TECHNICKÁ UNIVERZITA V LIBERCI

Fakulta mechatroniky, informatiky a mezioborových studií

Studijní program: B2612 – Elektrotechnika a informatika

Studijní obor: 1802R022 – Informatika a logistika Název

Webová aplikace pro řízení praxí na střední škole

Web Application for Management of Student´s Practice at the Secondary Technical School

Bakalářská práce

Autor: **Jan Fojtík**

Vedoucí práce: Ing. Petr Kretschmer

V Liberci 1. 5. 2013

Zadání

Prohlášení

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

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu TUL.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

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

Datum 10. 5. 2013

Podpis:

Poděkování

Na tomto místě bych rád poděkoval vedoucímu práce, panu Petru Kretschmerovi, za ochotu spolupracovat na této práci a za hodnotné připomínky v průběhu jejího tvoření. Veliký dík si zaslouží i má rodina a kolegové, kteří mě po celou dobu nejen podporovali a motivovali k odevzdání co nejlepšího výkonu, ale taktéž mi dopřáli potřebný klid a tvůrčí prostředí.

Abstrakt

Bakalářská práce se zabývá problematikou agendy pro řízení praxí na střední škole. Poukazuje na neefektivní využití dostupných informačních technologií ve stávajícím způsobu zpracování agendy a snaží se navrhnout systém nový, který je reprezentován jednoduchou webovou aplikací. Ta je založena na skriptovacím jazyku PHP a rychle se rozvíjejícím českém frameworku Nette. Výsledkem je zjednodušení agendy při sběru dat od žáků, následné přiřazení praxe žáka a vygenerování smluv s příslušnou firmou.

Klíčová slova

webová aplikace, PHP, Nette framework, MySQL, řízení praxí, střední škola

Abstract

This thesis deals with the agenda for management practice in high school. It refers to the inefficient use of available information technology in the existing method of processing data and attempts to propose a new system which is represented by a simple web application. It is based on the PHP scripting language and fast-paced Nette Framework, Czech. The result should be a simplification agenda in collecting data from students, the subsequent assignment of the student practice and generate contracts with the respective company.

Key words

web aplication, PHP, Nette framework, MySQL, management practice, high school

Obsah

Seznam použitých zkratek

AJAX – Asynchronous JavaScript and XML

CSRF – Cross-Site Request Forgery

CSS – Cascading Style Sheet

CSV – Comma-separated values file

FTP – File Transfer Protocol

GNU-GPL – GNU General Public License

HTML – HyperText MarkUp Language

HTTP – HyperText Transfer Protocol

IE – Internet Explorer

IMAP – Internet Message Access Protocol

JSON – JavaScript Object Notation (JavaScriptový objektový zápis)

MD5 – Message Digest 5 Algorithm

MVC – Model-View-Controller

MVP – Model-View-Presenter

ODBC – Open Database Connectivity

PDF – Portable Document Format

PHP – Hypertext Preprocessor

POP3 Post Office Protocol

SMTP – Simple Mail Transfer Protocol

SNMP – Simple Network Management Protocol

SQL – Structured Query Language

URL – Uniform Resourse Locator

XML – eXtensible markup language

XSS – Cross-site scripting

Úvod

Princip realizování a zpracování agendy povinných studentských praxí na Střední průmyslové škole Česká Lípa, Havlíčkova 426, přísp.org. záležel doposud pouze na manažerských a informačně-komunikačních dovednostech a znalostech pověřeného pedagogického zaměstnance, v poslední době spíše na více zaměstnancích a veškeré využití informační a komunikační technologie začínalo a zároveň končilo u funkce hromadné korespondence kancelářských aplikací. Dále během několikaletého vývoje došlo ke změnám jak v legislativě, v čerpání finančních prostředků, tak i v přístupu školy k žákům. To vedlo k vyprofilování dalších požadavků a úkonů, které více zainteresovávaly další zaměstnance, nově však i žáky či firemní partnery. V této agendě rázem vzniklo hned několik nových pracovních rolí. Nestačil už pouhý koordinátor přidělování praxí žákům, ale bylo potřeba řešit další úkony jako generování a podepisování smluv s firemními partnery, propojit evidenci se školním informačním systémem, vytvářet různé statistiky, aktualizovat informace od žáků či firemních partnerů a plno dalších.

Všechny tyto aspekty volaly po větší synchronizaci a propojení zdrojových dat a především k jednotnému přístupu k nim. Různé varianty souborů vytvořených v tabulkových či textových editorech se staly přežitkem a příčinami různých komplikací a chyb. Nový rozměr do všeho vnesl až rok 2012, v kterém došlo na Střední průmyslové škole k pořízení centrálního databázového serveru, a tudíž k možnosti data centralizovat, sdílet a aktualizovat. S tímto přišel i nápad zpřehlednit a strukturovat nepřehledný systém agendy povinných studentských praxí a vytvořit pro ni jednoduchou webovou aplikaci.

1. Analýza problému

1.1 Stávající systém

Stávající systém přidělování praxí pracoval na bázi:

a) jednoho sešitu tabulkového procesoru o několika listech, obsahujícího nestrukturovaná data všech žáků a firem, a v těchto listech bylo sporadicky použito jen několik elementárních funkcí, jako SUMA pro součty položek a PODMÍNĚNÉ FORMÁTOVÁNÍ pro zvýraznění hodnoty položek,

b) několika variant textových dokumentů pro vytisknutí smluv s firmami a pro informování žáků a jejich rodičů,

c) data od žáků byla shromažďována pomocí papírového formuláře, který musel poté určený pracovník přepisovat do elektronické podoby,

d) pracná a neprůkazná kontrola stanovených termínů různých fází realizace,

e) přiřazování žáka k praxi vykonával pracovník dle svého úsudku, kdy se snažil zahrnout alespoň bydliště žáka a možnost dojíždění na dané pracoviště, vše však bez zpětné vazby a kontroly aktuálních dat.

1.2 Nový systém (SPS-Praxis)

Pro nový systém prezentovaný a spravovaný databázovou webovou aplikací jsem realizování a zpracování agendy povinných studentských praxí rozdělil do několika fází vyplývajících z harmonogramu školního roku a z potřeby získávat žákovská data:

1.2.1 Fáze 1 – Příprava a sběr dat (září až říjen)

- vytvoření aktuálních seznamů žáků 2. a 3. ročníků filtrováním a exportem do formátu .CSV z informačního systému Bakalář o těchto položkách:
    - jméno
    - příjmení
    - datum narození
    - bydliště

    - email
    - třída
    - obor
- vytvoření seznamu (.CSV) firem a partnerů pro praxi o položkách:
    - přesný název firmy podle obchodního rejstříku
    - obor praxe (strojírenství x informační a komunikační technologie)
    - kontaktní osoba (titul, jméno, příjmení)
    - telefon
    - email
    - ulice a číslo popisné
    - směrovací číslo
    - město
    - osoba, na kterou bude psána smlouva (titul, jméno, příjmení)
    - požadavky na vybavení, oblečení a bezpečnost v provozu

1.2.2 Fáze 2 – Vytvoření databáze (listopad)

- nahrání dat do databáze s možností úprav jednotlivých položek
- určení termínů probíhajících praxí (2 týdny na přelomu dubna a května)

1.2.3 Fáze 3 – Vytvoření žákovských účtů (listopad)

- vytvoření uživatelských jmen a hesel žákům ze seznamu
- kontrola funkčnosti přihlášení k žákovským účtům a kontrola osobních údajů
- rozeslání emailu žákům s informacemi o povinnosti praxi absolvovat (případná nedocházka znamená neklasifikaci z předmětu atd.)

1.2.4 Fáze 4 – Vyplnění formuláře soukromé praxe (prosinec)

- žák může vyplnit a odeslat formulář s daty o firmě (dle seznamu CSV), kterou si domluvil individuálně, vytiskne si ho pro nutnost potvrzení od uvedené firmy
- po předložení firmou podepsaného formuláře odpovědnému pracovníkovi je praxe schválena
- po uplynutí příslušného termínu je tato možnost již blokována
- generování smluv podle firmy a oboru s uvedením počtu žáků a příslušných termínů
- statistika obsazených žáků

1.2.5 Fáze 5 – Přiřazení praxe neobsazeným žákům (březen)

- určený pracovník přiřadí ze seznamu neobsazeným žákům partnera pro praxi, kterou během ledna a února nasmlouval manažer školy dle výstupů dosavadní obsazenosti žáků na praxi (zohlední se obor a bydliště žáka)
- rozeslání emailu žákům s informacemi o přidělené praxi

1.2.6 Fáze 6 – Generování smluv s firmami (duben)

- generování smluv podle firmy a oboru s uvedením počtu žáků a příslušných termínů
- statistika obsazených žáků a termínů

1.2.7 Fáze 7 – Potvrzení o absolvování praxe (červen)

- potvrzení o splnění praxe, hodnocení žáka
- výstupní statistiky
- generování poděkování zúčastněným firmám

1.3 Zvolené platformy

Pro daný problém jsem zvolil v současnosti velmi oblíbený databázový systém MySQL a programovací skriptovací jazyk PHP částečně reprezentovaný českým, rychle se rozvíjejícím frameworkem Nette.

1.3.1 Databázový systém MySQL

Tento multiplatformní databázový systém byl vytvořen švédskou firmou MySQL AB, nyní je vlastněn společností Sun Microsystems, což je dceřiná společnost Oracle Corporation. Komunikace s databází probíhá pomocí jazyka SQL. Podobně jako u ostatních SQL databází se jedná o dialekt tohoto jazyka s některými rozšířeními. Oblíbenost získal pro svou velmi snadnou implementaci, rychlost a především díky možnosti volného šíření pod licencí GNU General Public License. Nejčastěji se využívá v kombinaci s PHP a Apache jako základní software webového serveru. *(1)*

1.3.2 Skriptovací jazyk PHP

PHP je pravděpodobně nejrozšířenějším skriptovacím jazykem pro vytváření webových aplikací. Verze PHP 4 a PHP 5 jsou šířeny pod licencí PHP License v3.01, copyright the PHP Group, což je Open Source licence, certifikována Open Source

Initiative. Oblíbeným se stal především díky jednoduchosti použití a bohaté zásobě funkcí. Interpret PHP skriptu je např. možné volat pomocí příkazového řádku, dotazovacích metod HTTP nebo pomocí webových služeb.

Syntaxe jazyka je inspirována několika hned několika programovacími jazyky jako Perl, C, Java. PHP podporuje mnoho knihoven pro zpracování textu, grafiky, práci se soubory, dále také má implementovány přístup k většině databázových systémů (MySQL, ODBC, Oracle, PostgreSQL, MSSQL) a podporuje celé řady internetových protokolů (HTTP, SMTP, FTP, SNMP, IMAP, POP3 aj.). *(2)*

1.3.3 Nette framework

Český, odborníky pozitivně hodnocený framework pro vývoj webových aplikací v PHP 5. Nette používá revoluční technologii, která eliminuje výskyt bezpečnostních děr a jejich zneužití, jako je např. XSS, CSRF, session hijacking, session fixation atd. Má propracované ladící nástroje jako vizualizaci chyb a výjimek, měření času vykonávání skriptů, debugger bar či tzv. dumpování proměnných. Pracuje s vyzrálým objektovým návrhem a je licencován v kategorii Open-source licence (New BSD a GNU-GPL). *(3)*

1.3.4 Verze instalovaného softwaru

Pro svůj projekt vzhledem k možnostem práce, vývoje a testování jsem zvolil variantu WAMPserveru – triády Apache, MySQL, PHP pro operační systém Windows konkrétně v konfiguraci Apache 2.2.22, Mysql 5.5.24, PHP 5.3.13, XDebug 2.1.2, XDC 1.5, PhpMyadmin 3.4.10.1, SQLBuddy 1.3.3, webGrind 1.0. *(4)*

Tento balíček aplikací jsem si oblíbil pro jeho snadnou a rychlou instalaci a především pro ovládací panel, díky němuž se konfigurace webového serveru stala velmi rychlá, přehledná a komfortní i pro běžného uživatele. Přehledný přístup ke všem serverovým konfiguračním souborům vystihuje naprosto přesně mou představu o významu termínu „User-friendly“.

Vzhledem k tomu, že toto je moje první webová aplikace většího rozsahu, dal jsem přednost i osvěčenému Nette frameworku, v současnosti už verzi 2.0 a jeho doplňkům, které dokáží programátorovi zpříjemnit práci jak s formuláři, tak třeba i na únosnou míru snížit bezpečnostní rizika celé webové aplikace.

2. Návrh databázové struktury

Během několika prvních konzultací s vedoucím pracovníkem, který mě uváděl do problematiky fungování praxí na střední škole, jsem si sice vyslechl požadavky na funkčnost webové aplikace, ale v průběhu školního roku postupně přidával další podněty na různá vylepšení, čímž se navyšovaly nároky na vývoj aplikace. Pro tuto práci jsem se tedy snažil navrhnout databázi tak, aby se funkčnost aplikace dala rozšiřovat i v budoucích letech bez větších zásahů do databázové struktury. Tím se vytvoří prostor pro další podněty a nově požadované funkce, které vyplynou časem při užívání aplikace. S velkou pravděpodobností to bude aplikace velmi živá a požadavky na funkčnost budou záviset především na konkrétním pracovníkovi delegovaném vedením školy, který tuto problematiku bude mít zrovna v popisu práce.

Při návrhu databázové struktury jsem vzal nejdříve v potaz pravděpodobné celkové množství dat ukládané do databáze webové aplikace.

Maximální počet uživatelů aplikace se odvíjí od počtu tříd a množství žáků v nich. Na Střední průmyslové škole v České Lípě se praxe týká pouze 2. a 3. ročníků, a tomu odpovídá aktuálně počet šest tříd (v novodobé historii jich nebylo nikdy více jak 8) o celkovém počtu 150 žáků. Těchto 150 potencionálních uživatelů bude do aplikace přistupovat pouze sporadicky. Každý z nich uskuteční několik přístupů (řádově do deseti) pouze v době, kdy si bude chtít zapsat či zkontrolovat výběr praxe. Nejčastější přistupování do aplikace spadá do agendy manažera praxí a administrátora, případně ještě asistentky při tisknutí smluv či dalších výstupů.

Počet firem a partnerů pro praxi a s tím spojený počet zainteresovaných osob s jejich identifikačními údaji se pohybuje také jen v řádu desítek. Čili databáze nebude obsahovat velké množství dat. Také nebude muset odolávat velkému náporu uživatelů a jejich dotazům a navíc pravděpodobně bude využívána spíše po lokální síti, což mě oprošťuje od jakýchkoliv náročných optimalizačních řešení či kompromisů jako je např. cachování dat či jejich indexace. *(5)*

Poté jsem si vytvořil první variantu seznamu potřebných položek pro evidenci (viz Tabulka 1) a začal kreslit náčrtek možných tabulek a jejich polí. Při pojmenování tabulek a jejich polí jsem vycházel z obecně platného a praktikovaného anglického jazykového úzu, ale pro větší srozumitelnost a vyjádření logiky datové struktury a celé

aplikace jsem pro některé položky zvolil i češtinu bez diakritiky ve tvaru Lower-CamelCase.[1]

Už během náčrtu jsem narážel na problém s přehledností a vyjádřením vztahů. Jak databázový model rostl, poohlédl jsem se po nějakém efektivním nástroji pro navrhování relačních databází. Z těchto užitečných nástrojů mě oslovil především jednoduchý wwwsqldesigner-2.6 psaný JavaScriptem, ale tím, jak se projekt rozrůstal a model databáze byl potřeba pro větší přehlednost editovat po částech, jsem dal přednost osvědčenému MySQL Workbench ve verzi 5.2.42.

Tabulka 1 – 1. normální forma (1NF) položek databáze (první varianta)

username	Uživatelské jméno generované administrátorem ve tvaru *jmeno.prijmeni*
password	Heslo uživatele hashované
mail	Email uživatele (dle systému Bakalář)
jmeno	Křestní jméno uživatele (pro oslovení, vyhledávání a generování smluv)
prijmeni	Příjmení uživatele (pro oslovení, vyhledávání a generování smluv)
created	Datum vytvoření uživatele
login	Datum posledního přihlášení uživatel
role	Role uživatele (admin, editor, zak)
trida	Třída, kterou žák navštěvuje
firma	Celé jméno firmy
obor	Obor praxe (Strojírenství, Technické lyceum, IT)
kOsoba	Titul jméno příjmení kontaktní osoby ve firmě
kMail	Email kontaktní osoby
kTelefon	Telefon kontaktní osoby
ulice	Adresa firmy
psc	Poštovní směrovací číslo
mesto	Město, kde sídlí firma
sOsoba	Titul jméno příjmení osoby oprávněné podepisovat za firmu smlouvu
pozadavky	Požadavky na oblečení, vybavení a bezpečnost
dPodani	Datum zadání praxe žákem
pSchvalena	Praxe schválená manažerem
pPridelena	Praxe přidělená manažerem
done	Praxe uznána za splněnou v rámci školní klasifikace

[1] CamelCase - označuje způsob psaní víceslovných frází a nadpisů, kdy jednotlivá slova nejsou oddělena mezerami, ale každé z nich začíná velkým. Lower-CamelCase (malé počáteční písmeno prvního slova) je současně konvence psaní různých identifikátorů v programovacích nebo skriptovacích jazycích jako např. Java, PHP, Ruby atd.

2.1 ER – Model vyjádřený diagramem

S rostoucí složitostí projektu jsem zvolil metodu práce vycházejícího z principu Entity - Relationship Model[2] ve formě diagramu. Postupně jsem se snažil odhalit potřebné entity s jejich atributy a následně určit vztahy (relace) mezi nimi. Dá se říci, že každá entita představuje vlastně jednu tabulku databáze a atribut je jakákoliv další informace, která s entitou může souviset. Například v mém případě entita *Osoba* má atributy *jméno* a *příjmení*. Každá entita musí mít alespoň jeden atribut, který ji odliší od ostatních. Ten pak představuje primární klíč. Po vytvoření této kostry logických vazeb bylo potřeba vyjádřit typy relací mezi entitami. Nejdříve jsem určil tzv. kardinalitu relace čili minimální a maximální počet výskytu entity pro obě strany relace. Tomu odpovídá, že např. entita *Žák* relací (1..1) *patří do* entity *Třída* (žák patří pouze do jedné třídy) a naopak jedné *Třídě* může *patřit* jakýkoliv *Žák* (1..N). V tomto případě lze vyvodit, že minimální hodnota relace je (1..1) a maximální (1..N) čili výsledná relace je 1..N. Takto jsem postupoval ve všech dalších případech, dokud jsem nepostihl všechny zamýšlené entity, jejich atributy a relace mezi nimi v celé databázi. Vše jsem pak graficky ztvárnil do ER - diagramu (viz Příloha 1), kde entita je reprezentována zeleným obdélníkem, atribut růžovou elipsou, relace modrým kosočtvercem a výsledný typ relace je v okrovém rámečku.

2.1.1 Postup při tvorbě diagramu

Vzhledem k nutnosti autentizace uživatele jsem jako první začal uvažovat nad entitou ***User*** s atributy *login (uživatelské jméno)*, *heslo*, *datum vytvoření*, *datum přihlášení*. Dále bylo potřeba uživatele rozdělit alespoň na žákovské a administrační čili jim přidělit určitou ***Roli*** (*název*) s daným ***Oprávněním*** (*název*, *popis*). Nato už mohla navazovat entita ***Žák*** a od ní odvozené normalizované entity ***Třída*** (*název*, *ročník*, *označení*, *obor*), ***Žádost o praxi*** (viz Obrázek 1) a ***Praxe*** (*schválení*, *začátek*, *konec*, *splněna*). Každá praxe je provozována ***Firmou*** (název) a ta může mít i několik pracovišť na různé ***Adrese*** (*ulice*, *město*, *psč*) a vše zaštiťuje ***Osoba*** (*jméno, příjmení*, *email, telefon*) a od ní odvozená ***Kontaktní osoba***.

[2] metoda datového modelování, která vytváří jeden z typů konceptuálních schémat či sémantických datových modelů systému (obvykle relační databáze) a požadavků na něj stylem shora dolů. Techniky datového modelování se používají pro popis ontologie (tj. přehled a klasifikace použitých pojmů a jejich vztahy mezi sebou) pro určitou oblast zájmu.

Obrázek 1 - Entita Žádost o praxi a její atributy

2.1.2 Postup normalizace tabulek při návrhu databáze

Během normalizace tabulek jsem postupoval podle těchto obecně známých pravidel:

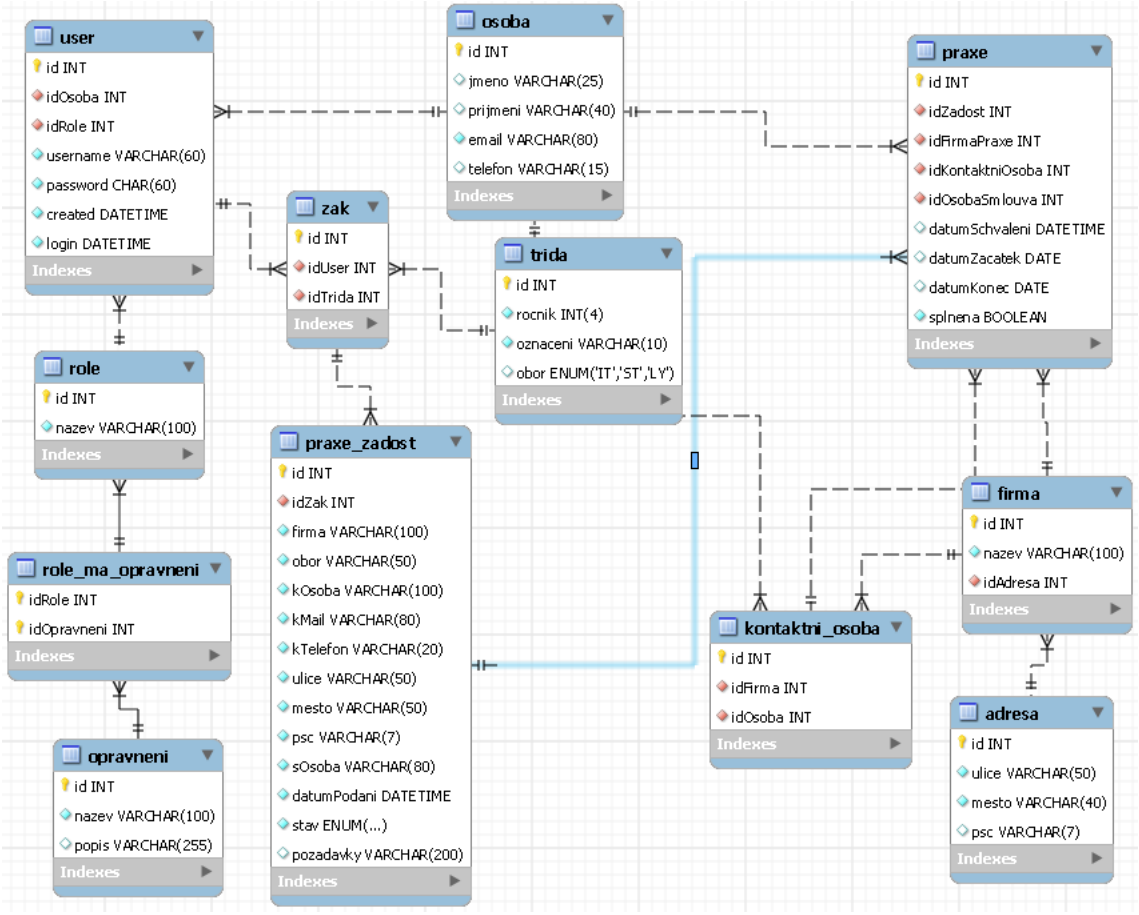
- data by měla být atomická čili dále nedělitelná (1NF)

Tuto podmínku jsem splnil všude až na pole *ulice*. Vzhledem k charakteru pole a jeho následnému využití v aplikaci je zbytečné oddělovat jméno ulice od čísla popisného, což ani v praxi většina programátorů běžně nečiní, pakliže to nesouvisí s dalšími konkrétními funkcemi tvořené aplikace.

- všechna data by měla záviset na celém klíči, aby nedocházelo k redundanci dat (2NF)
- neklíčová data jsou závislá pouze na klíči a ne na sobě samých (3NF)

Pomocí tohoto postupu *(6)* jsem se dostal do stavu 3. normální formy (3NF), kdy jsem postupně redukoval možné duplicitní (redundantní) data. Pro větší srozumitelnost jsem některé názvy atributů převedl na vhodnější pojmenování, kdy

během tvorby ER-diagramu se mi databáze rozrostla na 12 tabulek s prozatímním počtem 46 atributů.



Obrázek 2 - Výsledný návrh databáze s relacemi

2.2 Tabulky – datové typy polí

Pro dokončení návrhu databáze je potřeba přiřadit atributům správné datové typy, indexy a počáteční hodnoty.

Datové typy používané v SQL se neliší od typů používané u jiných programovacích jazyků. K dispozici máme typy pro celá i reálná čísla, pro textové řetězce, pro datum a čas, pro binární data (jako jsou např. obrázky, zvukové soubory apod.) Přehled nejpoužívanějších typů je uveden v Tabulce 2.

Tabulka 2 – Přehled základních datových typů SQL *(7)*

int	celé číslo v rozsahu od −2 147 483 648 do 2 147 483 647
smallint	celé číslo v rozsahu od −32 768 do 32 767
tinyint	celé číslo v rozsahu od 0 do 255
float	číslo s pohyblivou řádovou čárkou
char (n)	textový řetězec o délce n (maximálně však 255 znaků)
varchar (n)	textový řetězec o maximální délce n (max. však255 znaků)
decimal (p)	desetinné číslo s p platnými číslicemi
decimal (p,d)	desetinné číslo s p platnými číslicemi a s d desetinnými místy
money	peněžní částka (tento typ nepodporují zdaleka všechny servery, můžeme ho však snadno nahradit například pomocí decimal (12,2))
datetime	údaj o čase a datu ve formátu RRRR-MM-DD HH:MM:SS
time	údaj o čase ve formátu HH:MM:SS
date	údaj o datu ve formátu RRRR-MM-DD
blob, image	speciální typy pro uložení dlouhých binárních dat (každý server používá vlastní typ)

Při volbě datových typů pro vzniklé tabulky jsem vycházel především z možností a potřeb aplikace, ale také i z běžných životních zvyklostí.

U stringových položek souvisejících s Osobou jako jsou jméno, příjmení a z nich utvořené v tabulce User pole username jsem se pokusil vyhledat nejdelší možné české varianty jako je např. příjmení Červenokostelecký (17 znaků), dále netradiční česká křestní jména nedosahují větší délky než 10 znaků jako Drahoslava, Konstantin, Květoslava, Stanislava a předpokládám, že lidé mohou mít dvě i více křestních jmen. Problém by mohl nastat u žáků původem z ciziny, např. z Asie, kteří mají hned několik jmen a ani sami někdy nedokážou správně přiřadit, co je křestní jméno a co příjmení.

Proto považuji za dobrý kompromis u těchto položek rezervovat rozsah tímto způsobem:

Tabulka Osoba

```
`jmeno` VARCHAR(25) NULL ,
`prijmeni` VARCHAR(35) NULL ,
```

Tabulka User

```
`username` VARCHAR(60) NOT NULL ,
```

Obdobně v tabulce Adresy u položky město jsem zvolil VARCHAR (40), protože například nejdelší název obce v Libereckém kraji mají Albrechtice v Jizerských horách (32), dále u ulice VARCHAR (50), protože obsahuje zároveň i číslo popisné a orientační a může být i v takovém tvaru jako v Kostelci nad Černými lesy ulice Generála Jaroslava Sázavského Vedrala (37).

Pro uživatelské heslo, které bude ještě navíc upraveno hashovací funkcí, bylo namístě zvolit typ CHAR o rozsahu 60 znaků.

Mimo zmiňovaný VARCHAR a CHAR jsem dále využil u položek, kde je potřeba evidovat datum (např. Začátek a konec praxe), typ DATE. V tomto případě stačí pouze rozsah DD-MM-RRRR. Pro položky charakteru Vytvořeno, Přihlášen, Schváleno apod. jsem dal přednost DATETIME vzhledem k možnosti uchovávat údaj o čase a datu ve formátu RRRR-MM-DD HH:MM:SS.

Z výčtových typů jsem nakonec přiřadil poli Stav praxe typ ENUM s možnostmi ('schválena','zamítnuta','čekající') a taktéž poli Obor ('IT','LY','ST'), který vyjadřuje kategorii oboru na dané škole přidělovanou každé třídě.

Booleovský typ jsem se rozhodl použít pouze v tabulce Praxe u pole Splněno (BOOLEAN), které uzavírá celý proces průběhu praxí a s hodnotou TRUE je příznakem pro úspěšné vykonání a dokončení žákovských praxí.

2.3 Tabulky – přiřazené indexy a klíče

Aby byla databázová struktura kompletní a plně funkční, je důležité ještě přiřadit správné indexy a klíče dle požadované logiky aplikace. Pokud by databáze neudržovala pro tabulky žádný index, pak by i u jednoduchého dotazu musela projít všechny dostupné záznamy a rozhodnout teprve na základě získaných dat, zda záznam patří do požadované relace a až poté zda odpovídá dalším podmínkám, čili databázový index je vlastně jakákoliv datová struktura, která uspořádá hodnoty jednoho či více zvolených atributů za účelem rychlejšího prohledání, případně řazení, záznamů na těchto atributech. *(8)*

Tyto struktury je nutné udržovat aktuální při každém vložení, upravení nebo odmazání záznamu, což s sebou přináší zvýšený počet zápisů do paměti databáze nebo zpomalení operací jako je například vykonání SQL příkazů UPDATE či INSERT. To právě odpovídá tomu, že se databáze musí postarat nejen o změny v datech tabulky, ale i o změny v datech indexu, tudíž nadměrné definování indexů v tabulce může vést paradoxně ke zpomalení celé aplikace.

2.3.1 Druhy klíčů a indexů

- **Primární klíč (PRIMARY KEY)**

Dobrým adeptem na zvolení primárním klíčem je pole, které jednoznačně identifikuje každý záznam v databázové tabulce. Z toho vyplývají dvě základní vlastnosti, jako jsou jedinečnost (v rámci tabulky) a „nenulovost“ čili nesmí obsahovat možnost zapsání prázdné hodnoty NULL respektive musí být nastaven na NOT NULL. Některá pole jako Rodné číslo, SPZ vozidla, Email či Číslo bankovního účtu pro svoji unikátnost se vybízejí přímo stát se primárním klíčem a jsou označována za tzv. přirozený primární klíč, ale vzhledem ke globalizaci a nejednotnosti formátů vstupních dat těchto atributů se nejčastěji můžeme setkat s tzv. umělým primárním klíčem v podobě sloupce pojmenovaného jako ID, které je auto-inkrementován.

V neposlední řadě se primární klíč může v některých případech skládat i z kombinace hned několika atributů.

V mém případě při navrhování databáze jsem se při přiřazování primárních klíčů držel druhé zmiňované možnosti, a to vytvoření umělých primárních klíčů pojmenovaných jako ID. Obsahují ho vlastně všechny tabulky s výjimkou jedné, a to *role_ma_opravneni*, kde je primární klíč tvořen hned dvojicí atributů *idRole* a *idOpravneni*, čili se jedná o klíč složený. Koresponduje to též s relací N:M, kdy tabulka role_ma_opravneni je vlastně propojovací tabulka mezi tabulkami Role a Oprávnění.

Zdrojový kód SQL pro vytvoření tabulky *role_ma_opravneni*:

```
CREATE  TABLE IF NOT EXISTS `PraxeZaci`.`role_ma_opravneni` (
  `idRole` INT NOT NULL AUTO_INCREMENT ,
  `idOpravneni` INT NOT NULL ,
  PRIMARY KEY (`idRole`, `idOpravneni`) ,
  INDEX `fk_{IDOPRAVNENI}` (`idOpravneni` ASC) ,
  CONSTRAINT `fk_{IDOPRAVNENI}`
    FOREIGN KEY (`idOpravneni` )
    REFERENCES `PraxeZaci`.`opravneni` (`id` ));
```

- **Cizí klíč (FOREIGN KEY)**

Cizí klíč určuje vztah mezi dvěma tabulkami, a to tak, že hodnota v daném sloupci musí být i v jiné (primární) tabulce. Tento klíč vlastně slouží jako reference či odkaz pro vyjádření relací mezi databázovými tabulkami. Jedná se o pole či skupinu

polí, která nám umožní identifikovat, které záznamy z různých tabulek spolu navzájem souvisí. Tím vzniká tzv. integritní omezení databáze, které do tabulky položky umožní vložit jen povolené hodnoty.

Cizí klíč také umožňuje definovat akce, které mají nastat při pokusu o změnu nebo mazání záznamů v cizí tabulce. Například, po smazání záznamu z cizí tabulky budou ve zdrojové tabulce řádky s odpovídající hodnotou cizího klíče taktéž smazány, nebo budou jejich odkazy nastaveny na určitou (neutrální) hodnotu či se může smazání řádků v cizí tabulce i zabránit, pokud daná reference stále existuje.

V mém návrhu databáze tedy cizí klíče vytvářejí reference mezi tabulkami na atributy vzniklé při normalizaci do 3. normální formy. Konkrétně jsou tyto vztahy patrné už z pojmenování cizího klíče, kdy jsem jejich názvy volil shodně dle názvů atributů tak, jak jsou uvedené v primárních tabulkách. Pro lepší představu uvádím následující příklad.

Zdrojový kód SQL pro vytvoření tabulky Praxe

```
CREATE  TABLE IF NOT EXISTS `PraxeZaci`.`praxe` (
  `id` INT NOT NULL AUTO_INCREMENT ,
...
  PRIMARY KEY (`id`) ,
  INDEX `fk_{IDZADOST}` (`idZadost` ASC) ,
  INDEX `fk_{IDFIRMAPRAXE}` (`idFirmaPraxe` ASC) ,
  INDEX `fk_{IDKONTAKTNIOSOBA}` (`idKontaktniOsoba` ASC) ,
  INDEX `fk_{IDOSOBASMLOUVA}` (`idOsobaSmlouva` ASC) ,
  CONSTRAINT `fk_{IDZADOST}`
    FOREIGN KEY (`idZadost` )
    REFERENCES `PraxeZaci`.`praxe_zadost` (`id` ),
  CONSTRAINT `fk_{IDFIRMAPRAXE}`
    FOREIGN KEY (`idFirmaPraxe` )
    REFERENCES `PraxeZaci`.`firma` (`id` ),
  CONSTRAINT `fk_{IDKONTAKTNIOSOBA}`
    FOREIGN KEY (`idKontaktniOsoba` )
    REFERENCES `PraxeZaci`.`kontaktni_osoba` (`id` ),
  CONSTRAINT `fk_{IDOSOBASMLOUVA}`
    FOREIGN KEY (`idOsobaSmlouva` )
    REFERENCES `PraxeZaci`.`osoba` (`id` ));
```

- **Unikátní klíč (UNIQUE KEY)**

Jedná se o speciální druh indexu, který dal vzniknout zmiňovanému klíči primárnímu. Liší se od sebe tím, že klíč unikátní operuje pouze s vlastností vyjadřující jedinečnost záznamu, ale oproti primárnímu klíči jich v tabulce může být uvedeno hned několik a může obsahovat i prázdnou hodnotu NULL.

Tento typ klíče vzhledem k jeho vlastnosti jsem použil například v tabulce Oprávnění na pole Název, v tabulce User na pole Username či v tabulce Firma na pole Název, kde je potřeba zamezit duplicitnímu vkládání údajů a zajistit jejich jedinečnost.

2.4 Kódování databáze

Od verze MySQL 4.1 funguje podpora různých kódových stránek, a to nejen na úrovni databázového serveru, ale i na úrovni jednotlivých databází, tabulek a sloupců. Syntakticky lze toho dosáhnout příkazem COLLATE a za něj uvést odpovídající kódovou stránku spolu s výchozím tříděním, jak uvádí příklad níže:

databáze: `CREATE DATABASE db_name COLLATE utf8_general_ci`

tabulka: `CREATE TABLE table_name (…) COLLATE utf8_czech_ci`

sloupec: `jmeno varchar(50) COLLATE utf8_slovak_ci`

Kromě uložených dat lze určit ještě kódování klienta (tedy v jakém kódování se data posílají) a přenášených a vracených dat. Všechna tři kódování lze nastavit příkazem SET CHARACTER SET[3]. Pro optimální výkon je tedy vhodné používat všude stejné kódování jak na serveru, tak během přenosu i u klienta. *(9)*

Pro navrženou databázi jsem zvolil v dnešní době nejčastěji používanou univerzální jazykovou sadu z rodiny Unicode, a to UTF-8 s porovnáním češtiny utf8_czech_ci čili case insensitive, kdy nezáleží na velikosti porovnávaných znaků. Navíc aplikace bude využívat PHP Framework Nette, který UTF-8 přímo vyžaduje. *(10)*

[3] z hlediska syntaxe se příkazy SET CHARACTER SET nebo SET NAMES pro určení hodnoty píší bez apostrofů, jak by bylo v SQL zvykem (např. SET CHARACTER SET utf8)

3. Realizace aplikace

Pro tvorbu aplikace jsem využil vývojového prostředí Netbeans IDE 7.3, které jsem si oblíbil pro přehlednou správu a editaci projektových souborů, kdy např. v okně Navigátor jsou zobrazovány informace o používaných metodách a atributech.

3.1 Instalace frameworku Nette

V současné době existuje Framework Nette ve verzi 2.0.10 ve dvou rovnocenných variantách, a to pro PHP 5.2 a starší a pro PHP 5.3 a 5.4. Obě jsou dostupné na http://nette.org/cs/download. Pro ověření minimálních požadavků na prostředí webového serveru slouží integrovaný nástroj checker.php, který vyhodnotí např. informace o verzi PHP, přístupu k souborům typu .htaccess aj. Podrobný přehled požadavků je uveden v Příloze 2.

3.2 Princip práce v Nette

Nette vychází z tzv. Model-View-Controller architektury, kde je u aplikací s grafickým rozhraním oddělen kód obsluhy (controller) od kódu aplikační logiky (model) a od kódu, který zobrazuje data (view), což vede k větší přehlednosti pro další vývoj či umožňuje testovat fragmenty aplikace samostatně. *(11)*

3.2.1 Model

Model by měl představovat datový a funkční základ celé aplikace a navíc je v něm obsažena aplikační logika. Uchovává si svůj vnitřní stav a pro okolí nabízí pevně dané rozhraní. Voláním funkcí tohoto rozhraní se může zjišťovat či měnit jeho stav. Model o existenci dalších vrstev, jako je view nebo kontroler, neví. Pojem Model představuje celou vrstvu aplikace, nikoliv jen samostatnou třídu.

3.2.2 View (šablony Latte)

View, tedy pohled, je aplikační vrstva, která má zajišťuje zobrazení výsledku zaslaného požadavku. Obvykle používá nějaký šablonovací systém a ví, jak se mají zobrazit komponenty nebo výsledky získané z vrstvy modelu.

Nette framework k vytváření šablon využívá jazyk Latte, který oproti standardní PHP syntaxi zkrátí a zpřehlední zdrojový kód. Navíc zabezpečí výstup před zranitelnostmi jako je např. Cross-site scripting (XSS), kdy ošetří uživatelské vstupy tak, že odfiltruje

(tzv. „escapuje“) nežádoucí znaky, aby nemohlo dojít k podstrčení nežádoucího javascriptového kódu. Využívá k tomu unikátní technologii Context-Aware Escaping, která rozezná, ve které části dokumentu se nachází a podle toho zvolí správný typ escapování. Nesporné výše zmiňované výhody jazyku Latte jsou patrné už na tomto krátkém skriptu pro bezpečné vypsání proměnné `var`.

```
PHP syntaxe       <?php echo htmlspecialchars($var, ENT_QUOTES); ?>
Latte syntaxe     {$var}
```

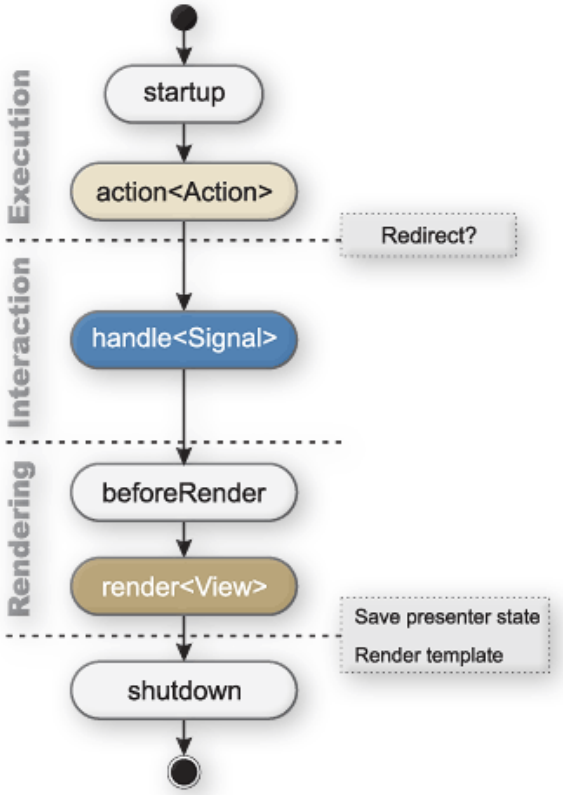
3.2.3 Controller vs. Presenter

Kontrolér neboli řadič má za úkol zpracovávat požadavky uživatele a na jejich základě pak volat patřičnou aplikační logiku reprezentovanou modelem a poté požádat view o vykreslení dat. Obdobou kontrolérů v Nette Framework jsou **tzv. presentery**.

Presenter představuje objekt, který převezme požadavek HTTP a vymyslí na něj odpověď. Odpovědí se pak rozumí například HTML stránka, obrázek, XML dokument, soubor na disku, JSON, přesměrování aj. Podle příchozích požadavků též spouští odpovídající akce a vykresluje šablony. Při tvorbě presenterů lze v Nette využít hned několik metod, které na sebe navazují a jako celek vytvářejí tzv. životní cyklus presenteru (viz Obrázek 3).

Metody presenteru:

- ***startup()*** – inicializuje proměnné nebo ověřuje uživatelská oprávnění
- ***action<Action>()*** – obdoba metody render s nevizuálním výsledkem jako je např. zapsání dat do databáze, odhlášení uživatele apod.
- ***handle<Signal>()*** – zpracovává tzv. signály neboli subrequesty komponent a AJAXových požadavků
- ***beforeRender()*** – metoda obsahující například nastavení šablony, předání proměnných společných pro více view, než se zavolá metoda render<View>
- ***render<View>()*** – metoda, která předá do šablony potřebná data
- ***shutdown()*** – metoda volána při ukončení presenteru



Obrázek 3 - Životní cyklus presenteru a jeho metody v Nette *(11)*

3.2.4 Adresářová struktura

Základní adresářová struktura projektu v Nette se odvíjí od pojetí aplikační architektury MVC respektive MVP. Hlavním adresářem projektu je tzv. `sandbox` a ten obsahuje podadresáře `app` , `libs`, `log`, `temp` a `www`. Přehlednější logické rozložení adresářů a vztahy mezi nimi popisuje níže uvedené Schéma 1. Tato struktura je však pouze doporučená a spěje k větší přehlednosti a bezpečnosti vyvíjené aplikace. Lze ji jakkoli modifikovat, a to pomocí konstant uvedených v souborech `index.php` a `bootstrap.php`.

U složitějších aplikací můžeme složky s presentery a šablonami rozčlenit do podadresářů a vytvořit tzv. moduly. Tím lze od sebe snadno oddělit například část aplikace určenou pro roli administrátora a běžného uživatele.

Schéma 1 – Adresářová struktura standardní Nette aplikace

```
sandbox/
├── app/                    ← adresář s aplikací
│   ├── config/             ← konfigurační soubory
│   │   └── config.neon     ← hlavní konfigurační soubor
│   │
│   ├── model/              ← modelová vrstva a její třídy
│   ├── presenters/         ← třídy presenterů
│   │   └── HomepagePresenter.php  ← třída presenteru Homepage
│   │
│   │   templates/          ← adresář se šablonami
│   │   ├── @layout.latte ← šablona společného layoutu
│   │   └── Homepage/       ← šablony presenteru Homepage
│   │       └── default.latte  ← šablona akce default
│   │
│   └── bootstrap.php       ← zaváděcí soubor aplikace
│
├── libs/               ← adresář na knihovny třetích stran
│   ├── Nette/          ← framework
│   └── ...
│
├── log/                  ← obsahuje logy, error logy atd.
├── temp/                 ← pro dočasné soubory, cache, ...
└── www/                  ← veřejný adresář, document root projektu
    ├── .htaccess         ← pravidla pro mod_rewrite
    ├── index.php         ← soubor spouštějící aplikaci
    └── images/           ← další adresáře, třeba pro obrázky
```

3.3 Datový model aplikace

Datový model je funkční základ celé aplikace a reprezentuje problém, který aplikace řeší. V mém případě sem patří entity, což jsou např. objekty reprezentující praxi, seznam praxí a uživatele. Ta to vrstva je nezávislá na prezentační logice, tedy té části aplikace, která model prezentuje uživateli a zpětně překládá jeho požadavky.

3.3.1 Připojení k databázi

Konfiguraci pro připojení k požadované databázi nalezneme v souboru `config.neon`, který je standardně umístěný v adresáři `app/config`. Zde stačí vyplnit požadované konstanty (viz níže) a databázi lze začít používat. Z hlediska syntaxe považuji za vhodné ještě uvést, že v tomto souboru je striktně vyžadováno odsazovat text od levého okraje pouze tabulátory a při použití jakéhokoliv jiného znaku z rodiny white-spaces připojení skončí chybou.

```
common:
	...
	nette:
		...
		database:
			dsn: 'mysql:host=localhost;dbname=db_praxe'
			user: root
			password:

		session:
			expiration: 14 days
	...
```

Připojení k databázi přes soubor config.neon

3.3.2 Model - třídy a služby

V Nette existuje řada možností, jak realizovat objektový návrh modelu a jak přistupovat k databázi. Vhodné je vytvořit třídy reprezentující jednotlivé entity a vazby mezi nimi. Přičemž perzistence, čili práce s databází, se přenechá samostatným třídám, tzv. data-mapperům, k čemuž lze využít například knihovnu Doctrine 2. Obvyklé činnosti prováděné nad entitami se svěří opět samostatným třídám, tzv. fasádám.

Jednodušší možností, kterou jsem se rozhodl aplikovat, je použít nástroj `Nette\Database\Table`, jenž představuje chytrý „průzkumník pro databázi“. *(12)*

Jako základ datového modelu jsem vytvořil abstraktní třídu `Repository`, v které jsou definovány tři elementární funkce pro práci s databázovými tabulkami. Dle dokumentace k frameworku Nette funkce `getTable()` odvodí název třídy rovnou z názvu databázové tabulky a využije se dále ve funkcích `findAll()`, která vrací všechny záznamy z tabulky, respektive `findBy()` s parametrem `array $by`, který v sobě zahrnuje asociativní pole hodnot už podle konkrétního filtru.

```
/*využívá třídy Nette\Database\Table\Selection
dle vzoru array ('jmeno_pole'=> 'hodnota_pole')*/

public function findBy(array $by)
{     return $this->getTable()->where($by);     }
```

Dotaz na tabulku pomocí funkce findBy

Pak už pro práci s konkrétními tabulkami vytvořím pro každou z nich v adresáři `app/model` novou třídu v samostatném repositáři (např. `UserRepository`, `TridaRepository` apod.), přičemž využiji dědičnosti právě od zmiňované abstraktní třídy `Repository` a jelikož Framework Nette 2.0 podporuje nové možnosti PHP verze 5.3, nadefinuji i jmenné prostory včetně použití Nette. Záměrně upouštím od používání koncové značky PHP `?>`, jelikož ji jazyk přímo nevyžaduje a eliminuji tím situaci, kdyby se za ni při editaci kódu dostal jakýkoli bílý znak, jenž by byl příčinou nefunkčnosti aplikace. *(12)*

```
<?php
namespace Todo;
use Nette;

class UserRepository extends Repository{ }
```

Deklarace třídy UserReposity

Dále Nette podporuje tzv. služby (services), které lze využívat například při komunikaci s databází, kdy se nemusí pro parametry spojení definovat globální proměnné či statické proměnné tříd, ale použije se právě daná služba. Tomuto způsobu se říká Depency Injection, jenž je založen na principu odebrat třídám zodpovědnost za získávání objektů, které potřebují ke své činnosti (tj. služeb) a místo toho jim služby předávat při vytváření. *(13)*

V mém případě tohoto principu využiji například při autentizaci uživatele, kdy se tato služba bude starat o ověřování platnosti uživatelského jména a hesla nebo při propojování presenterů s třídami datového modelu. V podstatě se nejedná o nic jiného, než v konfiguračním souboru `config.neon` doplnit sekci `services` následujícím způsobem:

```
services:
            authenticator: Todo\Authenticator
            userRepository: Todo\UserRepository
            tridaRepository: Todo\TridaRepository
            ...
```

Tím jsem vlastně zaregistroval službu `userRepository`, která bude obsahovat instanci `Todo\UserRepository`. Analogicky jsem postupoval u i dalších položek. Všechny třídy vyžadují pouze jeden parametr v konstruktoru, a tím je objekt `Nette\Database\Connection`, který právě zajišťuje spojení s databází.

3.4 Presentéry a šablony

Další krokem při tvorbě aplikace je vytvoření presenterů, které zajistí předání dat z modelové vrstvy do příslušných šablon (view).

3.4.1 HomepagePresenter

Opět jako u modelové vrstvy bude základním presenterem abstraktní třída `BasePresenter` a z ní děděním vznikne třída `HomepagePresenter`, která bude zajišťovat zobrazení úvodní stránky aplikace v šabloně `templates\Homepage\default.latte` respektive i jejího prapředka `templates\@layout.latte`, z kterého byla odvozena.

Hlavní stránka se skládá ze čtyř samostatných bloků:

- záhlaví s logem a informacemi o uživateli
- záznam o praxi přihlášeného uživatele
- formulář pro žádost o praxi
- datagrid pro správu praxí (pouze pro roli administrátora a editora)

Záhlaví je umístěno právě v základní šabloně `@layout.latte`, protože je žádoucí, aby se vykreslovalo na každé stránce.

Základ této šablony tvoří klasické HTML elementy rozšířené o makra šablonovacího jazyka latte, které se v kódu šablony vyznačují složenými závorkami a umožňují plno nadstandardních možností od zobrazování proměnných, podmíněných příkazů, cyklů, ovládacích prvků a formulářů, odkazů či ladění jako je tzv. dumpování proměnných či vkládání breakpointu.

Ukázka zdrojového kódu základní šablony `@layout.latte` a využití maker v záhlaví aplikace:

```
<!DOCTYPE html>
 <html>
  <head> <title>SPS-Praxis</title>
  ...
// {$basePath} vypíše escapovanou proměnnou
<script type="text/javascript"
     src="{$basePath}/js/netteForms.js"></script>
  ...
// definuje a vypíše blok
   {block head}{/block}
  </head>
  <body>

// propojení loga a
   <div class="title"><img src="../images/sps.png"
     height="32px">  PRAXE SPŠ</div>

// n:if="..." blok se zobrazí jen autentizovanému uživateli
  <div class="user" n:if="$user->isLoggedIn()">...

// n:href="User:password" odkazuje na UserPresenter,
   který umožňuje změnit heslo
  <span class="icon user">{$user->getIdentity()->name}</span>
   |<a n:href="User:password">Změna hesla</a> ...
   ...
// vkládá obsah z dalších dceřiných šablon např. default.latte
{include #content}
   ...
</body>
</html>
```

Zbylé tři bloky jsou zakódovány v šabloně `Homepage\default.latte` a jejich vykreslení obstarává zmiňovaný `HomepagePresenter`. Ten zahrnuje funkci pro vytvoření seznamu žádostí o praxi podle uživatele `createComponentUserTasks()`. Ta na základě modelové třídy `TaskRepository` pracuje s funkcemi této třídy, a to:

```
...
findIncomplete() // vrací zadané nesplněné praxe
...
// výsledek filtruje dle uživatele
public function findIncompleteByUser($userId)
      {
            return $this->findIncomplete()->where(array(
                  'user_id' => $userId          ));
      }
...
```

Vzhledem k tomu, že seznam žádostí o praxi či z něho odvozené seznamy se v aplikaci budou využívat i na jiných místech, lze využít v Nette tzv. komponent. Jsou to vlastně třídy, které reprezentují vykreslitelný objekt, jenž se dá opakovaně vložit kamkoli do stránky. *(14)*

Pro komponenty v adresářové struktuře Nette je potřeba vytvořit adresář `app\components` a následně do něj umístit jak soubor se zdrojovým kódem včetně rodičovského konstruktoru a metody render(), tak i soubor se šablonou latte.

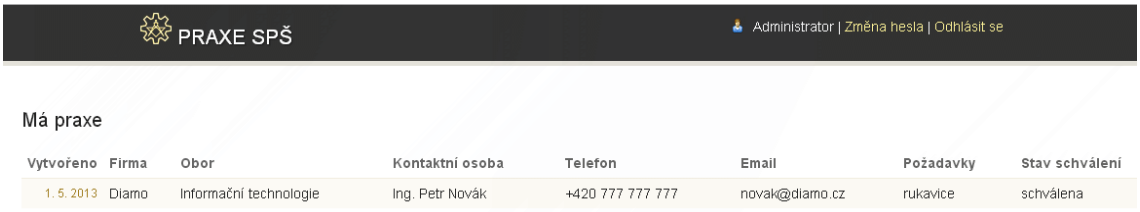
```
public function render() //metoda vykreslení
{ //přesměrování na správnou šablonu latte
      $this->template->setFile(__DIR__ . '/TaskList.latte');
      $this->template->tasks = $this->selected;
      $this->template->displayUser = $this->displayUser;
      $this->template->displayList = $this->displayList;
      $this->template->render();
}
```

Následně už stačí jen vytvořit v `HomepagePresenteru` funkci `createComponentNazev()` a komponentu lze používat.

```
// komponenta pro seznam praxí dle uživatele
public function createComponentUserTasks() {
$incomplete=$this->taskRepository
                  ->findIncompleteByUser($this->getUser()
                  ->getId());
```

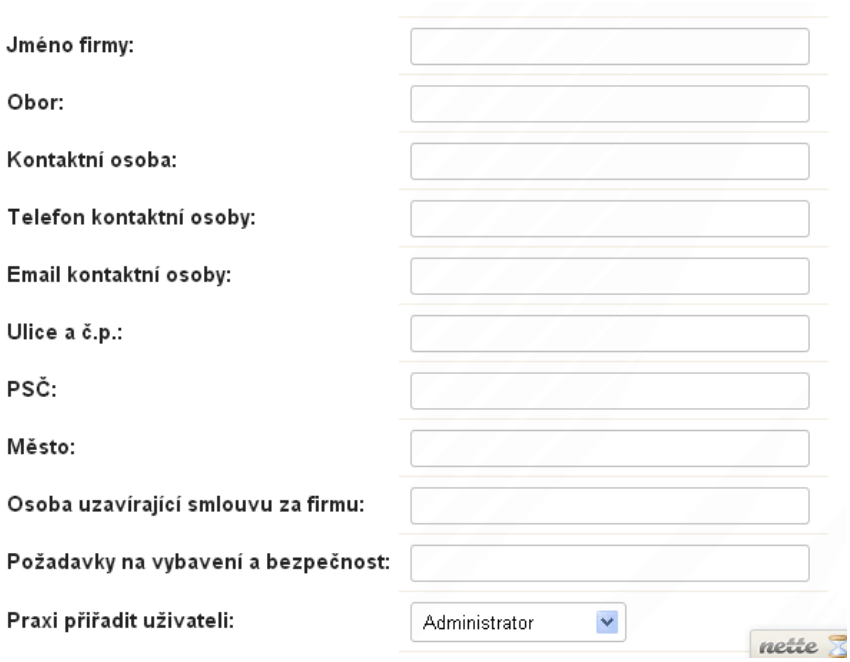
```
$control =
new Todo\TaskListControl($incomplete, $this->taskRepository);
$control->displayList = TRUE; //zobrazení seznamu všech praxí
$control->displayUser = FALSE; //zobrazení seznamu dle uživatele
      return $control;
}
```

Výsledkem je blok (Obrázek 4), ve kterém uživatel uvidí, zda má už praxi zadanou a dále její status čili stav přidělený pracovníkem s možnostmi čekající, schválena, zamítnuta.



Obrázek 4 - Blok zobrazující hlavičku a informace o praxi

Pro možnost zadávat praxi jsem zavedl formulář nazvaný žádost o praxi (Obrázek 5), z kterého se budou data ukládat do databáze. Toto oprávnění by měl mít jak samotný žák, tak i editor nebo administrátor s tou výjimkou, že u žáka nesmí být atribut ‚Přiřadit praxi uživateli' aktivní.



Obrázek 5 - Blok zobrazující zasílací formulář

Tento problém jsem vyřešil jednoduchým podmíněným příkazem, který toto pole změnil z rozbalovacího seznamu na skryté.

```
// porovná uživatele dle oprávnění (<=2 -> admin, editor)
if ($this->getUser()->getId() <=2)
  //rozbalovací seznam
  {$form->addSelect('userId', 'Praxi přiřadit uživateli:',
  $userPairs)
      ->setPrompt('- Vyberte -')
      ->addRule(Form::FILLED, 'Je nutné vybrat, komu je praxe
přiřazena.')
      //přiřazení aktuálně přihlášeného uživatele k praxi
      ->setDefaultValue($this->getUser()->getId());}
else
 //skryté pole, přiřazení praxe aktuálnímu uživateli
 {$form->addHidden('userId', $this->getUser()->getId());}
```

U validace a odesílání vyplněného formuláře reprezentované funkcí `taskFormSubmitted(Form $form)` bylo potřeba navíc ošetřit, aby žák nemohl praxi zadat vícekrát než jednou a nemohl ji kdykoli změnit. Použil jsem opět podmínku a využil funkce `findIncompleteByUser` pro seznam nesplněných praxí s parametrem vracející počet uživatelů, a ten jsem položil větší nule.

```
...// ověření, zda už uživatel nemá přiřazenu praxi
  if ($this->taskRepository->findIncompleteByUser($this-
>getUser()->getId())->count()>0) ...
```

Posledním blokem na úvodní stránce pro správu praxí je DataGrid. Pro Nette je jich dostupných hned několik. Vzhledem k potřebě zobrazovat na stránce větší počet polí v řádku jsem se rozhodl použít NiftyGrid, který podporuje jak AJAX, řádkové akce, hromadné akce, filtrování, řádkovou editaci záznamu, řazení dle sloupců, funkci Autocomplete, tak i potřebné SubGridy, v kterých nechám zobrazovat méně žádaná data, abych nepřekročil maximální šířku stránky 1280 pixelů, což je i standardní zobrazovací hodnota, kterou splňuje hardware lokální sítě dané školy.

Zavedení NiftyGridu do projektu se skládalo pouze z nakopírování zdrojových souborů do adresáře `libs` a vytvoření nového adresáře `app\grids` pro jednotlivé gridy a v poslední řadě vytvoření komponenty v příslušném presenteru.

V mém případě jde o jeden hlavní Grid pro editaci polí s větší prioritou jako jsou Firma, Žák, Vytvořeno, Třída, Obor, Požadavky, Stav, a jeden SubGrid, kde budu zobrazovat kontaktní údaje k uvedené praxi.

Pro hlavní Grid i SubGrid jsem vytvořil v adresáři `app\grids` nové třídy, které dědí už z vytvořené třídy `NiftyGrid\Grid`, a ty obsahují zdrojový kód konkrétních ovládacích prvků Gridu a jejich akcí. Jelikož chci umístit Grid na hlavní stránku, musím vytvořit ještě komponentu v `HomepagePresenteru`, což zajistí tento skript.

```
//HomepagePresenter
protected function createComponentTaskGrid()
{
return new TaskGrid($this->context->database->table("task"));}
```

Vytvořený Grid pracuje s daty následujícím způsobem, kdy využije třídy `NiftyGrid\NDataSource` a potřebná data přiřadí do proměnné `$source`.

```
protected function configure($presenter)

/*zdroj načte uvedené tabulky pomocí názvu pole, pakliže se
název shoduje, musí se rozlišit názvem tabulky např. task.id*/
{$source = new \NiftyGrid\NDataSource($this->tasks
->select('task.id, firma, ... , list.trida AS trida,
user.username, user.id AS user_id'));

$this->setDataSource($source);
```

Tím už jsou data pro Grid dostupná a lze je použít pro ovládací prvky Gridu.

Ukázka vytvoření pole Gridu a jeho konfigurace:

```
$this->setWidth("1130px"); //nastavení šířky Gridu
$this->setDefaultOrder("task.id DESC"); //řazení dle pole
$this->setPerPageValues(array(20, 50, 100));//stránkování
//vytvoření sloupce(pole, popisek, šířka, oříznutí textu
$this->addColumn('firma', 'Firma', '140px', 25)
      //možnost editace
      ->setTextEditable()
      //možnost filtrování
      ->setTextFilter()
      //automatické doplňování dle obsahu pole
      ->setAutocomplete(3);
```

Pro jednoduché ovládání Gridu jsem vytvořil tlačítka Editovat, Schválit, Smazat a Tisk do PDF a přiřadil jim příslušné akce.

Tlačítko Editovat

Funkce umožňuje editovat celý řádek Gridu a uloží případné změny do databáze.

```
//přidání tlačítka, popisek, nastavení CSS třídy = ikonka
$this->addButton(Grid::ROW_FORM, "Rychlá editace")
                 ->setClass("fast-edit");

//vybrání obsahu upravovaného řádku
$this->setRowFormCallback(function($values)
                          use ($self, $presenter)
{$vals = array(   "id" => $values["id"],
                  "firma" => $values["firma"],
                  "created" => $values["created"],
                  "user_id" => $values["username"],
                  "list_id" => $values["trida"],
                  "obor" => $values["obor"],
                  "pozadavky" => $values["pozadavky"],
                  "status" => $values["status"],);

//provedení updatu do databáze
$presenter->context->database->table('task')->where("id",
$vals["id"])->update($vals);

//potvrzení vykonané akce uživateli ve flash zprávě
$self->flashMessage("Záznam byl úspěšně uložen. ","grid-
successful");
```

Tlačítko Schválit resp. Zamítnout

Tuto funkci využije uživatel-editor při schvalování žádosti o praxi. Umožňuje přiřadit poli status hodnoty „schválena“ nebo „zamítnuta“.

```
$this->addButton("publish")
//nastavení podmínky pro zobrazení popisku tlačítka
->setLabel(function ($row) use ($self) {return $row['status'] ==
"čekající" || $row['status'] == "zamítnuta" ? "Schválit" :
"Zamítnout";})

//odkaz na AJAX, který se vykoná (např. publish!)
->setLink(function($row) use ($self){return $row['status'] ==
"čekající" || $row['status'] == "zamítnuta" ? $self-
>link("publish!", $row['id']) : $self->link("unpublish!",
$row['id']);})
```

```
      //změna ikonky dle podmínky
      ->setClass(function ($row) use ($self) {return $row['status'] ==
      "čekající" || $row['status'] == "zamítnuta" ? "publish" :
      "unpublish";});
```

Tlačítko Smazat

Funkce odstraní záznam ve vybraném řádku Gridu

```
$this->addButton("delete", "Smazat")

      //dle třídy nastavení ikonky
      ->setClass("delete")
      //odkaz na AJAX
      ->setLink(function($row) use ($self){return $self-
>link("delete!", $row['id']);})
      //potvrzující dialog
      ->setConfirmationDialog(function($row){return "Určitě chcete
odstranit praxi '$row[firma]'?";});
```

Tlačítko Tisk do PDF

Tato akce spustí Pdfpresenter a jeho metodu Pdfbrowser.

```
      $this->addButton("tisk", "Tisk PDF")
                        ->setClass("tisk")
                        ->setLink(function($row) use
($presenter){return $presenter->link("Pdf:Pdfbrowser", $row['id']);})
                        ->setAjax(FALSE);
```

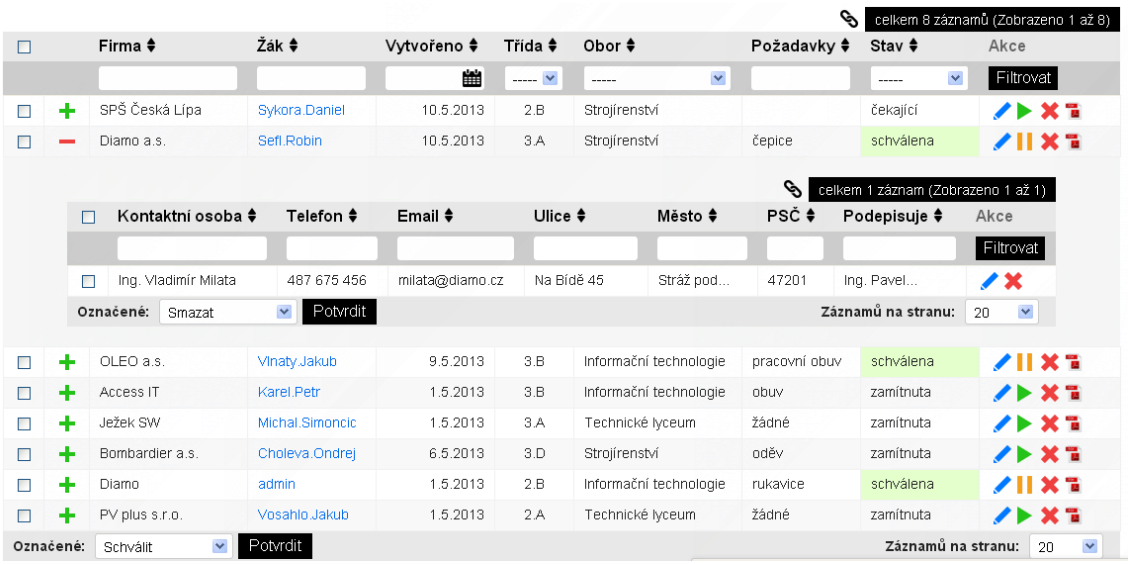
Dále Grid podporuje i tzv. **hromadné akce**, které pracují se všemi záznamy Gridu. Tuto eventualitu jsem využil pro možnost hromadně schválit či zamítnout praxi a samozřejmě též u hromadného smazání praxe. V samotném Gridu je využito metody `addAction` a funkce handleru `handleDelete`

```
      //metoda addAction
      $this->addAction("delete","Smazat")
      ->setCallback(function($id) use ($self){return $self-
      >handleDelete($id);})
      ->setConfirmationDialog("Určitě chcete smazat všechny vybrané
      záznamy?");

      //funkce Adleru
      public function handleDelete($id)
      //dotaz do databáze
      {$this->presenter->context->database->table('task')->where("id",
      $id)->delete(array("status", "obor" ,"firma" ));

      if(count($id) > 1) //podmínka rozeznává počet vybraných záznamů
      {$this->flashMessage("Vybrané praxe byly smazány.","grid-info");
      }else{
            $this->flashMessage("Praxe byla smazána.","grid-info");}
            $this->redirect("this"); //refresh nové stránky
```

3.4.2 Vykreslení hlavní stránky (Homepage)

Pro vykreslení hlavní stránky stačí upravit šablonu umístěnou v `Homepage\default.latte` následujícím způsobem:

```
//makro na propojení do základní šablony @layout.latte
{block #content}

//testování, zda je uživatel přihlášen při kladném vyhodnocení
//se vykreslí následující prvky šablony
{if $user->isLoggedIn()}

//nadpis a makro na vykreslení praxe zadané uživatelem
<h2>Má praxe</h2> {control userTasks}
{/if}

{snippet flashMessages} //vložení šablony pro zprávy uživateli
            <div  n:foreach="$flashes  as  $flash"  class="flash
{$flash->type}">{$flash->message}</div>
{/snippet}

//vykreslí formulář pro žádost oo praxi
<div n:class="praxe">
{control taskForm}

</div

<br>

// pouze pro admina a editora vykreslí Grid pro editaci praxí
{if $user->id <=2 }

{control taskGrid}{/if}{/block}
```



Obrázek 6 - Blok s DataGridem

3.4.3 Tisk do PDF (PdfPresenter)

Pro export HTML do univerzálního formátu Portable Document Format (PDF) lze v Nette využít knihovny PdfResponse *(15)*, založené na PHP knihovně mPDF. *(16)*

Po nakopírování knihovny `mPDF` a souboru `PdfResponse` do adresáře `libs` a drobné úpravě posledně jmenovaného souboru, kde se export nejdříve vykonal bez přípony souboru, se knihovna mohla začít používat.

V adresáři `app\presenters` jsem založil třídu `PdfPresenter` odvozenou ze třídy `Nette\Application\UI\Presenter` a jelikož knihovna umožňuje hned několik způsobů ukládání PDF, vytvořil jsem i patřičné metody `actionPdfbrowser($id)` a `actionPdfdownload($id)`. První zmiňovaná zobrazí PDF přímo do webového prohlížeče, druhá otevře `OpenPrintDialog`, přes který lze soubor uložit na disk.

V požadavku vedení školy bylo tisknutí dokumentů na hlavičkový papír, což jsem vyřešil metodou `setBackgroundTemplate`, které předám vytvořenou PDF šablonu hlavičkového papíru (Obrázek 7) a ta ji dá jako pozadí původní exportované šabloně.

Princip exportu je tedy následující:

1. vytvoření šablony latte pro tiskový dokument
2. předání šablony Presenteru, který do ní přes proměnné vloží data
3. přiřazení akce Presenteru danému ovládacímu prvku (např. tlačítku)

Konkrétně:

1. šablona `templates\PDF\Pdf.latte`

```
...
<h2 style="text-align: center;">Vzorový tisk do PDF!</h2>
<p style="text-align: center;">{$vzor}</p>
...
```

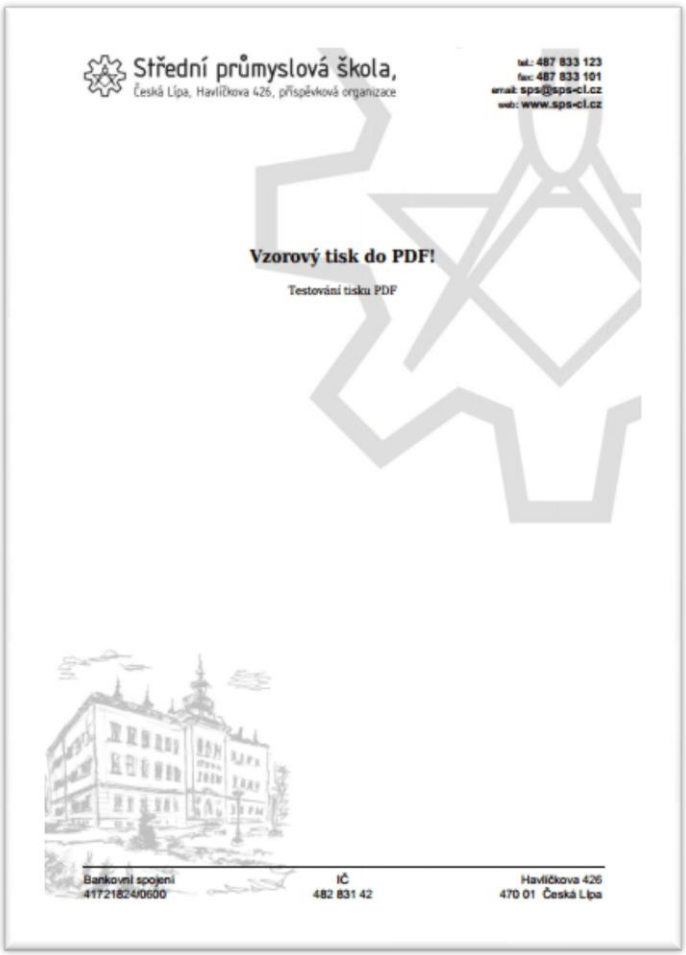
2. presenter `presenters\PdfPresenter.php`

```
class PdfPresenter extends Nette\Application\UI\Presenter {
//metoda, která zobrazí pdf v prohlížeči
public function actionPdfbrowser($id)
//předání správné šablony latte
{$template = $this->createTemplate()->setFile(APP_DIR .
"/templates/Pdf/myPdf.latte");
//předání dat přes proměnné
$template->vzor = "Testování tisku PDF";
$pdf = new \PdfResponse($template);
```

```
//vytvoření názvu dokumentu dle masky
$pdf->documentTitle = date("dmY") . " Praxe";
//metadata dokumentu (autor)
$pdf->documentAuthor = "SPŠ Česká Lípa";
//způsob uložení (INLINE= webový prohlížeč)
$pdf->setSaveMode(PdfResponse::INLINE);
//nastavení pozadí dokumentu dle šablony PDF
$pdf->setBackgroundTemplate(APP_DIR .
"/templates/Pdf/temp.pdf");
$this->sendResponse($pdf);}
}
```

3. přiřazení akce k tlačítku v Gridu

```
      ...
$this->addButton("print", "Tisk PDF")
      ->setClass("print")
      ->setLink(function($row) use ($presenter)
//odkaz na PdfPresenter a jeho metodu Pdfbrowser
{return $presenter->link("Pdf:Pdfbrowser", $row['id']);})
      ...
```



Obrázek 7 - Šablona hlavičkového papíru

3.5 Přihlašování (autentizace uživatele)

Ač autentizace uživatele zcela jistě zasahuje do všech vrstev aplikace, pro větší přehlednost ji uvádím v samostatné kapitole.

Při vytváření jsem postupoval podle obecných doporučení uvedených v dokumentaci a manuálech PHP *(17)* a tutoriálech k Frameworku Nette. *(18)*

3.5.1 Třída UserRepository

Tato třída je odvozena od rodičovské třídy `Repository` a jen rozšiřuje metodu `FindBy()` na `FindByName()`, která získá z datového modelu pole uživatelů dle jména.

```
public function findByName($username){
return $this->findBy(array('username' => $username))->fetch();}
```

3.5.2 Třída Authenticator

Tato třída je odvozena hned od několika tříd Nette (Nette\Security, Nette\Utils\Strings) a obsahuje standardní metody `authenticate()`, `setPassword()`a `calculateHash()`. Poslední zmiňovaná metoda řeší tzv. hashování, kdy dostane heslo jako parametr a pomocí funkce crypt(), konkrétně CRYPT_BLOWFISH ($2a$07$), jej vrátí pomocí sedmimístného $salt (tzv. „soli“) v zahashované podobě. *(19)* Tato varianta s funkcí crypt() doplněnou o salt by měla být bezpečnější než často užívaná hashovací funkce MD5.

Metoda `calculateHash`

```
public static function calculateHash($password, $salt = NULL){
      if ($password === Strings::upper($password)) {
      $password = Strings::lower($password);
      }
return crypt($password, $salt ?: '$2a$07$' .
Strings::random(22));
      }
}
```

3.5.3 Třída SignPresenter

V této třídě odvozené z `BasePresenteru` se definuje přihlašovací formulář `createComponentSignInForm()` s následnou svou validací `SignInFormSubmitted($form)`, která navíc při kladném vyhodnocení nastaví dobu

expirace session a přesměruje na hlavní stránku. V poslední řadě je zde uvedena metoda `actionOut()`, která odhlásí přihlášeného uživatele a přesměruje ho na úvodní přihlašovací formulář `Sign\in.latte`.

3.5.4 Třída UserPresenter

Pakliže by si chtěl uživatel změnit heslo, tak je tu pro něj třída `UserPresenter`. Zahrnuje formuláře `createComponentPasswordForm()` a `passwordFormSubmitted(Form $form)`, které využívají modelové třídy `UserRepository` a `Authenticator`. Při využití formulářových prvků Nette a jejich validačních pravidel je následující kód velmi snadno čitelný.

Validační podmínka minimální délky hesla (MIN_LENGTH = 6 znaků)

```
$form->addPassword('newPassword', 'Nové heslo:', 30)
    ->addRule(Form::MIN_LENGTH, 'Nové heslo musí mít alespoň %d
    znaků.', 6);
```

Validační podmínka shodujících se polí (EQUAL)

```
$form->addPassword('confirmPassword', 'Potvrzení hesla:', 30)
    ->addRule(Form::FILLED, 'Nové heslo je nutné zadat ještě
    jednou pro potvrzení.')
    ->addRule(Form::EQUAL, 'Zadaná hesla se musejí shodovat.',
    $form['newPassword']);
```

3.5.5 Šablony in.latte a password.latte

Tyto šablony zobrazují formuláře předchozích dvou Presenterů a opět pomocí maker jsou začleněny do výchozí šablony aplikace `@layout.latte`.

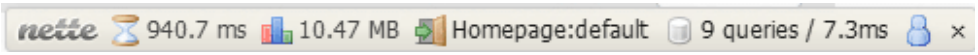
3.6 Vzhled aplikace (stylování CSS)

Při konzultacích s vedením školy nebyl sice na vzhled aplikace kladen žádný důraz, přesto jsem v základní šabloně pro většinu elementů definoval třídy a připojil soubory grid.css a print.css. Druhý jmenovaný je varianta černobílého vzhledu optimalizovaného pro tisk přímo z webového prohlížeče. Navíc jsem zachoval i původní styl icons.css pro ikony komponenty DataGridu. V rámci Nette soubory CSS patří do adresáře `www\css`. Výsledný vzhled je uveden v Příloze 4.

4. Otestování a ověření funkčnosti aplikace

Abych mohl otestovat aplikaci, naplnil jsem databázi fiktivními, ale typově podobnými daty. Celý vývoj aplikace probíhal v režimu localhost. Pro testování a kontrolu jsem využíval jak zmiňovaného Wampserveru nebo Netbeans IDE, tak především debuggovacího nástroje Nette, jež představuje knihovna Nette\Diagnostics\Debugger, mezi programátory známa jako tvz. „laděnka“. Tento užitečný nástroj běží ve dvou režimech, a to ve vývojovém nebo produkčním.

Při vývojovém režimu využívá tzv. plovoucího debuggovacího panelu (viz Obrázek 8), v kterém se standardně zobrazuje doba skriptu, maximum přidělené paměti či vykonané dotazy nad databází. Dále se k tomuto nástroji dají doinstalovat různé další doplňky jako CallbackPanel, ComponentTreePanel, SessionPanel, RequestPanel aj. Avšak během vývoje aplikace nebylo potřeba těchto doplňků využít.



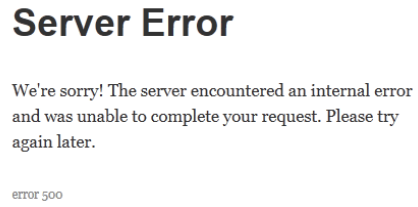
Obrázek 8 - Debugger Bar v Nette

Pokud se jednalo o syntaktické chyby PHP, tak většinu je odchytilo už samotné vývojové prostředí Netbeans, pakliže šlo o chyby logické, velkou důležitost sehrála zmiňovaná laděnka, která přehledně vykreslila daný problém (viz Obrázek 9) i určila místo ve zdrojovém kódu (viz Obrázek 10).

Obrázek 9 - Nette Laděnka s pojmenovanou chybou SQL

Obrázek 10 - Nette Laděnka odkazující na místo zdrojového kódu

Po dokončení jsem aplikaci nahrál na server a funkčnost testoval v produkčním režimu. Pokud došlo k nějaké chybě, tak zde Nette informuje uživatele pouze univerzálním výpisem Server error 500 (viz Obrázek 11) a z bezpečnostních důvodů dál nic neuvádí. Ovšem všechny chyby podrobně zaloguje a vypíše do adresáře `log`, navíc vytvoří stejnou HTML stránku jako v testovacím vývojovém režimu.



Obrázek 11 - Laděnka v produkčním režimu

Jedinou malou komplikací bylo odhalování chyb v AJAXu, kdy změny probíhají pouze v prohlížeči, ale chybu jsem rychle odhalil pomocí zakomentování problematické zdrojového kódu a postupného odkomentovávání, čili jsem nepotřeboval žádný další debuggovací nástroj.

Při ověření uživatelského prostředí a logiky aplikace jsem nenašel žádné chyby, kromě několika překlepů u hlášek při validaci formulářů, jež byly obratem opraveny.

Zobrazení aplikace bylo testováno v prohlížečích Internet Explorer 7,8,9, Opera 9, 10, Opera Mobile 12, Google Chrome 25, 26 a Firefox 20.0. Jedinou malou komplikací bylo zarovnání popisků formuláře žádosti u IE7, což jsem řešil úpravou souboru CSS.

Ošetření vstupů a další bezpečnostní prvky aplikace *(15)*

Samotný framework Nette řeší problematiku hned několika možných útoků na webové aplikace, jako jsou:

- Cross-Site Scripting (XSS)
- Cross-Site Request Forgery (CSRF)
- URL – attack
- Session hijacking, session stealing, session fixation

Závěr

Výsledkem této práce je jednoduchá webová aplikace, která by měla zjednodušit řízení agendy praxí na střední škole, především sběr dat od žáků a snadnější a přehlednější manipulaci s daty koordinátorem praxí. Přínosným počinem bylo především zmapování celé problematiky přímo na střední škole a společně s koordinátorem praxí vytvoření návrhu modelu, jak by vše mohlo v praxi fungovat. Na základě toho jsem pečlivě navrhl databázovou strukturu, která by měla být schopna operativně reagovat na nové požadavky vedení školy a která by nebrzdila budoucí vývoj stávající aplikace o další moduly.

Při vývoji aplikace jsem vůbec poprvé použil rychle expandující český framework Nette, který mi usnadnil práci především při řešení bezpečnosti aplikace, kdy v podstatě sám spolehlivě řeší problematiku Cross Site Scripting (XSS) i Cross-Site Request Forgery (CSRF), dále zjednodušuje validaci formulářů i vykreslování šablon. Je však znát, že je to opensource a některé doplňky obsahují chyby, které jsou pak těžko odhalitelné a spíše práci znesnadňují.

Aplikace je v tomto školním roce 2012/2013 v testovacím režimu a připravena k produkčnímu nasazení. Vzhledem k tomu, že sám zadavatel dále počítá s postupným rozšiřováním funkčnosti aplikace, tak nebylo cílem práce navrhnout nějaké definitivní řešení, ale spíše v objektovém pojetí vytvořit základní logiku aplikace.

Budoucím směřování práce by mohlo být především doplnění aplikace o modul, který bude řešit při přiřazování praxe dopravní dostupnost pracoviště z bydliště žáka či jeho docházkový systém, a dále se také vybízí více sjednotit agendu školy a propojit vše s nově vznikajícím školním webem nebo s evidenčním systémem Bakalář.

Seznam použité literatury

1. Vrána, Jakub. Využití databázových indexů. *root.cz.* [Online] 22. červenec 2003. [Citace: 15. duben 2013.] http://www.root.cz/clanky/vyuziti-databazovych-indexu/.

2. Kosek, Jiří. *PHP - Tvorba interaktivních internetový aplikací.* Praha : Grada Publishing, 1999.

3. Kafka, Aleš. Informační systém Masarykovy univerzity. *IS.MUNI.CZ.* [Online] 28. květen 2012. [Citace: 30. duben 2013.] http://is.muni.cz/th/359535/fi_b/thesis.pdf.

4. Vrána, Jakub. MySQL 4.1 – kódování. *php.vrana.cz.* [Online] 2. prosinec 2012. [Citace: 30. duben 2013.] http://php.vrana.cz/mysql-4-1-kodovani.php.

5. Grundl, David. Nette Framework: zvyšte svoji produktivitu. *Zdrojak.cz.* [Online] 10. Březen 2009. [Citace: 30. Duben 2013.] http://www.zdrojak.cz/clanky/nette-framework-zvyste-svoji-produktivitu/.

6. MVC aplikace & presentery. *Nette.org.* [Online] 13. březen 2012. [Citace: 2. květen 2013.] http://doc.nette.org/cs/presenters#toc-sablony.

7. Databáze a model. *Nette.org.* [Online] 12. březen 2012. [Citace: 2. květen 2013.] http://doc.nette.org/cs/book/database.

8. Dependency Injection. *Nette.org.* [Online] 3. duben 2012. [Citace: 23. duben 2013.] http://doc.nette.org/cs/dependency-injection.

9. Komponenty. *Nette.org.* [Online] 21. březen 2012. [Citace: 8. květen 2013.] http://doc.nette.org/cs/book/components.

10. Kuchař, Jan. PdfResponse. *Nette.org.* [Online] 24. únor 2013. [Citace: 4. květen 2013.] http://addons.nette.org/cs/pdfresponse.

11. Back, Ian. mPDF. *PHP class to generate PDF files from HTML with Unicode/UTF-8 and CJK support.* [Online] 26. leden 2013. [Citace: 3. květen 2013.] http://www.mpdf1.com/mpdf/index.php.

12. Vrána, Jakub. PHP triky - Přihlašování uživatelů. *php.vrana.cz.* [Online] 31. srpen 2005. [Citace: 13. duben 2013.] http://php.vrana.cz/prihlasovani-uzivatelu.php.

13. Grundl, David. Přihlašování uživatelů. *Nette.org.* [Online] 29. duben 2013. [Citace: 7. květen 2013.] http://doc.nette.org/cs/book/authentication.

14. PHP.net. PHP: crypt - Manual. *PHP.net.* [Online] 10. květen 2013. [Citace: 11. květen 2013.] http://cz1.php.net/crypt%28%29.

15. Grundl, David. Zabezpečení před zranitelnostmi. *Nette.org.* [Online] 11. duben 2012. [Citace: 2. květen 2013.] http://doc.nette.org/cs/vulnerability-protection.

16. Stažení a instalace. *Nette.org.* [Online] 12. únor 2008. [Citace: 12. březen 2013.] http://doc.nette.org/cs/installation.

17. MySQL. *Wikipedia.org.* [Online] 7. březen 2013. [Citace: 14. duben 2013.] http://cs.wikipedia.org/wiki/MySQL.

18. PHP : Hypertext Preprocessor. *Php.net.* [Online] 13. duben 2013. [Citace: 14. duben 2013.] http://php.net/.

19. Wampserver. *Wampserver.com.* [Online] 2013. [Citace: 14. duben 2013.] http://www.wampserver.com/en/.

20. Grundl, David. Nette Framework: zvyšte svoji produktivitu. *Zdrojak.cz.* [Online] 3. březen 2009. [Citace: 30. duben 2013.] http://www.zdrojak.cz/clanky/nette-framework-zvyste-svoji-produktivitu/.

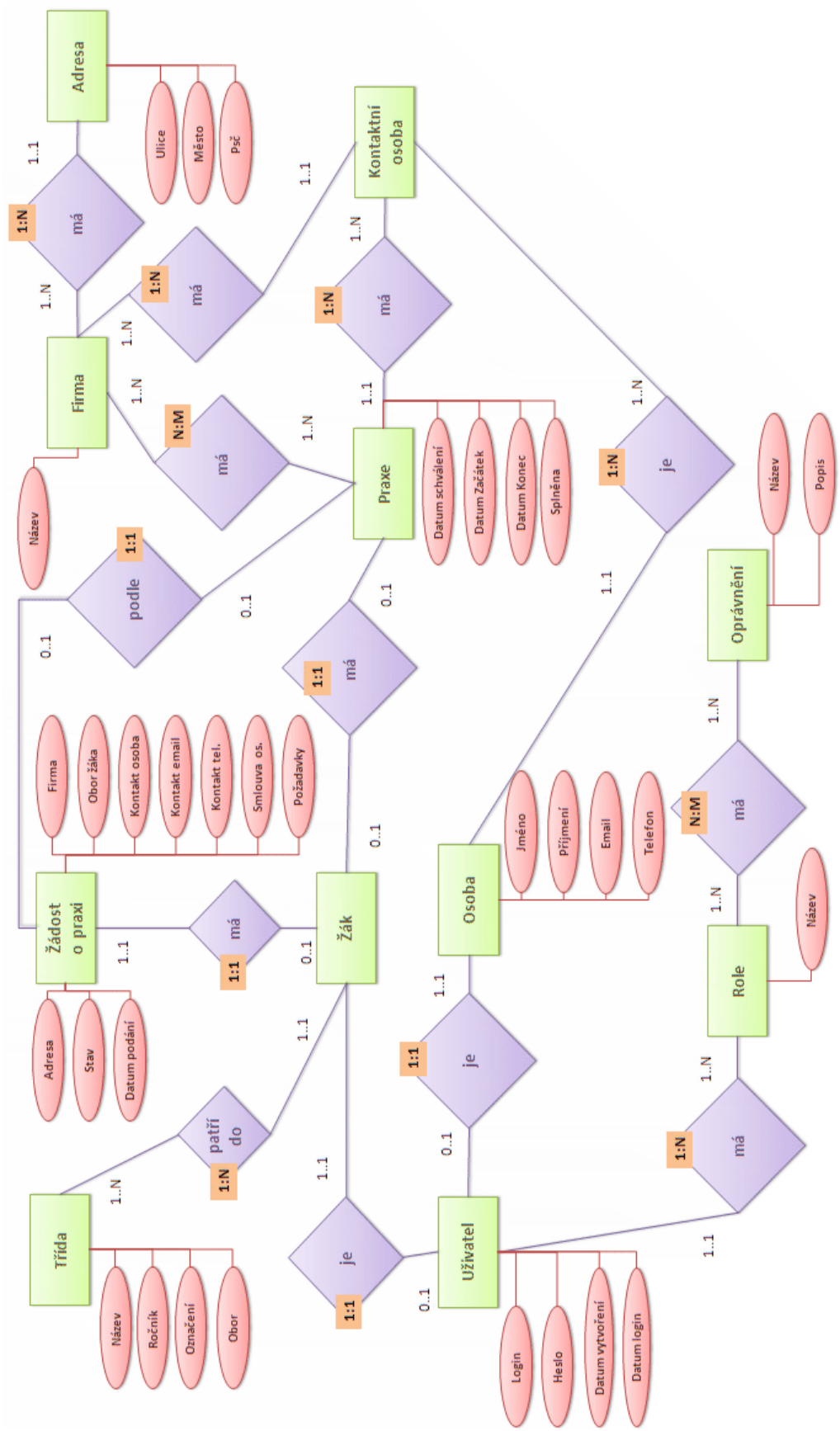
21. O frameworku. *Nette.org.* [Online] 2013. [Citace: 14. duben 2013.] http://nette.org/cs/.

22. Kulhan, Jakub. Normalizace relačních databází. *programujte.com.* [Online] 23. červenec 2008. [Citace: 15. duben 2013.] http://programujte.com/clanek/2008071900-normalizace-relacnich-databazi/.

Seznam obrázků a schémat

Seznam tabulek

Příloha 1 – ER –diagram databáze



Příloha 2 – Požadavky frameworku Nette na webserver

Požadavek	Vysvětlení
PHP version	Lze použít obě verze, doporučujeme PHP 5.3.
.htaccess file protection	Kontroluje, zda lze pomocí .htaccess zakázat přístup do chráněných složek
.htaccess mod_rewrite	Kontroluje, zda je k dispozici mod_rewrite pro routování
Function ini_set()	Přítomnost této funkce je důležitá z bezpečnostních důvodů
Function error_reporting()	Přítomnost této funkce je důležitá z bezpečnostních důvodů
Function flock()	Vyžaduje cache
Register_globals	Nebezpečná konfigurační direktiva PHP, která musí být vypnutá
Zend.ze1_compatibility_mode	Kompatibilita s PHP 4, musí být vypnutá
Session auto-start	Z bezpečnostních důvodů je doporučeno nepoužívat
Reflection extension	Extenze PHP vyžadovaná frameworkem
SPL extension	Extenze PHP vyžadovaná frameworkem
PCRE extension	Extenze PHP vyžadovaná frameworkem
ICONV extension	Extenze PHP vyžadovaná frameworkem
PHP tokenizer	Extenze PHP vyžadovaná frameworkem
PDO extension	Extenze PHP vyžadovaná Nette\Database
Multibyte String extension	Extenze PHP vyžadovaná funkcemi Strings::lower() a upper()
Multibyte String function overloading	Nebezpečná konfigurační direktiva PHP, musí být vypnutá
Memcache extension	Extenze PHP podporovaná úložištěm cache
GD extension	Extenze PHP vyžadovaná Nette\Image
Bundled GD extension	Extenze PHP vyžadovaná metodamiNette\Image::filter() a rotate()
Fileinfo extension or mime_content_type()	Funkce používané k MIME-type detekci uploadovaných souborů

převzato z http://doc.nette.org/ cs/requirements

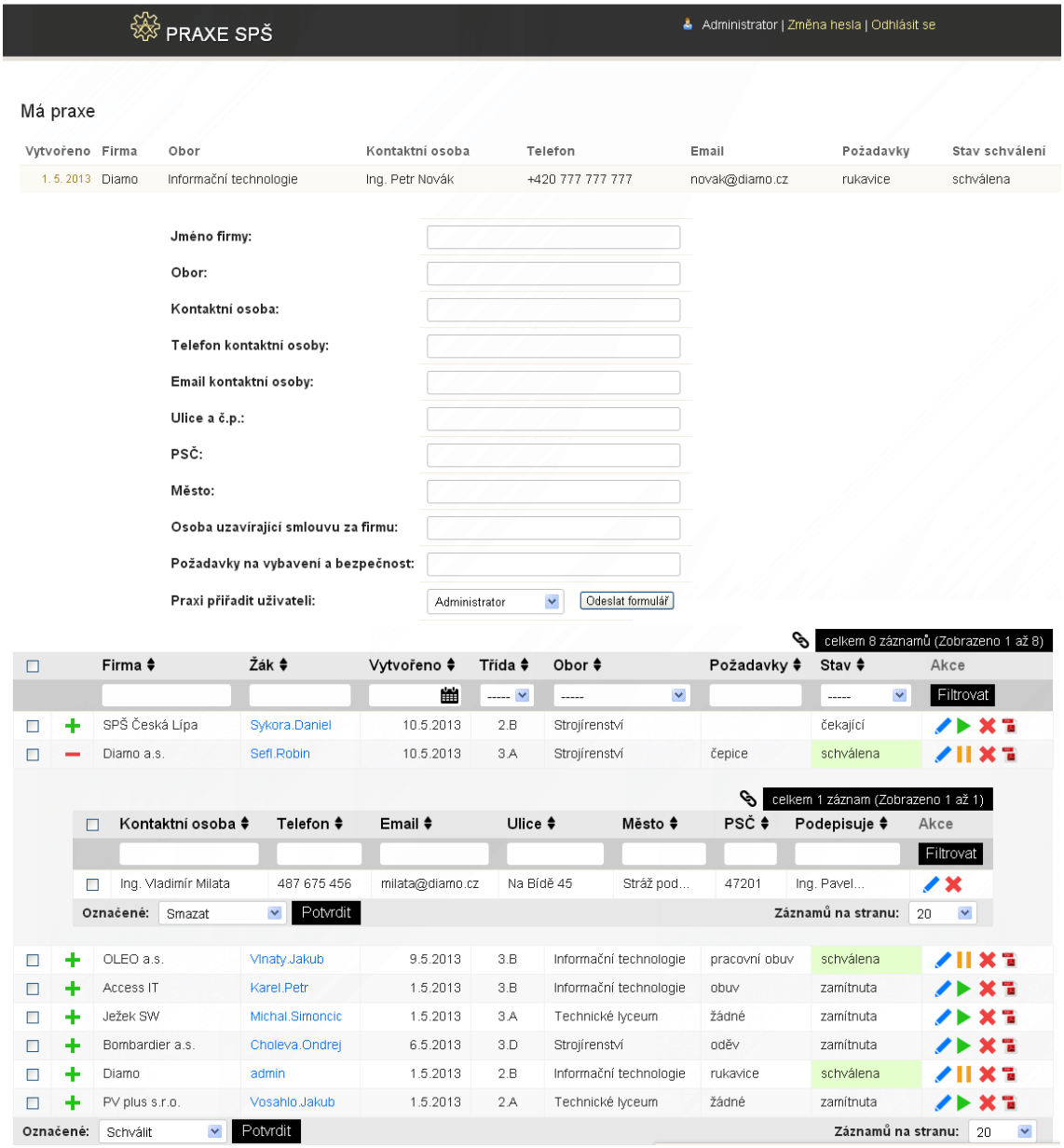
Příloha 3 – Struktura souborů a složek frameworku Nette

```
www/                    ← kořenový adresář webového serveru
└── nette/              ← rozbalený archív
    ├── API-reference/  ← offline verze API dokumentace
    ├── client-side/    ← skript pro validaci formulářů
    ├── examples/       ← příklady, rovnou spustitelné
    ├── Nette/          ← samotný framework
    │   ├── Application/
    │   ├── Caching/
    │   ├── ...
    │   └── loader.php
    │
    ├── Nette-minified/ ← minimalizovaná verze frameworku
    │   └── nette.min.php
    │
    ├── sandbox/        ← předpřipravený projekt
    │   ├── app/
    │   ├── libs/
    │   ├── log/
    │   ├── temp/
    │   └── www/
    │
    └── tools/          ← užitečných nástrojů
        ├── Code-Checker/
        ├── Code-Migration/  (jen v distribuci pro PHP 5.3)
        ├── Ini2Neon/
        └── Requirements-Checker/
```

(16)

převzato z http://doc.nette.org/cs/installation

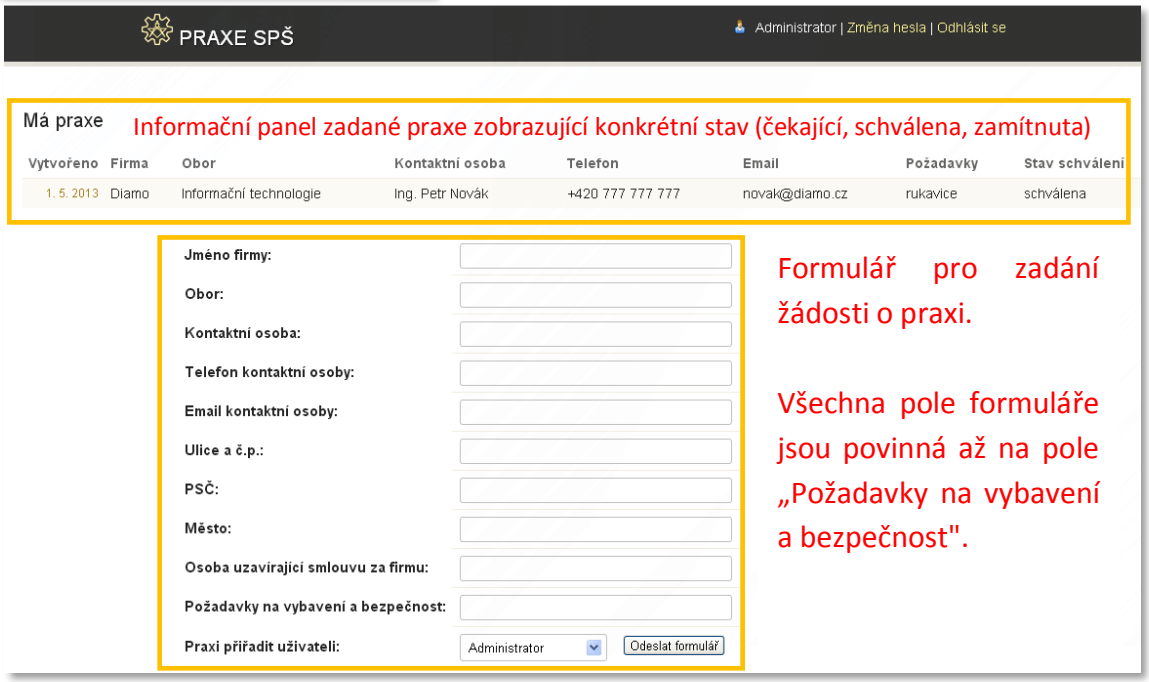
Příloha 4 – Výsledný vzhled webové aplikace



Příloha 5 – Stručný manuál k webové aplikace

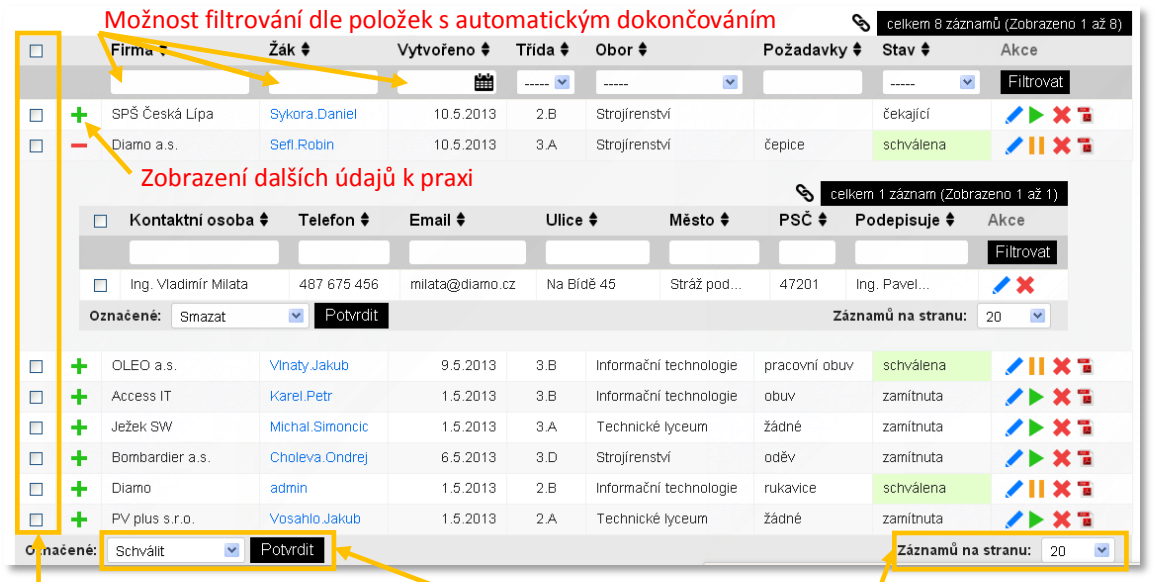
Do webového prohlížeče zadejte adresu:
http://www.sps-cl.cz/praxe

Přihlašovací údaje jsou generovány z evidenčního systému Bakaláři. V případě komplikací kontaktujte správce sítě.



Formulář pro zadání žádosti o praxi.

Všechna pole formuláře jsou povinná až na pole „Požadavky na vybavení a bezpečnost".



Hromadný výběr položek

Hromadné akce se záznamy

Možnost stránkování

Editace praxe

Schválení praxe

Zamítnutí praxe

Odstranění praxe

Nástroj kalendář

Tisk sestavy do PDF