymi-presentery

[5] DÍTĚ, Mikuláš. Nette Scaffold. In: *Screencast* [online]. Zveřejněno 18. 11. 2011 [vid. 8. 11. 2014]. Dostupné z: http://www.screencast.com/t/txWYhA7Q3JQ

[6] SMITKA, Jan. [„Základ poloautomatické administrace konečně realizován. Z tohoto: https://gist.github.com/0567712c954b130fc376 … se stane http://ukaz.at/1mm http://ukaz.at/1mn"]. In: *Twitter* [online]. 26. 7. 2011, 5:52[vid. 8. 11. 2014]. Dostupné z: https://twitter.com/jansmitka/status/95838788139491328

[7] SMITKA, Jan. [„After 6 months of painful experiments, the traits for admin system gen. are now scrapped off. That means, we will be compatible with 5.3."]. In: *Twitter* [online]. 27. 12. 2011, 15:41 [vid. 8. 11. 2014]. Dostupné z: https://twitter.com/LightbulbTM/status/151810057544273921

[8] SKVORC, Bruno. Best PHP Frameworks for 2014. *SitePoint – Learn HTML, CSS, JavaScript, PHP, Ruby & Responsive Design* [online]. 28. 11. 2013 [cit. 2014-11-08]. Dostupné z: http://www.sitepoint.com/best-php-frameworks-2014/

[9] GRUDL, David. Zabezpečení před zranitelnostmi. *Nette Framework* [online]. 1. 8. 2013, 6. 8. 2014 [cit. 2014-11-08]. Dostupné z: http://doc.nette.org/cs/2.2/vulnerability-protection

[10] Scaffolding - CakePHP Cookbook 2.x documentation. *CakePHP Cookbook 2.x documentation* [online]. [cit. 2014-11-08]. Dostupné z: http://book.cakephp.org/2.0/en/controllers/scaffolding.html

[11] Code Generation with Bake - CakePHP Cookbook 2.x documentation. *CakePHP Cookbook 2.x documentation* [online]. [cit. 2014-11-08]. Dostupné z: http://book.cakephp.org/2.0/en/console-and-shells/code-generation-with-bake.html

[12] SensioGeneratorBundle - Symfony.com. *SensioGeneratorBundle - Symfony.com* [online]. [cit. 2014-11-08]. Dostupné z: http://symfony.com/doc/current/bundles/SensioGeneratorBundle/index.html

[13] RAMIREZ, Gregorio. How to Generate an Entity in Symfony2. In: *Youtube* [online]. Zveřejněno 17. 1. 2012 [vid. 11. 8. 2014]. Dostupné z: https://www.youtube.com/watch?v=0SE09AlJSEI

[14] RAMIREZ, Gregorio. How to Generate CRUD for an Entity in Symfony2. In: *Youtube* [online]. Zveřejněno 18. 1. 2012 [vid. 11. 8. 2014]. Dostupné z: https://www.youtube.com/watch?v=Jqp781knyPM

[15] Object Relational Mapper. *Doctrine Project* [online]. 2014 [cit. 2015-03-22]. Dostupné z: http://www.doctrine-project.org/projects/orm.html

[16] Database Abstraction Layer. *Doctrine Project* [online]. 2014 [cit. 2015-03-22]. Dostupné z: http://www.doctrine-project.org/projects/dbal.html

[17] Getting Started with Doctrine. *Doctrine 2 ORM 2 documentation* [online]. 2014 [cit. 2015-03-22]. Dostupné z: http://doctrine-orm.readthedocs.org/en/latest/tutorials/getting-started.html

[18] 3. Configuration. *Doctrine DBAL 2.1.0 documentation* [online]. 2014 [cit. 2015-03-22]. Dostupné z: http://docs.doctrine-project.org/projects/doctrine-dbal/en/latest/reference/configuration.html

[19] 5. SQL Query Builder. *Doctrine DBAL 2.1.0 documentation* [online]. 2014 [cit. 2015-03-22]. Dostupné z: http://docs.doctrine-project.org/projects/doctrine-dbal/en/latest/reference/query-builder.html

[20] 4. Basic Mapping. *Doctrine 2 ORM 2 documentation* [online]. 2014 [cit. 2015-03-22]. Dostupné z: http://docs.doctrine-project.org/projects/doctrine-orm/en/latest/reference/basic-mapping.html

[21] 5. Association Mapping. *Doctrine 2 ORM 2 documentation* [online]. 2014 [cit. 2015-03-22]. Dostupné z: http://docs.doctrine-project.org/projects/doctrine-orm/en/latest/reference/association-mapping.html

[22] 14. Doctrine Query Language. *Doctrine 2 ORM 2 documentation* [online]. 2014 [cit. 2015-03-22]. Dostupné z: http://docs.doctrine-project.org/projects/doctrine-orm/en/latest/reference/dql-doctrine-query-language.html

[23] FOWLER, Martin. *Patterns of enterprise application architecture*. Boston: Addison-Wesley, c2003, xxiv, 533 s. The Addison-Wesley Signature Series. ISBN 978-0-321-12742-6.

[24] Database\Table. *Nette Framework* [online]. 2015 [cit. 2015-03-22]. Dostupné z: http://doc.nette.org/en/2.3/database-table

[25] Replicator/index.md at master - Kdyby/Replicator. *GitHub.com* [online]. 2015 [cit. 2015-03-22]. Dostupné z: https://github.com/Kdyby/Replicator/blob/master/docs/en/index.md

[26] Translation/index.md at master - Kdyby/Translation. *GitHub.com* [online]. 2015 [cit. 2015-03-22]. Dostupné z: https://github.com/Kdyby/Translation/blob/master/docs/en/index.md

[27] Translations. *The Symfony Book* [online]. 2015 [cit. 2015-03-22]. Dostupné z: http://symfony.com/doc/current/book/translation.html

[28] Secured-links/README.md at master - nextras/secured-links. *GitHub.com* [online]. 2015 [cit. 2015-03-22]. Dostupné z: https://github.com/nextras/secured-links/blob/master/README.md

[29] 26. Tools. *Doctrine 2 ORM 2 documentation* [online]. 1. 8. 2013, 6. 8. 2014 [cit. 2014-11-08]. Dostupné z: http://doctrine-orm.readthedocs.org/en/latest/reference/tools.html#database-schema-generation

[30] PHP: Introduction - Manual. *PHP.net* [online]. 2010 [cit. 2015-01-26]. Dostupné z: http://php.net/manual/en/intro.pdo.php

[31] MySQL 5.6 Reference Manual :: 11.2.5 Numeric Type Attributes. *MySQL.com* [online]. 2010 [cit. 2015-01-26]. Dostupné z: http://dev.mysql.com/doc/refman/5.6/en/numeric-type-attributes.html

[32] MySQL 5.6 Reference Manual :: 13.1.17 CREATE TABLE Syntax. *MySQL.com* [online]. 2010 [cit. 2015-01-26]. Dostupné z: http://dev.mysql.com/doc/refman/5.6/en/create-table.html

[33] MySQL 5.6 Reference Manual :: 14.6.6 InnoDB and FOREIGN KEY Constraints. *MySQL.com* [online]. 2010 [cit. 2015-01-26]. Dostupné z: http://dev.mysql.com/doc/refman/5.6/en/innodb-foreign-key-constraints.html

[34] MySQL 3.23, 4.0, 4.1 Reference Manual :: 1.9.5.6 Foreign Keys. *MySQL.com* [online]. 2010 [cit. 2015-01-26]. Dostupné z: http://dev.mysql.com/doc/refman/4.1/en/ansi-diff-foreign-keys.html

[35] MySQL 5.6 Reference Manual :: 15.2 The MyISAM Storage Engine. *MySQL.com* [online]. 2010 [cit. 2015-01-26]. Dostupné z: http://dev.mysql.com/doc/refman/5.6/en/myisam-storage-engine.html

[36] File Database/Reflection/ConventionalReflection.php. *Nette 2.2.7 API* [online]. 2014 [cit. 2015-01-26]. Dostupné z: http://api.nette.org/2.2.7/source-Database.Reflection.ConventionalReflection.php.html#31-42

[37] Kdyby/Translation. *Github.com* [online]. 2014 [cit. 2015-01-26]. Dostupné z: https://github.com/Kdyby/Translation

[38] Začínáme | *Nette Framework* [online]. 2015 [cit. 2015-01-26]. Dostupné z: http://doc.nette.org/cs/2.2/quickstart/getting-started#toc-obsah-sandboxu

[39] Kdyby/Replicator. *Github.com* [online]. 2014 [cit. 2015-01-26]. Dostupné z: https://github.com/Kdyby/Replicator

[40] The world's most popular mobile-first and responsive front-end framework. *Bootstrap.com* [online]. 2015 [cit. 2015-01-26]. Dostupné z: http://getbootstrap.com/

[41] Composer. *Nette Framework* [online]. 2015 [cit. 2015-03-08]. Dostupné z: http://doc.nette.org/cs/2.3/composer

7 Přílohy

1) CD nosič se zdrojovými kódy Nette generátoru
    a. */bin* – spouštěcí soubory pro Windows a Linux
    b. */src* – zdrojové kódy
    c. */How to create templates.md* – návod na tvorbu vlastních sad šablon
    d. */README.md* – stručný popis projektu
    e. */composer.json* – nastavení závislostí pro nástroj Composer
    f. */index.php* – hlavní spouštěcí soubor
2) Veřejný GitHub.com repositář s poslední dostupnou verzí Nette generátoru: https://github.com/r-bruha/nette-generator

FIM UHK
UNIVERZITA HRADEC KRÁLOVÉ
Fakulta informatiky a managementu
Rokitanského 62, 500 03 Hradec Králové, tel: 493 331 111, fax: 493 332 235

Zadání k závěrečné práci

Jméno a příjmení studenta: **Radek Brůha**

Obor studia: Aplikovaná informatika
Jméno a příjmení vedoucího práce: **Pavel Kříž**

Název práce:
Generátor kódu pro webové aplikace v Nette

Název práce v AJ:
Code-generator for Nette Web Applications

Podtitul práce:

Podtitul práce v AJ:

Cíl práce: Navrhnout a implementovat generátor kódu pro urychlení vývoje webových aplikací ve frameworku Nette vycházejících z existujícího databázového schématu.

Osnova práce:
1. Úvod
2. Existující řešení
3. Návrh a implementace vlastního řešení
4. Výsledky testování na základních příkladech užití
5. Závěr

Projednáno dne: 14.10.14

Podpis studenta Brůha

Podpis vedoucího práce