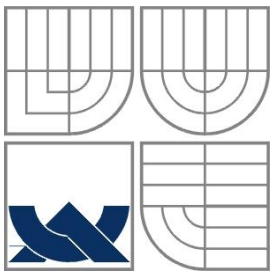
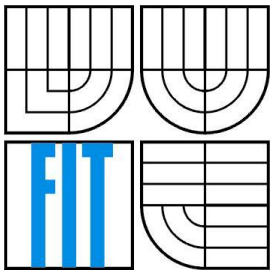


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

GENERÁTOR INFORMAČNÍHO SYSTÉMU

INFORMATION SYSTEM GENERATOR

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — JIŘÍ DAJČ
AUTHOR

VEDOUCÍ PRÁCE — Ing. JIŘÍ ŠEVCOVIC
SUPERVISOR

BRNO 2011

Abstrakt

Role generátorů zdrojového kódu se stává stále významnější. Ať jde o generování tříd na základě objektového modelu v jazyce UML nebo o vizuální tvorbu uživatelských rozhraní. Tato práce se zabývá generováním zdrojového kódu pro základní operace nad jednou tabulkou. Předkládá informace o technologiích pro implementaci informačních systémů a zaměřuje se na PHP MVC frameworky. Zabývá se generováním zdrojového kódu obecně a popisuje návrh generátoru kódu základních operací nad jednou tabulkou a jeho implementaci v prostředí Qt Creator.

Abstract

The role of the source code generators is becoming increasingly important. Generation of classes based on object model in UML or visual creation of user interfaces is today commonly used. This work deals with the generation of source code for basic operations on one table. It presents information about technologies for the implementation of information systems, focusing on PHP MVC frameworks. It deals with the source code generation in general and describes the design of the generator and its implementation in Qt Creator.

Klíčová slova

Informační systém, generování zdrojového kódu, MVC, PHP frameworky, Kohana

Keywords

Information system, source code generation, MVC, PHP frameworks, Kohana

Citace

Dajč Jiří: Generátor informačního systému, bakalářská práce, Brno, FIT VUT v Brně, 2011

Generátor informačního systému

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Jiřího Ševcovice. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Jiří Dajč
15. Května 2011

Poděkování

Chci poděkovat svému vedoucímu Ing. Jiřímu Ševcovicovi za mnoho užitečných rad a připomínek a také za možnost zpracovat tak zajímavé téma bakalářské práce.

Obsah

1 Úvod

V dnešní době téměř každá malá firma používá informační systém nebo o jeho nasazení alespoň uvažuje. Přestože se objevují krabicové verze ekonomických systémů a různých systémů pro řízení podniku, poptávka po informačních systémech na zakázku je stále poměrně velká. Je to především díky tomu, že náklady na vývoj IS se snižují a spolu s nimi se snižuje i jejich pořizovací cena a dostupnost. Pokud chceme být konkurenceschopní a tedy úspěšní na poli vývoje těchto systémů, musíme náklady na vývoj udržovat co nejnižší.

Vzhledem k tomu, že požadavky na zakázkový IS se mohou značně lišit, jsme často nuceni nový systém implementovat téměř od začátku. A právě na začátku implementace IS se skýtá možnost ušetřit náklady v podobě času. V této fázi většinou vytváříme aplikační rozhraní pro přístup k databázovým tabulkám a pro správu obsahu vybraných tabulek vytváříme základní uživatelské rozhraní. Zdrojový kód této základní části systému vychází pouze ze struktury databáze a na základě znalosti této struktury je ho tedy možné vygenerovat. S tímto přístupem nejen ušetříme čas, ale často také předejdeme chybám, které vznikají při kopírování a úpravách kódu již vytvořeného v rámci minulých projektů.

V této práci jsou představeny motivy pro vytvoření generátoru zdrojového kódu základních operací nad jednou tabulkou. Jeho výstupem by měl být kód v jazyce PHP pro framework Kohana 3.1. Práce se snaží přiblížit pojmem „informační systém“, objasňuje přirozené příčiny, které vedly k rychlému rozvoji těchto systémů, a předkládá kritéria pro jejich klasifikaci. Představuje klient-server architekturu portálových informačních systémů a prostředky pro jejich realizaci. Další kapitola se věnuje PHP MVC frameworkům. V té je představena architektura Model-view-controller (MVC) spolu s několika rozšířenými frameworky. Závěr kapitoly se blíže věnuje frameworkům Kohana a CakePHP, čímž nabízí možnost jejich srovnání. Poté se zaměřuji na problematiku generování zdrojového kódu.

Cílem této práce je navrhnout a implementovat generátor zdrojového kódu informačního systému. Tím se zabývají poslední dvě kapitoly práce, ve kterých se věnuji výběru vhodného PHP frameworku a databázového systému, návrhu uživatelského rozhraní generátoru a návrhu struktury generovaného kódu. V závěru stručně shrnuji dosažené výsledky a předkládám návrhy na další vylepšení generátoru.

2 Informační systém

S postupným rozvojem naší společnosti stále narůstá potřeba zpracovávat a uchovávat nejrůznější druhy informací. Dříve k tomuto účelu sloužily často velmi rozsáhlé kartotéky zaměstnanců, pacientů, součástek, produktů a tak dále. Z dnešního pohledu můžeme jasně konstatovat, že tento způsob uchovávání informací má několik zásadních nevýhod. Za nejvýraznější považuji tyto:

- Velká prostorová náročnost (kartotéka může zabrat celé místnosti),
- špatná dostupnost informací (karty lze prohlížet pouze přímo v kartotéce),
- velmi neefektivní vyhledávání.

S rozvojem počítačů se naskytla možnost tyto kartotéky nahradit počítačovým systémem, který dnes označujeme jako „informační systém“ (přestože se o informační systém jednalo i bez použití počítačů, dnes tímto pojmem rozumíme především systémy využívající moderních informačních technologií). Informace, které se dříve uchovávaly v papírové podobě, se převedly do podoby digitální a začaly se ukládat na pevný disk počítače. Tento způsob práce s informacemi odstranil výše zmíněné vlastnosti, které dnes vnímáme jako nevýhodné. S rozvojem internetu se informační systémy dostaly do stavu, kdy je možné, aby člověk na jedné straně planety zadával do systému nová data a na druhé straně planety s nimi jiný člověk ihned pracoval. To bylo dříve nemyslitelné.

Výhody tohoto přístupu jsou dalekosáhlé a vysoká dostupnost nejrůznějších informací vedla k zrychlování rozvoje společnosti ve všech oblastech. V dnešní době existuje obrovské množství různých informačních systémů, které můžeme dělit podle mnoha kritérií do mnoha různých skupit. Pro představu uvádím vybraná kritéria a skupiny podle [1].

Členění podle informačního prostředí:

- Geografie a zeměměřičství,
- knihovna,
- účetnictví,
- banka, pokladna a platby,
- mzdy a správa lidských zdrojů,
- majetek a odpisy,
- pacienti a styk se zdravotní pojišťovnou.

Členění podle organizační úrovně řízení:

- Informační systém centra koncernu,
- podniku,
- soukromé firmy do 20 zaměstnanců apod.

Členění podle typu zpracování:

- *Zpracovávající převážně informace a znalosti* – faktografické,
- *zpracovávající převážně procesy* – měřicí, regulační.

S postupným rozvojem internetu (narůstající dostupností a rychlostí připojení) se značně rozšířila portálová řešení IS. Portálové řešení je postaveno na klient-server architektuře a využívá v zásadě stejných technologií jako běžné internetové stránky. Klientem se tedy rozumí běžný internetový

prohlížeč a rozhraní pro práci s informačním systémem je definováno v jazyce HTML (HyperText Markup Language) [2] s použitím příbuzných technologií. Serverem rozumíme aplikaci spuštěnou na vzdáleném počítači (běžném webovém serveru), se kterým klient komunikuje nejčastěji prostřednictvím aplikačního protokolu HTTP (HyperText Transfer Protocol) [3].

2.1 Protokol HTTP

HTTP je textový protokol (klient a server komunikují prostřednictvím textových příkazů) pro výměnu hypertextových dokumentů ve formátu HTML. Komunikace probíhá způsobem dotaz-odpověď a protokol je bezstavový – klient zažádá o určitý dokument a server mu dokument poskytne. Tím komunikace končí. Z hlediska informačních systémů je to zásadní nedostatek – protokol neumožňuje rozlišit kontext požadavků od jednotlivých klientů. Tato nepříjemná vlastnost je dána tím, že protokol nebyl z počátku určen pro realizaci informačních systémů.

Pro řešení problému bezstavovosti lze ukládat na straně klienta krátké textové soubory, které mohou nést informace o stavu autentizace uživatele (nebo libovolné jiné informace). Tyto soubory se nazývají cookies. Při každé další návštěvě daného serveru jsou všechny cookies daného serveru klientem zaslány spolu s požadavkem na html dokument. Cookies však přináší řadu bezpečnostních rizik. Jedná se o běžné textové soubory, které mohou být snadno zfalšovány nebo ukradeny.

Na straně serveru je možné pro uchování kontextu využít tzv. session. Jedná se většinou o krátké informace o uživatelích, kteří k serveru přistupují, ale může sloužit i k ukládání jiných informací, spojených s aktuálním klientem. Identifikace session patřící danému uživateli probíhá prostřednictvím informace předávané v URL nebo v cookie. Tato varianta je bezpečnější než pouhé použití cookies, protože data jsou uložena na serveru a není je tedy třeba při každém požadavku přenášet. I tato varianta má řadu bezpečnostních rizik, kretá však lze poměrně snadno minimalizovat.

2.2 Serverová část IS

Před implementací serverové části IS se musíme zabývat volbou vhodného programovacího jazyka a volbou databázového systému. Za nejrozšířenější technologie pro implementaci serverové části IS jsou dnes považovány ASP (Active Server Pages) [5] od společnosti Microsoft a skriptovací jazyk PHP s otevřeným zdrojovým kódem [4].

Následující podkapitola blíže představí jazyk PHP a databázovým systémům se věnuje podkapitola další.

2.2.1 Jazyk PHP

Zkratku PHP dnes chápeme jako rekurzivní zkratku ze slov „PHP: Hypertext Preprocessor“, původní význam zkratky však byl „Personal Home Page“. PHP je interpretovaný skriptovací jazyk určený především ke tvorbě dynamických internetových stránek. Lendorf Rasmus na něm začal pracovat v roce 1994, jako na nástroji pro údržbu osobních stránek [6]. Jazyk kombinuje vlastnosti několika jazyků (Perl, C, Pascal, Java) a nechává programátorovi částečnou svobodu v syntaxi. Díky tomu a díky bohaté zásobě funkcí (přes pět tisíc v základní knihovně) se jazyk stal velmi oblíbeným. Dnes je vedle ASP jedním ze dvou nejrozšířenějších skriptovacích jazyků pro web a jsou v něm napsány i ty největší internetové projekty, včetně Wikipedie. Rozsáhlá dokumentace jazyka včetně mnoha příkladů je dostupná online [7].

Hlavní vlastnosti jazyka PHP:

- Dynamicky typový jazyk – typ proměnné se určí v okamžiku přiřazení hodnoty, krom toho, že nemusíme deklarovat proměnné se tedy ani nemusíme starat o jejich typ.
- Heterogenní pole – mohou obsahovat prvky různých typů, mohou fungovat jako hashovací tabulka (klíčem může být číslo, řetězec i logická hodnota).

Výhody jazyka PHP:

- Specializovaný na webové stránky,
- rozsáhlý soubor funkcí,
- nativní podpora mnoha DB systémů,
- multiplatformnost,
- strmá křivka učení,
- rozsáhlá dokumentace.

Nevýhody jazyka PHP:

- Jazyk není kompletně definován – je popsán svou implementací,
- nekonzistentní pojmenování funkcí,
- v základní distribuci chybí ladící nástroj,
- neudržuje kontext požadavků - tuto nevýhodu lze odstranit použitím jednoho z mnoha PHP frameworků, kterým se věnuje kapitola 3 PHP frameworky.

I když jazyk PHP trpí mnoha nedostatky, je dnes velmi rozšířený a oblíbený. Jeho velká obliba je z velké části dána dostupností mnoha frameworků, které implementaci dynamických stránek v tomto jazyce velmi zjednodušují.

2.2.2 Databázový systém

Každý informační systém musí mít nějaké přostředky pro ukládání dat a pro základní práci s nimi. Dříve se k tomuto účelu využíval běžný souborový systém, přičemž způsob uložení dat a způsob práce s daty záležely na programátorovi, který tyto problémy musel řešit. Postupným vývojem tato problematika dospěla do stavu, kdy základní funkcionalitu pro práci s daty zajišťují databázové systémy. Programátor se těmito otázkami tedy již nemusí zabývat do takové hloubky a může věnovat více času vlastní logice vyvíjené aplikace.

Databázi můžeme definovat jako určitou uspořádanou množinu informací (dat) uloženou na paměťovém médiu. V širším slova smyslu jsou součástí databáze i softwarové prostředky, které poskytují rozhraní pro přístup k těmto datům a pro manipulaci s nimi. V české odborné literatuře se tento systém nazývá systém řízení báze dat (SŘBD). Pojmem databáze jsou často myšlena jak vlastní uložená data, tak i SŘBD [8].

Z hlediska ukládání dat a vazeb mezi nimi, lze databáze rozdělit na několik typů. Nejčastěji se setkáme s databázemi relačními. Termín relační databáze definoval Edgar F. Codd již v roce 1971 [8]. Data v relačních databázích jsou uložena v tabulkách, přičemž jednotlivým záznamům odpovídají jednotlivé řádky. Záznamy mohou prostřednictvím cizích klíčů odkazovat na jiné záznamy nebo jejich množiny (tabulky).

Mezi běžně dostupné databázové systémy patří například MySQL, PostgreSQL, Oracle nebo Microsoft SQL Server. Liší se ve svých vlastnostech a v ceně licence. Co se týče malých a středně velkých informačních systémů, je dostačující použít zdarma dostupný systém MySQL od společnosti Oracle. V praxi se databázový systém MySQL často používá i pro značně rozsáhlé projekty.

2.3 Klientská část IS

Klientskou částí portálového řešení IS rozumíme v podstatě uživatelské rozhraní. To bývá implementováno pomocí běžných webových technologií, kterými se ve stručnosti zabývá tato kapitola.

(X)HTML

HTML (Hypertext Markup Language) je značkovací jazyk pro hypertext, jenž je charakterizován množinou značek (tagů) a jejich atributů. Tyto značky můžeme rozdělit na **strukturální**, **popisné** a **stylistické**. Značky původně sloužili jak k definici struktury textu (nadpisy, odstavce, tabulky, seznamy atd.) tak pro definici vzhledu dokumentu (rozměry prvků, barvy, okraje, zarovnání atd.). Později se začala projevovat snaha, definici vzhledu od obsahu oddělit, čímž by se usnadnilo zpracování dokumentů a vyhledávání informací v nich obsažených. To vedlo ke vzniku kaskádových stylů. Zároveň vznikl jazyk XHTML (extensible hypertext markup language), který zachovává zpětnou kompatibilitu s jazykem HTML, avšak odstraňuje většinu formátovacích (stylistických) značek a atributů. V současnosti lze použít pro tvorbu webových stránek jak jazyk HTML, tak XHTML a oba jazyky se stále vyvíjejí.

CSS

Kaskádové styly (Cascading Style Sheets) jsou jazyk pro definici vzhledu dokumentu napsaném v jazyce HTML, XHTML nebo XML. Tento jazyk slouží jako náhrada za zaztaralé HTML značky a atributy a přidává k nim další rozšíření. CSS bývá často definováno v samostatném souboru a k dokumentu je připojeno pomocí speciálního tagu. Je možné definovat různé styly pro různá výstupní zařízení.

JavaScript

JavaScript je objektově orientovaný skriptovací jazyk, který se převážně používá jako interpretovaný programovací jazyk pro webové stránky. Syntaxí jazyk patří do rodiny jazyků C/C++/Java. Jakkoli tomu název napovídá, jazyk, vyjma syntax, nemá nic společného s jazykem Java. Kód v JavaScriptu může být zapsán, podobně jako kód CSS, v externím souboru nebo přímo v HTML dokumentu a bývá interpretován prohlížečem po kompletním načtení stránky.
Pomocí JavaSriptu můžeme měnit jak strukturu HTML dokumentu, tak CSS styl jeho jednotlivých elementů. Jazyk nám poskytuje možnosti reakce na uživatelské události v rámci stránky (pohyby/klikání myší, stisk kláves, aktivace tlačítka atd.) a nabízí řadu funkcí pro práci s datem a časem, pro navigaci v historii prohlížeče a vytváření oken, pro matematické operace a mnoho dalších.

jQuery

Jquery [17] je knihovna napsaná v JavaScriptu. Jejím účelem je zjednodušit přístup k jednotlivým prvkům dokumentu, obsluhu událostí, tvorbu animací a použití technologie AJAX. Tato knihovna je

navržena tak, aby změnila způsob zápisu JavaScriptu a tím rapidně urychlila vývoj aplikací. Proto se o ní často mluví jako o jQuery frameworku.

Jelikož je knihovna jQuery velmi rozšířená, existuje pro ni mnoho volně dostupných plug-inů. Jedním z velmi vydařených je plug-in DataTables [18]. Ten značně usnadňuje tvorbu interaktivních tabulek a postará se za programátora o stránkování, řazení podle jednotlivých sloupců a o vyhledávání podle hodnot v jednotlivých řádcích tabulky. Plug-in umí pracovat jak se statickými tabulkami, jejichž obsah je kompletně vypsaný v HTML, tak s dynamickými tabulkami, jejichž obsah se průběžně načítá ze serveru prostřednictvím technologie AJAX.

3 PHP frameworky

Přestože je jazyk PHP velmi mocný a poskytuje nám obrovské množství funkcí, implementace rozsáhlých systémů je v něm poměrně náročná. Proto se poslední dobou velké oblibě těší PHP frameworky, jenž nám nabízí větší míru abstrakce než samotný jazyk PHP a díky tomu snižují množství kódu které jsme nuceni napsat (a později udržovat).

Mezi hlavní výhody PHP frameworků patří především:

- Velká míra abstrakce dat – při použití ORM pracujeme výhradně s objekty,
- průhledné mapování URL na controller/metodu (to nemusí platit vždy),
- často dobře zpracované knihovny pro podporu multilingválnosti,
- přehledná adresářová struktura,
- strmá křivka učení – díky intuitivnosti, kvalitní dokumentaci a mnoha příkladům od různých autorů (záleží na konkrétním frameworku). Pokud navíc někdo již má zkušenosti s MVC frameworkem, velmi rychle se zorientuje v jiném frameworku.
- Dostupnost dalších knihoven (generování formulářů, validace, autorizace, práce s E-maily, online platby, …).

Použití frameworku s sebou však nese i jisté nevýhody. Když zanedbáme vliv na zvýšení spotřeby výpočetního výkonu, který je u velkých projektů poměrně zanedbatelný, stále si musíme dát pozor na několik věcí. Především bychom se s frameworkem měli dostatečně seznámit a zjistit, jaká má omezení, než v něm začneme implementovat rozsáhlý informační systém. V půlce bychom mohli narazit na problém, jehož řešení bude obnášet procházení a úpravy zdrojových kódů frameworku nebo, v horším případě, změnu frameworku.

Mimo to bychom si měli uvědomit, že framework může obsahovat chyby – něco prostě nemusí fungovat tak, jak by mělo. Proto je vhodné pro velké projekty použít alespoň několik měsíců „zaběhnutou“ stabilní verzi, která je dostatečně vyzkoušená. Ze stejného důvodu se rovněž vyplatí použít nějaký hodně rozšířený framework, jehož komunita většinu chyb poměrně rychle odhalí, což vede k jejich odstranění. Pro lepší představu toho, jak se PHP MVC frameworky mohou vzájemně koncepčně lišit, se poslední podkapitoly detailněji zaměřují na frameworky CakePHP a Kohana.

3.1 MVC Architektura

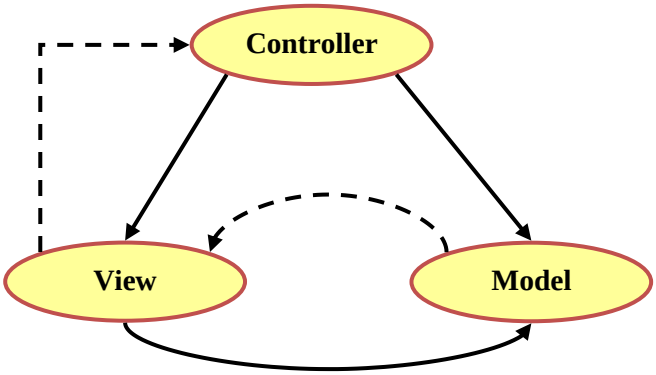
Frameworky často využívají architekturu MVC (Model-View-Controller) [9]. Ta rozděluje aplikaci na tři základní části – datový model, řídící logiku(Controllers) a pohledy na data(Views). Jednotlivé části je možné implementovat nezávisle a tedy i paralelně.

Controller se stará o zpracování klientových požadavků. K tomu může využívat modely pro přístup k datům a jako odpověď zašle klientovi pohled (view).

Model je vrstva nacházející se mezi databází a naší aplikací. Všechny přístupy k datům v databázi by měli probíhat prostřednictvím modelu. Modely jsou využívány nejčastěji v controllerech, ale je možné k nim přistupovat i z pohledů.

Pohledy (Views) slouží k prezentaci dat. V případě PHP frameworků se nejčastěji skládají z HTML kódu, ve kerém se na určitých místech vypisují proměnné, které jsou do pohledu předány

kontrolérem. Pohledy by měli obsahovat minimum logiky. Jejich smyslem je umožnit snadnou změnu struktury výstupů, a proto by měly být co nejpřehlednější. Koncept MVC je znázorněn na obrázku 1.



Obrázek 1: Koncept MVC. Plné šipky znázorňují přímou asociaci, čárkované nepřímou (tou se zabývat nebudeme).

ORM

Frameworky nám často nabízí možnost použít na úrovni modelů ORM (Object-Relational Mapping). Jedná se o vrstvu mezi relační databází a naší aplikací. Stará se o převádění dat z relační databáze do objektové reprezentace a naopak. Se záznamy v relační databázi pak můžeme pracovat jako s objekty jazyka PHP. Tato vrstva v podstatě zastává funkci modelu v MVC architektuře. ORM může být implementováno jako bázová třída, ze které se odvozují modely. Tato bázová třída zpravidla má množství metod pro skládání SQL dotazů a jejich provádění. Použití ORM bude ukázáno v následujících podkapitolách, které se podrobněji věnují několika PHP frameworkům.

3.2 Přehled PHP frameworků

V dnešní době existuje velké množství PHP frameworků a stále vznikají nové. Určit, který z nich je nejlepší, není možné, protože každému vyhovuje něco jiného a každý projekt klade na framework jiné nároky. Nyní se ve stručnosti podívámé na několik frameworků, které patří mezi nejznámější. Tato podkapitola je částečně převzata z [10].

Symfony

Symfony je MVC framework, který se zaměřuje na zrychlení vývoje a údržby webových aplikací. Je kompatibilní s téměř každým databázovým systémem. Zvláštností tohoto frameworku je, že vyžaduje přístup k příkazovému řádku, přes který se provádí mnohá nastavení a údržba. V knihkupectví Amazon je dostupných nekolik knih zabývajících se tímto frameworkem .

CodeIgniter

CodeIgniter je výkonný PHP framework s velkými možnostmi rozšíření. Je vytvořen pro PHP kodéry, kteří potřebují jednoduchý a elegantní nástroj pro tvorbu moderních webových aplikací. Pokud jste programátor žijící ve skutečném světě sdílených hostingů a klientů s deadliny a pokud jste unaven rozsáhlými a nedokumentovanými frameworky, pak je CodeIgniter pro vás tím pravým.

CakePHP

Framework CakePHP si dává za cíl minimalizovat čas potřebný pro vývoj webových projektů. Zavádí mnoho konvencí, díky čemuž se mu úspěšně daří minimalizovat množství zdrojového kódu, který jsme nuceni napsat. CakePHP je jedním z nejrozšířenějších frameworků a na internetu je pro něj k dispozici mnoho českých návodů. Je dostupný pro PHP 4 i PHP 5.

Zend

Zend framework se zaměřuje na tvorbu bezpečných, spolehlivých a moderních web 2.0 aplikací a webových služeb. Je připraven na velmi rozsáhlé projekty a je vytvořen agilními metodami. Zend framework byl od samého začátku unit testován se striktními požadavky na jeho kód. Komunita kolem tohoto frameworku je obrovská a na internetu je o něm tedy mnoho dostupných informací.

Prado

Prado není klasickým MVC frameworkem. Je založen na komponentovém přístupu a programování událostí. Pro svůj běh vyžaduje PHP 5.1.0 nebo vyšší. Je možné použít ho pro vývoj open-source i komerční aplikací.

Kohana

Kohana je elegantní HMVC PHP5 framework který poskytuje bohaté množství komponent pro tvorbu webových aplikací. Vyžaduje minimální konfiguraci, plně podporuje utf-8 a i18n a poskytuje mnoho užitečných nástrojů, které vývojáři potřebují. Integrované automatické načítání tříd, strukturovaný souborový systém a snadná integrace s dalšími knihovnami dělají Kohanu použitelnou pro libovolně velké projekty.

Akelos

Aplikace stojící na frameworku Akelos mohou běžet na většině sdílených hostingů, jelikož Akelos vyžaduje pouze dostupnost PHP bez dalších rozšíření. To znamená, že Akelos je ideální pro distribuci osamotě stojících aplikací, které ke svému běhu nevyžadují žádnou nestandardní PHP konfiguraci.

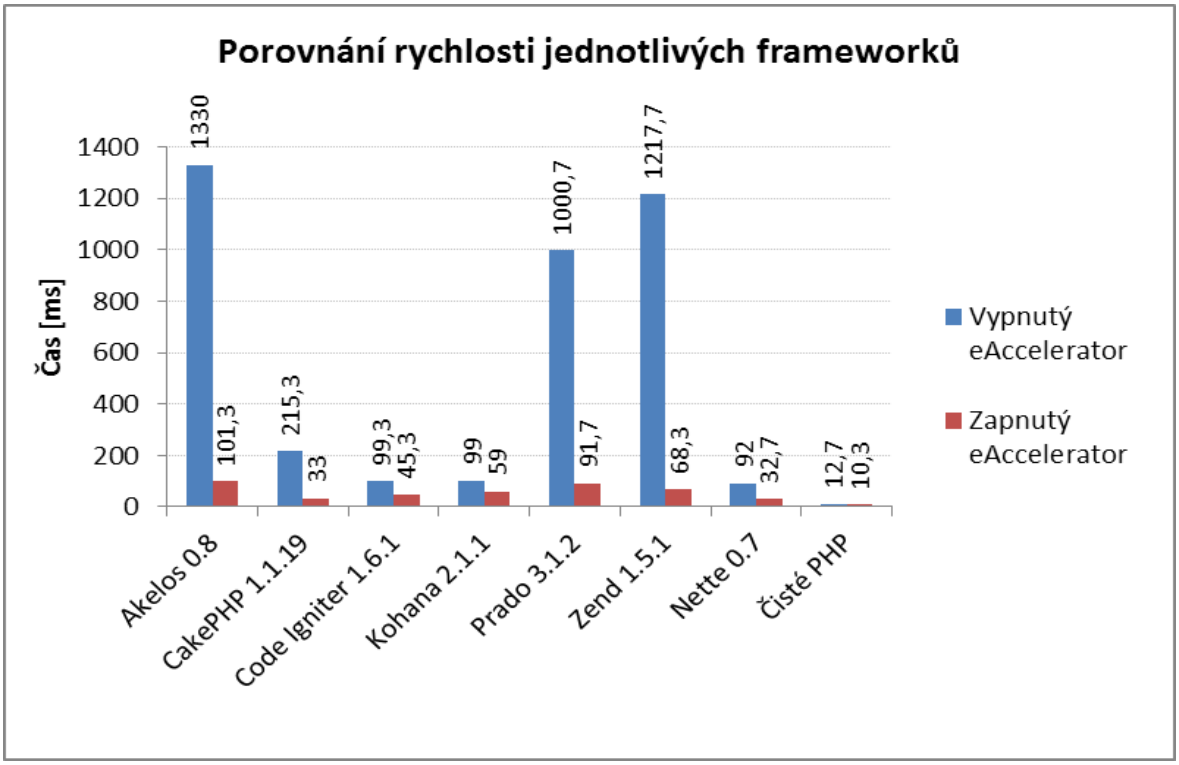
Nette

Nette framework spatřil světlo světa v roce 2008. Je dílem českého autora a je určen pro PHP 5.2 a vyšší. Nette se zaměřuje na eliminaci bezpečnostních rizik, jeho architektura je velice blízká konceptu MVC a během několika let kolem něj vyrostla nejaktivnější česká komunita. Dokumentace a množství návodů je u Nette nadstandardní a proto bych ho doporučil i začátečníkovi.

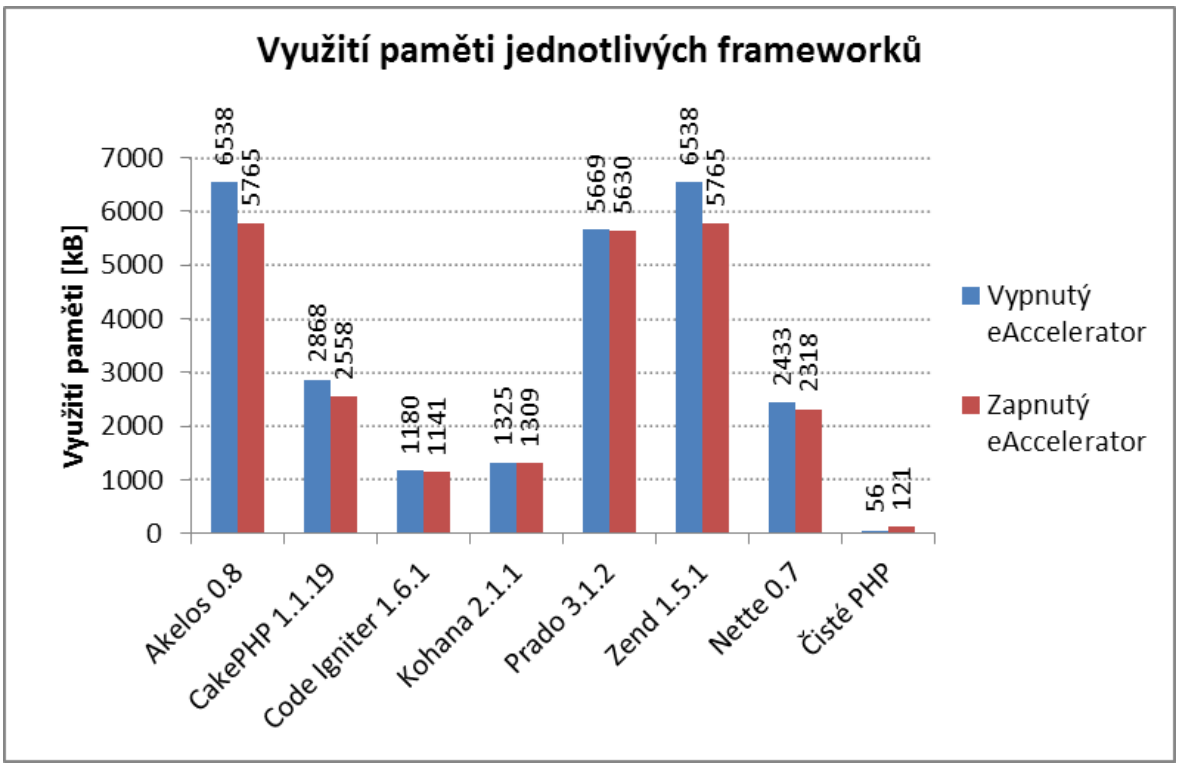
3.3 Srovnání PHP frameworků

Vytvořit vypovídající srovnání rychlosti a paměťové náročnosti několika PHP frameworků je časově velice náročné. Aby srovnání bylo objektivní, musel by jeho autor všechny srovnávané frameworky podrobně nastudovat a implementovat v každém z nich stejnou aplikaci. V této kapitole se nachází srovnání frameworků podle [12]. Autor v několika frameworcích implementoval stejnou aplikaci a poté pomocí speciálních nástrojů měřil rychlost a spotřebu operační paměti jednotlivých frameworků. Výsledky měření jsou vidět v grafech na obrázku 2 a na obrázku 3.

V testu zároveň bylo provedeno měření se spuštěným eAcceleratorem, což je nástroj, který se snaží běh aplikace zrychlit za pomoci cachovaní. Srovnání bylo vytvořeno v roce 2008 a jeho výsledky jsou dnes pouze orientační.



Obrázek 2: Srovnání rychlosti PHP frameworků [12].

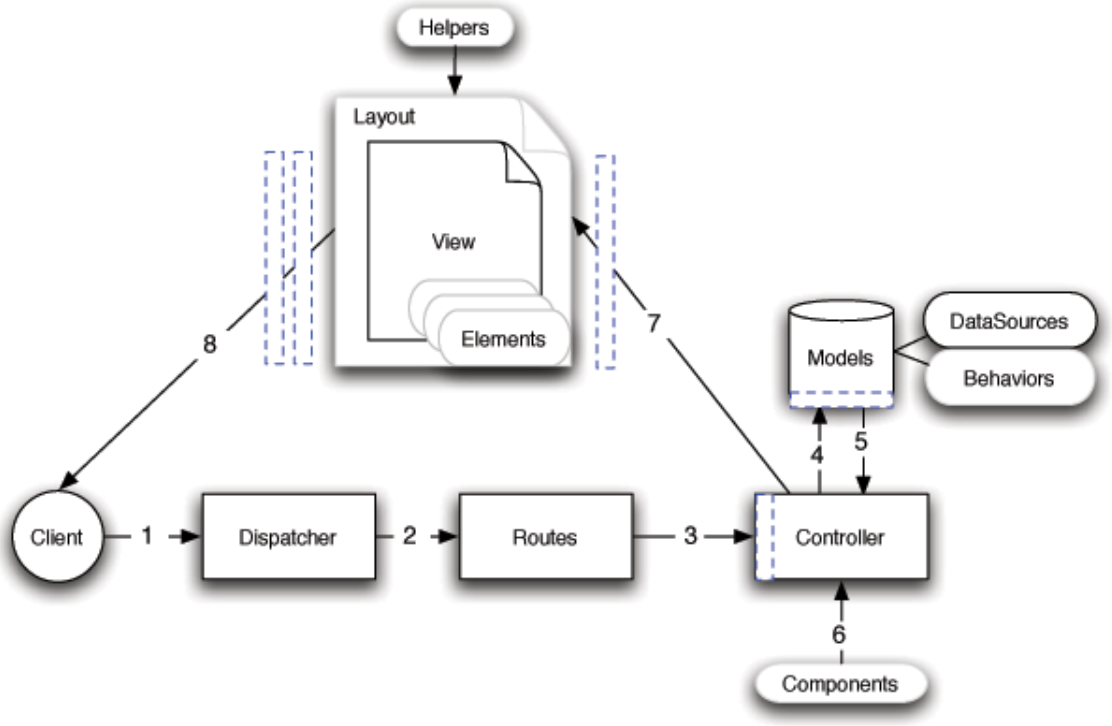


Obrázek 3: Srovnání využití paměti PHP frameworků [12].

3.4 CakePHP

CakePHP je klasický framework postavený na MVC architektuře. Vyznačuje se vysokou mírou konvencí – například je určeno, kde používat jednotná a kde množná čísla pro názvy tříd, jak pojmenovávat primární a cizí klíče v tabulkách a je konvencí, umisťovat pohledy do adresáře, jehož název odpovídá názvu controlleru a název pohledu by měl odpovídat názvu metody, ve které je použit. Při dodržování těchto pravidel je kód pro člověka, který s frameworkem neumí, poměrně nejasný, protože se mnoho věcí děje automaticky. Na druhou stranu člověk, který tyto konvence zná, se velmi rychle zorientuje v každém projektu, který se jimy řídí. Dodržování většiny konvencí není nutné, ale v určitých situacích pro vývojáře mohou být velmi omezující.

Běžné zpracování požadavku v CakePHP je znázorněno na obrázku 4. Tento koncept je společný téměř všem MVC frameworkům. Liší se většinou v pojmenování některých částí (např. místo „Components“ se často používá označení „Libraries“) nebo přidáním či odebráním určitích částí. Jde v podstatě o znázornění v praxi aplikované MVC architektury a můžeme pozorovat podobnost s obrázkem 1 – Koncept MVC.



Obrázek 4: Zpracování požadavku v CakePHP – převzato z [15]

Ukázka kódu kontroléru v CakePHP

Jelikož jednoduché modely a pohledy jsou si ve většině frameworků velmi podobné, rozhodl jsem se přiložit pouze ukázku controlleru (ukázka kódu 1), kterou považuji za nejvíce vypovídající.

```
class UsersController extends AppController {
    var $name = 'Users';
    // Seznam modelů, které tento controller používá
    var $uses = Array('User', 'Gallery', 'Foto', 'Contact');
```

```
// Seznam komponent které tento controller používá
var $components = Array('image');
// Zobrazí detailní informace o uživateli
function detail($user_id) {
    // Nastaveni layoutu
    $this->layout = 'default';
    // Nacte uzivatele z DB podle jeho ID
    $user = $this->User->findById($user_id);
    // Zpřístupní $user ve view - tím je podle konvencí
    // views/users/detail.ctp
    // I přes příponu .ctp jde o běžný PHP soubor
    $this->set('user', $user);
}
...
```

Ukázka kódu 1: Controller ve frameworku CakePHP

Práce s pohledy

CakePHP rozlišuje pojmy layout, view a element. Každá stránka má jeden layout (v něm je většinou hlavička, menu a patička – to co je společné pro více stránek) a jeden view (v něm je vlastní obsah stránky - „to co se mění“), který se může skládat z několika dalších elementů. Toto řešení na první pohled vypadá elegantně, ale při tvorbě větších projektů se ukáže jako velmi omezující.

Pokud budeme chtít jeden „pohled“, například detail produktu, použít jako view (pro stránku o produktu) a jinde jako element (pro stránku se srovnáním produktů), narazíme na problém (nelze použít dva views najednou). Jeho řešení není složité – detail produktu uděláme jako element a ve stránce o produktu tento element jako jediný načteme do příslušného view. Tento přístup však považuji za zbytečně komplikovaný a podobných komplikací je v CakePHP mnoho. Proto bych tento framework pro projekty většího rozsahu spíše nedoporučoval. Naopak je díky svým konvencím vhodný pro začátečníky, kteří získávají své první zkušenosti s frameworky a nemají zatím představu jak správně využít potenciál MVC architektury.

3.5 Kohana

Druhým PHP frameworkem, který blíže představím je Kohana. Ta ve verzi 3.1 vychází z HMVC (Hierarchical MVC) architektury [13]. Od MVC konceptu se liší tím, že při zpracování požadavku je možné vytvářet požadavky další. Tato vlastnost však není klíčová a nebudu se jí zde více věnovat. Většina zde uvedených informací nemusí platit pro starší verze tohoto frameworku a některé dokonce ani pro verzi 3.0.

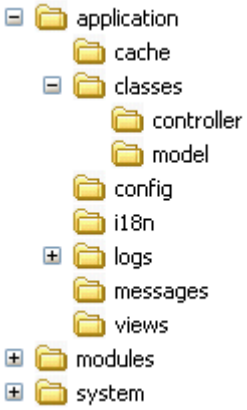
Narozdíl od CakePHP je v Kohaně často mnoho způsobů, jak udělat aplikaci správně. Většinu funkcionality můžeme často implementovat jak na úrovni modelů, tak na úrovni controllerů nebo pohledů. Nelze přesně určit, kde jakou funkcionalitu implementovat, ale je možné stanovit základní pravidla, která značně zvýší přehlednost, rozšiřitelnost a udržovatelnost naší aplikace.

Z MVC architektury PHP frameworků vycházejí následující pravidla pro tvorbu snadno udržovatelného kódu:

- V pohledech by mělo být co nejméně logiky - omezit se na několik větvení a iterací. Příliš rozsáhlé a nepřehledné pohledy vypovídají o nepochopení MVC architektury.
- Controllery by neměly obsahovat žádný HTML kód. Naopak by měly obsahovat většinu logiky aplikace.
- Veškerá práce s daty by měla být implementována v modelech. Ať už jde o data v databázi, v souborovém systému nebo někde jinde.
- Obecná funkcionalita, která nesouvisí přímo s jedním modelem by měla být definována ve zvláštních helperech/knihovnách/komponentách aby byla přístupná napříč celou aplikací.

Adresářová struktura

Adresářová struktura Kohany 3.1 je vidět na obrázku 5. Veškerý náš kód by měl být umístěn v adresáři `application`. Všechny vlastní třídy bychom měli umisťovat do adresáře classes, přičemž podtržítka („_“) v názvech tříd jsou Kohanou při vkládání souborů transformovány na znaky oddělovače adresářů („/”). To v praxi znamená, že třídu s názvem `Controller_Admin_news` musíme umístit do souboru s cestou `classes/controller/admin/news.php`.



Obrázek 5 Adresářová struktura frameworku Kohana 3.1

Ukázka modelu

Následující model by sloužil pro přístup k tabulce `users` a pro přístup k souvisejícím tabulkám prostřednictvím jejich modelů. V ukázce není nutné definovat hodnotu atributu `$_table_name`, neboť Kohana automaticky předpokládá, že název tabulky odpovídá množnému číslu od názvu modelu. Při převodu na množné číslo se však řídí anglickou gramatikou, a tak je někdy definice vlastního názvu nutná. Atribut `$_primary_key` také není třeba uvádět, neboť uvedená hodnota (`id`) je implicitní. Atributy `$_belongs_to` a `$_has_many` obsahují seznamy souvisejících modelů, ke kterým je možné přistupovat prostřednictvím tohoto modelu. Uvedený kód modelu by se nacházel v souboru `classes/model/user.php`.

```
<?php
class Model_User extends ORM {
    protected $_table_name = 'users';
    protected $_primary_key = 'id';
    protected $_belongs_to = Array();
    protected $_has_many = Array(
```

```
            'messages' => Array('model'=>'message',
'foreign_key'=>'user_id'),
        );
}
?>
```

Ukázka kódu 2: Model ve frameworku Kohana 3.1

Ukázka controlleru

Controller v následující ukázce obsahuje jedinou metodu, která zpracovává požadavek na zobrazení všech uživatelových zpráv. Jediným parametrem metody je identifikátor uživatele, který odpovídá hodnotě primárního klíče v uživatelově databázovém záznamu. Následující kód by se nacházel v souboru `classes/controller/user.php`.

```
<?php
class Controller_User extends Controller {
    public function action_messages($user_id)
    {
        // Automaticky načte záznam uživatele s PK=$user_id
        $user = ORM::factory('User', $user_id);
        // Načte všechny uživatelovy zprávy
        $messages = $user->messages->find_all();
        // Vytvoří instanci pro přístup k pohledu
        $view = View::factory('user/messages');
        // Předá do pohledu odkazy na proměnné $user a $messages
        $view->bind('user', $user)->bind('messages', $messages);
        // Nastaví tělo odpovědi
        $this->response->body($view);
    }
}
?>
```

Ukázka kódu 3: Controller ve frameworku Kohana 3.1

Ukázka pohledu

Tato ukázka obsahuje kód pohledu, který se stará o výpis informací o uživateli a výpis seznamu uživatelových zpráv. Pro výpis jednotlivých zpráv je použit další pohled, jehož obsah není uveden, neboť není zajímavý. Uvedený kód by byl umístěn v souboru `views/user/messages.php`.

```
<div class="user">
    <div class="name"><?= $user->name ?></div>
    <div class="last_logged"><?= $user->last_logged ?></div>
</div>
<div class="messages">
    <?php
```

```
        foreach ($messages as $m) {
            // HTML struktura zprávy je popsána ve vlastním pohledu
            echo View::factory('user/message')->set('message', $m);
        }
    ?>
</div>
```

Ukázka kódu 4: Pohled (view) ve frameworku Kohana 3.1

V ukázkách kódu ve frameworku Kohana můžeme vidět, že pro přístup k pohledům slouží statická metoda factory třídy View, která je dostupná napříč celou aplikací. Nejinak je tomu u metody factory třídy ORM, která slouží pro přístup k modelům. Díky tomuto faktu můžeme s pohledy a modely pracovat kdekoli v naší aplikaci. To je základní koncepční rozdíl mezi frameworky CakePHP a Kohana neboť v CakePHP můžeme k modelům přistupovat pouze z controlleru a použitý pohled je dán již názvem metody controlleru, ve které se nacházíme.

4 Generování zdrojového kódu

Generováním zdrojového kódu lze nazvat proces, jehož vstupem je model systému a výstupem je zdrojový kód v určitém jazyce. Modelem může být například datový model, objektově orientovaný model v jazyce UML nebo model uživatelského rozhraní. Typu modelu pak odpovídá typ jazyka vygenerovaného kódu. Jednotka, která generování provádí se označuje jako generátor zdrojového kódu [16].

4.1 Význam generování zdrojového kódu

Motivem pro automatické generování zdrojového kódu může být snaha o zjednodušení práce, která se projevuje v možnostech tvorby uživatelských rozhraní v moderních vývojových prostředích. Hlavním důvodem proč automaticky generovat zdrojový kód však bývá úspora času. Při vhodném použití generátorů zdrojového kódu můžeme u velkých projektů ušetřit velké množství času a tím i práce.

Vizuální programování

S jistou formou generování zdrojového kódu se můžeme setkat ve všech moderních vývojových prostředích jako je například Microsoft Visual Studio, Qt Creator, NetBeans a mnohá další, bez ohledu na programovací jazyk pro který jsou určena.

Tato prostředí často umožňují generovat kód uživatelského rozhraní prostřednictvím tzv. vizuálního programování. Programátor má na vybranou, zda kód uživatelského rozhraní napíše sám, nebo pro jeho vytvoření použije vizuální designer, ve kterém jednotlivé prvky rozhraní umisťuje na formulář prostřednictvím myši a ihned vidí jak rozhraní bude, po případném překladu a spuštění aplikace, vypadat. Tato technika tvorbu uživatelských rozhraní do velké míry zjednodušuje, často ovšem na úkor přehlednosti výsledného kódu, což později znepříjemňuje jeho ruční úpravy.

UML nástroje

Na jinou formu generování zdrojového kódu lze narazit v nástrojích pro modelování v jazyce UML. Tyto nástroje často umožňují vygenerovat SQL kód se strukturou databáze na základě datového modelu (ER Diagramu) nebo kód tříd s deklaracemi či hlavičkami metod na základě objektového modelu aplikace. Dá se říct, že čím větší projekt vytváříme, tím více času může kód takovýmto způsobem vygenerovaný ušetřit. Toto pravidlo ovšem platí pouze za předpokladu, že se při generování kódu vyhneme úskalím, která nastíníme v následující části kapitoly [16].

4.2 Problémy s generováním kódu

Tato podkapitola byla převzata z [16].
Jak se říká – nic není zadarmo. Toto úsloví platí v oboru softwarového vývoje vždy a všude. Jakýkoli přínos je vždy vykoupen něčím, co je nutné obětovat. Nejinak je tomu při využívání automatického generování zdrojového kódu. Problémy na které můžeme při používání generátoru kódu narazit se dají rozdělit podle následujících tří oblastí:

- *Oblast technologie,*

- *oblast organizace,*
- *oblast psychologie.*

Nyní se postupně zaměříme na každou z oblastí.

Oblast technologie

Vývojové technologie jsou jedním z klíčových aspektů vývoje softwaru. Jedná se o programovací jazyky, kompilátory, vývojová prostředí a další. S nimi souvisí některé z překážek, na které při genrování kódu můžeme narazit:

1. *Vygenerovaný kód neodpovídá námi používaným technologiím. Jinými slovy, kód je generován v jazyce, jenž není podporován nástroji, které při vývoji používáme. Může se ale jednat p ouze o rozdílné verze jedhoho a téhož jazyka. Potom stačí malá odlišnost a překladač již vygenerovaný kód nemusí zkompilovat.*
2. *V případě, že náš kompilátor jazyk generovaného kódu podporuje, je vše v pořádku. Pokud se však rozhodneme o upgrade vývojových nástrojů s podporou jazyka vyšší verze, mohou nastat problémy popsané v bodě 1.*
3. *V oblasti vývoje softwaru občas dojde k nahrazení stávající technologie novou, úplně odlišnou. Ta bude podporovat naprosto jiný jazyk, než se používal doposud. Komplikace popsané v bodě 1 nastanou za této situace zcela jistě v mnohem větším měřítku, než v předchozím případě.*

Způsob řešení problémů týkajících se této oblasti závisí na vlastnostech používaného generátoru. Pokud je generovaný kód v jiném jazyce či jiné verzi jazyka než používáme při implementaci, v zásadě mohou existovat dvě varianty řešení. První z nich je, že generátor přestaneme používat a nahradíme jej jiným, který nám po stránce jazyka generovaného kódu bude vyhovovat. Mnohem schůdnější je situace v případě, že generátor kódu je uživatelsky otevřený, což znamená, že formát výstupu lze uživatelem definovat. To je umožněno například tak, že se ke generátoru připojí textový soubor, který obsahuje popis formátu výstupu pomocí metajazyka. Tento soubor je možné později modifikovat č nahradit jiným, což řeší problém s inovací či změnou vývojových technologií.

Oblast organizace

Softwarový vývoj středně velkých a velkých projektů zpravidla předpokládá týmovou spolupráci. Každý člen týmu vystupuje zpravidla v některé z následujících rolí:

- *Vedoucí projektu,*
- *analytik,*
- *vývojář,*
- *tester.*

Dobré fungování vývojového týmu ovšem předpokládá jeho dobrou organizaci. Musí být dané, který člen týmu za co odpovídá, jaká jsou pravidla a v jakých termínech se musí stihnout jednotlivé dílčí práce. Pokud je při vývoji projektu používáno generování kódu, lze se setkat s těmito problémy:

1. *Není zcela jasné, kdo je za generování kódu odpovědný. Je to analytik, který ručí za správnost návrhu systému, nebo vývojář, který má na starosti implementaci čili vytvoření zdrojových kódů a následné odladění?*
2. *Úprava vygenerovaného zdrojového kódu neodpovídá kódovacím konvencím dohodnutým v rámci týmu. Díky tomu zdrojové kódy ztrácí na jednotnosti a tím i na přehlednosti.*

3. *Generátor kódu plně nepodporuje používanou metodiku vývoje. Jako příklad lze uvést iterativní vývoj, kde po končeném počtu iterací postupně dospějeme k požadované podobě vytvářeného systému. V první iteraci se vytvoří prvotní návrh, ten se po dokončení implementuje a implementace se otestuje. V každé další iteraci se návrh upraví, poté se úpravy implementují a otestují. V poslední iteraci se provede poslední verze návrhu a implementace a konečné testování. Generování kódu z prvotního návrhu bude probíhat stejně jako u neiterativního vývoje. Použití generátoru v dalších iteracích už bude komplikovanější, jelikož se budeme muset vyhnout nebezpečí přepsání dosud napsaného kódu.*

Jak již bylo řečeno výše, v projektovém týmu musí být jasné, kdo nese za co odpovědnost. To platí i pro generování kódu. Je víceméně záležitostí vedoucího projektu, zda odpovědným v tomto případě stanoví analytika či vývojáře. Pokud jsou konvence kódování odlišné od úpravy generovaného kódu, nabízí se řešení změnit konvence. To je sice možné, avšak v případě projektu v pokročilejší fázi poněkud nepraktické. Jestliže používáme uživatelsky otevřený generátor kódu, můžeme jej našim konvencím přizpůsobit. Pro zmíněný iterativní vývoj by se nejlépe hodil generátor, který umožňuje synchronizaci modelů návrhu s odpovídajícími zdrojovými kódy. Jinými slovy, generátor by byl schopen generovat pouze změny v modelu tak, aniž by stávající kód byl narušen. Pokud takový generátor nemáme k dispozici, pak je nutné vygenerovat kód do nových souborů a pomocí některého nástroje tyto soubory sloučit se zdrojovými kódy z minulé iterace.

Oblast psychologie

Psychologie má významné postavení ve všech oblastech lidského konání. Výjimkou není ani softwarový vývoj, kde úspěšné dokončení projektu je mimo jiné ovlivněno psychikou každého člena týmu. Nyní uvedeme případy, kdy psychologická stránka generování kódu může mít na vývoj negativní vliv:

1. *Význam generování kódu je přeceněn. Skutečnost, že výsledkem generování kódu je pouze jeho „kostra“ a že většinu stejně musí vývojář dopsat ručně, zjistí každý velice brzy, často ještě dříve, než vůbec začne generátor používat. Avšak to, že generátor je pouze výplodem člověka, člověk je tvor nedokonalý a tím pádem i generátor a potažmo vygenerovaný kód může mít nedostatky, si už každý neuvědomí. Pak se lze setkat s případy toho typu, kdy se ve vygenerované hlavičce metody swap() předávaly parametry hodnotou místo odkazem a vývojář strávil několik hodin nad hledáním chyby ve vlastnoručně napsaném kódu.*
2. *Vygenerovaný kód omezuje vývojáře. Na vygenerovaný kód se často pohlíží jako na něco, co je dané, a tudíž neměnné. Tento předsudek tak činí z generovaného kódu jakési dogma, které poněkud zužuje zorné pole kreativity vývojáře, jenž modifikaci generovaného kódu obchází pomocí jakýchsi záplat v podobě složitých konstrukcí v místech, kde jsou změny dovoleny. A to není dobré.*

Překonat překážky týkající se psychologie bývá mnohem náročnější než přizpůsobit generátor kódu technologiím či kódovacím konvencím. Rozhodně je nutné si uvědomit, že automatické generování neznamená bezchybné generování a že se jedná o nástroj, který nám má pomoci a nikoliv o mantinel, který bude omezovat naše tvůrčí myšlení. Je důležité, aby vedoucí projektových týmů na tyto zásady své podřízené upozorňovali a dohlíželi na jejich dodržování.

Ačkoliv automatické generování zdrojového kódu s sebou přináší svá úskalí, rozhodně stojí za to se jím zabývat a používat jej při vývoji projektů středně velkého a velkého rozsahu. Je ovšem potřeba vybrat správný generátor, který by měl být určitě uživatelsky otevřený a případně mít i schopnost

synchronizace s dosud napsaným zdrojovým kódem. Avšak i při práci s tím nejlepší generátorem kódu je potřeba mít na paměti, že o něm platí přesně to, co o ohni: je dobrý sluha, ale špatný pán.

5 Analýza a návrh aplikace

Tato kapitola se zabývá analýzou požadavků a návrhem aplikace, která má umožnit vygenerovat zdrojový kód základních operací nad jednou tabulkou. Na úvod se budu zabývat výběrem PHP frameworku a databázového systému, pro něž bude koncipován výstup generátoru. Poté upřesním, jaké požadavky budou na generátor kladeny a jak by mělo probíhat vytváření informačního systému pomocí generátoru. Další část kapitoly se věnuje návrhu uživatelského rozhraní a návrhu struktury výstupů.

5.1 Volba PHP frameworku a databáze

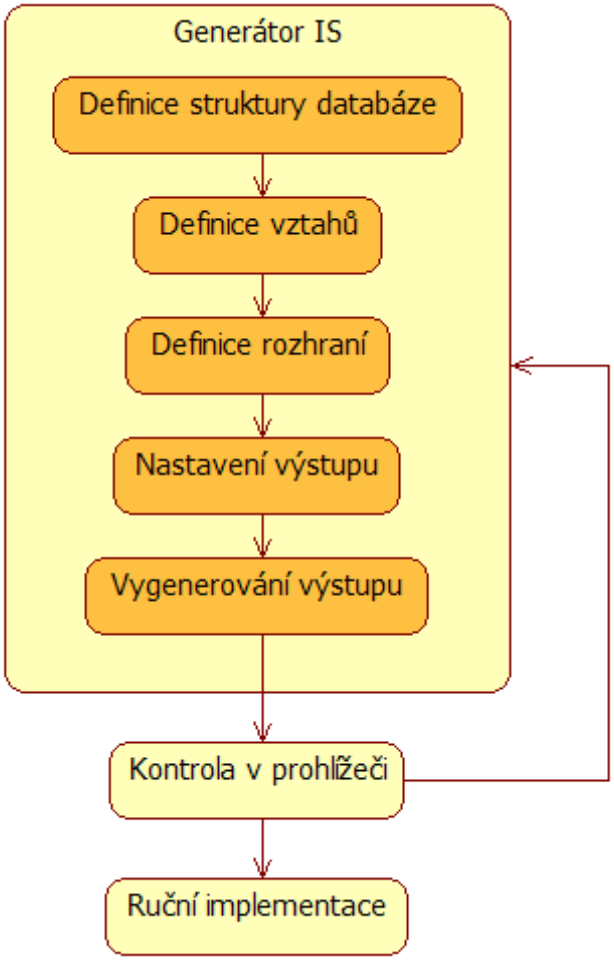
Volba frameworku, pro který bude určen generovaný zdrojový kód, není tak důležitá, jak by se mohlo zdát. Jistě by bylo vhodné zvolit framework, který je nejpoužívanější, aby má aplikace mohla být k užitku co největšímu množství vývojářů. MVC frameworky však mají velmi podobnou strukturu zdrojových kódů a změnit formát výstupu nebo přidat další podporovaný formát by tedy neměl být velký problém. Proto jsem bez váhání jako hlavní cílovou platformu zvolil framework **Kohana 3.1**. Tuto volbu jsem učinil především z toho důvodu, že se aktivně podílím na několika projektech v Kohaně a myslím, že i v blízké budoucnosti se tímto frameworkem budu zabývat.

S použitím PHP frameworku se volba databáze stává méně významnou. O práci s databází se zpravidla starají knihovny frameworku, které komunikaci s databázovým systémem zapouzdřují a umožnují jednotný přístup k mnoha různým databázovým systémům. Jako jedno z rozšíření generátoru mám však v úmyslu umožnit generování SQL kódu s příkazy pro vytvoření databázových tabulek. Tento SQL kód bude koncipován pro databázový systém **MySQL**, který považuji v oblasti malých informačních systému za nejrozšířenější. Nicméně generovaný SQL kód bude pravděpodobně kompatibilní i s dalšími databázovými systémy.

5.2 Role generátoru

Než se začnu zabývat návrhem uživatelského rozhraní a návrhem výstupů aplikace, považuji za vhodné ujasnit si jakou roli v implementaci informačního systému bude generátor hrát. Cílem by mělo být vygenerovat jakousi kostru systému, která implementuje základní, uživatelem definovanou funkcionalitu. Tato kostra je určena k rozšiřování a programátor by měl mít možnost do vygenerovaného kódu jakkoli zasahovat.

Pokoušet se o oddělení vygenerovaného kódu od kódu ručně napsaného nepovažuji za vhodné, neboť by to značně snížilo přehlednost výsledných zdrojových souborů a programátor by musel mnoho zaběhnutých postupů nahradit jinými, zbytečně složitějšími postupy.



Obrázek 6: Diagram implementace IS s použitím generátoru zdrojového kódu

Z diagramu na obrázku 6 je patrné, že role generátoru končí jakmile programátor začne vygenerovaný kód ručně upravovat a rozšiřovat. Díky tomu se vyhnu problému jak rozlišit vygenerovaný kód od ručně napsaného. Pro projekty, u kterých se specifikace požadavků výrazně mění během implementační fáze, bude však generátor téměř nepoužitelný nebo použitelný jen s obtížemi. Také je patrné, že vygenerovaný výstup by měl být snadno spustitelný v prohlížeči, aby mohla ihned po vygenerování výstupu proběhnout jeho kontrola.

5.3 Požadavky na generátor

Hlavním výstupem generátoru má být kód umožňující základní operace nad jednou tabulkou. Tyto operace často bývají označovány zkratkou CRUD (Create, Retrieve, Update, Delete)[14]. Hlavní část generátoru by tedy měla umožnit generování modelů, controllerů a pohledů, které požadované operace nad vybranými tabulkami umožní.

Je velice pravděpodobné, že ve vygenerovaném informačním systému se uplatní modely pro přístup ke všem definovaným tabulkám. Generátor by tedy měl vygenerovat modely pro všechny tabulky. Tyto modely by zároveň měly obsahovat definici vazeb na další modely. Určitě by bylo vhodné, aby aplikace umožnovala uložit rozpracovaný projekt, včetně různých nastavení. K tomuto účelu může posloužit formát XML. Dále by aplikace měla poskytovat prostředky pro definici

kompletní struktury databázových tabulek, ze které bude možné vygenerovat SQL skript pro jejich vytvoření.

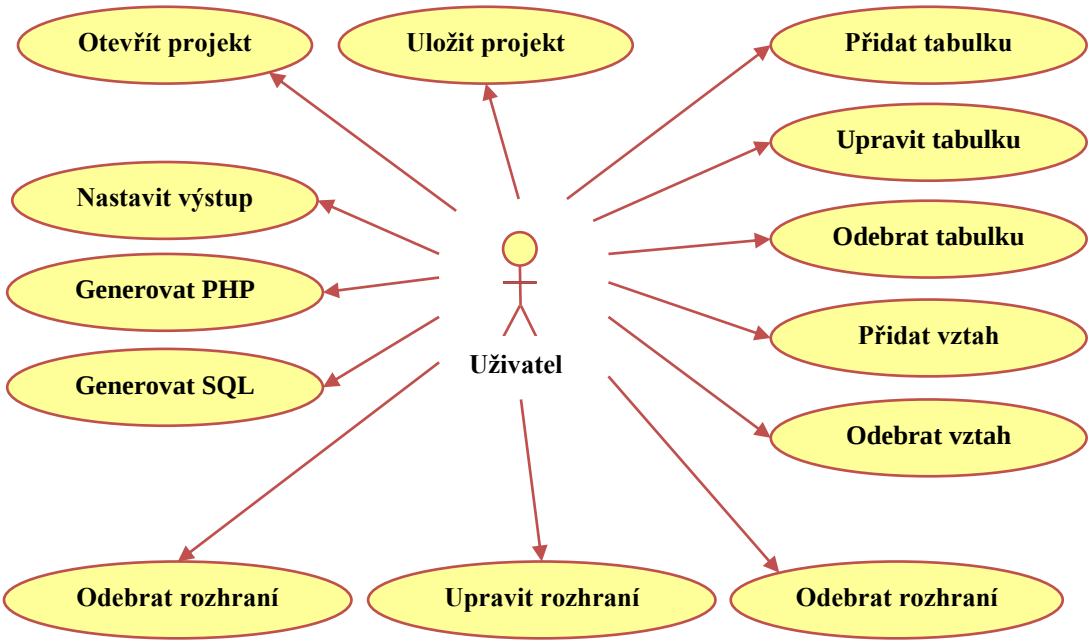
Generátor by tedy měl poskytnout vhodné uživatelské rozhraní, které umožní uživateli pohodlným způsobem zadat všechny nutné informace.

5.4 Návrh generátoru

Nyní se budu zabývat návrhem generátoru, přičemž se zaměřím na návrh jeho uživatelského rozhraní. Na úvod připomínám, že uživatelské rozhraní by mělo umožnit především:

1. Definici kompletní struktury tabulek,
2. definici vazeb mezi tabulkami (ty odpovídají vazbám mezi modely),
3. definici jednotlivých rozhraní pro CRUD operace nad tabulkami.

Pro jasnou představu, co vše by měl generátor uživateli umožňovat, jsem v jazyce UML vytvořil diagram případů užití, který je vidět na obrázku 7. Klíčovým případům užití se potom věnují následující části kapitoly.



Obrázek 7: Generátor IS – diagram případů užití

5.4.1 Definice struktury tabulky

Klíčovou roli v celé aplikaci budou hrát databázové tabulky. UI by tedy mělo v hlavním okně zobrazovat seznam vytvořených tabulek, mělo by umožnit tyto tabulky editovat a vytvářet nové. Abych se vyhnul problémům s navigací uživatele v rámci aplikace, rozhodl jsem se pro definici struktury tabulky použít formulář v samostatném okně. Toto okno by navíc mělo být modální, aby uživatel po dobu jeho zobrazení nemohl pracovat s ostatními částmi aplikace.

5.4.2 Definice vztahů

Zajímavějším problémem bude návrh rozhraní pro definici vazeb mezi tabulkami. Nabízejí se dvě možná řešení, která se nyní pokusím přiblížit. U obou řešení vycházím z faktu, že vazby jsou realizovány prostřednictvím jednoduchých (ne složených) primárních a cizích klíčů.

Formulářové řešení

Vazbu mezi tabulkami lze definovat prostřednictvím formuláře pro definici struktury tabulky. Toto řešení by obnášelo u každého sloupce tabulky zobrazit komponentu umožnující výběr jedné z definovaných tabulek. Výběrem nějaké z tabulek by uživatel dal najevo, že daný sloupec je cizím klíčem, odkazujícím na primární klíč zvolené tabulky. Toto řešení se jeví jako poměrně jednoduché a přehledné, nicméně neposkytuje uživateli ucelený pohled na vazby v jeho databázi. Dokonce neposkytuje ani ucelený pohled na vazby týkající se jedné tabulky.

Grafické řešení

Druhým možným řešením je použití grafické scény, která by strukturu databáze zobrazovala formou diagramu. Uživatel by měl možnost měnit pozici tabulek ve scéně a definice vztahu by probíhala tažením myši z jedné tabulky na druhou. Přesněji řečeno tažením myši ze sloupce jedné tabulky na sloupec tabulky druhé, neboť pro definici vztahu nám pouhá znalost tabulek nestačí. Vztah by poté byl v diagram reprezentován čárou spojující dané tabulky. V tomto případě by šlo o čáru spojující přímo dva sloupce různých tabulek. Čára reprezentující vztah musí být „označitelná“, aby šlo vztah odebrat. Vzhledem k tomu, že vytváření vztahů tímto způsobem je velmi snadné, není ani nutné poskytovat prostředky pro jejich změnu. Pokud uživatel bude chtít vztah změnit, jednoduše ho odstraní a vytvoří znovu.

I přes počáteční obavy z náročnosti implementace jsem se rozhodl pro grafické řešení. Bylo by samozřejmě možné použít obě řešení zároveň, ale přidaná hodnota formulářového řešení je dle mého názoru zanedbatelná.

5.4.3 Definice požadovaných rozhraní

Aplikace by měla podporovat tyto čtyři základní typy rozhraní pro práci s databázovými tabulkami (v závorce je uveden mnou zavedený a zkrácený název typu rozhraní, který se dále používá pro jeho označení):

- Zobrazení obsahu formou tabulky (*list*),
- zobrazení detailu jednoho záznaku (*detail*),
- přidání záznamu (*add*),
- editace záznamu (*edit*).

Ke každé tabulce bude možné definovat neomezený počet rozhraní. Ta mohou být stejného typu a lišit se pouze v některých parametrech. Na první pohled se může zdát zbytečné, generovat k jedné tabulce více rozhraní stejného typu, ale tato vlastnost může být užitečná třeba v případě, kdy chceme mít k dispozici výpis všech uživatelů a zvlášť mít výpis uživatelů, kteří mají v systému novou objednávku. Někdy také můžeme chtít různá editační rozhraní pro různé úrovně oprávnění nebo pro různé typy záznamů - jiné editační rozhraní u nově vytvořené objednávky než u uzavřené

objednávky, přestože oba typy jsou v jedné tabulce. atd. Každé definované rozhraní bude mít svůj název, který bude dále sloužit pro generování názvu odkazu v menu IS.

Při definici rozhraní by měl uživatel mít možnost zvolit, se kterými sloupci tabulky bude dané rozhraní pracovat, protože v rozhraní typu list většinou nechceme vidět všechny sloupce, ale pouze vybrané, podle kterých nalezneme záznam, který nás zajímá a pokračujeme zobrazením jeho detailu nebo jeho editací. Stejně tak v rozhraní typu add a edit nemusíme vidět sloupce, jejichž hodnoty se generují automaticky. Často také nechceme zobrazovat sloupce obsahující cizí klíče.

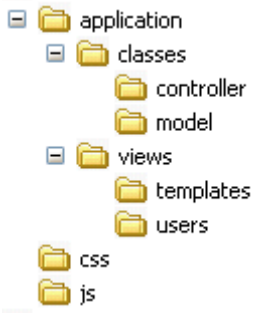
Pro vybrané sloupce by dále uživatel mohl chtít definovat název, pod kterým se budou zobrazovat v rozhraní. Například ke sloupci „name“ by tento název mohl být „Jméno“. Pod těmito názvy budou jednotlivé sloupce prezentovány uživateli generovaného IS. Jednotlivé typy rozhraní budou mít různé možnosti dalších jednoduchých nastavení. Výčet těchto nastavení je následující:

- *List*
    a. Použít jQuery plug-in DataTables,
    b. generovat odkazy na smazání záznamu,
    c. generovat odkazy na editaci záznamu,
    d. generovat odkazy na detail záznamu,
    e. generovat odkaz na přidání záznamu.
- *Detail*
    a. Generovat tlačítko pro editaci,
    b. generovat tlačítko „zpět“.
- *Add, edit*
    a. Generovat tlačítko „zpět“.

Část UI pro definici rozhraní by bylo vhodné umístit do hlavního okna aplikace, protože v ní bude uživatel odvádět velkou část své práce s generátorem a neustálé otevírání a zavírání nových oken by dle mého názoru snižovalo přehlednost a efektivitu práce s generátorem.

5.5 Návrh struktury výstupu

Hlavním výstupem generátoru bude v podstatě kompletní zdrojový kód informačního systému, jenž bude umožňovat základní operace nad jednou tabulkou. Slovem kompletní rozumíme, že výstup bude složen ze všech zdrojových souborů nutných pro spuštění generovaného informačního systému. Výstupem tedy bude adresářová struktura a zdrojové soubory projektu. Automaticky generovaná adresářová struktura bude vycházet ze struktury frameworku Kohana 3.1 a je znázorněna na obrázku 8.



Obrázek 8: Generovaná adresářová struktura

Uživatel před zahájením generování specifikuje adresář, do kterého chce výstup generovat a v něm generátor vytvoří kompletní strukturu uvedenou na obrázku 8. V případě, že některý z adresářů bude již existovat (při generování do adresáře s Kohanou nebo při opakovaném generování), jeho vytváření se přeskočí a ručně vytvořené soubory v daném adresáři budou ponechány.

Modely, controllery a pohledy

Pro každou definovanou tabulku bude vygenerován model a pro každou tabulku která má definováno alespoň jedno rozhraní bude vygenerován jeden controller. Model i controller budou mít stejný název jako daná tabulka.

Metody controlleru pro obsluhu akcí jednotlivých rozhraní se budou jmenovat action_TYP(X), kde „action_" je Kohanou definovaný prefix, TYP odpovídá typu rozhraní (*list, detail, edit, add*) a X je rostoucí celé číslo pro odlišení metod v případě generování více rozhraní stejného typu nad jednou tabulkou.

Pro každý controller bude vytvořena složka v adresáři `views`, jejíž název také odpovídá názvu tabulky. Do té budou poté vygenerovány jednotlivé soubory pohledů, použité v daném controlleru. Název souborů s pohledem bude odpovídat hodnotě výrazu TYP(X).php

Další potřebné soubory

Krom modelů, controllerů a pohledů, implementujících uživatelem požadovaná rozhraní, se vždy vygenerují další soubory nutné pro běh systému. Jejich přehled je uveden v tabulce 1.

Relativní cesta ze složky application	Stručný popis souboru
classes/session.php	Třída implementující metody pro zobrazování informativních zpráv.
classes/web.php	Třída obsahující definici menu a další pomocné metody.
classes/controller/template.php	Bázová třída pro uživatelské controllery implementující použití šablon.
views/templates/ajax.php	Šablona pro odpovědi na AJAXová volání.
views/templates/main.php	Šablona pro ostatní odpovědi – obsahuje HTML hlavičku, hlavičku a patičku stránky a menu.
css/default.css	Výchozí CSS určené k přepsání nebo nahrazení jiným souborem.
js/jquery.js	jQuery framework pro JavaScript.
js/jquery.dataTables.js	Plug-in pro jQuery, jenž zjednodušuje tvorbu interaktivních tabulek.

Tabulka 1: Seznam pomocných souborů nutných pro běh informačního systému

Další požadavky na generovaný IS:

- **Oddělení vzhledu od obsahu** – kde změně vzhledu vygenerovaného IS by měla stačit pouhá úprava CSS, bez jakýchkoli zásahů do zbytku vygenerovaného kódu.
- **Multilingválnost** - všechny textové výstupy pro uživatele budou ve vygenerovaném systému vypisovány pomocí funkce pro překlad. Ta má v Kohaně 3.1 s použitím standardní knihovny i18n název „__" (dvě podtržítka). Takže například text odkazu „Uživatelé" bude vygenerován

jako „__(‘Uživatelé‘)“. Tím bude zajištěna možnost snadného překladu informačního systému do jiného světového jazyka pomocí jazykových souborů.

- **Struktura menu** - pro umožnění základní navigace bude generováno také menu. Hlavní menu bude obsahovat odkaz pro každou tabulku s alespoň jedním rozhraním. Podmenu pak bude obsahovat odkazy na jednotlivá rozhraní tabulky zvolené v hlavním menu. Rozhraní pro editaci odkaz v menu mít nebude. Jediný způsob jak se dostat na editační rozhraní bude prostřednictvím odkazu z rozhraní typu list.

6 Implementace generátoru

Implementaci generátoru jsem se rozhodl provést v C++ toolkitu Qt Creator [19], jehož vývoj od roku 2008 probíhá pod záštitou společnosti Nokia. Zvolil jsem aktuálně poslední verzi, což byla verze Qt Creator 2.0.1 založena na Qt 4.7.0 (32 bit) a implementaci jsem provedl v operačním systému Windows 7. Qt je multiplatformní knihovna, jež podporuje lokalizaci aplikací a poskytuje obrovské množství užitečných funkcí. Za výhodu lze považovat také velmi kvalitní dokumentaci a mnoho dobře komentovaných ukázkových aplikací, které se standardně nainstalují společně s toolkitem pod názvem Qt Demo.

Sám jsem se během implementace několika demo aplikacemi inspiroval. Ukázková aplikace Diagram Scene z kategorie Graphics View mi byla oporou při implementaci grafické scény s diagramem, který poskytuje ucelený pohled na strukturu databáze a slouží k definici vztahů mezi tabulkami. Demo aplikace DOM Bookmarks z kategorie XML mi pak posloužila jako zdroj informací k problematice parsování XML souboru.

Všechny zdrojové soubory se nachází v jednom adresáři a název souboru až na výjimky odpovídá názvu třídy kterou implementuje. Většina tříd je standardně rozdělena na hlavičkový soubor (*.h) obsahující deklaraci třídy a zdrojový soubor (*.cpp) s definicí jejích metod. Ikony použité v uživatelském rozhraní generátoru se nacházejí v adresáři `images` a zdrojové soubory (PHP, HTML/CSS/JS) nutné pro běh IS se nacházejí v adresáři `sources`.

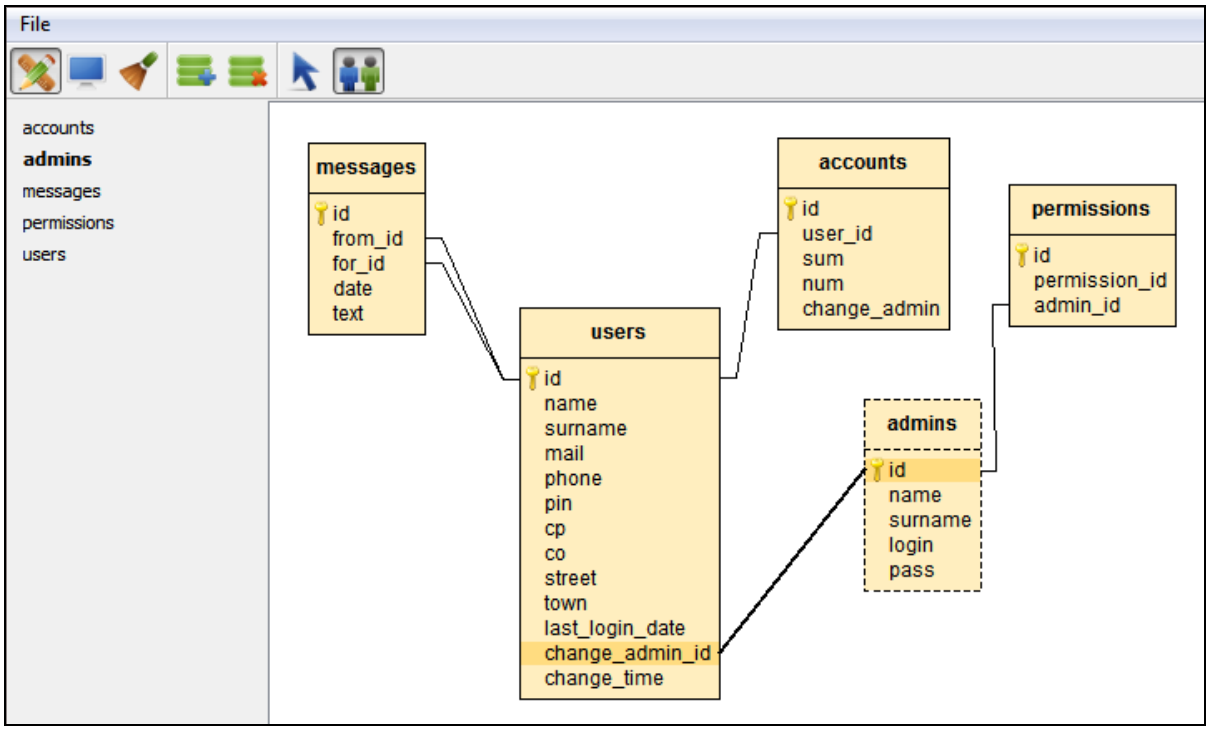
6.1 Finální podoba UI generátoru

Přestože je Qt označováno jako multiplatformní, překlad aplikace se mi podařil pouze v OS Windows 7 (32 bit) a Windows XP SP 3 (32 bit). Ke spuštění aplikace je nutné aby se v adresáři se spustitelným souborem (exe) nacházelo několik přiložených DLL knihoven. Ikony uživatelského rozhraní a zdrojové soubory pro běh IS jsou po překladu součástí spustitelného souboru.

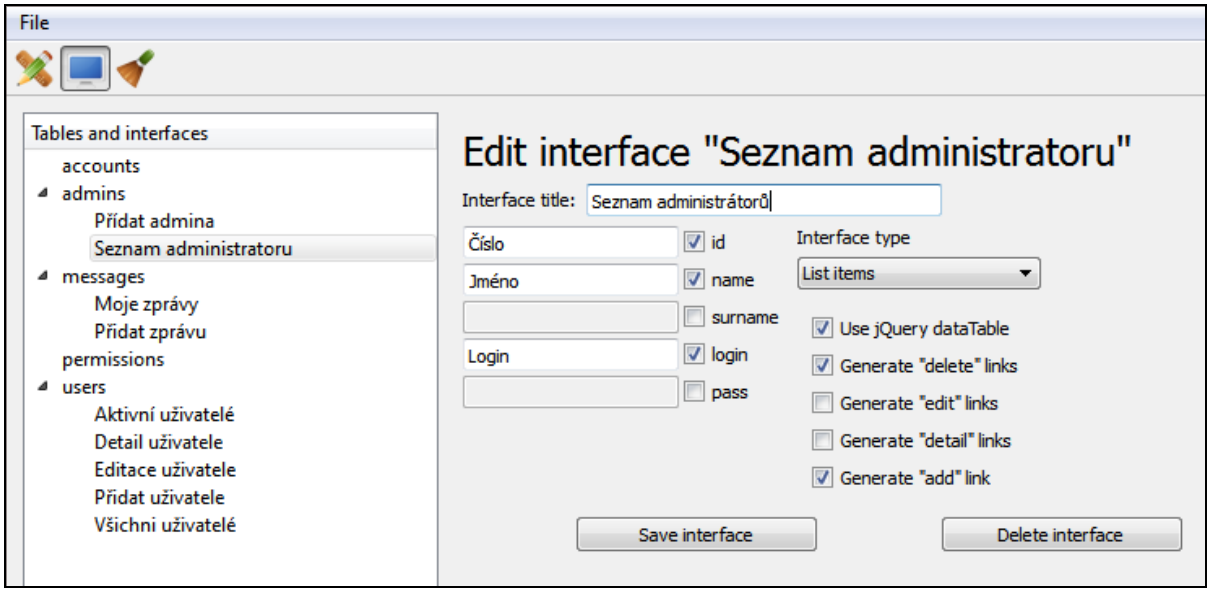
Podoba rozhraní pro definici vztahů mezi tabulkami je vidět na obrázku 9. Na obrázku 10 je podoba formuláře pro definici generovaných rozhraní a na obrázku 11 potom vzhled okna pro definici struktury databázové tabulky.

6.2 Vzhled generovaného IS

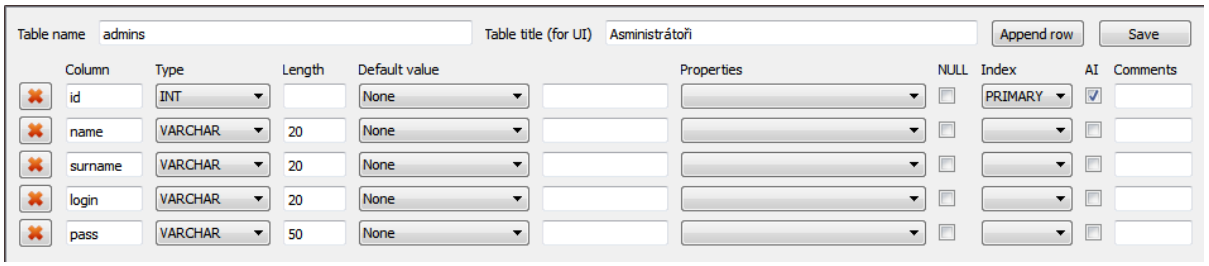
Výstupem generátoru je kompletně funkční informační systém. Díky tomu uživatel může ihned po vygenerování zkontrolovat zda výstup odpovídá jeho požadavkům a případně ho vygenerovat znovu. Systém má poměrně strohý design, který je určen pouze k otestování. Očekává se, že po úspěšném vygenerování si programátor vytvoří design vlastní. Kompletní rozložení stránky s vygenerovaným seznamem administrátorů (použit plugin DataTables pro jQuery) je vidět na obrázku 12. Na obrázku 13 je potom ukázka formuláře pro přidání uživatele.



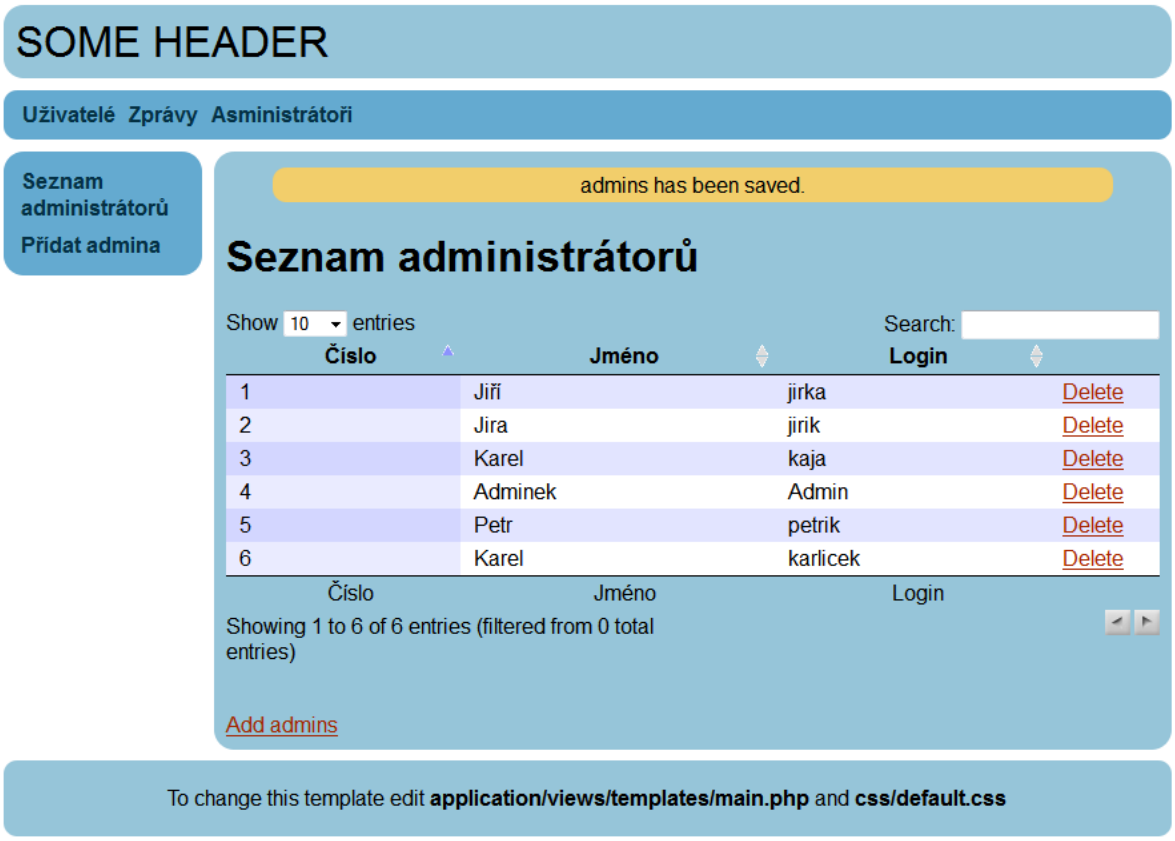
Obrázek 9: Diagram databáze pro definici vztahů



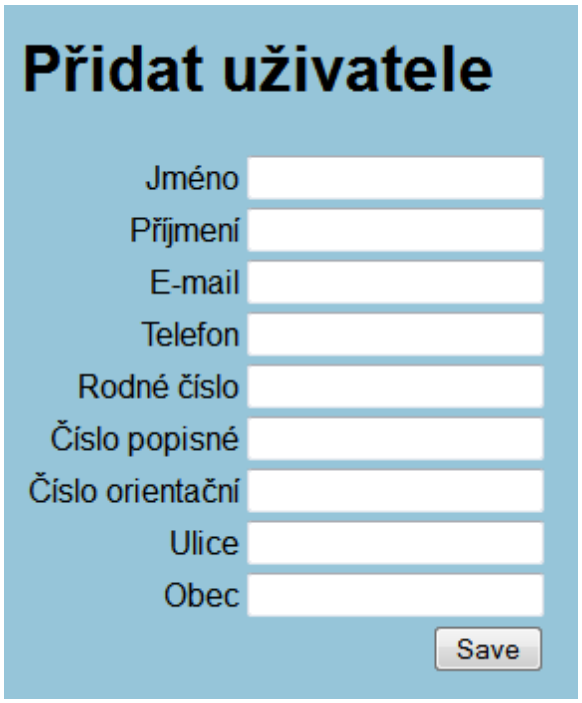
Obrázek 10: Formulář pro definici rozhraní informačního systému



Obrázek 11: Formulář pro definici struktury tabulky



Obrázek 12: Rozložení stránky vygenerovaného IS



Obrázek 13: Rozhraní pro přidání uživatele

7 Závěr

Cílem práce bylo navrhnout a implementovat generátor zdrojového kódu základních operací nad jednou tabulkou. Před vlastním návrhem jsem si ujasnil, jakou roli v implementaci informačních systémů bude generátor hrát. Vlastní návrh jsem poté zaměřil na strukturu generovaného kódu a na organizaci uživatelského rozhraní generátoru. Implementaci generátoru jsem provedl v prostředí Qt Creator pod OS Windows 7. Během ní jsem nenarazil na žádné zásadní problémy a generátor by měl splňovat všechny body zadání.

Výsledná aplikace umožňuje generovat několik typů rozhraní, které pokrývají všechny základní operace nad jednou tabulkou. To je přidávání, editace, mazání a procházení záznamů. Krom vlastních rozhraní pro zvolené tabulky je automaticky generován layout, v němž se nachází menu pro základní navigaci umožňující snadné otestování funkčnosti.

Uživatelské rozhraní generátoru zobrazuje strukturu databáze formou diagramu, díky čemuž lze pohodlně definovat vazby mezi jednotlivými databázovými tabulkami. Definice těchto vazeb je poté součástí generovaných modelů. Pokud se uživatel navíc rozhodne definovat v generátoru kompletní strukturu databázových tabulek, má pak možnost nechat si vygenerovat SQL kód pro jejich vytvoření. Aplikace také nabízí možnost uložení rozdělané práce do souboru ve formátu XML, ze kterého lze později celý projekt načíst a v práci pokračovat.

Během času, který jsem tomuto projektu věnoval mě napadlo mnoho potenciálních rozšíření, která bych rád implementoval v budoucnu, možná i v rámci své diplomové práce. Jedná se především o import struktury DB z formátu SQL, generování přihlašovacího formuláře do IS, možnost generování oprávnění pro jednotlivé akce a dále o možnost definice validačních pravidel u rozhraní umožňujících změny informací uložených v DB.

Literatura

[1] POKORNÝ, J. *Databázové systémy a jejich použití v informačních systémech*. Praha : Academia, 1992. 313 s. ISBN 80-200-0177-8

[2] HOLZSCHLAG, Molly E. *HTML a CSS : jdi do toho*. Praha : Grada, 2006. 264 s. ISBN 80-247-1454-X

[3] FIELDING, R, et al. *Hypertext Transfer Protocol : HTTP/1.1* [online]. 1999 [cit. 2011-05-16]. RFC 2616. Dostupné z WWW: <http://www.faqs.org/rfcs/rfc2616.html>

[4] BRÁZA, Jiří. *PHP 5 : Začínáme programovat*. Praha : Grada, 2005. 244 s. ISBN 80-247-1146-X

[5] BLAHOUT, Michal . *Jemný úvod do ASP* [online]. 2002 [cit. 2011-04-16]. Dostupné z WWW: <http://www12.brinkster.com/mibla/>

[6] *PHP* [online]. 2011 [cit. 2011-04-11]. History of PHP. Dostupné z WWW: <http://cz.php.net/manual/en/history.php.php>

[7] ACHOUR , Mehdi , et al. *PHP* [online]. 1997, 2011-04-15 [cit. 2011-04-16]. PHP Manual. Dostupné z WWW: <http://www.php.net/manual/en/>

[8] PROCHÁZKA, David. *Oracle : průvodce správou, využitím a programováním*. První vydání. Praha : Grada, 2009. 168 s. ISBN 978-80-247-2762-2

[9] SWEAT, J. E. *Architect's Guide to PHP Design Patterns*. USA, 2005 ISBN 0-9735898-2-5

[10] *Propeople : Blogging about web design and development* [online]. 2010 [cit. 2011-04-16]. Top 10 PHP Frameworks You Should Know About. Dostupné z WWW: <http://blog.wearepropeople.com/php-frameworks/>

[11] *Symfony : Web PHP Framework* [online]. [cit. 2011-04-16]. Dostupné z WWW: <http://www.symfony-project.org/>

[12] DANĚK, Petr. *Root.cz* [online]. 2008-09-11 [cit. 2011-04-16]. Velký test PHP frameworků. Dostupné z WWW: <http://www.root.cz/clanky/velky-test-php-frameworku-2008>

[13] *Unofficial Kohana 3 Wiki* [online]. 2010, 2011-03-21 [cit. 2011-04-18]. HMVC in Kohana 3.0. Dostupné z WWW: <http://www.kerkness.ca/wiki/doku.php?id=hmvc_in_kohana>

[14] HELLER, Martin. *InfoWorld* [online]. 2007-01-29 [cit. 2011-04-18]. REST and CRUD: the Impedance Mismatch. Dostupné z WWW: <http://www.infoworld.com/d/developer-world/rest-and-crud-impedance-mismatch-927>

[15] *CakePHP* [online]. 2011 [cit. 2011-05-14]. A Typical CakePHP Request. Dostupné z WWW: <http://book.cakephp.org/view/898/A-Typical-CakePHP-Request>

[16] NOUZA, O. Generování zdrojového kódu : pomůcka, nebo překážka?. In *Sborník příspěvků z XXVI. konference EurOpen.CZ*. První vydání. Plzeň : IMPROMAT CZ, 2005. s. 147. ISSN 80-86583-08-2

[17] *JQuery* [online]. 2010 [cit. 2011-05-08]. Dostupné z WWW: <http://jquery.com/>

[18] JARDINE, Allan. *DataTables* [online]. 2011 [cit. 2011-05-08]. Dostupné z WWW: <http://www.datatables.net/>

[19] *Qt : Cross-platform application and UI framework* [online]. 2011 [cit. 2011-05-12]. Dostupné z WWW: <http://qt.nokia.com/>

Seznam příloh

Příloha 1. DVD se zdrojovými kódy, technickou zprávou a manuálem v elektronické podobě