TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky a mezioborových studií

Studijní program: B2612 – Elektrotechnika a informatika

Studijní obor: 1802R022 – Informatika a logistika

Aplikace PHP – Telefonní seznam na TUL

PHP Application - Phonebook

Bakalářská práce

Autor: **Tomáš Fejfar**

Vedoucí práce: Ing. Petr Kretschmer

Konzultant: Ing. Igor Kopetschke

V Liberci 25. 5. 2009

Originál zadání práce

Prohlášení

Byl jsem seznámen s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 o právu autorském, zejména § 60 (školní dílo).

Beru na vědomí, že TUL má právo na uzavření licenční smlouvy o užití mé bakalářské práce a prohlašuji, že **souhlasím** s případným užitím mé bakalářské práce (prodej, zapůjčení apod.).

Jsem si vědom toho, že užít své bakalářské práce či poskytnout licenci k jejímu využití mohu jen se souhlasem TUL, která má právo ode mne požadovat přiměřený příspěvek na úhradu nákladů, vynaložených univerzitou na vytvoření díla (až do jejich skutečné výše).

Bakalářskou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím bakalářské práce a konzultantem.

Datum:

Podpis:

Poděkování

Na tomto místě bych rád poděkoval svému vedoucímu Ing. Petru Kretschmerovi za pomoc a podporu během vypracovávání mé práce. Také bych rád poděkoval své rodině a přátelům, že mě po celou dobu studia podporovali a své přítelkyni za trpělivost.

Abstrakt

Tato bakalářská práce se zabývá tvorbou telefonního seznamu Technické univerzity v Liberci – PHP aplikace. Existující aplikace telefonního seznamu nedostačuje současným požadavkům. Je složitá na správu pro administrátory a nepřehledná pro uživatele.

Při tvorbě jsem se proto zaměřil na opravu existujících nedostatků. Především jsem se snažil o zlepšení návrhu, aby bylo možné v budoucnu pružně a rychle reagovat na případné změny v požadavcích na aplikaci. Dále jsem se zaměřil na to, aby byl systém přehledný a intuitivní pro uživatele a aby pro administrátory byla jeho správa efektivní.

Mnou vytvořená aplikace telefonního seznamu je postavena na kvalitním frameworku a použití abstraktních tříd umožňuje snadnou rozšiřitelnost. Navíc díky využití principu helperů je chování napříč aplikací konzistentní a jakákoli úprava se ihned promítne do všech částí aplikace. Uživatelům je k dispozici snadné hledání a procházení telefonního seznamu. V administrátorské části nabízí přehledné rozhraní pro úpravu záznamů a administrátory nijak neomezuje v možnostech propojení. Umožňuje propojovat osoby a technologické telefony s libovolným množstvím pracovišť a neomezeným počtem linek. Součástí administračního rozhraní je také modul pro vytváření statických stránek pro uživatelskou část aplikace.

Výsledkem mé práce je aplikace, která splňuje současné požadavky na systém telefonního seznamu Technické univerzity v Liberci. V návrhu byly brány v potaz jak současné nedostatky, tak možné požadavky, které se mohou vyskytnout v budoucnu. Aplikace je tak připravena na další rozšiřování a úpravy.

Abstract

This bachelor thesis presents the development process of the university phonebook – a PHP application. The existing application does not fit the current requirements. For one thing, it is difficult for administrators to maintain and on the other hand it is complicated for users to use.

Therefore I focused on eliminating the existing mistakes in my application. Especially I tried to improve the application design in order to make it possible to react to the prospective changes and adapt the application if necessary. In addition, I focused on making the application well-arranged for users and effective for administrators to maintain.

The application I created is based on quality framework. Using abstract classes allows easy extensibility. Another point is that the usage of the helper principle ensures consistent behaviour across the application because every change of helper class is projected into each part of the application. There is simple and useable interface for searching and browsing the phonebook. The administration part offers well-arranged interface that does not limit the coupling possibilities as it allows the administrator to connect people with many workplaces, many different grades and also more than one phone line. Moreover, the application offers a module for generating static content.

The outcome of my project is an application that fulfils the current demands on the university phonebook system. The concept takes into account the drawbacks of the system used today along with possible future demands. The application is ready for extensions and changes.

Obsah

Seznam symbolů, zkratek a termínů

- **Action** – metoda controlleru, která zapouzdřuje jednu nebo více z jeho činností
- **AJAX** – Asynchronous JavaScript and XML – je způsob jak upravovat části stránky a získávat data, aniž by došlo k načtení celé stránky znovu
- **Business objekt** – Rozšíření DTO – objekt, který v sobě zpracovává data
- **CGI** – Common Gateway Interface – protokol pro propojení externích aplikací s webovým serverem
- **CLI** – Command Line Interface – rozhraní příkazového řádku
- **CMS** – Content Management Systém – systém správy obsahu
- **Controller** – součást vzoru MVC, která zajišťuje zpracování požadavků
- **DAO** – Data Access Object – implementace konkrétního přístupu k datům, se kterým pracuje model, odděluje model od dat a umožňuje ještě snadnější výměnu formátu zdrojových dat
- **DBMS** – Data Base Management Systém – Systém řízení báze dat – rozhraní mezi aplikací a daty uloženými v databázi
- **Document root** – místo na webovém serveru, kam směřují požadavky
- **DTO** – Data Transfer Object – objektová obálka nad daty
- **FastCGI** – vylepšení protokolu CGI především co se týče rychlosti
- **GPL** – GNU General Public License – licence určená pro svobodný software
- **GUI** – Graphical User Interface – grafické rozhraní programu
- **Hash** – výsledek hashovací funkce, která jednosměrně převede velký objem dat do relativně malého řetězce, ukládání hashe zaručuje nemožnost dekódování původních dat
- **HTML** – HyperText Markup Language – značkovací jazyk pro tvorbu internetových stránek
- **HTTP** – HyperText Transfer Protocol – protokol pro přenos HTML stránek internetem
- **IIS** – Internet Information Services – webový server integrovaný v operačním systému Windows
- **JSON** – JavaScript Object Notation – formát pro výměnu dat používaný často v internetových aplikacích
- **Model** – datová vrstva MVC vzoru

- **MPTT** – Modified Preorder Tree Traversal – způsob uložení hierarchických dat v relační databázi
- **MVC** – architektonický vzor Model-View-Controller
- **MVP** – architektonický vzor Model-View-Presenter
- **PHP** – PHP Hypertext Preprocessor – skriptovací jazyk
- **RDBMS** – Relational Data Base Management System – Relační DBMS
- **SQL** – Structured Query Language – dotazovací jazyk pro přístup k databázi
- **TDD** – Test Driven Development – vývoj řízený testy
- **Use-at-will** – filozofie frameworku, která umožňuje používat jednotlivé části aplikace odděleně a nenutí vývojáři jeden hotový model implementace
- **View** – prezentační vrstva MVC vzoru
- **Wireframe** – webová stránka bez funkčnosti, případně papírový model, který se používá při uživatelském testování
- **XML** – Extensible Markup Language – jazyk určený pro vytváření vlastních značkovacích jazyků využívaný také k přenosu dat mezi různými platformami
- **XSS** – Cross Site Scripting – metoda prolomení bezpečnosti webu využitím bezpečnostních chyb a podstrčením vlastního kódu

Úvod

Čím více se zvětšuje množství informací na internetu, tím složitější je informace najít. Kromě vyhledávačů vznikají i proprietární informační systémy určené k hledání konkrétních podmnožin informací. Takovým informačním systémem je i telefonní seznam Technické univerzity v Liberci. Současná podoba aplikace však nedostačuje požadavkům, které jsou na ni kladeny. Tato bakalářská práce se věnuje návrhu a zpracování nové webové aplikace telefonního seznamu.

V teoretické části jsou rozebrány přístupy a softwarové prostředky, které jsem při návrhu a vývoji aplikace použil. Druhá část práce se věnuje návrhu databáze, a také jsem zde ukázal, proč jsem zvolil pro tvorbu aplikace právě Zend Framework a jaké z toho plynou výhody. V poslední části se zabývám konkrétní implementací aplikace a popisuji jednotlivé části informačního systému. Na konkrétních příkladech je také ukázáno, jak jsem využil možností Zend Frameworku.

Cílem práce je vytvořit aplikaci telefonního seznamu, která bude poskytovat uživatelům snadný přístup k informacím o telefonních číslech v rámci Technické univerzity v Liberci a umožní administrátorům systému efektivní správu těchto informací. Výsledná aplikace by měla být připravena na možné rozšíření a činit úpravy co možná nejsnadnější.

1 Teorie

1.1 PHP

Rekurzivní zkratka *PHP* znamená *PHP Hypertext Preprocessor*. Jedná se o interpretovaný jazyk, původně určený pro vývoj dynamických webových stránek procedurálním programováním, avšak v současnosti je jeho součástí také *CLI* (od 4.3.0), objektový model (primitivní objekty od 3.0, vylepšený objektový model od 5.0.0). Nejčastěji je interpret jazyka *PHP* používán jako modul webového serveru (např. Apache) nebo pomocí protokolu *FastCGI* (v případě IIS).

PHP je dynamicky typovaný jazyk, což se projevuje tím, že není třeba definovat typ proměnné při její inicializaci. Na druhou stranu se však během kompilace typy kontrolují a není tak možné provádět nesmyslné operace (např. součet pole a celého čísla). Pokus o takovou operaci skončí fatální chybou. Výhodou tohoto jazyka je strmá křivka učení, velké množství hotových řešení a jednoduchost nasazení změn v kódu. Nevýhody jsou především nižší rychlost v porovnání s kompilovanými jazyky, a to, že vyžaduje disciplínu ze strany programátora. Kvůli netypovosti jazyka může totiž snadno dojít v programu ke vzniku těžko odhalitelné chyby.

PHP je šířeno pod *PHP* licencí, která ale není kompatibilní s *GPL* vzhledem k omezením v použití termínu *PHP*. Zajímavostí je, že jazyk *PHP* nebyl, narozdíl od jiných jazyků (např. *SQL*), nikdy standardizován a jeho struktura je určována jeho implementací.

1.2 Relační databáze

Relační databáze je soubor dat uložený pomocí relačního databázového modelu. Pojem databázová relace vychází z matematického pojmu relace (určitá podmnožina kartézského součinu množin), avšak zavádí navíc strukturu relace. Struktura relaci pojmenovává a určuje, jaké jsou atributy a domény relace (tj. množiny přípustných hodnot). Relační databázový model sdružuje data do relací (nazývaných častěji tabulky), kde každý záznam (řádek) je pevně definovaný seznam hodnot pro jednotlivé atributy relace (často nazývané sloupce). Hodnota atributu musí vždy náležet do domény atributu.

V relační databázi je často nutné zachytit vztahy mezi daty. K tomuto účelu existují podpůrné prostředky jako primární klíče, cizí klíče, normalizace nebo referenční integrita. Všechny tyto části spolu tvoří relační databázi. Relace může znamenat kromě jedné tabulky stejně tak i výsledek libovolného dotazu nad relační databází. Pojmem relační databáze se v současnosti často nepřesně označuje některý z *RDBMS* (např. *MySQL*).

K získávání dat z relační databáze se používá dotazovací jazyk *SQL*. Ten byl od první standardizované verze (*SQL-86*) z roku 1986 postupně rozšiřován a vzniklo několik standardizovaných dialektů. Jmenovitě jde o *SQL-86*, *SQL-89*, *SQL-92*, *SQL:1999*, *SQL:2003*, *SQL:2006* a *SQL:2008*. Krom toho většina *RDBMS* používá svůj vlastní dialekt, který sice vychází z některého z *SQL* dialektů, avšak přidává některé vlastní funkce nebo datové typy, které nejsou standardizované a tím jej činí nekompatibilní s ostatními *RDBMS*. Motivací pro takové jednání je především princip proprietárního uzamčení (angl. *vendor lock-in*). Ten činí zákazníka závislého na službách či produktech jednoho výrobce tím, že vytváří značné náklady při přechod na jiný systém vzhledem k nestandardním součástem produktu.

Pro zachování integrity databáze je nutné v případě smazání záznamu cizího klíče vzít v úvahu také záznamy, které jsou na daném záznamu závislé v ostatních tabulkách. Dialekt SQL:2003 proto definuje akce, které mají být vykonány při mazání (*ON DELETE*) a úpravě (*ON UPDATE*) hodnoty cizího klíče. Je implementováno pět akcí, které mohou být provedeny. Pokud žádnou akci nedefinujeme, použije se *NO ACTION*, tedy žádná akce. V tu chvíli je povinností programátora, aby se postaral o dodržení integrity databáze v aplikaci. Další možností je nastavení *CASCADE*. V případě změny nebo smazání řádku v hlavní tabulce dojde analogicky k úpravě nebo smazání odkazujících řádků v podřízené tabulce Akce *RESTRICT* jakékoli změny řádků v hlavní tabulce zakáže, pokud existují v podřízených tabulkách záznamy, které na tento řádek odkazují. Ještě jsou k dispozici akce *SET DEFAULT* a *SET NULL*. První pro sloupec cizího klíče nastaví výchozí hodnotu. Druhá pak nastaví prázdnou hodnotu.

1.3 MVC architektura

Zkratka je složená z anglických slov *Model*, *View* a *Controller*. Pojem *MVC* poprvé formuloval Trygve Reenskaug. Principem tohoto vzoru architektury je, že *model* poskytuje a zpracovává data, *controller* řeší aplikační logiku a *view* data zobrazuje uživateli [1]. Nespornou výhodou oddělení jednotlivých složek je to, že je poměrně snadné měnit *model* a *view*, což umožňuje například výstup upravit pro sázecí program *TeX* místo *HTML* nebo výstup dat v *JSON* pro zpracování *AJAXem*. Stejně tak je možné načítat data místo z databáze např. z *XML* souboru. Pro návrhové vzory neexistuje jasná norma a proto v případě *MVC* existuje spor o tom, zda by měl v*iew* komunikovat přímo s *Modelem*. Obecně se této možnosti využívá, avšak na webu je častěji k vidění situace, kdy veškeré požadavky na model procházejí přes c*ontroller*. Je to dáno především tím, že v *HTML* je třeba při každém požadavku znovu generovat celou stránku a je tedy nutné provést další operace, které do v*iew* nepatří. Toto se částečně mění s odesíláním požadavků pomocí *JavaScriptu* (*AJAX*), kdy lze měnit i jednotlivé části stránky. I přesto jsem se v aplikaci snažil pracovat s *modelem* pouze z prostředí *controlleru*. Tento postup jsem zvolil z čistě praktického hlediska, protože pokud se aplikační logika provádí na jednom místě, je kód snadněji pochopitelný.

1.4 Návrhové vzory

Návrhový vzor popisuje problém, se kterým se opakovaně setkáváme, a popisuje princip řešení problému tak, abychom mohli vytvořené řešení opakovaně použít, aniž bychom dělali stejnou věc vícekrát.

Původně byl tento pojem použit v architektuře Christopherem Alexanderem jako řešení pro „protichůdné síly“, které ovlivňovaly řešení problému. Například jak udělat místnost dostatečně slunečnou a přitom zabránit tomu, aby se přehřívala v letních měsících. Pojem *návrhový vzor (design pattern)* ve spojení s informačními technologiemi se objevil v roce 1987 na konferenci OOPSLA. Kent Beck a Ward Cunningham ho použili jako pomůcku pro programátory začínající s objektově orientovaným programováním (OOP).

Častěji bývají však návrhové vzory spojovány s takzvaným *Gang of Four* (GoF), který tvoří Erich Gamma, Richarda Helm, Ralph Johnson a John Vlissides. Ti jako

první definovali několik základních návrhových vzorů [2], které se používají až do současnosti.

Příkladem návrhového vzoru je **Singleton**, který řeší, jak zajistit existenci právě jedné instance objektu a vytváří jednotné rozhraní pro přístup k ní [2]. Principem je, že třída má *statickou privátní proměnnou* obsahující instanci, *privátní konstruktor* a *statickou metodu*, která získá existující instanci buď z proměnné, nebo vytvoří novou instanci pomocí volání *privátního konstruktoru*. Kód 1 je ukázkou zdrojového kódu vzoru **Singleton** tak, jak je implementován v jazyce *PHP*.

Kód 1Singleton v PHP
Zdroj: vlastní zpracování

```php
<?php
/**
 * Singleton class
 */
class Singleton
{
	/**
	 * @var Singleton
	 */
	private static $_instance = null;
	private function __construct()
	{}
	public static function getInstance()
	{
		if(null === self::$_instance){
			self::$_instance = new self();
		}
		return self::$_instance;
	}
}
```

Příkladem použití návrhového vzoru **Singleton** v aplikaci telefonního seznamu je implementace třídy **Zend_Auth**. Použitím **Singletonu** je v této třídě zajištěno, že informace o přihlášení uživatele bude přístupná a unikátní v celé aplikaci.

1.5 Zend Framework

Zend Framework je open-source webový framework zpracovaný v *PHP*. Používá filozofii *use-at-will*. Není nutné používat jednu předem danou strukturu aplikace a je na vývojáři, aby zvolil nejvhodnější cestu pro konkrétní projekt. Jednotlivé komponenty frameworku jsou na sobě nezávislé (*loosely coupled*) a je možné je používat samostatně. Framework je vyvíjen pod *New BSD licencí*, což zaručuje bezpečné použití pro komerční projekty.

Vývoj Zend Frameworku je zaštítěn firmou Zend, jež stojí za vývojem samotného jádra *PHP* – Zend Engine. Na návrhu jeho struktury se podílejí vývojáři, kteří dříve používali frameworky postavené na jiných jazycích, z nichž do Zend Frameworku přenášejí nejlepší myšlenky (např. .NET, Ruby on Rails…)

Technologickými partnery vývoje jsou takové firmy jako IBM, Adobe, Google nebo Microsoft. Celý kód je z více jak 80% pokryt unit testy, které zajišťují, že nebude docházet k regresím a umožňují využití metody *TDD*. Další nespornou výhodou jsou velmi krátké vývojové cykly, takže nové verze jsou vydávány s velmi malým zpožděním a tak jsou chyby rychle opravovány. Manuál Zend Frameworku také obsahuje pro každou z komponent speciální oddíl, který se věnuje nutným změnám ve zpětné kompatibilitě (především jde o opravy v oblasti bezpečnosti) a tomu, jak je překonat při migraci na vyšší verzi.

1.5.1 Helpery v Zend Frameworku

Zend Framework obsahuje velké množství tříd v mnoha souborech. S načítáním souborů a instancováním tříd je spojena nezanedbatelná režie. Aby byla snížena náročnost aplikace, je v Zend Frameworku použit koncept *helperů*. *Helpery* jsou pomocné třídy, které zapouzdřují určitou funkcionalitu, která může a nemusí být použita ve všech částech aplikace. Snížení náročnosti je dosaženo načítáním a instancováním ve chvíli, kdy je třída *helperu* poprvé použita a její instance se uchovává napříč celou aplikací. Zend Framework zavádí dva druhy *helperů* - *view helpery* a *action helpery*.

View helpery se používají ve *view* objektu. Jsou určeny především pro generování určitého obsahu. Z konkrétních *view helperů* existujících v Zend Frameworku budu jmenovat například **Zend_View_Helper_Partial** pro vykreslení dílčího view scriptu, **Zend_View_Helper_Url** pro sestavování *URL* nebo **Zend_View_Helper_HeadStyle** pro nastavení *CSS* stylů v hlavičce *HTML* dokumentu.

Action helpery jsou využívány ve třídě controlleru a slouží k různým změnám stavu aplikace. Z konkrétních implementací bych jmenoval například **Zend_Controller_Action_Helper_ViewRenderer** pro instancování a automatické nastavení *view*, **Zend_Controller_Action_Helper_FlashMessenger** pro ukládání zpráv mezi jednotlivými požadavky nebo **Zend_Controller_Action_Helper_Json**

určený pro odesílání dat ve formátu *JSON* pro zpracování *AJAXem*. Koncept *helperů* jsem použil také v aplikaci telefonního seznamu. Konkrétní implementaci zmíním dále.

2 Návrh aplikace

Před samotným návrhem jsem si stanovil několik požadavků, které musí aplikace splňovat:

- musí v ní být možné v budoucnu snadno dělat potřebné úpravy
- musí být možné prohledávat telefonní seznam nebo ho procházet pomocí jednoduchého a přehledného uživatelského rozhraní
- aplikace by neměla omezovat možnosti přiřazení pracovišť, funkcí nebo linek, pokud to není nutné

Pro pojmenování tříd, metod, proměnných, tabulek a sloupců v databázi jsem zvolil anglické pojmenování. Samotný program je postaven na frameworku s názvy tříd, metod a proměnných v anglickém jazyce, a anglické názvy tabulek do tohoto schématu lépe zapadají. Výjimku tvoří názvy, které se překládají do *URL*.

Při tvorbě jsem se snažil zachovávat jmennou konvenci, která je nastavena v Zend Frameworku. U databáze je primární klíč označen *id* a cizí klíč označen *nazevtabulky_id*. Názvy tabulek jsou v množném čísle. Tabulky propojení mají příponu *bindings*.

2.1 Platforma pro vývoj aplikace

Jako platformu pro vývoj aplikace jsem zvolil programovací jazyk PHP (konkrétně verze 5) s navázáním na relační databázi *MySQL*, jednak protože s vývojem v tomto prostředí mám nejvíce zkušeností a také proto, že jak *PHP* tak *MySQL* jsou open-source. *PHP 5* oproti starší verzi 4 nabízí lépe propracovaný objektový model s podporou dědičnosti, rozhraní a různými úrovněmi viditelnosti proměnných třídy, který při vývoji využiji.

Protože telefonní seznam je poměrně složitá aplikace a do budoucna existuje možnost jejího dalšího rozvoje, rozhodl jsem se, že ho postavím na Zend Frameworku. Framework je nadstavba nad samotným jazykem, která řeší časté problémy, se kterými se programátor může setkat. Kromě toho také pomáhá v aplikaci udržet jednotný styl a v případě Zend Frameworku navíc programátora svou strukturou motivuje

k dodržování základních pravidel, ať se jedná o *OOP*, používání návrhových vzorů nebo *coding standards*.

2.2 Výběr frameworku

Chtěl bych nyní podrobně rozebrat, proč jsem pro aplikaci telefonního seznamu zvolil právě Zend Framework. Do užšího výběru možných frameworků jsem vybral Symfony, Zend Framework, Nette Framework a CodeIgniter. Při výběru jsem hodnotil několik kritérií, která by měl splňovat. Jedním ze základních požadavků bylo, aby byl jednoduše použitelný pro výstavbu aplikace na architektuře *MVC*, případně dalších podobných (např. *MVP* v případě Nette). Tento požadavek splňovaly všechny čtyři.

Dalším požadavkem byla existující dokumentace a její kvalita. Protože pokud bude třeba aplikaci dále rozšiřovat, bude se s ní muset vývojář nutně seznámit. Dokumentace bývá často pověstnou Achillovou patou jakéhokoli programu a platí to i o frameworku. V tomto ohledu je na tom nejlépe Zend Framework, který za skvělou dokumentaci vděčí především podpoře firmy Zend, která na tuto nepopulární činnost dodává lidské zdroje a také potřebné finance. Zend Framework nabízí rozsáhlou dokumentaci, která každou z komponent rozebírá v samostatné sekci a na příkladech demonstruje její fungování. Vysvětluje také, jak správně třídu podědit a rozšířit. Dokumentace Symfony je na nízké úrovni. Obsahuje pouze návod jak začít a několik návodů na často používané činnosti (vytváření formulářů nebo nákupního košíku). CodeIgniter je na tom o něco lépe. Strukturou je dokumentace více podobná té u Zend Frameworku avšak je méně rozsáhlá. Dokumentace Nette je zpracovaná velmi přehledně a stručně a je rozdělená podle tříd a tematických celků. Obsahuje nejčastější použití komponent, ale chybí další informace pro hlubší pochopení fungování jednotlivých tříd. Všechny frameworky mi přišly nedostatečně dokumentované a v tomto ohledu nevyhovující. Výjimku tvoří Zend Framework.

Za vývojem frameworku by neměl podle mého názoru stát pouze jediný vývojář. Toto kriterium vyloučilo z výběru Nette, které vyvíjí přední český *PHP* vývojář *David Grudl*, který stojí mimo jiné za velmi úspěšným formátovačem textu zvaným *Texy!* nebo databázovou knihovnou *Dibi*. Vývoj závislý na jediném člověku je nevýhodný jak z hlediska dalšího vývoje samotné aplikace, tak z pohledu bezpečnosti. Pokud pan Grudl z nějakého důvodu s vývojem přestane, může velmi snadno nastat situace,

kdy bezpečnostní mezery ve frameworku nebudou opravovány nebo nebude zaručena stejná kvalita jako doposud.

Nezanedbatelným faktorem jsou také možnosti samotného frameworku. V tomto ohledu exceluje Zend Framework, který nabízí velké množství adaptérů, jež lze libovolně měnit. Například se jedná o databázové adaptéry pro *MySQL*, *Oracle*, *MSSQL* nebo o autentizační adaptéry pro *HTTP* autentizaci, autentizaci proti databázi nebo *LDAP*. Taktéž nabízí možnost napsat vlastní adaptér rozšířením abstraktní třídy adaptéru. Ostatní frameworky v tomto ohledu silně zaostávají. I když ve všech je možné části frameworku rozšířit nebo přepsat, tak to není tak jednoduché a dobře dokumentované jako v případě Zend Frameworku a není k dispozici takové množství různých hotových adaptérů.

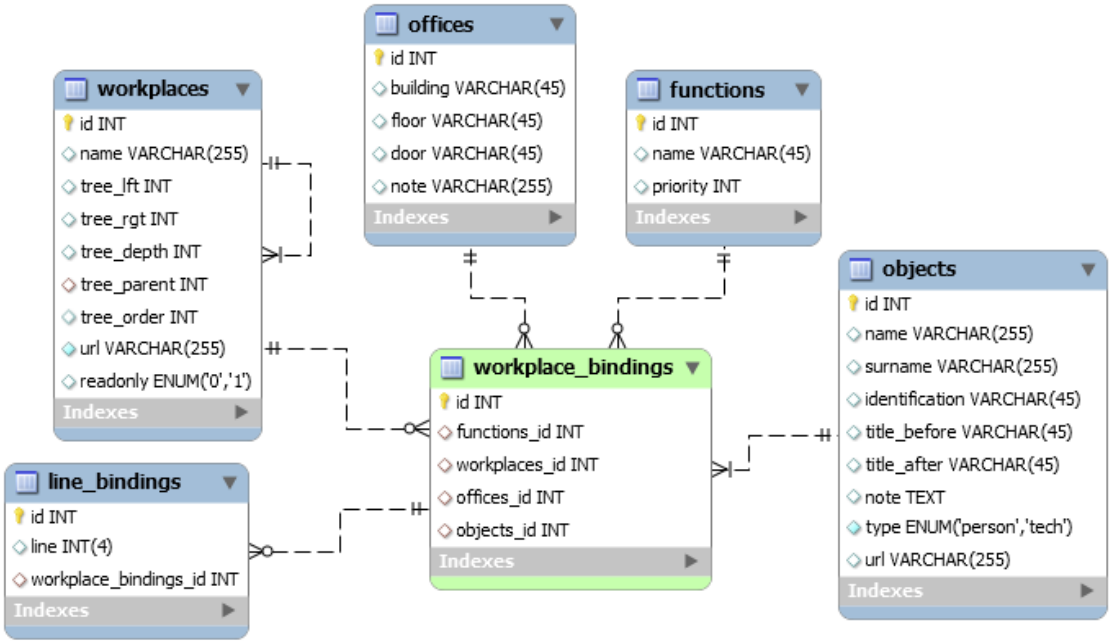
2.3 Návrh databáze telefonního seznamu

Při návrhu databáze jsem stál před problémem, zda postavit aplikaci nad existující tabulkou zaměstnanců na serveru **Moon** nebo zda navrhnout vlastní strukturu. Existující databáze na serveru **Moon** má určité nedostatky, které by mi vývoj značně komplikovaly. Jednou z nevýhod je velké množství nepotřebných dat v této tabulce – tabulka obsahuje kromě údajů o zaměstnancích také údaje o čipových kartách, fotografie a další data, která zbytečně zpomalují práci s tabulkou. Tato data by bylo velmi vhodné uložit do vlastní tabulky provázané 1:1 s tabulkou zaměstnanců, ze které by byla načítána pouze v případě potřeby. Sloupec typu *blob* (fotografie) má variabilní délku ve velkém rozsahu a to způsobuje vzhledem ke způsobu uložení na pevném disku delší přístupové doby a zpomalení práce s tabulkou. Tento problém by ale bylo možné v aplikaci telefonního seznamu vyřešit použitím databázového pohledu. Tabulka zaměstnanců je typu *MyISAM*, který sice podporuje definice integritních omezení cizích klíčů při vytváření, ale podle manuálu je po zpracování neukládá ani nepoužívá [3]. Závažným prohřeškem je uvádění některých redundantních dat (název pracoviště) přímo v tabulce. Další problém by nastal se zápisem do této tabulky. Ne snad z důvodů technických, ale spíše z důvodů bezpečnostních. Nad tabulkou zaměstnanců běží některé kritické aplikace univerzity, a proto by mohly úpravy v podobě integritních omezení a přidávání sloupců způsobit jejich nefunkčnost. Největší (a navíc zbytečná) komplikace je pro mne primární klíč této tabulky. Ten je tvořen *hashem* rodného čísla a osobním číslem pracovníka. Přitom je pracovník jasně definován svým osobním

číslem. Logickým řešením by bylo mít celočíselný primární klíč místo osobního čísla a *hash* rodného čísla zavést jako jeden z atributů s unikátním klíčem. Ale jsem si vědom toho, že tento stav vychází pravděpodobně ze systému univerzitní agendy STAG, kde je často využíváno jak osobního čísla, tak rodného čísla k hledání záznamů. Přesto toto řešení dle mého názoru není příliš šťastné. Po zvážení všech těchto hledisek jsem se rozhodl pro vlastní návrh.

Obr. 1 zachycuje *ERD diagram* hlavní části telefonního seznamu. Pro přehlednost jsou vynechány tabulky, které obsahují data, která přímo nesouvisí s telefonním seznamem. Všechny tabulky jsou typu *InnoDB*.

Obr. 1 ERD diagram
Zdroj: vlastní zpracování



Základem samotného seznamu jsou tabulky *line_bindings* (propojení linek) a *workspace_bindings* (propojení pracovišť). Tabulka propojení linek přiřazuje vazbou M:N čísla linek k odpovídajícím záznamům v tabulce propojení pracovišť. Tabulka *workplace_bindings* propojuje technologické telefony a zaměstnance (objekty) vazbou M:N s pracovišti (*workplaces*). Jsou v ní uloženy také další cizí klíče pro propojení s tabulkou funkcí (*functions*) a tabulkou kanceláří (*offices*).

Vztahy mezi tabulkami jsou řešeny pomocí cizích klíčů v jednotlivých tabulkách. Integrita databáze je zajištěna nastavením integritních omezení. Referenční integrita je uplatněna pro všechny cizí klíče a jsou nastaveny akce *ON DELETE* a *ON UPDATE*.

Propojením objektů a pracovišť relací M:N je umožněno přiřadit jednomu pracovníkovi více různých pracovišť a k nim různé funkce. Protože se čísla linek přiřazují k propojení pracovišť, je možné, aby jeden člověk měl zaregistrováno více linek pro více funkcí – např. jedno číslo bude mít pracovník zavedené jako zaměstnanec do své kanceláře a jiné číslo bude mít propojené k funkci správce školní sítě do dohledového centra nebo do laboratoře. Dokonce je možné mít zavedené několik čísel k jedné funkci, i když tento případ nastane v praxi asi velmi zřídka.

2.3.1 Objekty (objects)

Tabulka objektů obsahuje jak záznamy osob, tak záznamy technologických telefonů. To, zda se jedná o člověka nebo technologický telefon, řídí atribut *type* typu *ENUM* a rozdíly ve zpracování výsledných dat z databáze se řeší až na úrovni programu. Použití oddělených tabulek by významně snižovalo čitelnost a pochopitelnost kódu a přineslo by pouze zanedbatelný přínos.

I když je primárním klíčem záznamu sloupec *id*, má každý objekt přiřazený atribut *identification*, který je unikátní a je použit mimo jiné v URL detailu záznamu. Pro osoby je ekvivalentní osobnímu číslu. Zavedl jsem ho proto, aby detail záznamu byl na předvídatelné URL obsahující osobní číslo zaměstnance, což umožňuje např. snadnější odkazování z jiných univerzitních systémů. Ke každému objektu je možné přiřadit libovolný obsah do atributu *note* a adresu URL s podobnějšími informacemi do atributu *url*.

2.3.2 Funkce (functions)

Tabulka funkcí obsahuje názvy jednotlivých funkcí. Každá funkce má atribut *priority*, který se používá při řazení v telefonním seznamu. Díky tomu nenastane situace, kdy by byl např. záznam děkana v telefonním seznamu zobrazen až po záznamu účetní, což v některých případech ve starém systému nastávalo.

2.3.3 Kanceláře (offices)

Informace o kanceláři jsou narozdíl od původního systému, kde byly uloženy jako textový řetězec, uloženy ve zvláštní tabulce. Tato tabulka obsahuje kromě sloupců s informacemi o budově, patře a číslu dveří také sloupec *note*, který umožňuje dále upřesnit informace o kanceláři. To se hodí například v případě podkrovních kanceláří na budově A, kde je budova, patro i číslo dveří společné pro velké množství zaměstnanců.

I když se napříč aplikací používá zápis kanceláře budova-patro-dveře, z databáze se tyto informace přenáší jednotlivě a jsou zformátovány až na výstupu pomocí *view helperu*, o kterém se zmíním v další části práce.

2.3.4 Pracoviště (workplaces)

Při návrhu tabulky pracovišť a jejich struktury jsem postupoval tak, abych se vyvaroval chyb, které se existují v původním systému – jde především o špatné řazení položek a nevhodný způsob uložení stromové struktury pomocí řetězců.

Tabulka pracovišť obsahuje kromě názvu pracoviště a atributů souvisejících s hierarchií pracovišť (označené prefixem *tree*) také *url*, které se využívá při procházení telefonního seznamu. To by mělo mít pozitivní vliv na indexovatelnost seznamu vyhledavači a vyústit v lepší nalezitelnost telefonních čísel zadáním dotazu do vyhledavače.

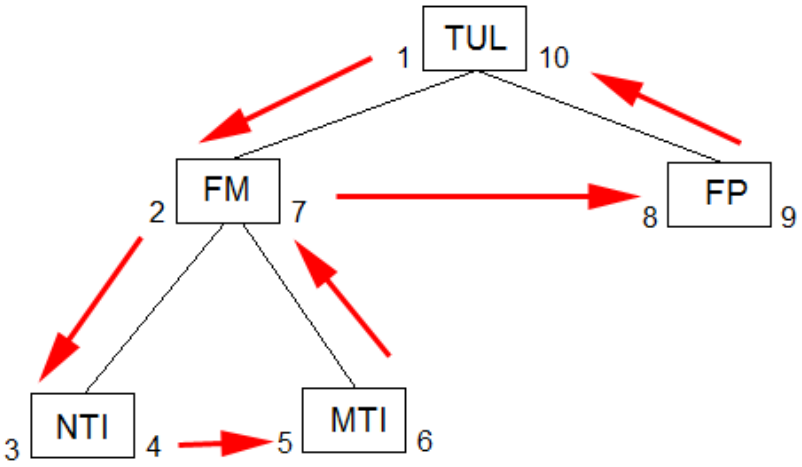
Zachycení stromové organizační struktury v databázi

Při ukládání hierarchických informací do tabulek nastává vždy problém, neboť relační databáze jsou nevhodné pro ukládání hierarchických dat. V praxi se pro uložení stromových struktur v relačních databázích používá několik metod. Nejjednodušším principem je **ukládání předka v hierarchii**, tento způsob ovšem vede na rekurzivní funkci. Protože v databázi je použití rekurzivní funkce přinejmenším problematické, používá se rekurze na straně aplikace a to vede k několikanásobným dotazům do databáze, což je značně neefektivní. Další možností je použití tzv. **ploché tabulky**, kdy každý záznam má uloženou svoji pozici v hierarchii, podle které se seřadí a také hloubku zanoření. Funguje velmi dobře při výpisu, ale přináší komplikace při složitějších operacích (např. hledání cesty). Třetí možností je **řazení podle řetězcového klíče**, také zvané **vyjmenování cesty**, kdy má záznam řetězcový klíč –

např. *A*, a ten se vždy připojuje s oddělovačem před klíče jeho potomků (A-A, A-B,…). Data se následně získávají pomocí dotazu s konstrukcí *LIKE*, což toto řešení činí opět velice neefektivním a nepřehledným. Poslední možností je **metoda vnořených množin** uváděná Joe Celkem [4], nazývaná také **MPTT**, kterou jsem s úpravami použil v aplikaci telefonního seznamu.

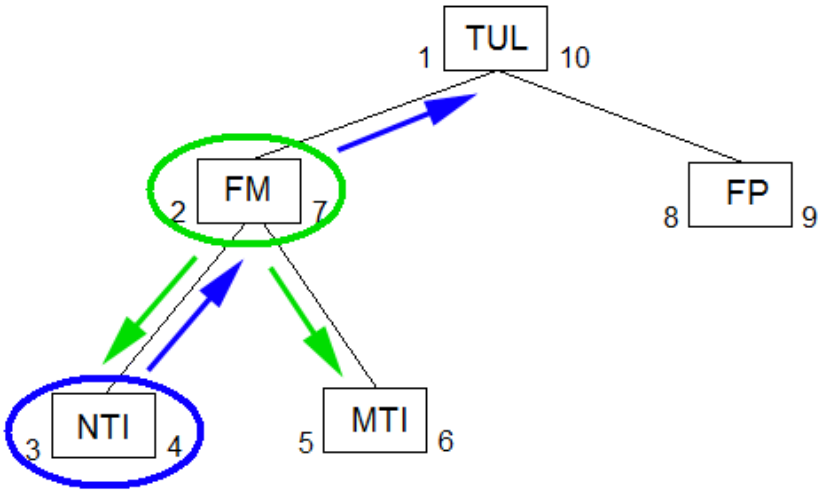
Používám svojí vlastní, částečně upravenou implementaci tohoto postupu. Struktura je brána jako strom. Každý vrchol stromu je, stejně jako v metodě Joe Celka, označen dvěma čísly – **left** a **right** (*tree_lft* a *tree_rgt*). Obr. 2 ukazuje názorně princip, jakým se strom čísluje. Obě hodnoty čísluji společným počítadlem. Hodnota **left** previzitou, hodnota **right** postvizitou. Tento způsob umožňuje vybírat velmi efektivně podstromy a cesty. Jeho nevýhodou je složitá manipulace a neefektivnost při úpravách.

Obr. 2 Princip číslování pro MPTT
zdroj: vlastní zpracování



Tento problém jsem vyřešil doplněním dalších atributů, a to **id nadřazeného vrcholu** a **pozice** (*tree_parent* a *tree_order*). Při práci se stromem (např. pro přesun podstromu) se používají tyto a následně se pomocí rekurzivní funkce přepočítají hodnoty *tree_lft* a *tree_rgt*. Díky tomu je spojena rychlost přístupu pomocí **MPTT** a jednoduchost úprav pomocí rekurzivních funkcí. Vzhledem k tomu, že k úpravám ve struktuře dochází velmi zřídka, není delší čas přepočítání stromu po úpravě velkým problémem. Přepočítávací rekurzivní funkce má značné výkonnostní rezervy, a pokud by bylo nutné, je možné jí optimalizovat použitím nerekurzivní funkce a přepočítáváním hodnot *tree_lft* a *tree_rgt* přímo.

Obr. 3 Hledání cesty v grafu a získávání podstromů
zdroj: vlastní zpracování



Obr. 3 ilustruje hledání cesty v grafu (modře). Z obrázku je patrné, že vrcholy stromu na cestě od vrcholu NTI ke kořenu stromu (tedy situace typická např. pro drobečkovou navigaci) mají hodnoty *tree_lft* menší než 3 a současně *tree_rgt* větší než 4. Všechny vrcholy na cestě včetně aktuálního tedy získáme jednoduchým *SQL* dotazem (Kód 2).

Kód 2 SQL dotaz pro hledání cesty v MPTT
zdroj: vlastní zpracování

```
SELECT
  title
FROM
  workplaces
WHERE
  (tree_lft =< 3) AND (tree_rgt => 4)
ORDER BY
  tree_lft
```

Podobně snadno lze získat podstrom. Pokud chceme získat všechny vrcholy náležící pod FM stačí na to opět jednoduchý dotaz do databáze, protože jak je z obrázku patrné, vrcholy podstromu mají hodnotu *tree_lft* větší než je hodnota kořene podstromu a současně hodnotu *tree_rgt* menší než hodnota kořene podstromu (Kód 3).

Kód 3 SQL dotaz pro získání podstromu
zdroj: vlastní zpracování

```
SELECT
  title
FROM
  workplaces
WHERE
  (tree_lft > 2) AND (tree_rgt < 7)
ORDER BY
  tree_lft
```

2.3.5 Propojení linek (line_bindings)

K propojení linek na propojení pracovišť, byť je realizované vazbou M:N, jsem nezvolil propojovací tabulku, protože k lince se nevážou žádné další údaje. Samotná hodnota linky je klíčem a její uložení ve zvláštní tabulce by bylo nadbytečné.

2.3.6 Tabulka administrátorů (admins)

Tabulka administrátorů obsahuje jméno a heslo používané při přihlášení. Za zmínku stojí způsob uložení hesla, které je zahashováno algoritmem *SHA-1* spolu s náhodnou hodnotou uloženou v atributu *salt*, a díky tomu není možné odhalit např. stejná hesla dvou uživatelů, a to ani v případě, že má útočník přímý přístup k hodnotám uloženým v databázi.

2.3.7 Tabulka statického obsahu (cms)

Tato tabulka obsahuje statické stránky, které se objeví v uživatelské části v menu. U každé stránky je možné definovat titulek, obsah, pozici v menu a další. Pokud se stránka tvoří průběžně, je možné ji nejprve nastavit jako neaktivní, aby se nezobrazila v menu a publikovat jí teprve po dokončení.

3 Popis aplikace

3.1 Struktura

Aplikace je postavena nad Zend Frameworkem verze 1.7.8 a kopíruje běžně používanou strukturu modulárních aplikací v Zend Frameworku. Obsahuje dva moduly – *default* a *admin*. Modul *default* zpracovává požadavky uživatelské části – hledání a procházení telefonního seznamu. Modul *admin* je pak zodpovědný za administrační rozhraní pro úpravy v databázi.

Aplikace je umístěna ve třech základních složkách – *public*, *application* a *library*. Složka *public* obsahuje soubory, které mají být přístupné *HTTP* požadavkem. Kromě souboru *index.php*, který směruje požadavky na jádro aplikace, se jedná také o soubory stylů, soubory *JavaScriptu* a obrázky. K souboru *index.php* lze přistoupit buď přímo, nebo (což doporučuji) je připraven soubor .htaccess, který nastaví přesměrování tak, aby se *index.php* neobjevoval v *URL*. Všechny ostatní soubory aplikace jsou umístěny mimo *dokument root* webu a díky tomu jsou chráněny před nepovoleným přístupem, a to i v případě, že dojde k poškození interpretu *PHP* kódu či jinému selhání a zdrojové kódy aplikace by se zobrazovaly v nezkompilované podobě. Složka *library* obsahuje zdrojové soubory tříd použitých v aplikaci (kromě modelů). Struktura složek kopíruje jmenné konvence Zend Frameworku. Jádro aplikace je umístěno ve složce *application*. Složka *models* je sice společná pro oba dva moduly, avšak samotné třídy modelů jsou oddělené. Pro administrační část aplikace mají prefix *Admin* a dědí funkcionalitu modelů modulu *default*. Tím je dosaženo konzistence v práci s daty.

3.2 Nastavení

Nastavení aplikace je řízeno konfiguračním souborem *application.ini* umístěným ve složce *configs*. Výhodou konfiguračního souboru je čitelnost a možnost editace i bez znalosti programovacího jazyka. V konfiguračním souboru jsou uloženy například přístupové údaje k databázi nebo nastavení stránkování.

Konfigurační soubor je rozdělen na sekce, které se načítají podle *URL* serveru, na kterém aplikace momentálně běží. To umožňuje mít např. v lokální sekci uvedeny údaje pro testovací databázi a současně v produkční sekci údaje pro produkční databázi.

Jejich přepínání pak probíhá automaticky a není nutné tyto hodnoty nastavovat při každém přesunu na testovací server a zpět. To ulehčuje vývoj a předchází chybám.

3.3 Využití výhod Zend Frameworku

Zend Framework umožňuje programátorovi plně využít možností OOP implementovaných v PHP5 pomocí množství předpřipravených abstraktních tříd a rozhraní. V aplikaci jsem této možnosti využil na řadě míst.

Nejlépe lze existující možnosti ilustrovat na příkladu CRUD controlleru. Třída **Tul_Controller_Crud** obsahuje funkcionalitu určenou k vytváření, procházení, úpravě a mazání položek v databázi. Třída dědí funkcionalitu **Tul_Controller_Abstract** (pomocné metody pro rychlejší přístup k přesměrovávání, session zprávám, atp.) a tím pádem také **Zend_Controller_Action** (tzn. samotná funkčnost controlleru). Třídy poděděné od této třídy díky návrhu podle principu znovupoužitelnosti obsahují jen základní nastavení a samotná logika je umístěna v rodičovské třídě.

Kód 4 Třída pro práci se seznamem funkcí
zdroj: vlastní zpracování

```
<?php
class Admin_FunkceController extends Tul_Controller_Crud
{
	protected $_url = '/admin/funkce/';
	protected $_modelClass = 'Admin_Functions';
	protected $_formClass = 'Tul_Form_Admin_Functions';
	protected $_filterFormClass = 'Tul_Form_Admin_FilterFunctions';
	protected $_viewKey = 'funkce';
	protected $_title = 'Funkce';
}
```

Kód 4 představuje celý kód, který je třeba v controlleru implementovat. Z kódu je patrné, že je třeba ještě doprogramovat třídy modelu a formulářů. Zvažoval jsem možnost implementovat pevně dané rozhraní pro model, avšak nakonec jsem se rozhodl, že v reálném nasazení by nebyl přínos tak velký, aby vyvážil zbytečné zpomalení a načítání další třídy. Vzhledem k tomu, že veškerý aplikační kód je v rodičovském controlleru, bylo by nasazení rozhraní a kontrola jeho implementace triviální.

Využití rodičovského controlleru neznamená, že by neexistovala možnost funkcionalitu dále rozšiřovat. Třída **Tul_Controller_Crud** implementuje sadu metod **_beforeAdd()**, **_beforeUpdate()**, **_beforeDelete()**, které jsou zavolány před

provedením jednotlivých akcí v controlleru, a jejich přepsáním ve třídě samotného controlleru lze snadno modifikovat data před jejich zpracováním. Na stejném principu fungují i analogicky pojmenované metody **_afterAdd()**, atd. Příkladem jejich využití může být controller pro správu administrátorů.

Kód 5 konkrétní implementace funkce _beforeAdd()
zdroj: vlastní zpracování

```
protected function _beforeAdd(&$data)
{
	$data['salt'] = md5(time());
	$data['password'] = sha1($data['salt'] . $data['password']);
	unset($data['password2']);
}
```

Kód 5 ukazuje konkrétní implementaci funkce **_beforeAdd()** k vygenerování zashashovaného hesla. Těsně před přidáním jsou data předána referencí funkci **_beforeAdd()**, kde jsou upravena. Je odstraněno pole kontroly hesla, je vygenerováno pole salt a to je následně využito při generování hashe hesla. Samotné přidání do databáze je implementováno v rodičovské třídě.

3.4 Využití helperů

Jak bylo již zmíněno v teorii, Zend Framework poskytuje systém rozšiřování funkcionality pomocí helperů načítaných podle potřeby. V aplikaci telefonního seznamu jsem helpery použil na několika místech.

3.4.1 View helpery Office, Line a FullName

Tyto helpery zapouzdřují zobrazení kanceláří, linek a celého jména včetně titulů. Jsou používány napříč aplikací a zajišťují, že formát zobrazení těchto dat bude vždy konzistentní. Helper *Office* například přijímá informace o kanceláři (budova, patro, číslo dveří a poznámka) buď jako objekt nebo jednotlivě a vrací je ve formátu např. B1-1-12 (poznámka). Ošetřen je i stav, kdy není vyplněna poznámka. Podobně funguje i helper *FullName*, který formátuje celé jméno, včetně titulů. Helper *Line* se stará o zobrazení čísla linky.

Kód 6 Upravený view helper Line
zdroj: vlastní zpracování

```
<?php
class Tul_View_Helper_Line extends Zend_View_Helper_Abstract
{
      public function line($lineNumber)
      {
        if(strlen($lineNumber) == 0){
          return Zend_Registry::get('Zend_Translate')->_('Není');
        } else if(mb_strlen($lineNumber) == 9){
          //máme 765123456
          return '+420' . $lineNumber;
        }
        return str_pad((string)((int)$lineNumber),4,'0',STR_PAD_LEFT);
      }
}
```

Kód 6 ukazuje, jak lze snadno upravit helper *Line* tak, aby bral v potaz také mobilní telefonní čísla. Možnost úprav formátování na jednom místě činí systém velmi variabilním a umožňuje to jednoduché úpravy.

3.4.2 Action helper UrlPersistence

Tento helper zapouzdřuje funkcionalitu spojenou s přesměrováním a udržováním návratového URL v session. Voláním metody *storeLastUrl()* se uloží současné *URL* do session a voláním metody *gotoLastUrl()* dojde k přesměrování na uložené *URL*. Tento způsob umožňuje komfortní přihlašování přes libovolné *URL*. Administrátor má například v záložkách uloženou adresu pro přidání nové osoby do seznamu. Pro přístup je však nutné přihlášení. Pokud přihlášení již vypršelo, tak dojde k uložení aktuální adresy a přesměrování na přihlašovací formulář. Po přihlášení je přesměrován na původní adresu.

3.5 Třídy modelů

Třídy modelů jsou v aplikaci poděděny od abstraktní třídy **Tul_Table_Abstract**. Tato třída rozšiřuje základní třídu **Zend_Db_Table_Abstract** o často používané funkce a definuje základní rozhraní.

Při každém vytvoření instance tabulky jsou z databáze získána metadata tabulky (pomocí dotazu *DESCRIBE TABLE*). Jak načítání metadat, tak samotné vytvoření instance je poměrně náročné na prostředky a opakovat tento postup při každém použití je značně neefektivní. Proto jsem vytvořil statickou třídu **Loader**, která se stará

o načítání všech modelů a uchovává jejich instance v sobě. Navíc je při načítání metadat využívána *cache* a tím je náročnost ještě snížena.

Většina metod modelu vrací výsledek typu **Tul_Paginator**, což je třída pro stránkování, která obaluje objekt *SQL* dotazu a podle nastavených parametrů v něm provádí úpravy. Díky tomu je načítání stránkovaných dat efektivní, protože k dotazu do databáze dochází až ve chvíli, kdy jsou data požadována, a navíc se načítá jen jejich nutná podmnožina. V ostatních případech vrací metoda modelu objekt řádku (**stdClass**) nebo pole objektů řádků. Výjimku tvoří metoda *getPairs*, která vrací asociativní pole, kde klíčem je hodnota prvního sloupce dotazu a hodnotou je hodnota druhého sloupce dotazu. Tento způsob je využíván např. pro plnění hodnot do formulářového prvku *select*.

3.6 Bezpečnost aplikace

Kromě *helperů* má použití Zend Frameworku nezanedbatelný přínos i co se bezpečnosti týče. Formuláře jsou opatřeny základní ochranou proti *XSS*. Hodnoty ve formulářích jsou při zpětném vyplňování ošetřeny funkcí htmlspecialchars() tak, aby nebylo možné podstrčením určité *URL* vyvolat neočekávanou akci (Kód 7).

Kód 7 Škodlivý XSS kód
Zdroj: http://cs.wikipedia.org/wiki/Cross-site_scripting

```
<script>alert('Toto je úspěšný XSS útok.');</script>
```

Vyhledáním takového výrazu by došlo v případě neošetřeného výstupu k jeho vykonání a bylo by zobrazeno okno se zprávou.

Díky důslednému ošetřování vstupních dat do databáze je web odolný i proti napadení pomocí SQL injection (Kód 8).

Kód 8 Škodlivý kód SQL injection
Zdroj: Vlastní zpracování

```
1234' OR '1'='1
```

Tento kód představuje výraz zadaný do pole pro vyhledání linky a využívá toho, že ukončí vyhledávaný výraz a doplní podmínku, která vrátí všechny záznamy. V aplikaci telefonního seznamu je však text před zpracováním ošetřen a nepředstavuje problém pro bezpečnost aplikace.

V administrační části nejsou vkládaná data ošetřována proti *XSS*, protože autentizovaný administrátor je důvěryhodnou osobou a pokud by využil např. možnosti zapsat *HTML* kód do názvu pracoviště, tak by tak nečinil proto, aby ohrozil bezpečnost aplikace nebo jejích uživatelů.

Pro ještě větší bezpečnost by bylo vhodné vytvořit zvláštní účet pro přístup k databázi s omezenými právy, aby prolomení bezpečnostní zranitelnosti nepřesahovalo svým rozsahem rámec aplikace.

Bezpečnost aplikace je i přes všechny tyto mechanizmy náchylná k chybě lidského faktoru. Proto je třeba dodržovat základní bezpečnostní návyky, jako je odhlašování po skončení práce nebo silné heslo.

3.7 Vícejazyčné prostředí

Aplikace je připravena na práci ve vícejazyčném prostředí. Překlad je řešen pomocí **GNU Gettext**. Je uložen v binárních souborech pro rychlejší zpracování. Binární soubory mohou být vytvářeny např. volně dostupným programem **PoEdit** ze zdrojových souborů s příponou *PO*. Texty, které jsou použity v aplikaci a přitom nejsou přeloženy, se vygenerují spolu s *URL* jako záznam do souboru *untranslated.po*. K samotnému překladu je v aplikaci použita třída **Zend_Translate** s napojením na *gettext* adaptér. Adaptér **Zend_Translate_Adapter_Gettext** má jistá omezení oproti standardní implementaci gettextu, ale poskytuje komfortní integraci například s formuláři nebo třídami pro formátování a filtrování dat. Lokalizována je pouze uživatelská část aplikace.

3.8 Cache

V případě potřeby urychlení lze jednoduše části aplikace ukládat do vyrovnávací paměti. Místem, kde by bylo možné ušetřit systémové prostředky a zrychlit generování, je například organizační struktura univerzity v postranním panelu zobrazená při procházení telefonního seznamu. Stejně tak by se mohlo uplatnit cachování tříd pro formuláře, jejichž generování je také spojeno s velkou režií. Vzhledem k jednoduchému a transparentnímu způsobu ukládání do cache pomocí třídy **Zend_Cache** není problém tuto funkcionalitu do aplikace v budoucnu dodat.

Nashromážděná data vyrovnávací paměti aplikace obsahuje složka *cache*. Ukládají se metadata databázových tabulek, což výrazně urychluje inicializaci modelů. Dále je také cachován vygenerovaný *HTML* kód na výstupu z *Texy!*, protože zpracování *Texy!* syntaxe je poměrně pomalé.

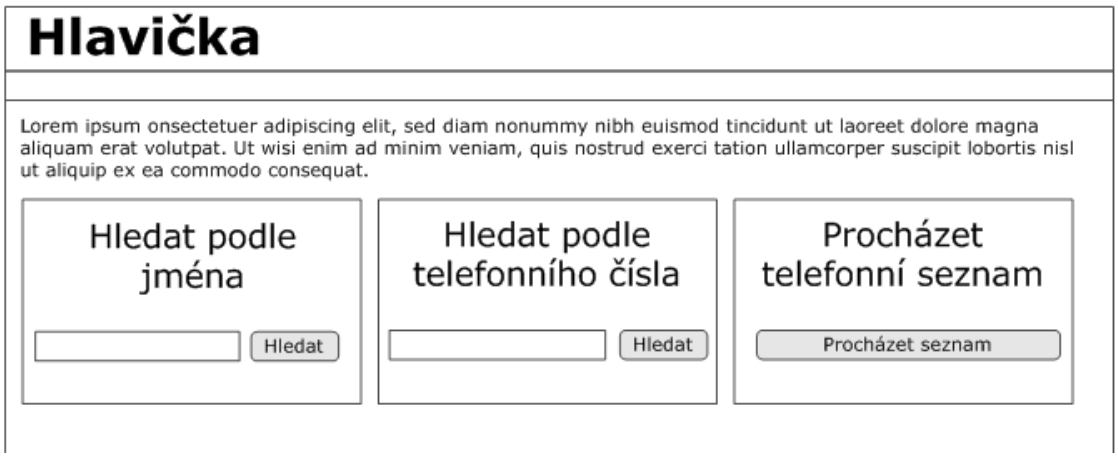
3.9 Uživatelská část

Uživatelskou část jsem navrhl především s ohledem na snadné použití. Analyzoval jsem současný stav aplikace telefonního seznamu, abych se nedopustil stejných chyb.

3.9.1 Optimalizace a uživatelské testování hlavní stránky

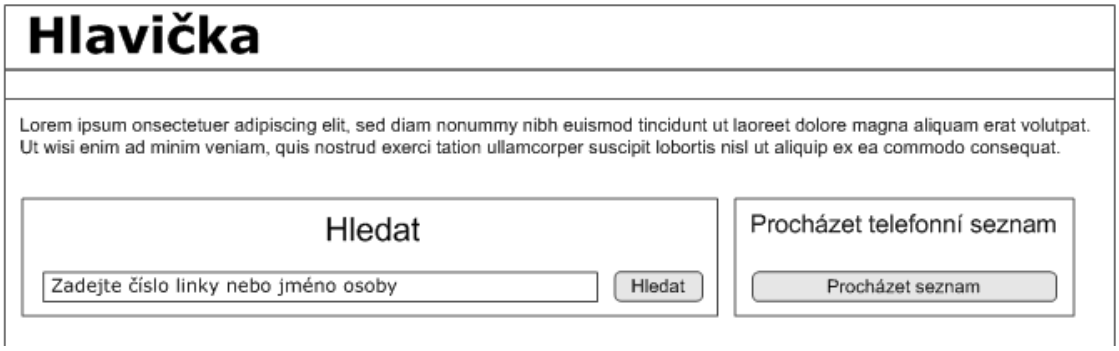
V současném systému telefonního seznamu je na úvodní stránce vizualizace informačního centra, která nemá ani informační hodnotu a ani není přínosem pro uživatele. Současný stav úvodní stránky je překážkou jednoduchého používání aplikace. Primárním cílem uživatele telefonního seznamu je nalézt telefonní číslo zaměstnance, a proto jsem se rozhodl umístit vyhledávání hned na úvodní stránku. Připravil jsem si několik jednoduchých *HTML prototypů*, s kterými jsem provedl krátké uživatelské testování na malém vzorku uživatelů. Samotné testování jsem doplnil také konzultacemi.

Obr. 4 Původní návrh hlavní stránky
Zdroj: Vlastní zpracování



Na Obr. 4 je původní návrh hlavní stránky. Během testování se ukázalo, že i přes velmi výrazné nadpisy mají lidé tendenci používat pouze první políčko. A to jak pro hledání podle jména, tak pro hledání podle čísla linky. Proto jsem původní návrh upravil tak, aby používal pouze jedno políčko.

Obr. 5 Upravená verze hlavní stránky
Zdroj: Vlastní zpracování



Na Obr. 5 je upravená verze hlavní stránky s jediným vyhledávacím políčkem. Druhé testování by již nemělo smysl, protože uživatelé byli ovlivněni prvním testováním a mým komentářem k němu. Ale i přesto valná většina uživatelů hodnotila druhý návrh jako na první pohled přehlednější a jednodušší.

3.9.2 Optimalizace hledání

Při testování současného systému telefonního seznamu jsem narazil na zajímavou věc – pokud v hledání použijeme pouze části jmen, tak se zdá, že nedochází k rozdělení vyhledávací fráze na jednotlivé části. Zaznamenal jsem to například pro frázi „ond nov“, na kterou se nezobrazí Ondřej Novák, byť hledané frázi odpovídá. I když je taková vyhledávací fráze nepravděpodobná, nelze spoléhat na to, že taková situace nenastane.

Při hledání proto v mé aplikaci postupuji vždy tak, že nejprve řetězec rozdělím na jednotlivé části (v místě mezer). Ve výsledku se následně zobrazí pouze ty záznamy, které obsahují všechny části vyhledávaného řetězce. Hodnoty se porovnávají bez ohledu na velikost písmen.

3.9.3 Výsledky vyhledávání

Výsledky vyhledávání se zobrazují v tabulce. Pro lepší přehlednost jsou za sebou následující řádky automaticky odlišeny různými barvami (tzv. *zebra stripping*). Přínos co do rychlosti nebo přesnosti hledání v tabulce je sice zanedbatelný, ale i přesto většina uživatelů tento způsob preferuje nebo jim alespoň nevadí [5]. Výsledky vyhledávání i veškeré výpisy linek se řadí podle priority funkce, dále pak podle čísla linky. Pokud to dává smysl (vyhledávání podle jména, procházení), zobrazují se ve výsledcích

i osoby bez přiřazené linky. Díky existenci položky *url* u záznamu zaměstnanců se může telefonní seznam snadno stát pohodlným rozcestníkem při hledání informací o zaměstnancích univerzity. Uživatel vyhledá profil zaměstnance a ihned má k dispozici odkaz na jeho domovskou stránku na webu jeho domácího ústavu.

K ulehčení vyhledávání slouží také funkce, která přesměruje uživatele z výsledků hledání podle čísla linky přímo na detail propojeného objektu v případě, že je výsledkem jediný záznam. O přesměrování je uživatel informován pomocí výrazného informačního boxu v horní části stránky. Pro případné vynucení nepřesměrování existuje parametr *no-redirect*, který zajistí, že nedojde k přesměrování ani v případě jediného výsledku.

3.10 Administrátorská část

V administrátorské části jsem se snažil administrátory co nejméně omezit v možnostech propojování linek a zaměstnanců. A spoléhám v tomto na jejich úsudek. Vycházel jsem z toho, že předchozí systém byl tak komplikovaný a omezující, že bylo snadnější provádět úpravy přímo v databázi. Je možné propojit neomezený počet linek k jedné vazbě na pracoviště a vazby na pracoviště pro jednoho zaměstnance taktéž nejsou omezené.

3.10.1 CMS

Součástí administrátorské sekce je také *CMS* určený pro správu statických stránek. Příkladem je stránka se seznamem tísňových volání. Umožňuje přímo do menu umisťovat také odkazy mimo aplikaci telefonního seznamu. Zde opět systém administrátory nijak neomezuje a spoléhá na jejich úsudek.

Je možné vytvářet prakticky libovolný statický obsah. K editaci se používá systém *Texy!*, který zaručuje, že výstupem bude vždy validní *XHTML* kód. Všechny stránky v CMS se objeví v menu v uživatelské části. Pokud chceme odkázat mimo telefonní seznam, stačí vyplnit URL včetně protokolu (http://, sftp:// nebo třeba call://) a systém automaticky rozpozná odkaz mimo web a podle toho s ním naloží. To je využito například pro odkaz na web univerzity.

3.10.2 Správa administrátorů

V aplikaci existují tři různé skupiny práv – **guest**, **admin** a **superadmin**. Guest nemá přístup do administrátorské sekce a je jím kdokoli kdo přijde na web. K získání práv **admin** nebo **superadmin** je třeba se přihlásit do administrace a liší se pouze v tom, že uživatel s právy **superadmin** může přidávat další administrátorské účty. Může přidávat administrátory a nastavovat jim hesla. Do budoucna se počítá s nasazením systému *Shibboleth*, takže tato část bude muset být přepracována, protože autentifikace bude poskytována systémem *Shibboleth* a tato část aplikace bude pouze spravovat práva administrátorů.

3.11 Možnosti urychlení

Použití Zend Frameworku s sebou přináší, mimo spousty výhod, i daň v podobě zvýšené náročnosti na prostředky. Díky architektuře *use-at-will* jsou jednotlivé třídy umístěny ve zvláštních souborech a jejich oddělená funkcionalita je zajištěna voláním **require_once** před použitím kterékoli jiné třídy. Zde dochází k velkému zpomalení, neboť funkce **require_once** musí projít při každém volání seznam všech vložených souborů a zkontrolovat ho. Urychlení by bylo možno docílit smazáním těchto volání v kódu frameworku a spolehnout se na třídu **Zend_Autoloader**.

3.12 API

Aplikace neobsahuje žádné konkrétní *API*, protože není zcela jasné, které údaje telefonního seznamu by v něm měly být takto veřejně poskytovány. Místo toho však v aplikaci existuje **ApiController**, který podle *API* klíčů deleguje různé třídy vyřízením požadavků. V tabulce *api_gateway* jsou uloženy *API* klíče a k nim příslušné třídy. Hotová je ukázková třída **Test**, která podle osobního čísla vrátí jméno a *URL* zaměstnance v systému ve formátu *JSON*. Je tak možné k různým *API* klíčům propojit různá *API*. Podobně snadno lze zrealizovat například třídu, která bude vracet seznam zaměstnanců v určité kanceláři ve formátu *XML*.

4 Závěr

Z velké části jsem splnil cíle, které jsem si vytyčil v úvodu práce, i když jsem v některých oblastech slevil ze svých původních plánů. Pokud bych upřednostnil jednoduché rozšíření, úpravy a dodržení principů *OOP* za každou cenu před rychlostí samostatné aplikace, implementoval bych model pomocí *DAO*, aby byla změna zdroje dat ještě jednodušší. Data bych předával pomocí *DTO* nebo *business objektů*, což by více odpovídalo objektovému modelu. Dle mého názoru by ale takový způsob přes své nesporné výhody způsobil zásadní zpomalení aplikace, což by s sebou neslo horší použitelnost pro uživatele.

Aplikace neobsahuje importní skripty pro přenos dat z aktuální databáze, protože to sahá za rámec této bakalářské práce. Navíc bohužel musím konstatovat, že během práce na aplikaci jsem se setkával s poměrně velkou neochotou, která se také částečně promítla do nekompatibility mé aplikace telefonního seznamu s existujícími daty. A proto bude na odpovědných pracovnících, aby před nasazením aplikace do ostrého provozu přesunuli současná data do nové databáze.

Nová aplikace telefonního seznamu TUL je dle mého názoru mnohem lépe použitelná a přístupná než verze stará. Ulehčuje uživatelům hledání informací a administrátorům agendu spojenou s úpravami telefonního seznamu. Doufám, že bude nasazena v praxi a bude sloužit svému účelu.

5 Bibliografie

[1] **REENSKAUG, Trygve.** *Models-views-controllers.* [Online] 10. Prosinec 1979. [Citace: 10. Květen 2009.] http://heim.ifi.uio.no/trygver/1979/mvc-2/1979-12-MVC.pdf.

[2] **GAMMA, Erich, a další.** *Design Patterns: Elements of Reusable Object-Oriented Software.* Boston : Addison-Wesley, 1995. 0201633612.

[3] **MySQL AB.** 1.7.5.4 Foreign Keys:. *MySQL 5.0 Reference Manual.* [Online] [Citace: 10. Květen 2009.] http://dev.mysql.com/doc/refman/5.0/en/ansi-diff-foreign-keys.html.

[4] **CELKO, Joe.** *SQL for Smarties: Advanced SQL Programming.* Third Edition. San Francisco : Morgan Kaufmann, 2005. stránky 631-637. 0123693799.

[5] **ENDERS, Jessica.** Zebra Striping: Does it Really Help? *A List Apart.* [Online] 6. Květen 2008. [Citace: 13. Květen 2009.] http://www.alistapart.com/articles/zebrastripingdoesithelp.