

Databázový systém pro správu výzkumných projektů

Database management system for research projects

Bc. Pavel Gavenda



Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
akademický rok: 2010/2011

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Bc. Pavel GAVENDA**
Osobní číslo: **A09469**
Studijní program: **N 3902 Inženýrská informatika**
Studijní obor: **Informační technologie**

Téma práce: **Databázový systém pro správu výzkumných projektů**

Zásady pro vypracování:

1. Analyzujte problematiku a vypracujte literární rešerši na dané téma.
2. Navrhněte strukturu databáze systému a aplikace.
3. Vytvořte databázi a aplikace a popište postup řešení.
4. Zajistěte správu a zabezpečení databáze.
5. Implementujte systém.
6. Vytvořte podporu uživatelů – helpdesk.

Rozsah diplomové práce:

Rozsah příloh:

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

1. CASTAGNETTO, Jesus. PHP : Programujeme profesionálně. Přeložil Luděk Roubíček. 2., opr. a aktualiz. vyd. Praha : Computer Press, 2002. 656 s. ISBN 80-251-0046-4.
2. CONOLLY, Thomas. Mistrovství – databáze : profesionální průvodce tvorbou efektivních databází. Přeložil Vilém Gutfreund. Vyd. 1. Brno : Computer Press, 2009. 584 s. ISBN 978-80-251-2328-7.
3. RESIG, John. JavaScript a Ajax : Moderní programování webových aplikací. Přeložil Ondřej Baše. Vyd. 1. Brno : Computer Press, 2007. 360 s. ISBN 978-80-251-1824-5.
4. PHP [online]. 2011 [cit. 2011-01-31]. Manual. Dostupné z WWW: [www.php.net/manual/en/].
5. MySQL 5.1 [online]. 2005 [cit. 2011-01-31]. Reference Manual. Dostupné z WWW: [dev.mysql.com/doc/refman/5.1/en/].
6. Nette Framework [online]. 2008 [cit. 2011-01-31]. Dokumentace. Dostupné z WWW: [nette.org/cs/dokumentace].

Vedoucí diplomové práce:

doc. Ing. Zdenka Prokopová, CSc.

Ústav počítačových a komunikačních systémů

Datum zadání diplomové práce:

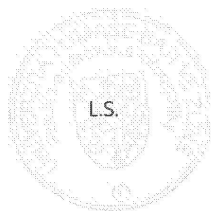
24. února 2011

Termín odevzdání diplomové práce:

18. května 2011

Ve Zlíně dne 24. února 2011

prof. Ing. Vladimír Vašek, CSc.
děkan



doc. Mgr. Roman Jašek, Ph.D.
ředitel ústavu

ABSTRAKT

Diplomová práce se zabývá tvorbou databázového systému pro správu výzkumných projektů. Teoretická část popisuje základní koncept modelu systému s popisem předešlého stavu a návrhem vhodné formy nového informačního systému včetně seznámení s relačními databázemi a návrhem konkrétního modelu dat, dále popisuje použitou architekturu aplikace a skriptovací technologie. Praktická část popisuje převedení modelu databáze na schéma, popis jednotlivých částí aplikace, rozbor databázové vrstvy aplikace a konfigurace připojení. Podstatnou část tvoří uživatelská příručka informačního systému pro správu výzkumných projektů.

Klíčová slova: databázový systém, relační databáze, informační systém, PHP, JavaScript, MVP

ABSTRACT

This thesis deals with the creation of a database system for managing research projects. The theoretical part describes the basic concept of a model, describing the previous state and the proposal should form a new information system, including introduction to relational databases and the design of a specific data model, then describes the architecture used by applications and scripting technologies. The practical part describes the transfer of the model database into schema, descriptions of applications, analysis of the database layer of application and connection configuration. An important part is the user's manual system to manage research projects.

Keywords: database system, relational databases, information system, PHP, JavaScript, MVP

Děkuji své vedoucí diplomové práce doc. Ing. Zdeně Prokopové, CSc. za její aktivní přístup při komunikaci a připomínky k mé práci. Poděkovat také musím své rodině za velkou podporu ve studiu.

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové/bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová/bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji diplomovou/bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou/bakalářskou práci nebo poskytnout licenci k jejímu využití jen s předchozím písemným souhlasem Univerzity Tomáše Bati ve Zlíně, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše);
- beru na vědomí, že pokud bylo k vypracování diplomové/bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové/bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové/bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně

.....
podpis diplomanta

OBSAH

ÚVOD.....	9
I TEORETICKÁ ČÁST	11
1 ANALÝZA PROBLEMATIKY	12
1.1 PŮVODNÍ STAV	14
1.2 PLÁNOVANÝ STAV	14
2 NÁVRH DATABÁZE A APLIKACE	16
2.1 NÁVRH DATABÁZE	18
2.1.1 Metodika tvorby databáze	18
2.1.2 Relační model databáze.....	21
2.1.3 Relační datový model.....	22
2.1.4 Relační tabulky.....	23
2.1.5 Databáze výzkumných projektů	25
2.2 NÁVRH INFORMAČNÍHO SYSTÉMU	26
2.2.1 Uživatelské účty	26
2.2.2 Administrační rozhraní.....	27
2.2.3 Agenda výzkumných projektů.....	27
2.2.4 Výzkumný projekt	27
2.3 STRUKTURA APLIKACE	28
2.4 SOFTWARE POŽADAVKY.....	29
2.4.1 MySQL.....	30
3 SKRIPTOVACÍ JAZYKY PHP, JAVASCRIPT	31
3.1 PHP.....	31
3.1.1 Vývoj jazyka.....	32
3.1.2 Nette Framework.....	33
3.1.3 MVC a MVP architektura	35
3.2 JAVASCRIPT	37
3.2.1 jQuery.....	37
3.2.2 Ajax	38
3.2.3 jQuery UI.....	40
II PRAKTICKÁ ČÁST	41
4 VYTVOŘENÍ DATABÁZE A APLIKACE	42
4.1 TVORBA SCHÉMATU DATABÁZE	42
4.2 TVORBA APLIKACE ARCHITEKTUROU MVP	44
4.2.1 Vrstva modelu	44
4.2.2 Prezentační vrstva	48
4.2.3 Pohledy – views	52
4.2.4 Ajax	53
4.3 DIBI DATABÁZOVÁ VRSTVA PRO PHP.....	54
5 SPRÁVA A ZABEZPEČENÍ DATABÁZE	57

6	IMPLEMENTACE SYSTÉMU	60
6.1	TESTOVÁNÍ.....	60
6.2	INSTALACE SYSTÉMU	60
6.3	NASTAVENÍ DATABÁZE A APLIKACE	61
6.4	ÚDRŽBA SYSTÉMU	62
6.5	ZÁLOHOVÁNÍ.....	63
7	PODPORA UŽIVATELŮ – HELPDESK.....	64
7.1	PŘIHLÁŠENÍ	64
7.2	AGENDA VÝZKUMNÝCH PROJEKTŮ	65
7.3	PROJEKT	70
7.3.1	Zobrazení projektu	71
7.3.2	Návrh výzkumného projektu v PDF.....	72
7.3.3	Editace projektu.....	73
7.3.4	Stav projektu	74
7.3.5	Zadavatel projektu.....	74
7.4	ADMINISTRACE.....	75
7.4.1	Uživatelé	75
7.4.2	Poskytovatelé finančních prostředků.....	76
7.4.3	Předkladatelé	77
7.4.4	Měny.....	78
7.4.5	Záloha databáze	78
	ZÁVĚR	79
	ZÁVĚR V ANGLIČTINĚ.....	81
	SEZNAM POUŽITÉ LITERATURY.....	83
	SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK	84
	SEZNAM OBRÁZKŮ	85

ÚVOD

Společnost je závislá na informacích od pradávna, její nároky na možnosti získat je se zvětšují a souběžně s tím jak populace roste, se zvětšuje i množství informací. V dávné historii se informace nejprve začali šířit mezi jednotlivými osobami v podobě mluveného slova, ale tento způsob ji nedokázal zachovat nepozměněnou a časem došlo k její degradaci, případně úplnému ztracení. Přirozeným důsledkem bylo vyvinutí různých způsobů zachycení informace v nejranějších stádiích pomocí maleb, obrazů a s příchodem písma došlo k velkému rozmachu zachování informací v psané podobě, se kterou se setkáváme dodnes. V historii patřila schopnost psát těm nejvzdělanějším, dnes ji považujeme za běžnou úroveň vzdělanosti. Stejně tak jsou dnes ve společnosti běžné textové materiály dnes již v tištěné podobě v mnoha různých formách od velkých knih po informační letáky.

Dnešní společnost se na každém kroku setkává s novými informacemi a není v silách jednotlivce pojmout vše. Proto vyvstává problém nejen informace shromažďovat, ale dále se v nich orientovat a využít je. S tímto problémem se setkáváme běžně, v dnešní elektronice protkané době, kdy máme na dosah ruky velké množství informací a je na nás jak je využijeme. K tomu, abychom mohli s informacemi efektivněji pracovat, vznikají takzvané informační systémy. Informační systémy jsou dnes běžnou součástí našeho života, běžně využíváme jejich služeb například v podobě různých kartoték, seznamů a jiných ne nutně elektronických systémů, ale právě v papírové podobě, i když dnešním trendem je převádět je do elektronické podoby. Účelem těchto systémů je, abychom byli schopni z velkého množství dat dostat informace v pro nás použitelné podobě, systém by měl interpretovat vstupní data na lidem srozumitelné informace, tak abychom byly schopni rychle obdržet požadovanou informaci, aniž bychom museli znát celý soubor dat, ze kterých tato informace vznikla.

Elektronické informační systémy se dnes stávají běžnou součástí našeho dne díky schopnosti automatizovaně provádět úkony bez zásahu lidské ruky, jsou schopny sbírat vstupní data a transformovat je na výstup, samozřejmě toho dělají většinou mnohem více. Systém po sběru dat, ať už manuálním zadáním nebo například pravidelným automatizovaným měřením, může provést jejich vyhodnocení, uložení, odeslání zprávy na některý z výstupů či jiné akce. Jako příklad můžeme uvést různé informační tabule, kterými jsou velká města protkána a slouží například k měření, záznamu a zobrazení

meteorologických hodnot, systém elektronických jízdnicích řádů, kdy je systém částečně automatizován – zobrazení jízdnicích řádů na informačních tabulích, tak je i závislý na lidských vstupech – hlášení výluk, závad, změna jízdnicích řádů.

Informační systémy jsou pro dnešní společnost nezbytnou součástí, která ji ulehčuje práci s daty. Vznik těchto systémů je logickým vyústěním nutnosti pracovat s velkým objemem dat a do budoucna se tyto systémy budou vytvářet a profilovat pro nejrůznější odvětví lidského dění, tak abychom mohli co nejjednodušeji spravovat a poskytovat informace. Elektronické systémy nám v posledních desetiletích nabízí možnost velmi jednoduše ukládat velké množství dat a tak jsme schopni opět zefektivnit naši práci, na druhou stranu je potřeba mít dobře zpracované výstupy z těchto systémů, abychom dostávali výsledky s velkou informační hodnotou, které můžeme dále využít.

I. TEORETICKÁ ČÁST

1 ANALÝZA PROBLEMATIKY

Informační systém, kterým databázový systém pro správu výzkumných projektů je, má být složen z několika částí, které spolu velmi úzce spolupracují a pro celek jsou nutné z logického i funkčního hlediska. Základem tohoto systému je, jak z názvu vyplývá, samotná databáze, která tvoří základní prvek systému pro uložení dat a jejich dalšího zpracování. Podle Thomase Conollyho databáze je: *„jediné, případně veliké, úložiště dat, která mohou být používána současně mnoha odděleními a uživateli. Všechna data, která tito uživatelé požadují, jsou integrována s minimálním množstvím duplikací. Důležité je, že databázi obvykle nevlastní žádné oddělení nebo uživatel, nýbrž je sdíleným zdrojem společnosti“*. [2] Z toho plyne, že jde o jednotné úložiště dat nezávislé na uživateli a prostředí odkud k němu přistupuje. To je velká výhoda databází, protože nám umožňují přistupovat k jednotnému úložišti, takže se plně odbourává problém s decentralizováním dat a jejich duplicitou při existenci více úložišť. S tímto problémem se běžně setkáváme při práci v týmech bez využití metod pro spolupráci. Jednotná databáze tak zajistí aktuální stav informací pro všechny uživatele v daný okamžik, aniž by uživatel musel řešit synchronizaci více zdrojů. Tak můžeme nabídnout totožné aktuální informace na několika místech zároveň a zároveň pracovat s jedním úložištěm. Databáze je z pravidla provozována tak, aby byla neustále dostupná a vzdáleně přístupná. Uživatel se většinou nedostává do kontaktu s fyzickým úložištěm dat a přistupuje k datům vzdáleně pomocí odpovídajícího softwaru, respektive komunikačního protokolu. Přístup k datům bývá zpravidla také omezen, většinou mají pro nás data určitou hodnotu a chceme přístup k nim kontrolovat a povolit jen pro určitou skupinu osob a technického zařízení, které s ní budou moct pracovat a i dále podle přístupových údajů můžeme rozlišit přístupnost jednotlivých dat pro konkrétní uživatele. Databáze nám tedy nabízí jednoduše přístupné vzdálené centrální úložiště dat s omezením přístupu podle nastavených práv, slouží nám k ukládání informací o projektech a všech dalších informací, které se během fungování systému mohou měnit.

Nedílnou součástí našeho databázového systému je vytvoření aplikace pro přístup k databázi, tak aby se dala jednoduše spravovat. Samotný databázový systém nenabízí uživatelsky přívětivé rozhraní pro práci s daty, proto je potřeba tyto data pomocí aplikace uživateli vhodně a srozumitelně interpretovat, tak aby využití systému práci zjednodušilo, nikoliv komplikovalo. Aplikace má tedy nabídnout uživateli jednoduché prostředí – rozhraní pro přístup k databázi, je tedy důležitým prostředníkem mezi nimi. Aplikace je

tvořena, stejně jako struktura databáze, na základě informací poskytnutých žadatelem o systém, tak aby odpovídala daným představám. Aplikace by měla splňovat požadavky pro jednoduchou přístupnost dat a intuitivního použití systému, tak aby jej uživatel mohl využít s co nejmenší předchozí znalostí, u tohoto samozřejmě závisí na tom, jak moc jsou specifické požadavky zadavatele na funkce aplikace. S aplikací by také měla být dodána příručka k použití, případně provedeno školení uživatelů a úvodní nastavení systému.

Dnes se prosazují systémy, které stejně jako samotná databáze umožňují vzdálený přístup. Uživatelé se stávají stále více mobilní, pracují na více zařízeních a vyžadují od aplikací fungování co nejméně závislé na jejich zařízení. Z tohoto důvodu se stále více dostávají do popředí aplikace vytvořené pro webové prohlížeče, protože nabízí jednoduchou instalaci klientského rozhraní a společně s využitím sítě internet přístup z velké části světa. Webová aplikace má velkou výhodu v tom, že samotná aplikace běží vzdáleně a přes prohlížeč se k ní připojujeme. Protože prohlížeč je dnes běžnou součástí nejen osobních počítačů různých platform a operačních systémů, ale i různých přenosných zařízení můžeme nabídnout přístup k aplikaci s velkého množství zařízení. V tomto ohledu pro spoustu informačních systémů je webový klient skvělým řešením, protože se nemusí řešit vývoj klasických aplikací, které jsou specifické nejen pro různá zařízení, ale dnes i verze operačních systémů. Na druhou stranu takovéto systémy volíme pro případy, kdy nejsou vyžadovány speciální požadavky například spolupráce s hardwarem nebo využití konkrétních vlastností zařízení. Pro potřeby zobrazení informací ze systému, možnosti výstupů a vstupů do databáze je webový prohlížeč ideálním řešením.

Celý systém funguje na principu propojení několika komunikačních událostí mezi několika systémy – zařízeními. Uživatel skrze prohlížeč odešle požadavek na server na kterém je naše aplikace a ta mu zpět odpoví. V rámci vyřízení odpovědi aplikace podle požadavku ještě obstará komunikaci s dalším zařízením – databází, kterou požádá o potřebná data, vygeneruje výstupní kód a odešle na uživatelské zařízení, kde prohlížeč kód interpretuje. Toto řešení má výhodu v tom, že existuje jediná aplikace, ke které všichni přistupují, a je možné ji jednoduše upravovat, řešit její chyby, kontrolovat přístup k aplikaci, monitorovat kritické situace a pády a další. Totéž platí i o databázi, takže lze jednoduše spravovat tyto klíčové součásti systému. Jediné co nemůže kontrolovat a co závisí na uživateli je jestli použije podporovanou verzi prohlížeče, i když vyvinutá aplikace by měla podporovat všechny běžné prohlížeče, a hlavně přístup skrze internet k síti na které běží serverová

aplikace, samozřejmě je často využíván tento systém i pro vnitropodnikové intranetové aplikace, kde odpadá problém s připojením přes veřejnou síť.

1.1 Původní stav

Systém v původním stavu existuje jak v listinné podobě, tak v jednoduché elektronické podobě jako tabulky v příslušném editoru. Listinná podoba bude částečně zachována i nadále z důvodu zachování dokumentů s podpisy a razítky, nový systém tuto formu nenahrazuje a proces schvalování bude probíhat dále na základě tištěných dokumentů. V těchto dokumentech se těžko rychle hledají konkrétní informace, jedná se o velké množství tištěných dokumentů, proto v tuto chvíli jsou vedeny i záznamy o projektech v elektronické podobě v tabulkovém editoru. Původně jsou tabulky ručně vytvářeny, tak aby se dalo jednodušeji orientovat v projektech, ale tato ruční tvorba není moc uživatelsky příjemná a změny v tabulkách jsou časově náročné, tedy snižují produktivitu práce. I přes to elektronické tabulky nabízí aspoň jednoduchý dohled nad projekty a umožňují rychlejší přehled nad základními vlastnostmi projektů. Pokud je potřeba hledat podrobnější informace, je nutné hledat v tištěné verzi. Dalším problémem je přístup k vytvořeným tabulkám, které má tvůrce uloženy u sebe a pro případné další šíření je musí poskytnout určitou formou, je tedy potřeba manuálního zásahu.

1.2 Plánovaný stav

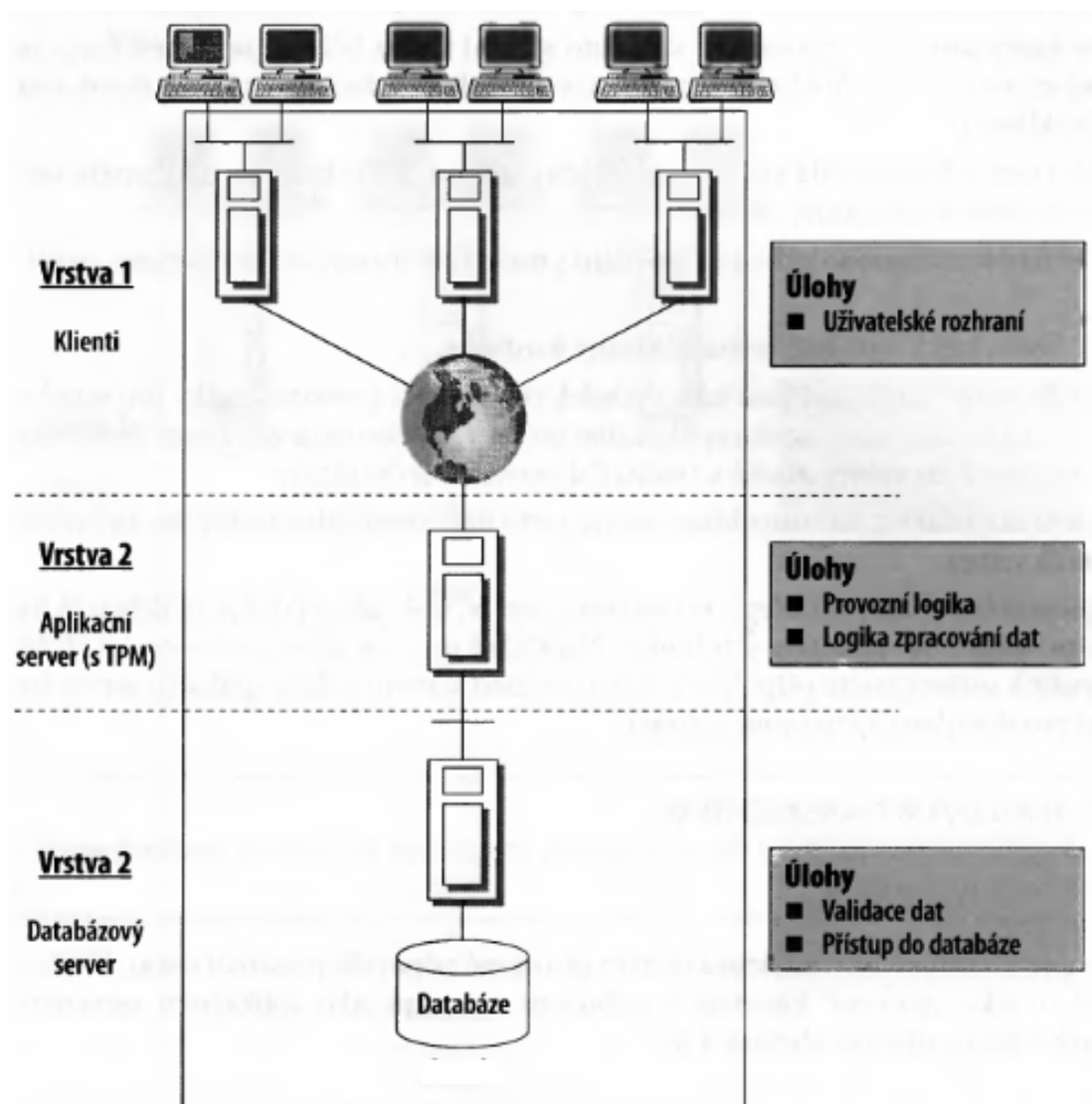
Nový informační systém by měl doplnit původní listinnou formu a nahradit ručně vytvářené elektronické tabulky jednoduchým uživatelským rozhraním, tedy vést podrobnější informace o projektech, zjednodušit a zpřehlednit jejich tvorbu, správu, vytváření přehledů o projektech. Přístup do tohoto systému bude mít kdokoli s příslušnými právy. Systém by měl běžet na centrální databázi, tak aby všichni měli přístup ke stejným datům a nevznikala duplicita. Databáze by měla být dostupná z jakéhokoliv osobního počítače nezávisle na operačním systému, z toho důvodu je zvolena varianta webové aplikace, díky které nemusíme řešit problémy tvorby aplikace pro konkrétní platformy. Nebudou tedy prováděny žádné změny na osobních počítačích uživatelů, které jsou běžně vybaveny webovým prohlížečem. Webový prohlížeč se bude připojovat k aplikačnímu serveru a ten k databázovému, uživatel bude plně odstíněn od případných problémů s aplikací či serverem a naopak serverové části odstíněny

od problémů uživatelského počítače – například zavirování. Takto systém tvoří nezávislé části, aplikace a databáze poběží na serverech k tomu určených s využitím běžně dostupných technologií. Systém musí umožňovat jednoduché zálohování databáze pro případ ztráty dat na serveru způsobenou například havárií datového úložiště.

2 NÁVRH DATABÁZE A APLIKACE

Celý informační systém je tvořen třemi částmi, které spolu úzce spolupracují. Systém jako takový můžeme v jednoduchosti popsat jako architekturu klient – server (uživatelská aplikace – systémová aplikace). V tradičním pojetí chápeme tuto architekturu jako dvouvrstvou, tedy na straně uživatele se používá takzvaný tlustý klient, který komunikuje s databázovým serverem. Tlustý klient má v tomto ohledu nevýhodu v podobě větší zátěže uživatelského hardwaru v podobě diskového prostoru, operační paměti a nebo času výpočtu na procesoru, většinou vše dohromady. Je to způsobeno tím, že klient dostane údaje z databáze a musí si je transformovat sám do výstupní podoby, takže je nutný kvalitně napsaný rychle pracující klient, který může být stejně zpomalen nedostatkem systémových prostředků při řešení náročnějších operací, což je dáno malou provázaností operačního systému s aplikací. Z toho důvodu se koncem 90. letech začalo upouštět od této dvouvrstvé architektury, podle Thomase Conellyho: *„Po roku 1995 se objevila nová podoba tradičního dvouvrstvého modelu klient – server, aby se překonaly některé z těchto problémů, nazývaná třívrstvá architektura klient – server. Tato nová architektura měla tři vrstvy, každá mohla běžet na jiné platformě.“* [2] Jednotlivé části třívrstvé architektury lze popsat:

- uživatelské rozhraní – tenký klient, který mnohem méně zatěžuje uživatelský počítač, nemá velké nároky na hardware, jeho úkolem je jen interpretovat obdržené údaje, již nemusí provádět žádné výpočty,
- aplikační server – většinou se jedná o server jako střední vrstva mezi klientem a databází, je vytvořen tak aby zvládal obsluhovat požadavky mnoha klientů a poskytoval vysoký výkon při výpočtech s daty, zde se vykonává aplikační logika,
- databázový server – poskytuje střední vrstvě data, je na něm nainstalován systém řízení báze dat pro efektivní uchování a práci s daty.



Obr. 1 Třívrstvá architektura klient – server [2]

Třívrstvá architektura nám nabízí jednoduchou údržbu díky existenci jediné kopie aplikace na serveru, není nutno při změnách nijak distribuovat software koncovým uživatelům, vše se provede na serveru. Stejně je to i s databází, která může být zálohována automaticky nezávisle na uživateli.

Formu aplikace máme danou, jedná se webovou aplikaci s využitím databáze k uložení dat. Nyní je potřeba si určit možnosti, funkce a požadavky na daný databázový systém pro správu výzkumných projektů a z tohoto poté vyjít při návrhu struktury dané databáze.

2.1 Návrh databáze

Databáze je dnes běžně využívané slovo, které za sebou ovšem skrývá velké množství pojmů, které spolu úzce souvisí, ale je potřeba se v nich orientovat, protože každý slouží k jinému účelu. Proto si nejprve uvedme několik termínů běžně užívaných při práci s databázemi:

- Databáze nebo také datová základna je soubor uživatelských dat, informací, které jsou někde uloženy. Běžně se tento pojem používá obecněji v širším kontextu jako softwarová součást, která slouží k přístupu k těmto datům.
- Software, kterým přistupujeme k datům, se nazývá systém řízení báze dat, což vychází z anglického database management system. Z názvu je jasné patrné, že se jedná o systém, který data spravuje, je tedy prostředníkem mezi dotazovatelem a samotnými daty.
- Relační model je založen na tabulkách, v nichž jednotlivé řádky chápeme jako záznamy a některé sloupce nám slouží k vazbě se sloupci s jiným tabulek, vzniká tak relace - propojení klíčů. Díky relacím vznikají množiny dat, které mají mezi sebou jasně definované vazby. Pomocí základních operací jsme schopni velmi jednoduše pracovat s daty v několika množinách.

2.1.1 Metodika tvorby databáze

Správný návrh databáze je klíčovou částí tvorby celé aplikace. Musí splňovat potřeby žadatele systému, kterým jde prvotně o data a druhotně o aplikaci jako takovou. Můžeme říct, že v těchto případech je databáze důležitější než aplikace. Nejprve je tedy tvořena databáze a na jejím základě je tvořena aplikace. Při návrhu je také nutné myslet nejen na to co má databáze obsáhnout, ale i na to, aby se data v ní dala využít k tvorbě důležitých informací, tak aby nám informační systém nejen data ukládal, ale nabízel i výstupy z těchto dat, které nám zjednoduší další práci.

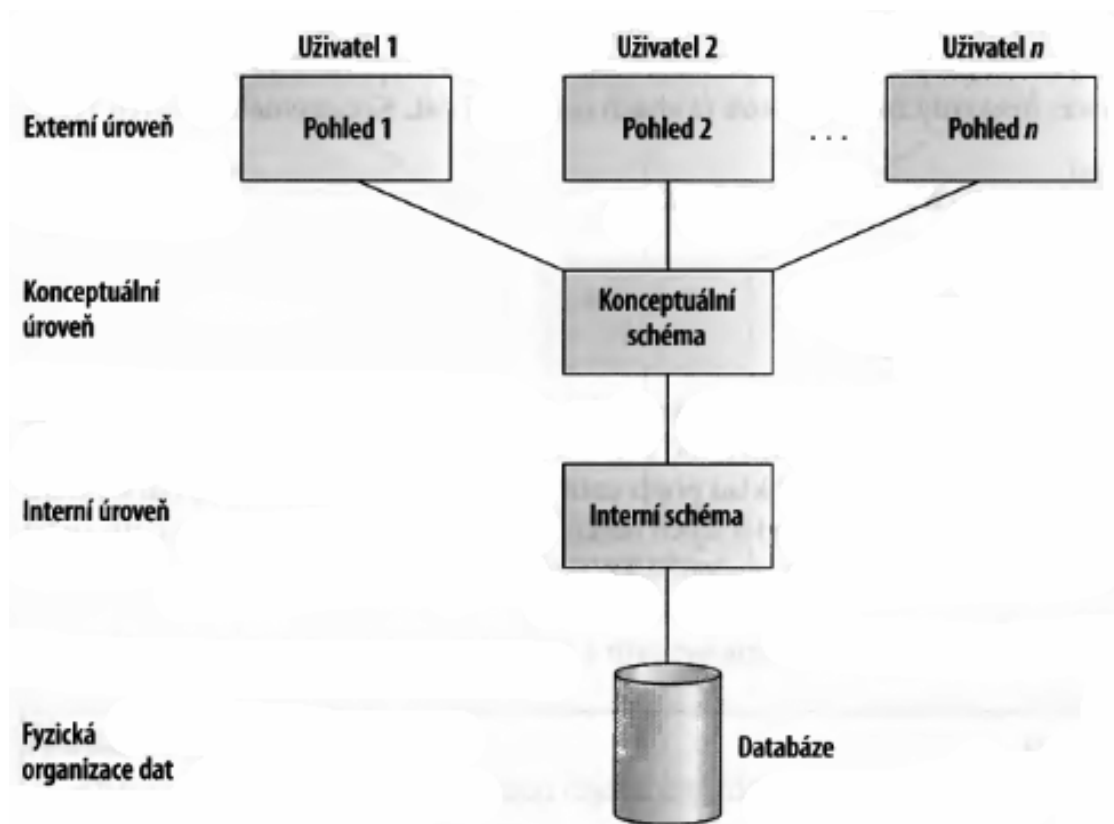
Metodiku návrhu databáze můžeme rozdělit do tří úrovní.

- Externí úroveň je podle Thomase Conellyho: „*Uživatelský pohled na databázi. Tato úroveň popisuje tu část databáze, která je podstatná pro každého z uživatelů*“. [2]
Jinak řečeno, jedná se o vnější pohled uživatelů na databázi. Každý uživatel má svou

osobní představu o formě databáze v závislosti na jeho znalosti systému, uživatelův pohled tedy obsahuje jen prvky, jejich vlastnosti a spojitosti které jej zajímají, jedná se tedy z velké části o subjektivní pohled, ze kterého začínáme vycházet. Další prvky a jejich relace můžou být v databázi obsaženy, aniž by uživatele zajímali a věděl o nich. Dalším z problémů může být rozdílný pohled na tytéž data případně různě transformovaná data například výpočtem nebo odvozením. Taková data nemusí být v databázi vůbec uložena, ale jsou v případě potřeby vytvářena. Jednoduchým příkladem je například údaj věku osoby. Uložení věku do databáze není problém, ale tam, kde potřebujeme znát aktuální hodnotu, by se museli údaje procházet a každoročně všechny upravit, to navíc většinou v návaznosti na datum narození nebo jiné, čímž by docházelo ke každodennímu procházení databáze. Tuto hodnotu mnohem jednodušeji získáme výpočtem právě z data narození nebo jiného porovnáním s aktuálním datem. Uživatel tak často vidí údaje, které nejsou přímo v databázi, na tuto skutečnosti si musíme dát pozor.

- Konceptuální návrh podle Conellyho: *„Obecný pohled na databázi. Tato úroveň popisuje, jaká data jsou uložena v databázi, a relaci mezi daty“*. [2] Střední úroveň nám slouží k poznání databáze s pohledu jejího tvůrce, správce. Jde o kompletní náhled na datové požadavky všech uživatelů se zakomponováním všech dalších dat, odstranění nepotřebných a provázání jednotlivých položek. Jedná se o abstraktní návrh, bez souvislosti s fyzickou podobou uložení dat. Tato úroveň představuje kompletní databázi na její abstraktní úrovni obsahující všechny její entity s jejich vlastnostmi a provázaností, omezení a sémantické informace o datech a v neposlední řadě bezpečnostní informace. Tato úroveň zahrnuje všechny vnější pohledy, tak aby nabídla uživatelům to, co od ní žádají, ať už v podobě uložených entit nebo hodnot z nich odvozených. Entity zde mají popsane vlastnosti, takže navrhujeme konkrétní typy číselných prvků, délky textových řetězců a další, stejně tak již jednotlivé položky tabulky mezi sebou můžeme provázat. Toto vše se děje stále na abstraktní úrovni návrhu a neřešíme možnosti a způsoby fyzického uložení databáze, řešíme jen potřeby systému a uživatelů.
- Interní úroveň je dle Conellyho: *„Fyzická reprezentace databáze na počítači. Tato úroveň popisuje, jak jsou data uložena v databázi“*. [2] Jde tedy o návrh databáze na úrovni fyzické implementace, tak aby bylo dosaženo optimálního výkonu a využití

úložného místa navrhované databáze. Jedná se tedy o výběr vhodného systému řízení báze dat, který bude ideální pro navrženou strukturu databáze. Jde o výběr vhodného rozhraní pro přístup k metodám operačního systému pro práci se záznamy, tedy jejich ukládání a vyvolání na paměťové zařízení. Některé systémy nabízí úzkou spolupráci s metodami operačního systému a jsou na něj vázány, jiné se naopak snaží co nejméně využít systémových metod díky návrhu vlastní organizace souborů, čímž můžou docílit menší závislosti na systému. Na fyzické úrovni se pod systém řízení dat skrývají samotná data v podobě známé jen operačnímu systému.



Obr. 2 Třívrstvá metodika návrhu databáze [2]

Celkový popis databáze pak nazýváme schéma databáze. Díky třívrstvé architektuře vzniká několik schémat pro uživatelské pohledy, které jsou dále všechny obsaženy v jednom konceptuálním schématu. Zde je potřeba rozlišovat pojmy popis databáze a schéma databáze. Zatímco popis nebo schéma jsme navrhli a implementujeme jej, případně používáme s minimálními změnami, tak databáze je soubor dat, která jsou systémem řízení báze dat na základě schématu databáze ukládána, tento soubor se velmi často mění a obsahuje uživatelská data. Cílem třívrstvé architektury by měla být nezávislost dat

na jednotlivých úrovních, neboli změny v nižších vrstvách by neměli ovlivnit vrstvy vyšší. Existují dva druhy nezávislosti dat:

- Logická nezávislost by měla zajistit, aby při změně konceptuálního schématu nebylo nutné měnit externí schémata či databázovou aplikaci, například při přidání nebo odebrání entit či vazeb. Změna by měla být viditelná jen tam, kde je žádána.
- Fyzická nezávislost interního schématu například v podobě změny organizace souborů či způsobů uložení by měla být možné za předpokladu zachování konceptuálního schématu. Změny se mohou projevit ve výkonu databázového systému, ale nesmí se projevit v popisu databáze. Pokud ovšem dochází k poklesu výkonu, může to být také způsobeno špatným návrhem konceptuálního schématu a musí dojít k jeho změně.

2.1.2 Relační model databáze

Relační databáze se využívají v celém světě od 70. let 20. století a za tuto dobu se staly hlavním používaným modelem dat oproti jiným modelům, například hierarchickému nebo síťovému. V téže době se také začíná s vývojem jazyka SQL – structured query language, který se běžně využívá při práci s tímto modelem a nabízí velmi jednoduchou syntaxi dotazů. Od konce 70. let se již setkáváme s komerčními databázovými systémy s tímto modelem a dotazovacím jazykem. Cíle relačního modelu byly tyto:

- umožnit jednoduchou správu databázového modelu nezávisle na fyzickém uložení dat, tedy odstínit výstupy ze systému řízení báze dat od způsobu interního uložení dat. Data by měla být stále stejně reprezentována databázovým systémem nezávisle organizací souborů nebo systému ukládání.
- jednoduché zvládnutí sémantiky dat, jejich konzistence a redundance pomocí normalizace relací, tak aby se skupiny dat neopakovaly. Jde o jednoduchá pravidla, která vedou ke správné tvorbě databázových schémat, entit systému, relací a dalších, tak aby systém splňoval požadavky na soudržnost dat, nevznikali data s chybějícími souvislostmi, a nadbytečnost, tedy vznik duplicitních nebo nepotřebných údajů.

Dnes se již setkáváme se stovkami různých systémů řízení báze dat pracujících s relačním modelem, které jsou určeny jak pro serverové aplikace s nejrůznějšími kritérii na výkon či stabilitu až po osobní počítače, i když mnoho z nich nedodržuje striktně definici tohoto modelu. Na osobních počítačích se běžně setkáváme s Microsoft Office Access, v prostředí

serverových aplikací například Microsoft SQL Server, MySQL, PostgreSQL a spoustou dalších. Každý systém nabízí v základu stejné funkce, některé přidávají své další a odlišnosti najdeme taky právě v podobě způsobu spolupráce s operačním systémem a uložením dat.

2.1.3 Relační datový model

Pro snadnější pochopení a orientaci v prostředí relačních databází je potřeba si vysvětlit některé pojmy. Podle Conellyho je: „*relační model založen na matematickém konceptu relace, která je fyzicky reprezentována tabulkou*“. [2] Principy fungování těchto systémů jsou založeny na terminologii převzaté matematiky, včetně teorie množin a predikátové logiky. Pokud se v těchto oborech orientujeme, je pro nás pochopení logiky struktur v modelech mnohem jednodušší. Model je: „*integrovaná kolekce konceptů pro popis dat, relací mezi daty a omezení dat*“. [2] Jde o reprezentaci objektů ‚reálného světa‘, událostí a jejich souvislostí. Zaznamenává nám podstatné aspekty modelovaného objektu, kterou jsou pro nás významné. Měl by sloužit jako výchozí bod pro návrh databáze a slouží nám při komunikaci se zadavatelem systému a definování vlastností databáze. Pro uživatele při komunikaci model poskytuje srozumitelnější pohled na systém uložení dat a jejich prezentaci. Datový model můžeme dělit na tři části:

- strukturální část – základní množina objektů modelu, která nám definuje jeho pravidla pro tvorbu databáze, jde o popis vlastností,
- manipulační část – množina operací (transakcí) modelu, které mohou být a budou prováděny s daty, jedná se zavedení přípustných operací nad daty jako je výběr, vkládání či aktualizace,
- pravidla integrity – tato množina není nutná, ale větší modely ji k bezproblémovému chodu vyžadují, tyto pravidla zajišťují přesnost dat, definují souvislosti v modelu a případné restriktce, tak aby byla zajištěna integrita.

Relační model využívá pět hlavních složek, některé jsem zde již použil, proto si je uvedeme, abychom je mohli později využít při návrhu konkrétního modelu, jsou to:

- relační databáze – kolekce normalizovaných tabulek, která dále obsahuje
- relace – což jsou jednotlivé tabulky pevně danými sloupci a proměnlivým počtem řádků, strukturu relace lze popsat těmito složkami:

- atribut – pojmenovaný sloupec relace,
- doména – množina přípustných hodnot pro jeden nebo více atributů, při definování sloupce definujeme i jeho doménu, doména si můžeme představit jako množinu hodnot, z nichž se vybere pro objekt právě jedna pro daný atribut
- datová n-tice – řádek relace, může nabývat různých hodnot podle domény

Z tohoto tedy plyne, že relační databáze obsahuje relace pro uložení informací o objektech ‚reálného světa‘, které chceme v databázi uchovat. Relace je fyzicky reprezentována tabulkou, jejíž řádky jsou jednotlivé záznamy o objektech a sloupce jsou vlastnosti (atributy) objektů. Z toho plyne, že nezáleží na pořadí atributů ani řádků, vždy konkrétní řádek – datová n-tice obsahuje konkrétní informace o objektu nezávisle na pořadí atributů, relace má stále stejný význam. Stejně domény se mohou v jedné relaci vyskytnout vícekrát a stejně tak nemusí být obsaženy vůbec, záleží pouze na vlastnostech objektu a jejich označení pomocí atributů. Doména má smysl právě při kontrole údajů, kdy povoluje pouze konkrétní množinu hodnot a při pokusu uložit do ní hodnoty mimo množinu operaci nedovolí, tím je zajištěn smysl dat. Často se kromě formální terminologie relačního modelu můžeme setkat i s jinými výrazy, které jsou taky běžně používané a pro veřejnost srozumitelnější. Proto používáme místo výrazů relace, atribut a datová n-tice pojmy tabulka, sloupec a záznam, které jsou nám bližší z důvodů naší představivosti. Někdy se taky místo pojmu tabulky používá pojem soubor, jednotlivé řádky jsou záznamy a sloupce jsou pole.

2.1.4 Relační tabulky

Po pochopení relačního datového modelu, který nám slouží ke snazší komunikaci se zadavatelem systému při definování vlastností potřebných k zachycení systémem, přejdeme k návrhu databáze z modelu. Pro tento návrh bychom si měli doplnit další informace o relacích neboli relačních tabulkách, které mají tyto vlastnosti:

- tabulka má své jedinečné jméno v databázi, které ji identifikuje
- každá buňka tabulky obsahuje pouze jedinou hodnotu, tak je pevně daná struktura tabulky a vždy průsečík sloupce a řádku nám vrátí právě tuto jednu hodnotu
- každý sloupec jedné tabulky je jednoznačně pojmenován

- hodnoty v jednom sloupci mohou nabývat vždy stejným hodnot z konkrétní množiny (domény)
- pořadí sloupců nemá význam na informaci řádku, ten nese pořadí stejné hodnoty v jakémkoliv pořadí, sloupce můžeme tedy mezi sebou libovolně přesunovat společně s jejich hodnotami na všech řádcích
- každý záznam tabulky je jedinečný, v tabulce nemohou existovat duplicitní záznamy
- pořadí záznamů v tabulce nemá žádný logický význam, význam tohoto pořadí vzniká až na fyzické vrstvě databáze, kdy při velkém množství dat musíme volit vhodný systém řízení báze dat

Jedinečnost záznamu v každé tabulce je potřeba nějakým způsobem zajistit, tak abychom byli schopni jednoznačně identifikovat každý řádek tabulky. Tuto vlastnost zajistíme tím, že v tabulce zvolíme takzvaný klíč relace, což provedeme výběrem vhodného sloupce nebo kombinace sloupců. Při hledání těchto klíčů, které nazýváme kandidátní klíče, musíme dbát na to, aby opravdu zajišťovali jednoznačné určení záznamu. Kandidátní klíč musí splňovat dvě vlastnosti:

- jedinečnost – v každém záznamu musí kombinace hodnot v klíčových sloupcích identifikovat právě daný záznam
- neredukovatelnost – žádná podmnožina kandidátního klíče nesmí určovat jedinečný záznam, pokud k tomu dojde, máme špatně vybraného kandidáta a je nutné se zaměřit právě na onu podmnožinu, která může jednoznačně identifikovat záznam

Po výběru vhodných kandidátů zvolíme jednoho z nich nazvaného super klíč, což je: *„sloupec nebo množina sloupců, které jednoznačně identifikují záznam v relaci a který obsahuje jen minimální počet sloupců nutných k jedinečné identifikaci záznamů“*. [2] Kandidátní klíče je třeba volit na základě znalosti možných hodnot z ‚reálného světa‘, například pokud máme vzorek tabulky a některé sloupce se jeví jako kandidáti, je potřeba si uvědomit že se v nich mohou objevit později duplicitní hodnoty, proto musíme znát údaj který zaznamenávají, abychom pochopili zda opravdu nabývá jedinečných hodnot. Protože tabulka nemá duplicitní záznamy, jak bylo popsáno v jejich vlastnostech, tak u každé tabulky můžeme určit primární klíč. V nejhorším možném případě bude klíčem celá množina sloupců, ale obvykle stačí k definici její podmnožina. Pomocí tohoto můžeme

vytvářet také cizí klíče. Cizí klíč je: „*sloupec nebo skupina sloupců v jedné tabulce, která odpovídá klíči některé (případně téže) tabulky*“. [2] Tímto můžeme provést vytvoření vztahu mezi cizím klíčem jedné tabulky a primárním klíčem jiné tabulky, názvy sloupců se v jednotlivých tabulkách mohou lišit, jde o jejich logický význam, který musí být stejný. Tyto vztahy mezi tabulkami nám zároveň slouží ke kontrole redundance dat, pokud se nám někde objeví nadbytečné a duplicitní údaje, je potřeba změnit struktury tabulek a dochází k vytváření právě těchto vztahů. Zároveň nám tyto vztahy slouží pro systém řízení dat, který při operacích může kontrolovat provázanost klíčů (referenční integritu) a případně provádět další operace, aby byla zachována požadovaná forma dat. Například by neměla být vložena hodnota cizího klíče, která neexistuje v rodičovské tabulce s primárním klíčem. Tím by mohlo dojít k narušení integrity systému. Pokud těmto případům chceme předejít, tak systém řízení báze dat umožňuje při definici tabulek a jejich cizích klíčů nastavit opatření co v takovém momentě má systém s rodičovskými údaji provést.

2.1.5 Databáze výzkumných projektů

Databázi bude tvořit několik tabulek, které budou mít mezi sebou logické vazby na bázi primární klíč a cizí klíč. Významnou tabulkou databáze budou projekty do systému zadávané. Tato tabulka bude obsahovat na základě uživatelského pohledu jednotlivé údaje, které jsou na dodaném formuláři návrhu výzkumných projektů, tento formulář slouží jako podklad pro tvorbu relačního modelu. Po bližším zkoumání tohoto dokumentu bylo zjištěno, že některá pole budou nabývat stejných hodnot z množiny, která se bude minimálně měnit. Aby nevznikali opakující se údaje navíc při špatném zadání údaje s minimální odlišností, budou tyto množiny údajů vedeny v oddělených tabulkách a v projektové tabulce bude pouze reference v podobě cizího klíče na konkrétní hodnotu do oddělené tabulky. Tyto oddělené tabulky můžeme pro účely naší aplikace nazývat číselníky a při zadávání projektů z nich bude vybíráno, čímž odpadne problém vzniku redundantních dat a překlepů. Tyto číselníky se budou dále v informačním systému spravovat. K těmto údajům projektu se budou ještě zaznamenávat další informační položky pro účely systému, těmi bude kromě jednoznačného identifikátoru projektu také cizí klíč uživatele, který projekt do systému vložil, čas vložení do systému a stav projektu. Uživatelé budou také vedeni v tabulce podobné číselníku, která bude navíc obsahovat správu jejich vlastností.

Shrnutí tabulek s jednoduchým popisem vlastností:

- projekty – údaje z formulářů, některé sloupce obsahují cizí klíče z číselníků, údaj o datu vložení a cizí klíč uživatele, který projekt založil,
- číselníky – poskytovatelé finančních prostředků, předkladatelé, měna,
- uživatelé – jméno, heslo, role, zakázané přihlášení

2.2 Návrh informačního systému

Na základě konzultací se zadavatelem systému a poskytnutých materiálů jsem vytvořil návrh požadavků na systém, seznam jeho funkcí a možností. Funkce systému jsou seskupeny do několika logických celků podle funkční příbuznosti nebo podle podobnosti zpracovávaných dat.

2.2.1 Uživatelské účty

Systém bude mít uložené uživatelské účty v databázi, uživatel je identifikován na základě svého uživatelského jména a hesla, bude mít uloženy práva a bude možnost zakázat přihlášení. Z důvodů bezpečnosti budou hesla v databázi hashovaná, tak by je nebylo možno jednoduše přechít.

Role uživatelů v systému:

- zadavatel – omezená práva, může zadávat projekty do systému a editovat je do doby schválení, může pomocí agendy výzkumných projektů prohlížet projekty, které do systému zadal,
- editor – hlavní role se všemi právy, tento typ uživatele má přístup ke všem funkcím informačního systému bez jakéhokoliv omezení.

Editor bude mít plně pod svou kontrolou seznam uživatelů, bude moct uživatele vytvářet, nastavovat jejich práva, zakázat jim přihlášení, případně mazat. Smazání bude ošetřeno, aby nemohl být smazán uživatel s existujícími záznamy v projektech.

Přístup do systému bude výhradně po přihlášení. Přihlášení bude provedeno klasickým webovým formulářem zadáním uživatelského jména a hesla. Na základě těchto údajů dojde k ověření uživatele vůči databázi a zpřístupnění systému s právy, které mu byly zvoleny.

2.2.2 Administrační rozhraní

Obsahuje správu jednotlivých číselníků, které se využívají v projektech. Editor může jednotlivé údaje v číselnících měnit, přidávat či mazat. Při mazání údaje bude kontrolováno, zda není údaj využit u některého z projektů, aby nebyla porušena integrita.

Číselníky a jejich hodnoty:

- poskytovatelé finančních prostředků – jméno,
- předkladatelé (řešitelé) – jméno,
- měna – jméno, koeficient násobení.

2.2.3 Agenda výzkumných projektů

Stránka obsahující výpis projektů uložených v systému v podobě tabulky. Bude rozlišen výpis pro roli editora obsahující všechny projekty a výpis projektů zadaných konkrétním uživatelem, tak aby si uživatel s rolí zadavatele mohl zobrazit projekty jím do systému vložené. Zobrazovaná data se budou dát třídit podle jednotlivých sloupců, volit zobrazené sloupce a vytvářet filtry hodnot ve sloupcích, tak aby bylo možno zobrazit pouze údaje odpovídající kritériím. Seznam bude umožňovat exportovat údaje ve formátu XLS.

2.2.4 Výzkumný projekt

Hlavní stránka projektu zobrazuje kompletní informace o projektu, nabízí možnost vyexportovat tabulku jako návrhový list projektu do PDF formátu a následně ji vytisknout. Pokud má uživatel odpovídající práva, může projekt editovat, měnit jeho stav nebo zadavatele. Při prvotním vložení projektu do systému je evidováno datum založení a stav projektu nastaven na počáteční hodnotu. Editace projektu nabízí textová pole s kontrolou odpovídající hodnoty nebo výběr z číselníku. Každý projekt bude mít nastaven jeden ze tří stavů, ve kterém se právě nachází, a které budou i barevně odlišeny.

Stavy projektu:

- čeká na chválení – v tomto stavu může projekt editovat i zadavatel,
- schválen,
- nechválen.

2.3 Struktura aplikace

Systém aplikace bude využívat třívrstvé architektury, která umožní dobře odstínit uživatelského klienta od samotné aplikace a dat, takže uživatelův klient poslouží jen jako komunikační prvek uživatele s aplikací a bude zobrazovat její výstupy. Další velkou výhodou je rozdělení logiky aplikace a databázi, čímž se funkčnost aplikace stane na těchto datech nezávislá. I když v konečném důsledku jsou pro tuto konkrétní aplikaci data z databáze nezbytnou součástí, protože aplikace slouží k prezentaci dat z databáze. Pro tento model se přímo nabízí využití webové aplikace, která je pro tyto účely vhodná. Jedním z dalších požadavků na aplikaci je možnost ji využít pro provoz ve vícero organizacích, tedy s oddělenou databází, což díky zvolené architektuře je velmi dobře realizovatelné a jedná se tedy o zavedení této možnosti do aplikační logiky.

Webová aplikace by měla být pro uživatele jednoduše přístupná, tak aby byla mohla být provozována na mnoha typech prohlížečů, je tedy třeba dbát na to, aby generovala kvalitní vizualizovatelný kód a nejlépe validní dle standardů při zachování kompatibility s obvyklými prohlížeči. Generovaný kód bude využívat standardu XHTML, který je dnes běžně rozšířen a prohlížeče s ním běžně pracují. Dokumenty XHTML budou doplněny stylisem, který bude obsahovat grafickou definici stránky.

Struktura stránek by měla odpovídat jednotlivým celkům popsaných v návrhu informačního systému. Přístup k jednotlivým číselníkům a seznamu uživatelů bude přímo z menu aplikace, stejně jako přístup do agendy výzkumných projektů a možnost založení nového návrhu projektu. Uživatel s rolí zadavatele bude mít menu omezené pouze na agendu výzkumných projektů, které zadal, a vložení nového návrhu projektu. Přístup do aplikace bude po ověření vůči uživatelským účtům v databázi.

Design aplikace by měl být jednoduchý a přehledný, tak aby nabízel intuitivní prostředí pro práci a neobsahoval zbytečně mnoho grafických prvků, které by mohli působit rušivě a odvádět pozornost. Webové stránky by měli být graficky ucelené s možností využití moderních JavaScriptových knihoven pro vytvoření dynamicky reagujícího prostředí, ale se zachováním funkčnosti pro případ kdy uživatelův klient nebude podporovat tyto funkce, protože se jedná o knihovny spouštěné na straně klienta, kde nemůžeme zaručit podporu těchto funkcí a záleží pouze na volbě uživatele. Grafické pojetí by mělo odpovídat typu aplikace, tedy mělo by hlavně podporovat funkčnost a využití aplikace.

Databáze jako taková by měla sloužit konkrétní organizaci, resp. jejímu oddělení. Jedním z požadavků pro databázový systém je, aby aplikace mohla pracovat z více databázemi, a z toho důvodu je třeba vytvořit systém univerzálně bez návaznosti na konkrétní organizaci. Správa databází by se měla provádět v konfiguračním souboru aplikace, tak aby byl uživatel odstíněn od těchto nastavení. Uživatel bude moci zvolit, ke které databázi se připojuje a s ní bude dále pracovat. Specifické informace související s konkrétní databází respektive organizací se mohou přidat do konfiguračního souboru, může se například jednat o název organizace.

2.4 Softwarové požadavky

Aplikační i databázová vrstva systému bude vyžadovat ke svému běhu známé open-source projekty. Aplikační logika webu bude vyžadovat skriptovací programovací jazyk PHP a je tedy nutné provozovat na webovém serveru jeho interpret ve verzi 5.2.0 nebo vyšší. Některé starší servery mají ještě stále nižší verze systémy, na volně dostupných webových serverech se dokonce setkáme i s verzemi 4. řady. Pro tento systém je však vyžadována jedna z nejnovějších řad nejen z důvodů bezpečnosti, kterou novější verze přinášejí, ale také z důvodů využití některých funkcí a konstruktů, které starší verze nemají. Pokud na serveru běží verze 5.2 nebo novější v obvyklé konfiguraci, měla by stačit požadavkům aplikace, není nutné instalovat nebo povolovat specifické knihovny. PHP neboli hypertextový preprocesor je také velmi často využíván právě pro své množství knihoven, například pro připojení k databázím nebo práci s obrázky, které obsahuje již v základní instalaci. Jedním z často používaných databázových systémů neboli systémů řízení báze dat je MySQL, který se bude starat o řízení dat na databázovém serveru. PHP i MySQL nejsou omezeny na jedinou platformu, lze je snadno provozovat jak pod Linuxem, tak pod Windows i dalšími systémy a nabízí výhodnou licenční politiku, takže není potřeba investovat za zakoupení licence ani provoz. Díky těmto výhodám, se dá aplikace jednoduše nasadit na velkém množství již existujících serverů, na Linuxových serverech se často setkáváme již v základní instalaci s těmito serverovými aplikacemi. Proto by neměl být problém provozovat webové stránky i na již existujícím webovém serveru bez nutnosti instalovat další software. V navrhovaném řešení není vyžadováno automatické zálohování databáze pomocí pravidelně spouštěného skriptu, uživatel má tuto možnost jednoduše přístupnou přes informační systém.

Klientská část vyžaduje prohlížeč webových stránek nejlépe některý z dnes běžně užívaných jako je Internet Explorer, Mozilla Firefox, Chrome a další. Kód pro prohlížeč bude obvyklého formátu XHTML doplněný o JavaScriptové knihovny pro vylepšení dynamiky prostředí a možnosti využití Ajaxu, což by mělo přinést příjemnější užívání aplikace. Z těchto důvodů je vhodné, aby prohlížeče byly v co nejnovějších verzích, aby mohla být využity všechny funkce prostředí aplikace, a aby měl povolené spouštění JavaScriptů.

2.4.1 MySQL

MySQL je podle manuálu: „celosvětově nejoblíbenější open source databázový software s více než 100 miliony kopií svého softwaru staženého z internetu a distribuovaného po celou svou historii. S jeho vynikající rychlostí, spolehlivostí a snadným použitím se MySQL stal preferovanou volbou pro Web, Web 2.0, SaaS, ISV, telekomunikační společnosti a moderně smýšlejících podnikových IT manažerů, protože eliminuje hlavní problémy spojené s prostoji, údržbou a správou pro moderní, on-line aplikace“. [5] Tento databázový software od firmy Sun Microsystems využívá velké množství firem kvůli jeho nesporným kvalitám, jedná se například o rychle rostoucí organizace z oblasti informačních technologií jako je Google, Youtube, Nokia, Yahoo! nebo známé open-source projekty pro web jako Wikipedia či Drupal. Je tedy velmi vhodná pro velké množství různých aplikací, kde je potřeba pracovat s daty. Kromě databázového systému nabízí firma Sun také různé placené produkty pro podporu jejich databázového softwaru v podobě řešení pro monitorování, zálohu, testování a dalších funkcí, tak aby doplnili jejich databázový systém o služby, které si dnešní velké organizace žádají. MySQL je relační databázový systém využívající jazyka SQL a splňuje definovaný ANSI/ISO SQL Standard v jeho poslední verzi 2003, jejíž postupný vývoj začal v roce 1986 a během té doby bylo vytvořeno několik verzí.

3 SKRIPTOVACÍ JAZYKY PHP, JAVASCRIPT

V předchozích kapitolách byly zmíněny technologie PHP pro dynamické vytváření stránek prováděním skriptů na straně serveru a technologie JavaScript pro dynamické akce na straně uživatele. Z popisu je vidět, že obě technologie pracují na jiných vrstvách aplikace a postupně si je popíšeme včetně možnosti jejich použití pro vyvíjenou aplikaci databázového systému. Nebudu se zabývat jen samotnými jazyky, ale je i jejich vylepšeními v podobě předpřipravených knihoven neboli frameworků, které ulehčí práci díky předem vytvořeným běžně využívaným funkcím a částečně ovlivňují i samotný styl tvorby aplikace. Frameworky nejen ulehčují vývoj v podobě hotového kódu, ale také se snaží vést programátoru k určitým moderním programovacím technikám pro vývoj kvalitní aplikace z pohledu kódu, jeho strukturovanosti a znovupoužitelnosti a i dalších vlastností. Důležitou částí je i výběr správného frameworku, protože se začínají objevovat frameworky, které ne zcela korespondují s cílem zjednodušit práci, ale spíš vedou k problémům s jejich integrací do systému a využitelností.

3.1 PHP

PHP neboli Hypertext Preprocessor je programovací skriptovací jazyk, který je vykonáván (neboli skripty jsou interpretovány) na straně serveru. Z názvu jazyka je patrné jeho určení k přípravě hypertextových dokumentů před jejich odesláním klientovi, tedy tento jazyk nám dává možnost generovat dynamicky kód webových stránek a tím možnost tvořit obsahovat v závislosti na uživatelských vstupech případně datových vstupech například z databáze. Tímto se webové stránky stávají z jednoduchých statických dokumentů mocným nástrojem pro velké aplikace, které mohou být plně založeny na externích datech a stránka se tak stává programem, jenž obsluhuje uživatelské požadavky. Velkou výhodou vyhodnocování skriptů na straně serveru je možnost dohledu nad prováděním kódu a uložení zdrojového kódu pouze na serveru, k uživateli se odesílá pouze HTML výstup. Můžeme tedy plně ovlivnit chování prostředí, na kterém aplikace běží, tedy chování webového serveru případně operačního systému, a zaručit správnost provedení skriptů. Tato centralizovanost aplikace nám umožňuje také jednoduchou správu zdrojového kódu, protože je použit v jediné kopii a případné opravy a změny v kódu je tedy potřeba distribuovat pouze na server, není nutný žádný zásah z pohledu uživatele.

3.1.1 Vývoj jazyka

Rasmus Lerdorf, jenž je považován za původce PHP v polovině 90. let vytvořil jednoduché skripty v Perlu pro kontrolu přístupů na vlastní webové stránky, které nabídl k využití ostatním uživatelům a nazval je Personal Home Page tools, což je původní význam zkratky PHP. Díky oblibě těchto skriptů, které byly postupně využity i ke složitějším účelům, vznikla komunita, která postupně vytvořila jazyk PHP3. Tato komunita zdokonalila tyto skripty, nabídla jednoduché aplikační rozhraní a možnost dále tvořit tento jazyk. *„Syntaxe jazyka byla také zdokonalena konstrukcemi, které budou důvěrně známé lidem, přecházejícím z objektově orientovaných a procedurálních jazyků. Pokud umíte C, C++ nebo Javu, nebo jste už psali nějaké skripty v shell/awk, nebo jste psali programy v Pascalu nebo VBasicu, budou pro vás základní konstrukce PHP hračkou.“* [1] Jazyk tedy obsahuje běžné řídicí struktury, operátory, datové typy, deklarace funkcí, tříd a objektů, je obdobná ostatním jazykům. PHP se dá jednoduše pracovat i s ukazateli, které jsou běžné například v jazyce C, ale v jiných skriptovacích jazycích se s nimi moc nesetkáme.

PHP není jediným takovým programovacím jazykem, rozhodně se najdou jazyky s delší tradicí jako je Perl, Java, Python nebo skripty pro shell pro Linux. Ale tyto jazyky jsou od počátku koncipovány jako běžné programovací jazyky nebo skriptovací jazyky pro operační systém, takže nabízí mnoho věcí navíc, které nejsou pro webovou aplikaci využitelné a naopak nenabízí některé funkce, které jsou specifické pro použití na webu. Castagnetto říká: *„PHP bylo navrženo, aby pracovalo na webu a v této oblasti vyniká, připojování a dotazování databáze je jednoduchá úloha, která může být zpracována ve dvou až třech řádcích kódu. Skriptovací jádro PHP má dobře optimalizovanou dobu odezvy potřebnou ve webových aplikacích. Také může být přímo součástí webového serveru, což zvyšuje propustnost.“* [1] Velkému rozšíření jazyka také napomohl fakt, že kromě jeho rychlosti a jednoduchosti nabízelo nezávislost na jedné platformě a jeho vydání jako open source projekt. Díky tomuto společně s open source operačními systémy založenými na Unixu / Linuxu a jejich dobrou provázaností patří dnes k nejpoužívanějším skriptovacím jazykům na straně serveru pro webové aplikace, jediné velké konkurenty má v podobě technologií ASP – Active Server Pages respektive ASP.Net a JSP – JavaServer Pages, který není natolik rozšířen, a jsou částečně vázány platformou. Aktuálně poslední stabilní vývojovou větví PHP je verze 5.3, která kromě plné podpory objektově orientovaného programování, jehož podpora je od počátku vylepšována, přinesla také

velkou novinku v podobě jmenných prostorů, které opět přibližují práci programátora běžným vysokoúrovňovým programovacím jazykům. Jak je uvedeno v úvodu PHP manuálu, jedna z výhod oproti jiným jazykům je: *„jak se zápis skriptů liší od skriptů psaných v jiných jazycích jako Perl nebo C – místo psaní programu s množstvím příkazů pro HTML výstup, píšete soubor s vloženým kódem, který provádí určené funkce. PHP kód je uzavřen ve speciálních počátečních a koncových značkách, které vám umožňují vstoupit a opustit ‚PHP mód‘.“* [4] Jazyk tak dává plnou moc programátorovi, aby své aplikace tvořil jakýmkoliv způsobem a nabízí funkce na základní jednoduché úrovni, ale zároveň tak, aby měl možnost využít plně webového serveru a spousty funkcí. Díky tomu vzniklo velké množství knihoven – frameworků, nabízí před připravené složitější funkce a šetří čas při vývoji, mimo to také některé se snaží nabídnout určitou architekturu tvorby aplikace pro vyšší přehlednost a lepší škálovatelnost. Tím programátor získá mocný nástroj a může se více soustředit na tvorbu logiky systému a méně psát kód, respektive psát kód tak, aby jej mohl znovu jednoduše využít i v jiných aplikacích bez psaní opakujících se drobných úloh. Jedním z těchto frameworků, který je použit i v praktické části je Nette Framework.

3.1.2 Nette Framework

Stejně jako samotné PHP je i tento framework šířen volně a patří k frameworkům s moderními funkcemi i přesto, že vznikl už v roce 2004. V té době sloužil jeho tvůrci a až v roce 2008 byl uvolněn jako open-source pro veřejnost. Jde o český projekt a u nás kolem něj vyrostla široká komunita včetně největších odborníků, která na něm dále pracuje a vylepšuje jej. Jedna z myšlenek tvůrce je otevřenost a možnost výběru, framework tedy nabízí spoustu funkcí, ale je možné využít jen některé z nich a stejně tak je možnost jej využít s jiným frameworkem, stejně tak se nabízí i jeho možnosti využití od jednoduchých webů až po korporátní aplikace.

Výhody využití Nette pro různé typy webových aplikací:

- primitivní webové stránky – například statické informace bez použití databází, pro tyto účely nabízí spoustu malých funkčních vylepšení, se kterými vytvoříme web rychleji a kvalitněji s možností jednoduché další editace, toto využití se sice zdá nevhodné, ale pokud práci s Nette ovládáte, tak vám nabídne mnoho utilit využitelných i pro takto jednoduchou činnost,

- složitější aplikace – tím jsou myšleny různé systémy pro správu obsahu, od různých redakčních systémů, blogů až po elektronické obchody nebo evidenční systémy, zde se ukazuje Nette v celém svém měřítku a poskytuje velmi jednoduchý nástroj pro efektivní rychlou tvorbu, navíc jednou z myšlenek je bezpečnost, takže nám částečně ulehčí tuto dnes podstatnou část webových aplikací a také nás vede ke strukturování kódu, takže jednoduše tvoříme opět znovu využitelné kusy kódu, Nette nám u těchto aplikací nabízí množství funkcí, které bychom si zpravidla museli napsat sami a právě díky tomu se můžeme plně soustředit na tvorbu složitější logiky a designu aplikace,
- podnikové aplikace – není problém tvořit s Nette i velké systémy, ale jedná se o tvorbu obdobnou jako u předešlého bodu, což je dáno právě jazykem PHP, který je skriptovací a je určen pro webové prostředí, není tedy problém vytvořit velké informační systémy pro webové prostředí, ale jsme omezeni právě prováděním na serveru, připojení systémů například pro měření, bezpečnost nebo business musíme řešit přes jiné systémy a zde je na zvážení zda takový systém vyvíjet na takové programovací platformě, obvykle je takovou webová aplikace připojena na nějaký systém a slouží jako prezentační vrstva pro data ze systému s možností jednoduché manipulace se systémem.

Nette nám tedy velmi zjednoduší vývoj nabízenými funkcemi a navíc nás vede k využití nejmodernějších funkcí, které dnešní web nabízí. Kromě toho, že je sám kvalitně vytvořen pomocí objektového programování vede i uživatele k tvorbě takto robustního kódu, navíc nabízí funkce pro jednodušší práci s uživatelským výstupem – šablonování systém se kterým se tvorba HTML stránek stává jednodušší a přehlednější. Stejně tak podporuje dnes již běžné metody pro vytváření interaktivních aplikací za pomoci AJAXu nebo nabízí funkce pro ladění aplikace, což je rovina kde samotné PHP nenabízí takřka žádné nástroje kromě výpisu strohého chybového hlášení. Další z neméně významných kvalit je orientace na bezpečnost vytvořené aplikace, jak je psáno v příručce k Nette: „*používá revoluční technologii, která eliminuje výskyt bezpečnostních děr a jejich zneužití, jako je například Cross site scripting, session hijacking, session fixation a další*“. [6] Výčet dalších funkcí, které posouvají úroveň práce v PHP na obdobnou s dnešními běžnými vysokoúrovňovými jazyky, je rozsáhlý a mnohé stojí za pozornost. Jednou z hlavních předností tohoto frameworku je také způsob jeho využití při tvorbě. Jak jsem již psal, můžeme využít jen

jednotlivě jeho části nebo využít způsob rozčlenění aplikace pomocí využití návrhové architektury MVP, kterou Nette silně podporuje a nabízí díky ní velké zjednodušení při tvorbě rozsáhlých projektů.

3.1.3 MVC a MVP architektura

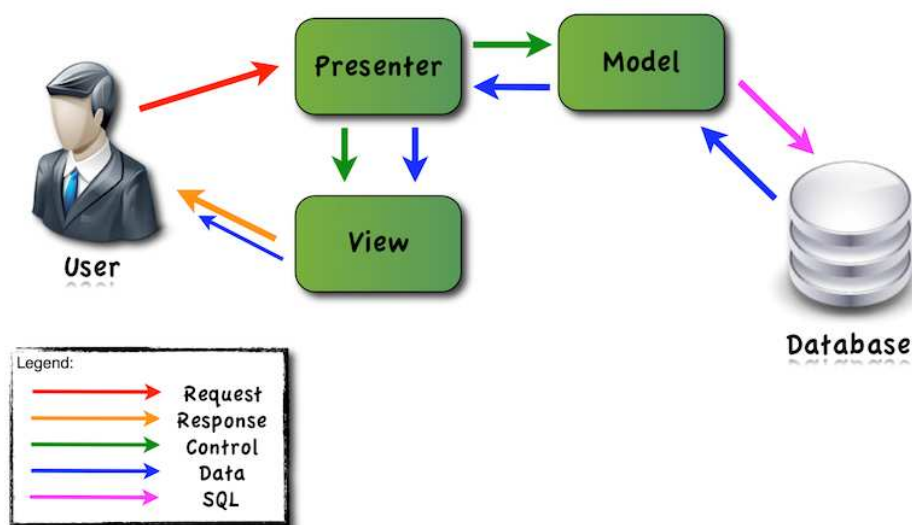
Větší aplikace je potřeba hned z několika důvodů členit do přehledné struktury nejlépe dle významu, tak aby jednotlivé části byly mezi sebou logicky propojeny, fungovaly jako celek a zároveň byly jako části použitelné i jinde. Toto dělení má výhodu, že se zvyšuje robustnost kódu, chyba v jedné části nemusí nutně způsobit chybu v jiné a taky vede ke tvorbě malých celků kódu, které lze velmi jednoduše použít jinde, také je díky tomu jednodušší práce v týmech, kdy každý má na starosti pouze svou část a ve výsledku dojde k propojení jednotlivých částí. Jedním ze způsobů členění aplikace, který se dnes využívá je i architektura MVC z anglického Model-View-Controller případně její odnož MVP neboli Model-View-Presenter, který je využit i v Nette Frameworku.

Podle Nette je MVC : „*spíše než návrhový vzor softwarová architektura, která rozděluje aplikaci do tří vrstev: na datový model, uživatelské rozhraní (view) a řídicí logiku (controller). Přičemž modifikace některé z nich má pouze minimální vliv na ostatní.*“ [6] Webové aplikace k tomuto dělení přímo vybízejí a tak je toto členění velmi vhodné použít. Popíšme si tedy jednotlivé vrstvy podrobněji:

- model – jde o model dat pro aplikaci, slouží nám k manipulaci s daty, tento model vychází z toho, co jsme nazývali modelem ‚reálného světa‘ při návrhu databáze a tato vrstva je jeho logickým vyústěním v programovém kódu, jde tedy o vrstvu, která nám zapouzdřuje komunikaci s databází nebo souborovým systémem a nabízí ji pomocí procedur ostatním vrstvám, model o existenci view nebo controlleru neví,
- view – uživatelské rozhraní nebo také pohled, vrstva systému která slouží k finální prezentaci dat z modelu a převádí je do podoby vhodné k zobrazení uživateli, tedy HTML kódu pro webovou aplikaci, komunikuje modelem a zároveň je jeho podoba ovlivněna controllerem,
- controller – řídicí logika nebo řadič systému, který slouží jako hlavní logika aplikace, tedy má za úkol vyřídit požadavky od uživatele, iniciovat změny v modelu nebo

pohledu, řídí a ovlivňuje jejich podobu, obvykle je volán již s určitým pohledem a controller má přehled o svých pohledech.

Návrhový vzor MVC je ale poněkud složitější a dnes je velký rozdíl mezi jeho původní definicí a využitím. „*Tento princip poprvé popsal Trygve Reenskaug v roce 1979. Dnes je velmi populární právě u webových aplikací, jenže často jde o tvrzení pramenící z jeho pochopení. Ve své původní podobě jej vlastně nepoužívá nikdo. Role a vztahy jednotlivých vrstev se často chápou velmi volně. To je také důvod, proč se Nette Framework hlásí k MVC jen jako k duševně spřízněné architektuře.*“ [6] Nette využívá méně známou variaci Model-View-Presenter neboli MVP, kde controller je nahrazen presenterem a logika provázanosti vrstev se liší, částečně připomíná třívrstvou komunikační architekturu klient - server. Uveďme si jednoduše jaký je rozdíl těchto návrhových vzorů, respektive co mění presenter v celé architektuře. Pokud využíváme způsob pasivních view, tak již pohled nekomunikuje přímo s modelem, ale obdrží data od presenteru. Tímto dojde k tomu, že ani pohled nemusí vědět o modelu. Naopak tím vzrůstá význam presenteru, který je hlavním prvkem systému a propojuje jednotlivé části a pohled by měl sloužit k prostému vložení dat z presenteru do šablony bez jakékoliv modifikace. Aplikace je zde volána skrze pohledy oproti MVC, kde se volá přímo controller a ten vybírá pohled, který odesílá. V obou případech se jedná spíše o vzory, které mají vytvořit most mezi lidským modelem myšlení a tím počítačovým, než o systém tvorby, který je nutné dodržovat, naopak by měl vést k lepšímu smýšlení nad strukturovaností aplikace.



Obr. 3 Návrhový vzor MVP [6]

3.2 JavaScript

JavaScript je programovací skriptovací jazyk určený dnes zpravidla pro webové stránky spouštěný na straně klienta prohlížečem. Jde tedy o kód, který se odešle uživateli společně s HTML kódem a webový prohlížeč uživatele jej vykoná. Síla tohoto jazyka je skryta právě v možnosti pracovat na straně klienta a často se využívá k různým interakcím chování uživatele a webové stránky například zobrazení či skrytí části stránek, různé dialogové okna, změna grafiky případně složitější skripty pro vytvoření působivých interaktivních prezentací. V posledních letech se také díky JavaScriptu začalo využívat způsobu komunikace mezi prohlížečem a serverem, zvaném Ajax, který je popsán v další podkapitole. Provádění skriptů na straně uživatele má i své stinné stránky a JavaScript z tohoto důvodu nebyl v předešlých letech moc oblíben, naštěstí toto období je pryč a dnes je na velkém vzestupu. V dřívějších dobách, zhruba před 10 lety, byla jeho implementace v prohlížečích špatná a často obsahovala chyby, takže bylo známo mnoho případů, kdy byl vytvořen kód, který působil uživateli škody. Proto obliba tohoto jazyka u uživatelů klesala a ti zakazovali v prohlížečích provádění těchto skriptů. Poté co prohlížeče začali implementovat jazyk tak, aby fungoval standardně a zároveň nedovoloval ohrozit bezpečnost prohlížeče nebo systému, například možností práce se souborovým systémem, se důvěra v něj vrátila a k čemuž dopomohli i některé velké firmy na trhu webových aplikací, jenž nabízeli aplikace, které JavaScripty vyžadovali. To byla postupná cesta k Ajaxu a kvalitní podpoře dynamické objektové modelu stránky. Dnes se setkáváme s JavaScriptem ve spoustě moderních aplikací. John Resig říká: *„jazyk JavaScript se vyvíjel postupně, ale trvale. Za minulé desetiletí se z jednoduchého programovacího jazyka na hraní stal respektovaný programovací jazyk, který používají společnosti a vývojáři na celém světě k vytváření neuvěřitelných aplikací. Moderní programovací jazyk JavaScript je solidní, robustní a neuvěřitelně mocný nástroj.“* [3]

3.2.1 jQuery

Knihovna jQuery je jedna z dnes nejpoužívanějších sad funkcí pro zjednodušení práce s objektovým modelem webové stránky pomocí JavaScriptu, první verzi vydal v lednu roku 2006 John Resig. Je vyvíjena tak, aby respektovala specifika použití JavaScriptu v jednotlivých prohlížečích a zároveň byla rychlá a jednoduchá, což se vývojářům této volně dostupné knihovny daří a svědčí o tom její velké využití na spoustě moderních webů,

které se snaží díky JavaScriptu zpříjemnit uživateli prostředí nebo využívají Ajax. Dnes tuto knihovnu využívají stránky firem jako je Google, Amazon, IBM nebo Microsoft. Navíc se dá tato knihovna velmi jednoduše rozšířit, takže do ní snadno implementujeme vlastní funkce. jQuery nám nabízí mnohem komfortnější využití JavaScriptových funkcí. Složitější operace, které bychom psali na několik řádků, takto zvládneme jediným jednoduchým zápisem a velmi efektně. Představme si základní funkcionalitu knihovny, která nám zjednodušuje práci s těmito oblastmi:

- výběr, změna a přidání do objektového modelu stránky, jednoduchá práce s výběrem elementů a jejich následným využitím, snadná možnost vložit do stránky další kód,
- obsluha událostí s možností dynamicky navazovat události na elementy stránky, zachytávat události uživatelem či prohlížečem samotným vyvolané,
- manipulace s CSS stylopisy sloužícími k definici grafické podoby webu,
- selektory pro jednoduché vyhledání konkrétní části stránky, elementu například podle hierarchie v dokumentu,
- efekty a animace – jednoduché použití s efektními výsledky, často využívané funkce pro dynamické uživatelské rozhraní v podobě různých menu či dialogů,
- Ajax – načítání obsahu ze serveru bez nutnosti načtení celé stránky,
- utility například s informacemi o prostředí prohlížeče nebo zjednodušení práce s poli.

3.2.2 Ajax

Ajax je zkratka anglického Asynchronous JavaScript and XML a jde o technologii vývoje webových aplikací, jenž reagují na uživateli požadavky načítáním obsahu ze serveru bez nutnosti znovunačtení celé stránky. Díky této funkcionalitě se posunula možnost vytvořit interaktivní aplikaci na velmi použitelnou úroveň, kdy dnes některé aplikace nabízí velmi propracované rozhraní působící dojmem práce s klasickou aplikací. Termín Ajax je znám od roku 2005, ale samotná myšlenka způsobu interakce uživatelského prostředí je mnohem starší. Ajax nabídl pouze myšlenku, jak využít dnešní programovací jazyk JavaScript pracující na straně uživatele společně s podporou odesílání XML požadavků prohlížečem pro asynchronní komunikaci se serverem. Jednotlivé části se mohou lišit, nemusí se například jednat o XML přenos, ale HTML, podle Resiga: „*Ajax*

zahrnuje tisíce různých datových komunikací, ale všechny mají společný předpoklad – dodatečné požadavky jsou prováděny směrem od klienta k server dokonce i v případě, kdy už je stránka kompletně načtena. To umožňuje aplikačním programům vytvářet další interakce, aniž by uživatelům zpomalili tradiční běh aplikace.“ [3] Velmi záhy se začala myšlenka stávat skutečností a do knihovny jQuery byla implementována podpora Ajaxu několik měsíců po vydání první verze. V době prezentace myšlenky Ajaxu se tímto způsobem tvorby aplikací již zabývala i firma Google, která je velkým průkopníkem v této oblasti, a měla již spuštěnou první interaktivní ukázkou – dnes běžně využívaného automatického doplňování slova ve vyhledávacím poli, takzvaný autocompleter. Autocompleter pracuje po každém stisku klávesy, kdy odesílá požadavek na server na seznam slov pro doplnění rozepsaného slova, odpověď pak zobrazí jako nápovědu.



Obr. 4 Autocompleter od společnosti Google

Možnost použít Ajax je tedy funkční již několik let a kromě zlepšení uživatelského prostředí stránky nabízí při dobrém návrhu serverové části aplikaci snížení objemu dat pro přenos, naopak špatný a někdy složitý proces implementace může vést nejen ke zpomalení a prodražení vývoje aplikace, ale nemusí přinést ani kýžený efekt menšího zatížení serveru. Při tvorbě klientské části, tedy HTML kódu i serverové aplikace také musíme dbát na to, že prohlížeč uživatele může být starší se špatnou podporou JavaScriptu a nebo uživatel může mít vypnuté provádění skriptů. Musíme tedy aplikaci psát, tak aby byla funkční i bez skriptů například pro webové stránky sloužící k prezentaci nebo u složitějších aplikací určených pro specifické účely si vyžádat použití odpovídajícího softwaru, tak aby byla aplikace funkční. Dnes běžné prohlížeče nemají s Ajaxem problém v podobě funkčnosti, ale stále nenabízí dobrou podporu zaznamenání stavu po Ajaxovém volání například pro možnost vrátit se zpět nebo posunout na další stránku, protože webová aplikace si již nechová jako posloupnost stránek, ale plnohodnotná aplikace s vnitřní logikou, která mění svůj stav bez změny adresy stránky. Proto se některé moderní Ajaxové aplikace toto snaží sami eliminovat za použití různých technik, prohlížeče se teprve v této době snaží na tento problém reagovat.

3.2.3 jQuery UI

jQuery UI je, jak již z názvu samotného vyplývá, nadstavbou knihovny jQuery a UI je zkratka anglického user interface neboli uživatelského rozhraní. Jde tedy o JavaScriptovou knihovnou doplněnou o grafický styl v podobě CSS souboru, pro zjednodušení tvorby grafického rozhraní a interakci s ním. Balík funkcí jQuery UI nabízí tři základní skupiny modulů pro práci s webovou stránkou:

- widgety pro webovou stránku, což jsou mále komponenty pro jednoduché obohacení grafického prostředí, které jsou v podobě několika běžně využívaných nástrojů, jako jsou dialogové okna, kalendáře a další s plnou podporou zvoleného vizuálního stylu, tak aby nabídli jednoduše interaktivní prvky na stránce bez nutnosti je složitě programovat,
- efekty, pro práci s elementy stránky například v podobě jednoduché práce s vizuálními styly a vlastnostmi nebo složitější animace jako explodování, rozložení a další,
- rozšíření interakce myši s prvky stránky což jsou například funkce drag and drop neboli táhni a pusť, možnost označení prvků, změna velikosti a jiné.

Tato knihovna vznikala nejprve jako plugin pro jQuery v roce 2007 a postupně díky oblibě se stala oficiálním rozšířením této knihovny díky velké oblibě uživatelů. Stejně jako jQuery se jedná velmi kvalitní JavaScriptový kód, který velmi dobře pracuje na spoustě dnešních prohlížečů. Také díky oblíbenosti mezi uživateli a podpoře v komunitě vývojářů jsou případné chyby rychle odstraňovány a knihovna je dále vyvíjena a rozšiřována o nové funkce, v poslední verzi 1.8 přibyla například funkce pro vytvoření tlačítka nebo automatického dokončení slova vyhledávacího pole. Při vývoji bylo myšleno i na možnost nefunkčnosti JavaScriptu, takže stránka při nenačtení této knihovny obsahuje obvyklý HTML kód a není třeba jej nějak výrazně upravovat pro přidání funkcionality knihovny. Vše se dá velmi jednoduše řešit zápisem několika řádků JavaScriptu, které promění jednoduše některé statické elementy stránky na interaktivní prvky.

II. PRAKTICKÁ ČÁST

4 VYTVOŘENÍ DATABÁZE A APLIKACE

Základem pro vytvoření databáze a samotné aplikace je mi předešlá teoretická i praktická znalost použitých technologií. Snažil jsem se přistoupit k problému otevřeně a s chutí, tak abych využil předešlých zkušeností a zároveň zkusil najít nové způsoby řešení a zvážil alternativní řešení, které by mohli posunout mou praktickou tvorbu a zkušenosti na vyšší úroveň a zároveň umožnili vytvořit aplikaci na kvalitní úrovni.

Programový kód je psán strukturovaně pomocí objektově orientovaného programování za použití návrhové architektury MVP, tak aby jednotlivé části aplikace byli strukturované podle logiky jeho funkce a mohl být jednoduše spravován. Celý kód je opatřen komentáři, podle standardu PHP-Doc jsem vytvořil informační hlavičky pro všechny důležité funkce zvláště v modelu je vše přehledně popsáno a definováno, tak aby byli vlastnosti procedur a atributů pochopitelné bez dalšího zkoumání. Celá aplikace je napsána tak aby nezasvěcená osoba programování znala se v kódu rychle zorientovala a pochopila jeho účel.

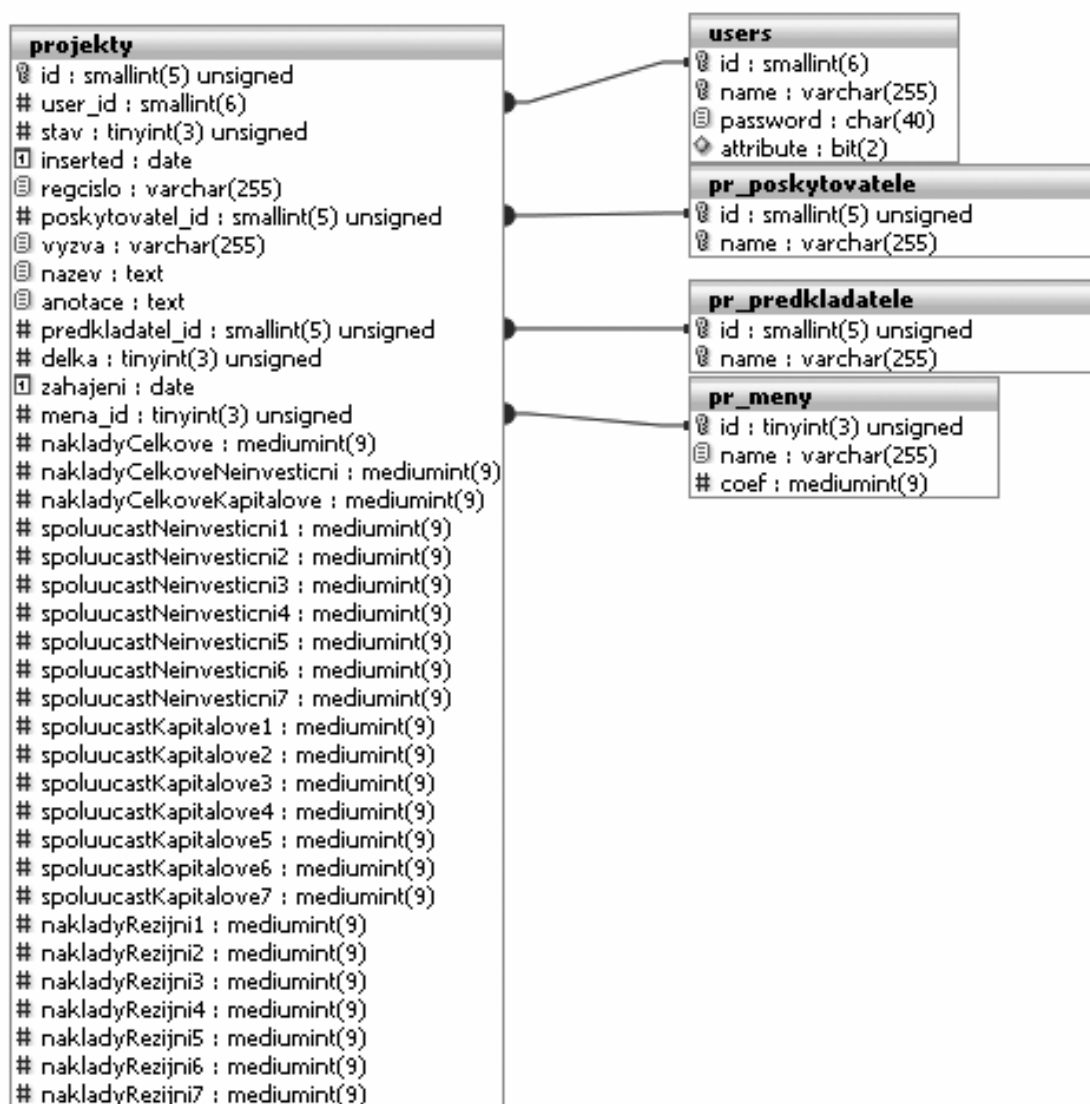
4.1 Tvorba schématu databáze

Entitě-relační databáze bude vytvořena a popsána jazykem SQL, se kterým pracuje většina dnešních databázových systémů pro tento model.

Schéma databáze obsahuje 4 tabulky:

- ‚projekty‘ – obsahuje všechny informace z formuláře návrhu výzkumného projektu, některé v podobě cizí klíčů z číselníků poskytovatelů finančních prostředků, předkladatelů, měny a uživatele, který vložil záznam do tabulky, datum vložení a stav projektu v podobě celočíselné hodnoty
- číselník ‚pr_poskytovatele‘ – obsahuje jméno poskytovatele finančních prostředků
- číselník ‚pr_predkladatele‘ – obsahuje jméno předkladatele
- číselník ‚pr_meny‘ – obsahuje jméno měny a koeficient násobku pro náklady projektu
- ‚users‘ – obsahuje jména a heslo uživatele systému, vlastnosti v podobě role v systému a zakázané přihlášení

Grafické znázornění databáze s jejím propojením a kompletním seznamem položek v tabulkách s jejich SQL datovým typem je na obrázku Obr.5.



Obr. 5 Schéma databáze

Po uvážení návrhu databáze a vhodnosti zavedení cizích klíčů pro zachování integrity údajů v systému jsem se rozhodl z důvodů výkonnosti do databáze nedefinovat cizí klíče a kontrolu integrity záznamů přesunout na vrstvu aplikace. Pokud by se tato možnost do budoucna ukázala nevhodná, resp. zavedení indexů na sloupce v tabulce projektů nezpůsobilo velké snížení výkonu, můžou se zpětně doplnit. Nyní se domnívám, že je netřeba tyto indexy udržovat a integritu zajistí aplikační logika.

4.2 Tvorba aplikace architekturou MVP

4.2.1 Vrstva modelu

Model nám slouží k zapouzdření operací s daty, tak aby aplikační logika nepracovala přímo s daty, ale byla odstíněna od jejich fyzického uložení a komunikačních protokolů. Prezentační vrstva vůbec nemusí vědět, jakým způsobem jsou data ukládána, zná pouze funkce modelu, kterými jej od něj dostane. V naší aplikaci nám zajišťuje model práci s dvěma typy úložišť dat, jedná se o relační databázi a souborový systém aplikačního serveru pro ukládání příloh k projektům.

Hlavní funkce databáze jsou ve třídě DBModel:

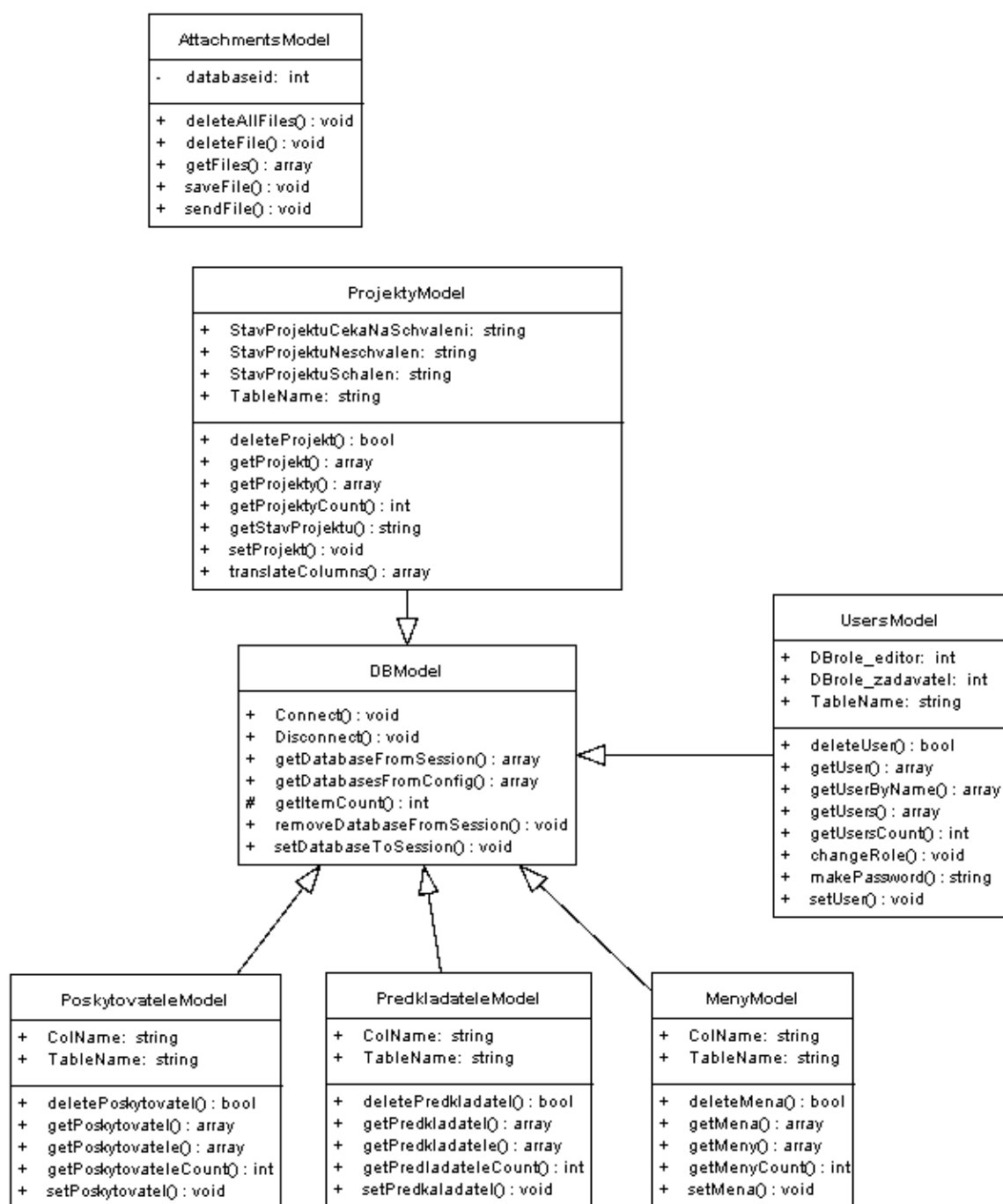
- připojení a odpojení databáze,
- vytvoření kompletní zálohy databáze včetně odeslání výsledného souboru uživateli,
- soukromé funkce pro potomky této třídy,
- načtení informací o dostupných databázích z konfiguračního souboru,
- uložení, načtení a smazání zvolené databáze pro aktuální sezení.

Z DBModelu dědí jednotlivé třídy logicky dělené podle tabulek, se kterými primárně komunikují, tedy pro které jsou určeny:

- ProjektyModel – zastřešuje práci s hlavní tabulkou, množství různých funkcí od základních pro uložení nebo smazání až po funkce pro seznam sloupců pro výstupní uživatelskou tabulku, překlad názvů tabulkových sloupců na uživatelská jména a další,
- UsersModel – práce s tabulkou uživatelů, obvyklé funkce pro přidání, úpravu, smazání, změnu práv či zakázání přihlášení, vytvoření seznamu všech uživatelů pro číselníky,
- PoskytovateleModel, PredkladateleModel, MenyModel – práce s jednotlivými číselníky, obdobné funkce pro výběr dat, uložení a smazání, při mazání se kontroluje, zda některý projekt neodkazuje na daný záznam.

Každý model tabulky navíc obsahuje konstanty se jmény tabulek, případně sloupců, u projektů i dalších vlastností – například hodnot odvozených ze sloupců, tak aby celá

aplikace pracovala právě s těmito konstantami a při případné změně například jména tabulky stačilo změnit jen definici jedné konstanty. Tyto konstanty vedou k tvorbě robustního kódu, který je méně náchylný na chyby v kódu v podobě překlepů, je přehledný a dá se jednoduše spravovat. Snažil jsem se také využít co nejvíce podobnosti kódu a zobecnit některé společné operace, ty poté využívají soukromé funkce z DBModelu. Například pro výpočet počtu záznamů dané tabulky se zavolá konkrétní model s tímto požadavkem a ten zavolá obecnou funkci v rodiči DBModel na zjištění počtu záznamů a předá ji jako parametr název tabulky.



Obr. 6 Model dat

Společným rysem všech modelů je jejich tvorba podobná návrhovému vzoru známému pod pojmem Singleton, což můžeme chápat jako jedince nebo unikát. Nejde přímo o vytvoření Singletonů, ale jejich variaci, kdy PHP umožňuje vytvářet statické metody a atributy ve třídách, takže můžeme přistupovat ne k instanci daného objektu, ale přímo ke statickým procedurám. To nám zaručí práci vždy s jednou konkrétní instancí, kterou si PHP interpret vytváří sám pro statické prvky. Tento návrhový vzor je vhodný právě

pro naše chápání modelu dat jako obálky nad konkrétním úložištěm dat, protože to je pro nás také právě jedno. Takto můžeme chápat danou třídu jako model dat pro aplikaci a stejně tak ji i využít.

Model dat zastřešující práci s tabulkou projektů si podrobněji probereme, protože obsahuje větší počet funkcí a konstant, které je vhodné uvést jako nezbytnou součást pro práci s projekty v aplikaci:

- konstanty pro název tabulky, zástupné názvy (aliasy) všech tabulek databáze – začínají prefixem ‚Table‘, indexy stavů projektů – prefix ‚StavProjektu‘,
- konstanty podle sloupců v databázi a případné sloupce od nich odvozené, které využívá celý systém pro práci s projektovými daty, mají prefix ‚Col‘
- vložení, změna projektu – provádí se validace vkládaných hodnot z číselníků, smazání projektu, zjištění zadavatele projektu pro ověření přístupu k některým částem webu,
- ověření klíčů z ostatních tabulek zda jsou použity v některém z projektů – funkce pro ostatní tabulky pro ověření integrity při mazání,
- práce se stavem projektu – převod na textový popis, barvy pozadí v závislosti na indexu stavu, vrácení všech stavů pro číselník při změně stavu,
- seznam sloupců pro zobrazení ve výstupní tabulce – definování výstupních sloupců pro uživatele v přehledu projektů,
- převod konstant sloupců třídy na databázové sloupce včetně odvozených hodnot případně hodnot z číselníků, soukromá funkce pro vytváření dotazů včetně propojení na ostatní tabulky, tentýž převod také na textové popisy sloupců,
- funkce pro ověření správného formátu data a hodnot nákladů při editacích projektů.

Tabulka tedy obsahuje komplexní správu projektových informací a neslouží jen k obstarání práce s jedinou tabulkou databáze, běžně komunikuje i s ostatními a hlavně přidává funkcionalitu v podobě definování sloupců pro výstupní sestavu a převod hlaviček sloupců do textového popisu.

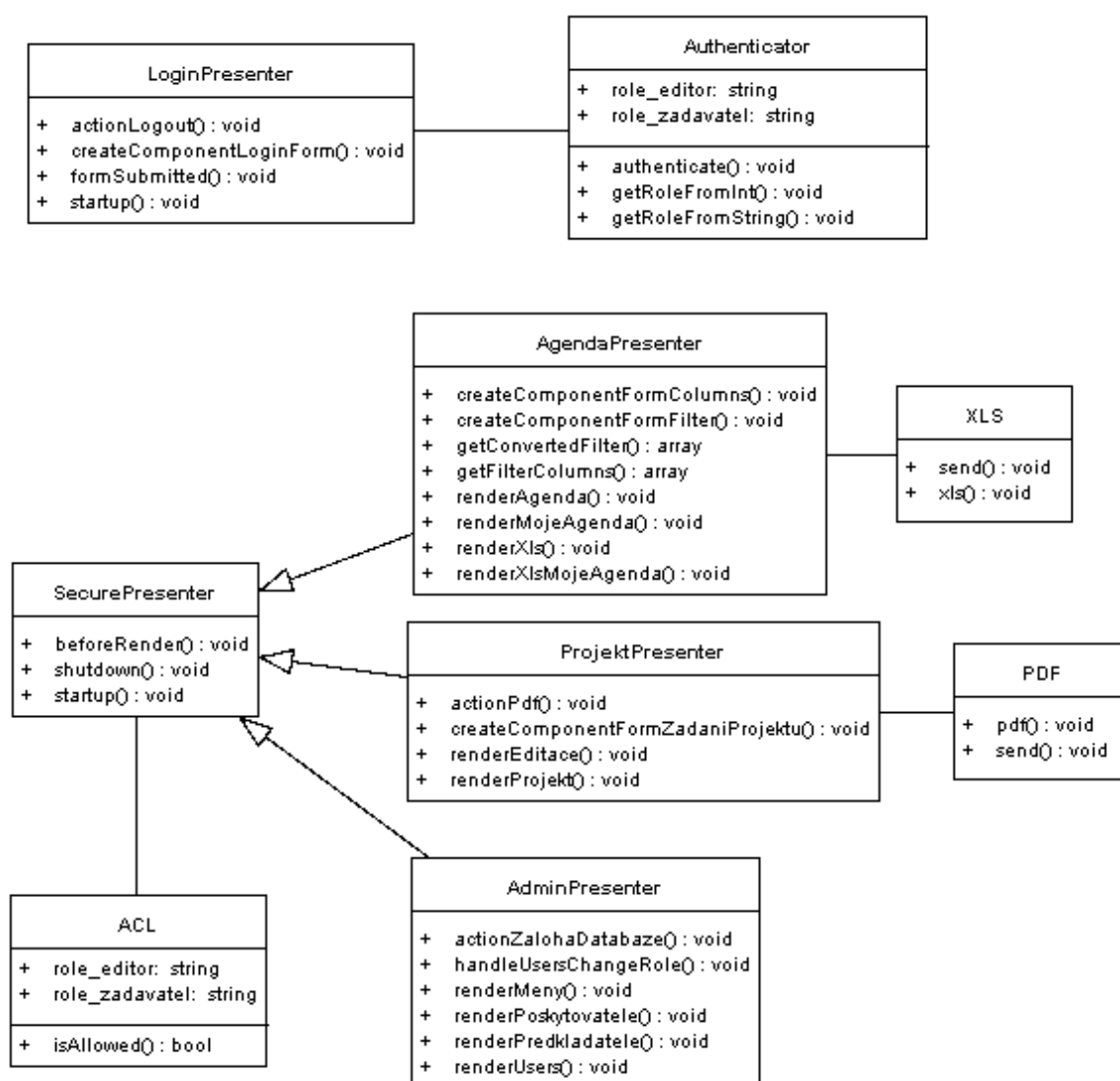
Na začátku jsem zmínil, že aplikace bude využívat dva typy úložišť, tím druhým je datové úložiště samotného serveru kde poběží webové server s interpretem PHP. Toto úložiště bude určeno pro přílohy k jednotlivým projektům. Z pohledu nastavení webového serveru

bylo nutné ošetřit přímý přístup k souborům, tedy zabránit mu, tak aby k souborům mohl přistupovat pouze oprávněný uživatel, což má na starosti prezentační vrstva. Model příloh je uložen ve třídě AttachmentsModel, která obsahuje základní funkce pro uložení, smazání, odeslání souboru uživateli nebo vytvoření seznamu příloh pro daný projekt. Tyto funkce pracují se souborovým systémem serveru pomocí PHP funkcí k tomu určených.

Jedním ze společných prvků všech modelů je vyhazování výjimek, které jsou pojmenované podle daného modelu, který je vyvolává. Jedná se o jeden ze způsobů jak zastavit provádění posloupnosti funkcí v případě neočekávaného chování nebo stavu a informovat o tom nadřazený blok. V našem případě může k těmto stavům dojít třeba při komunikaci s databází, kdy dojde k problémům v komunikaci. Poté funkce komunikující s databází vyhodí vlastní výjimku, kterou si model odchytí a následně provede její ošetření – například odešle prázdnou hodnotu prezentační vrstvě nebo v případě požadavku na důležitá data vytvoří model svou výjimku, kterou si musí ošetřit prezentační vrstva a ta dál informuje uživatele.

4.2.2 Prezentační vrstva

Členění této vrstvy odpovídá částečně návrhu informačního systému z jedné z předešlých kapitol. Tuto vrstvu tvoří dva celky – jeden slouží pro obsluhu nepřihlášeného uživatele a druhý velký celek samotná funkcionální aplikace pro správu výzkumných projektů.



Obr. 7 Třídy prezentační vrstvy

Z diagramu je patrné rozdělení jednotlivých částí systému s presentery:

- LoginPresenter slouží k obsluze nepřihlášeného uživatele případně k odhlášení, obsluhuje pohled s přihlašovacím formulářem, k ověření uživatele využívá objekt třídy Authenticator, který je definován v systému jako řadič práce s uživatelským účtem a obsahuje ověřovací rutinu,
- SecurePresenter je abstraktní rodič zbylých presenterů, nemůže být nikdy volán a slouží jen jako předek pro spouštění některých funkcí společných funkcí například jako hlavní funkci má při startu kontrolovat zda je uživatel přihlášen a poté zda má práva přístupu k dané sekci, což ověřuje pomocí třídy ACL, případně konkrétnímu projektu,

- ACL je zkratka anglického access control list což je v jednoduchosti řečeno seznam uživatelů, zdrojů a privilegií sloužící k ověření zda přihlášený uživatel má právo na vstup na konkrétní stránky - pohledy,
- AdminPresenter, AgendaPresenter a ProjektPresenter tvoří základní celky informačního systému, z jejich funkce jsou obsáhlejší a proto si je popíšeme jednotlivě.

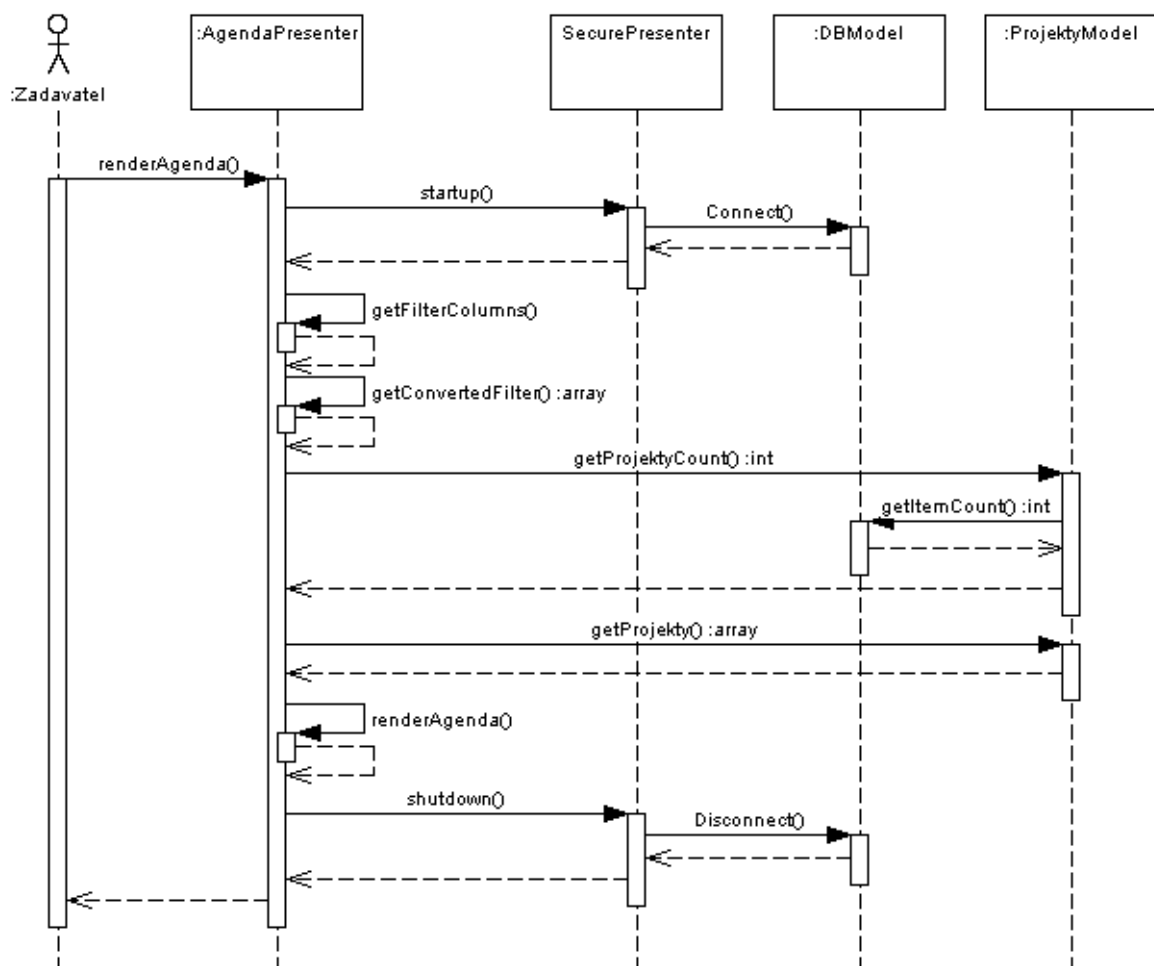
AdminPresenter zastřešuje administrační část systému, tedy obsluhuje jednotlivé pohledy pro správu číselníků a uživatelských účtů, obsahuje funkci pro obsloužení uživatelského požadavku na vytvoření zálohy databáze. Obsahuje funkce pro tvorbu jednotlivých formulářů, rutiny pro plnění šablon pohledů daty a řídí případné zobrazení formuláře na stránce. V pohledu uživatelských účtů nabízí kromě změny údajů také funkce pro změnu práv nebo zakázání přihlášení uživatele. Administrace je kompletně nepřístupná pro uživatele v roli zadavatele s omezenými právy, s touto rolí uživatel ani nevidí administrační menu pro přístup do jednotlivých pohledů.

AgendaPresenter obsahuje několik pohledů, celý presenter slouží k výstupům z tabulky projektů a její správě. Hlavní pohled s názvem agenda, respektive moje agenda s omezeným výpisem pro přihlášeného uživatele, slouží k zobrazení tabulky projektů a několika odkazů na zbylé pohledy, těmi jsou pohledy k výběru sloupců, které se v tabulce zobrazují, filtr hodnot ve sloupcích a poslední pohled vytvoří tabulku ve formátu excelového dokumentu a odešle ji uživateli, takže se nejedná o pohled v klasickém pojetí zobrazitelné stránky. Sice se jedná o malé množství pohledů, ale aplikační logika je vcelku velká hlavně v souvislosti s možností volby sloupců k zobrazení a filtrem hodnot. Jsou zde i klasické funkce pro tvorbu formulářů, je zde také funkce pro vymazání filtrů a nastavení zobrazení sloupců na základní hodnoty.

Přibližme si popis práce presenteru při zobrazování agendy projektů:

- požadavek na zobrazení od uživatele s možností volby sloupce a směru řazení hodnot,
- načtení seznamu sloupců k zobrazení z nastavení aktuálního uživatele, pokud nebylo vytvořeno, tak vezme inicializační seznam sloupců
- načtení filtru hodnot ve sloupcích a vytvoření výstupních textů s filtry a omezujícími podmínkami pro datový model

- inicializace stránkování
- informování pohledu o funkcích pro překlad názvů sloupců a stavů projektu
- načtení dat do pohledu – zavolání modelu projektů pro získání seznamu projektů, funkce se volá s parametry seznamu sloupců k zobrazení, filtrem hodnot, způsobem řazení a stránkování,
- procedura končí a předává řízení pohledu, který vezme HTML šablonu a provede dodatečné vytvoření tabulky s projekty, následně odesílá kód na výstup a končí běh aplikace.



Obr. 8 Sekvenční diagram zobrazení agendy

Obdobnou proceduru provádí i vytváření excelového souboru, ale od inicializace stránkování, kterou neprovádí, dojde k inicializaci třídy pro vytvoření dokumentu, poté je dokument naplněn daty, odeslán na výstup se specifickou hlavičkou dokumentu a presenter svou činnost ukončuje. Z tohoto rozboru procesu plyne jeho složitost, kdy se provádí

několik kroků, během kterých jsou data načítána a následně transformována na několik výstupních údajů. Například seznam sloupců k zobrazení je překládán jak na sloupce databázové při volání modelu, tak poté při výpisu do stránky jsou překládány na textové názvy v hlavičce tabulky, obdobně funguje i filtr hodnot sloupců. Z pohledu uživatele existují dva výpisy projektů – jeden obsahuje projekty zadané uživatelem samotným a slouží pro jeho přehled a druhý, který je přístupný jen uživateli v roli editora, kde jsou všechny projekty z databáze a přístupné všechny funkce ke změně projektů.

ProjektPresenter slouží k práci se samotným projektem, obsluhuje pohledy ke kompletnímu výpisu informací včetně příloh, vložení nebo editaci projektů či ke změně stavu, další funkce je export informací o projektu do PDF formátu v podobě dokumentu návrhu projektu. Editační formulář projektu je vcelku obsáhlý, obsahuje velké množství vstupů, které je potřeba ošetřit pravidly, tak aby vstupy byly nejenom korektní data pro databázi, ale i správné hodnoty některých údajů, které jsou na sobě závislé. Na stránce projektu pod informacemi je i seznam příloh a formulář pro vložení další. Presenter obsahuje funkce pro vyřízení požadavků uživatele na komunikaci s datovým modelem příloh a samozřejmě projektem. Tento presenter je částečně přístupný uživateli v roli zadavatele. Může takto přistupovat k projektům, které do systému zadal a případně je i editovat dokud jsou ve stavu čekajícím na schválení. Projekty, které do systému nezadal, nemá právo zobrazovat.

4.2.3 Pohledy – views

Jsou přímo závislé na presenterech pod které spadají a které je obsluhují, respektive poskytují jim data. Jedná se o holý HTML kód a šablonový skriptovací kód, který Nette Framework podporuje a překládá do klasického PHP kódu. Šablony jsou díky tomu velmi přehledné a jednoduše změnitelné. Aplikace také obsahuje takzvané layouts, což je kostra HTML kódu, která obaluje jeho obsah. Každá stránka obsahuje stejné prvky, jako jsou nezobrazované informace v hlavičce dokumentu, menu nebo patička stránky, tyto části tvoří layout a do něj je zasazena na konkrétní místo šablona pohledu. Tím je vytvořena hotová stránka a odeslána uživateli. Layouty jsou v aplikaci dva, jeden pro stránku s přihlášením, kde není například menu a některé další prvky a druhý je ten obvyklý pohled pro přihlášeného uživatele s menu, patičkou a informací o přihlášeném uživateli. Opět je vidět že layouts korespondují s celky presenterů, tedy rozdělením na část pro přihlášeného

a nepřihlášeného uživatele. Stejně tak platí i analogie s pohledy daného presenteru, kdy například správa uživatelů spadá pod administraci. Dále bychom mohli pokračovat i návazností na model, abychom ukázali, jak je systém logicky tvořen a zároveň jednotlivé části nejsou na sobě v případě problémů přímo závislé a lze je jednoduše nahradit nebo případně použít v jiné aplikaci. Aplikace je tak tvořena z robustních částí, které spolu úzce komunikují a tvoří kvalitní celek.

4.2.4 Ajax

Ajax je v aplikaci využit prostřednictvím knihovny jQuery, díky které můžeme tyto asynchronní požadavky na server velmi jednoduše zakomponovat do klientské části aplikace. Stejně tak i Nette Framework, který se snaží poskytovat podporu nejmodernějších technologií na webu, nabízí několik funkcí, díky kterým je vytvoření Ajaxové aplikace úpravou jen na několik řádků kódu. Není tedy problém vytvořit celou aplikaci jako Ajaxovou, ale toto řešení by mělo přinést jak zvýšení komfortu uživatele, tak i zmenšení zátěže serveru a to nejlépe s poskytnutím zpětné kompatibility aplikace pro její použití bez Ajaxu.

Vzhledem k tomu, že vyvíjený systém obsahuje na každé své stránce – pohledu jednoduchou prezentaci dat, aby byla jednoduše pochopitelná a použitelná, tak samotné načtení obsahu pohledu pomocí Ajaxu by znamenal vykonání stejného kódu aplikace na straně serveru a odeslání obdobného výstupu jako při klasickém požadavku, jen by se neodesílali části stránky jako je hlavička a patička. Tedy úspora zátěže serveru velmi mizivá, stejně tak přenos informací, kdy největší část výkonu a následně přenosu dat zabírá právě obsahová část. To byl jeden z důvodů, proč jsem nezavedl vykonávání všech uživatelských požadavků aplikace pomocí Ajaxu. Druhým byla právě podpora těchto volání v navigaci prohlížeče například pro tlačítka zpět a vpřed. Nechtěl jsem zbytečně zatěžovat prohlížeč složitou obsluhou těchto událostí, protože aplikace by si měla hlídat tuto navigaci a korektně ji ošetřit. Také by měl být stav aplikace prezentován jednoznačnou adresou, což se opět Ajaxovým voláním poněkud zhoršuje. Stejná adresa by nám měla vždy nabídnout stejný obsah, což pro zachování jednoduchosti aplikace hovoří dnes v neprospěch Ajaxu. V příštích letech prohlížeče budou lépe podporovat zaznamenávání stavu aplikace po Ajaxových voláních a možnost lépe pracovat s adresou aplikace, dnes se dá jednoduše zaznamenat stav aplikace pouze za adresu za znak mřížky. To se neslučuje

s moderním trendem vytvářet jednoduše čitelné webové adresy, tak aby i uživatelé podala informaci o obsahu stránky a toto jsem chtěl u systému zachovat. Nyní jsou adresy tvořeny jednoduchou strukturou, kdy nejprve je název sekce například projekt nebo agenda a poté za lomítkem konkrétní pohled na danou sekci, takže například projekt/editace nebo agenda/moje-agenda. Aplikace je tedy plně použitelná bez Ajaxu a splňuje standardy přístupnosti, ale i přesto je v ní aspoň částečně využito Ajaxových požadavků v několika případech, které jsem považoval za vhodné z uživatelského pohledu pro zpříjemnění práce. Toto použití je dále doplněno o funkce z knihovny jQuery UI pro vytváření dialogových oken do kterých je načtený obsah zobrazen. Uživatel tedy neopustí stránky, ale dostane obsah stránky z odkazu zobrazený v dialogovém okně. Toto je využito například u formulářů volby sloupců nebo jejich hodnot pro zobrazení v agendě výzkumných projektů, při změně stavu projektu nebo ověření akce mazání, tedy informací, které přímo souvisí s obsahem stránky na které uživatel právě je a zároveň to nejsou informace, které by si přímo žádali nové zobrazení stránky například v podobě přechodu z informační stránky o projektu na jeho editaci, kdy je vhodné načtení celé stránky i změny adresy.



Obr. 9 Dialogové okno

4.3 Dibi databázová vrstva pro PHP

Dibi je open source knihovna funkcí zastřešující práci s databází, autorem je zakladatel Nette Frameworku, jenž ji vytvořil pro zjednodušení práce programátora a možnosti širšího využití databází. Nette přímo tuto knihovnu nevyžaduje, i když většina moderních PHP frameworků nabízí právě obdobné knihovny pro modelovou vrstvu, protože nabízí komfortnější přístup k operacím, které jinak zabírají mnoho řádků, takto jsou pomocí i jediného příkazu dostupné. Dibi se nechce konkurovat velkým databázovým vrstvám, které se snaží vytvořit plnohodnotnou vrstvu nad databází a zajistit tak další funkcionalitu, ale snaží se zefektivnit rutinní práci s běžnými dotazy.

Přehled cílů Dibi:

- zjednodušení zápisu SQL příkazů nejen pomocí funkcí jako je insert nebo update, ale hlavně formou zjednodušení syntaxe SQL zavedením operandů a jednoduchým vkládáním proměnných – například při update příkazu nebo klauzuli where omezení výpisu stačí zadat pole hodnot s klíči jako názvy sloupců,
- zpřehlednění zápisu dotazů právě díky přehlednější struktuře a vkládání proměnných, případně možnost tvořit takzvané plynulé dotazy, kdy vytvoříme základní dotaz a na ten postupně aplikujeme vlastnosti od volby sloupců po omezení počtu řádků a to vše během provádění logiky aplikace a v místě potřeby dat teprve dotaz spustíme,
- jednoduchý přístup k metodám bez nutnosti vytvářet instance třídy, dibi je vytvořeno jako Singleton a tudíž stačí zavolat statickou metodu třídy,
- „automatická podpora konvencí (escapování/slasho-vání, uvozování identifikátorů), automatické formátování speciálních typů, například datum nebo řetězec, sjednocení základních funkcí (připojení k databázi, vykonání příkazu, získání výsledku)“ [7],
- rozšíření o často potřebné funkce, které běžné databázové funkce nenabízí a tudíž si je programátor musel sám vytvořit, Dibi se snaží zachovat jednoduchost a přitom ušetřit programátorovi čas s běžnými úkony – například vrácení výsledku dotazu jedinou funkcí v podobě asociativního pole kdy klíčem každého řádku může být zvolený sloupec,
- přenositelnost mezi databázovými systémy, pokud Dibi pro daný systém má ovladač, je možnost jednoduše jej využít, díky tomu můžeme Dibi využívat nezávisle na konkrétním databázovém systému, pokud tedy nevyžadujeme specifické funkce pro daný systém, což ve většině běžných webových aplikací není potřeba.

Jak jsem v posledním bodě cílů zmínil, Dibi využívá své ovladače pro práci s jednotlivými systémy řízení báze dat. PHP jazyk má pro podporované systémy konkrétní funkce, takové aplikace jsou část psány pro konkrétní systémy a využití na jiném je možné až po přeprogramování tedy výměně modelu za jiný při dobré architektuře aplikace. S Dibi tento problém odpadá a pomocí jediného příkazu, který můžeme odeslat společně s parametry pro připojení k databázi, mu nastavíme ovladač a tím je zvolen databázový

system. Tato vlastnost zajišťuje při využití modelové vrstvy její oproštění od nutnosti znát cílový databázový systém, pokud tedy nevyžaduje některé specifické vlastnosti.

Aplikace databázového systému pro správu výzkumných projektů je tak nezávislá na konkrétní databázi, plně podporuje nejen žádanou MySQL, ale je možností ji provozovat i na jiných běžných databázových systémech. Takto může v aplikaci využít více databází na několika fyzických strojích, jediná webová aplikace tak může být využita několika organizacemi a každá organizace může pracovat s jinou databází na jiném serveru, což je právě výhoda třívrstvé komunikační architektury.

5 SPRÁVA A ZABEZPEČENÍ DATABÁZE

Na správu databáze musíme nahlížet ze dvou pohledů:

- administrátor databáze nebo celého databázového systému – nemá souvislost s daty v databázi, ale má k nim plný přístup, měl by se starat o bezproblémový běh systému, jeho bezpečnost a řešit zálohování na úrovni systému,
- administrátor informačního systému – má přístup k databázi, jaký mu nabízí informační systém, tedy nemusí se jednat o plnou kontrolu nad databází, ale jeho úkolem je právě práce s daty v databázi.

Administrátor databáze je tedy technicky zaměřená osoba, která databázový systém spravuje a poskytuje nám přístup k jeho funkcím. Tato osoba vytvoří danou databázi a potřebné přístupové účty k ní, tak aby aplikace mohla s danou databází pracovat. Měl by se starat o bezproblémový běh systému, tak aby databáze byla stále přístupná pro naši aplikaci. Databázový systém by měl být pravidelně zálohován proti případnému neočekávanému problémovému stavu, kdy by mohlo dojít ke ztrátě dat. Toto je ovšem služba, kterou můžeme vyžadovat, ale není nutností. Vždy je vhodné zabezpečit zálohování i vlastním řešením, tedy aplikací.

Administrátor informačního systému má přístup k databázi v podobě funkcí systému, ale také zná hodnotu a smysl informací v databázi. Má na starosti správu systému z pohledu obsahu informací a práce s nimi, využití jeho funkcí. Jeho práce tedy není přímo s databází, ale pracuje s ní zprostředkovaně. Informační systém by se měl starat o to, aby práce s daty byla jednoduchá a uživatel – administrátor nemohl jednoduše způsobit škody na systému, aplikace by měla být kvalitně vytvořena, aby její funkce i v krajních případech nezpůsobila poškození dat.

V informačním systému pro správu výzkumných systémů je administrátor nazýván editorem, tedy uživatelská role s největšími právy. S těmito právy může uživatel využívat všechny funkce systému, tedy přidávat a mazat data z jednotlivých tabulek databáze, případně měnit jejich vlastnosti a tím smysl dat což může mít další následky při práci s těmito daty. Všechny tyto funkce jsou kontrolovány systémem například na zabezpečení integrity, takže uživatel nemůže poškodit celistvost informací. Jediným způsobem, kdy se editor skrze informační systém dostane k čistým datům v podobě SQL dotazů je příkaz pro zálohování databáze, kdy je prohlížeči odeslán soubor s kompletním výpisem databáze.

Jde tedy pouze o výstup pro individuální zálohování a není možnost opačného procesu, tedy spustit dotazy nad databází například v podobě načtení špatného souboru.

Zajištění bezpečnosti není jen zajištění samotných dat uložených v databázi, ty mohou být jen cílem, ale jde o bezpečnost celé aplikace. Castagnetto říká: „*zabezpečení databáze zahrnuje také hardware, software, pracovníky a data. Efektivní implementace zabezpečení vyžaduje odpovídající kontroly, které jsou specifikovány v dílčích cílech systému. Ačkoli společnosti potřebu zabezpečení v minulosti zanedbávaly nebo přehlížely, v současnosti si uvědomují jeho význam. Důvodem tohoto obratu je skutečnost, že rostoucí podíl klíčových dat společnosti je uložen na počítačích a ztráta nebo nedostupnost těchto dat může mít katastrofální následky.*“ [2] Databáze je tedy významný zdroj informací a měla by být odpovídajícím způsobem zabezpečena, měli by se minimalizovat rizika ztráty dat. Data by měla být tajná, proto je také přístup do aplikace ověřován přihlašovacími údaji a i dále pohyb v aplikaci kontrolován právy přístupu, tak aby se nepovolaná osoba nedostala k tajným informacím. Dalším problémem, který úzce souvisí se zabezpečením je zajištění integrity dat. Narušení integrity má za následek znehodnocení nebo úplnou neplatnost dat. V naší aplikaci se jedná o provázanost projektů s jednotlivými číselníky, kdy pokud by došlo ke smazání hodnoty číselníku na kterou je odkazováno z projektu, tak dojde ke ztrátě důležité informace o projektu a tedy znehodnocení dat. Narušení integrity může být způsobeno událostí, která znepřístupní systém a poškodí jeho data, tedy například různé havárie. Při zabezpečení dat se dnes také setkáváme s pojmem počítačová kriminalita a její aktivita stále stoupá s tím, jak se postupně zvětšuje podíl uložených informací na počítačích. Naší aplikaci doufejme nehrozí cílené útoky pro krádež dat z důvodů hodnoty obsažených informací, které jsou částečně veřejné, ale to neznamená, že není aplikace zabezpečena. Samozřejmě by aplikace mohla vyžadovat daleko větší standard zabezpečení, ale potom bychom museli úplně změnit specifika na její provedení. Míra zabezpečení by měla odpovídat ceně informací v systému. Cílem zabezpečení databáze proto těmito problémům má být minimalizace jejich škod v efektivním měřítku a bez omezení uživatelů aplikace.

Databázový systém pro správu výzkumných projektů se snaží minimalizovat zmíněná rizika několika způsoby:

- přístup do systému je ověřován jménem a heslem uživatele,

- uživatele do systému přidává pověřená osoba a má kontrolu nad možnostmi přihlašování do systému a právy uživatelů,
- uživatel v systému může mít omezená práva (role zadavatele), má přístup jen k některým částem systému a datům,
- zálohování systému může být prováděno automatizovaně přímo na serveru a zároveň manuálně ze systému, manuálně se provádí plná záloha databáze včetně struktury dat, data příloh jsou uloženy na webovém serveru a nelze je jednoduše zálohovat přes PHP skripty, proto je nutné jejich ruční zálohování pro případ ztráty, webový server by měl být chráněn proti možnostem ztráty dat automatickým zálohováním,
- integrita systému je zajištěna aplikační logikou,
- šifrování se provádí pouze pro heslo uživatele, tak aby při případném odhalení dat z databáze nebylo možné jednoduše zjistit tuto osobní informaci, zbylá data toto nevyžadují, protože se nejedná o soukromé informace a z velké části jsou to data veřejná,
- aplikace nevyžaduje specifické požadavky na její nepřetržitý běh nebo jiné specifické potřeby, dočasná nedostupnost nezpůsobí žádné škody.

6 IMPLEMENTACE SYSTÉMU

6.1 Testování

Součástí vývoje a spuštění provozu je také testování aplikace. Testování probíhá v několika krocích podle fáze vývoje systému:

- testování malých celků aplikace při vývoji – nemusí přímo souviset s požadovanými funkcemi informačního systému, může se jednat o testování malých částí kódu pro ověření jejich funkčnosti,
- testování částí systému po dokončení jejich vývoje – ověření některých požadavků na systém a správnosti funkce, nedochází k plnému prověření všech možných případů vstupních hodnot,
- testování systému jako hotové celku připravovaného k vydání – testuje se od úvodní instalace, vytváří se kopie systému a nová databáze, obvykle se provádí na jiném serveru než doted', ověřuje se celá funkcionalita, zda systém splňuje všechny požadavky, provádí se testování krajních vstupů a vytváří se mezní situace, tak aby se odzkoušela robustnost aplikace, v této fázi již může být k testování přizván i zadavatel systému a může aplikaci testovat,
- testování během nasazení systému do produkce – sleduje se, zda systém pracuje podle očekávání, monitorují se kolize a dochází k jejich reportování tvůrci aplikace, protože testování nemusí vždy obsáhnout všechny možné varianty použití systému.

6.2 Instalace systému

Instalace systému je velmi jednoduchá, pokud jsou splněny nefunkční požadavky na systém, na kterém má běžet aplikační serverová logika a databáze. Aplikaci stačí nakopírovat do složky přístupné přes webové rozhraní. Databázi stačí na databázovém serveru vytvořit jako prázdnou bez tabulek, stačí, aby administrátor sdělil údaje pro přihlášení a název databáze. Aplikace se postará o vytvoření schéma databáze a vloží do databáze uživatele s nejvyššími právy editor, tento uživatel je přístupný pod jménem ,admin' a heslem ,admin', velmi se doporučuje toto nastavení co nejdříve změnit, protože se jedná o známé často využívané pojmenování k přístupu k účtům.

6.3 Nastavení databáze a aplikace

Aplikace je vytvořena tak, aby umožňovala jednoduché nastavení přístupových údajů k databázi pomocí konfiguračního souboru, kde je vše přehledně zapsáno na jediném místě. Tento soubor je zabezpečen, aby se k němu nedalo dostat přes webový prohlížeč, protože obsahuje důležité ověřovací údaje k přístupu k serveru. Obsah konfiguračního souboru je kromě části ,common‘, kde jsou některá běžná nastavení aplikace rozdělen na části ,production‘ pro nastavení v produkčním prostředí a ,development‘ pro nastavení prostředí pro testování. Nastavení databáze funguje na stejném principu pro obě sekce, nyní si popíšeme jeho strukturu včetně názvů parameterů:

- `database.count = 1` – obsahuje počet databází v nastavení, podle tohoto počtu aplikace prochází dál nastavení a hledá konfiguraci jednotlivých databází s názvem `databaseX`, kde `X` nabývá hodnot 1 až po počet databází
- `database1.driver = mysqli` – ovladač databázového systému databázové vrstvy Dibi
- `database1.host = localhost` – jmenná adresa nebo IP serveru s databází
- `database1.charset = utf8` – znaková sada pro komunikaci
- `database1.lazy = true` – zda se má spojení vytvořit okamžitě při volání funkce pro spojení s databází (hodnota `false`) nebo takzvané líné spojení, kdy se má spojení vytvořit až při prvním dotazu pro databázi
- `database1.database = database-name` – jméno databáze na serveru
- `database1.username`, `database1.password` – přihlašovací údaje
- `database1.name = ,Databáze organizace‘` – název pro použití v aplikaci pro označení databáze, nemá žádnou souvislost s připojením k databázi

Takto můžeme velmi jednoduše nastavit aplikaci několik databází, ke kterým se může připojit, tudíž jedna aplikace může sloužit více organizacím, kdy každá bude mít vlastní databázi. Navíc není nutně vyžadován databázový server MySQL protože databázová vrstva využitá v aplikaci má možnost se připojit pomocí svých ovladačů i k jiným typům databází. Jedinou nevýhodou v tuto chvíli je, že aplikace v této verzi nepodporuje vytváření záloh pro jiný typ připojení přes PHP než je `mysqli` a to z důvodů odlišného vytváření zálohy pro jednotlivé databázové systémy. Požadavkem na tuto aplikaci bylo právě využití

MySQL, funkcionalita na jiných databázích je tedy navíc a chybějící zálohování je tedy jedna z možností jak aplikaci dál v budoucnu rozšířit.

Podporované databázové systémy knihovnou Dibi:

- MySQL v obou variantách pro PHP tedy klasické a objektové MySQLi,
- PostgreSQL,
- SQLite,
- Firebird,
- ODBC,
- PDO,
- Oracle,
- experimentálně MS SQL včetně odlišného ovladače pro verzi 2005.

Nastavení aplikace nevyžaduje žádná speciální nastavení kromě přístupu k databázi. Pokud webové úložiště aplikace bude umožňovat přístup do složek, které nebudou viditelné přes webový server, přesunout je a upravit cestu k nim v konfiguračním souboru. Obvykle kvalitní webový hosting nabízí prostor, který obsahuje zvolenou složku například s názvem www, která je přístupná právě přes webový server pod sjednanou adresou. Mimo tuto složku si tedy můžeme přesunout složky s aplikací, knihovnami a adresářem pro dočasné soubory, což vede ke zvýšení bezpečnosti uložení tajných dat o aplikaci. V případě změny struktury adresářů tedy stačí upravit soubor v kořenovém adresáři webu a nastavit v něm příslušné cesty. Některá webová úložiště nenabízí tuto možnost a proto je aplikace v základu koncipována tak, že jednotlivé tajné složky jsou chráněny nastavením vlastností přístupu přes webový server, což je provedeno skrze soubory v nich, které si server čte při vstupu do nich.

6.4 Údržba systému

Systém z technického hlediska nevyžaduje speciální požadavky na jeho udržování v chodu. Pokud hardware i software, na kterém aplikace běží nezpůsobí nějakou chybu v systému, což může způsobit i aplikace samotným při některém neočekávaném a neošetřeném stavu, tak by nemělo být zapotřebí se o aplikaci jakkoliv starat. V případě neošetřené události

nebo výjimky si aplikace vytváří zápis do logovacího souboru s podrobným popisem stavu aplikace a zdrojem události. Dalším z problémů, které mohou vzniknout, jsou problémy s databází. Může dojít například k jejímu naplnění, i když v tuto chvíli je její struktura koncipována tak, aby při jejím očekávaném běžném využití nedošlo k jejímu naplnění. Vzhledem k malému množství záznamů z pohledu databázového systému by aplikace mohla sloužit i několik desítek let, takže by měla bez problémů sloužit po dobu používání tohoto informačního systému. Pokud by ovšem tato nepředpokládaná situace nastala, bylo by potřeba změnit některé vlastnosti schématu databáze, tak aby umožňovala vložení více záznamů. Případně může dojít k poklesu výkonu databázového systému, potom by tedy bylo nutné buď pozměnit schéma databáze, nebo zvážit změnu systému, problémů může být několik a to nejen na straně databáze a jejího systému. Obdobný problém se týká přílišného vytížení aplikačního serveru, kdy by se měla monitorovat doba vykonávání skriptů a pokud výkon klesne pod přijatelnou úroveň, tak by měla být provedena příslušná opatření.

6.5 Zálohování

Informační systém využívá vícero datových úložišť. Servery by měli být automaticky zálohovány, ale pokud se nechceme spoléhat na tuto formu, můžeme manuálně zálohovat také. Jedno úložiště je na webovém serveru pro ukládání příloh, je společné pro všechny nastavené databáze. Z informačního systému nejde jednoduše zálohovat, protože na práci s větším objemem dat není PHP navrženo, proto se nabízí možnost přihlášení se na server a ručního stažení příslušné složky. U databáze je zálohování velmi jednoduché, stačí se přihlásit do systému s editorskými právy a zvolit odkaz „Záloha databáze“ a uložit soubor. Případnou obnovu zálohy může vytvořit administrátor databáze nebo ji lze nakopírovat místo inicializační kostry prázdné databáze, která se využívá při prvotním připojení k prázdné databázi bez struktury.

7 PODPORA UŽIVATELŮ – HELPDESK

7.1 Přihlášení

Vstup do systému je ověřován přihlašovacími údaji – jménem a heslem. Přihlašovací údaje Vám sdělí odpovědná osoba, která Vám může v systému vytvořit účet. Pokud systém spravuje více databází, je zobrazena volba databáze.

V systému může uživatel vystupovat pod jednou ze dvou uživatelských rolí:

- role editora – přístup ke všem funkcím webu,
- role zadavatele – může zadávat do systému projekty a měnit je pouze pokud čekají na schválení, prohlížet pouze jím zadané projekty.

Žádná část systému není přístupná bez ověření.

Agenda výzkumných projektů

Přihlášení

Jméno	<input type="text"/>
Heslo	<input type="password"/>
Databáze	Databáze1 <input style="display: inline-block; width: 15px; height: 15px; border: 1px solid black; vertical-align: middle;" type="button" value="v"/>
<input type="button" value="Přihlásit!"/>	

Obr. 10 Přihlašovací obrazovka

Odhlášení ze systému se provádí pomocí odkazů v pravém horním rohu nebo v patičce systému, kde je také zobrazeno jméno přihlášeného uživatele.



Obr. 11 Odhlášení

7.2 Agenda výzkumných projektů

Tato sekce je z menu aplikace přístupná pod odkazy ‚Správa projektů‘ a ‚Moje Projekty‘. Jedná se o obdobný pohled na tabulku projektů s tím, že správa projektů je přístupná pouze editorovi systému a díky ní je možno plně ovládat databázi s projekty. Moje projekty nabízí pohled projektů zadaných právě přihlášeným uživatelem.

Agenda výzkumných projektů Databáze1

◦ Správa projektů ◦ Moje projekty ◦ Nový návrh projektu

Správa projektů

Sloupce k zobrazení Filtr hodnot Základní nastavení

Položek: 32

« Předchozí | 1 | 2 | Následující »

			Stav projektu	Poskytovatel finančních prostředků	Registrační číslo	Název projektu
			čeká na schválení	GA ČR		Úplně jiný název
			čeká na schválení	EU		Starý název
			čeká na schválení	MŠMT	MEB051024	Information logistics of trans
			schválen	GA ČR		Nějaký název
			schválen	GA ČR		Nějaký jiný název
			schválen	GA AV		Nový název
			schválen	GA ČR	GP 102/09/P243	Prediktivní řízení nelineárních
			schválen	GA ČR	GA 102/09/1680	Evoluční návrh řídicích algori
			schválen	GA ČR	GA 202/09/1206	Nanocrystalické heterogenní
			schválen	MŠMT	MSW7088352102	Modelování a řízení zpracova
			schválen	MŠMT	1M0567	Centrum aplikované kyberne
			schválen	MŠMT	2C06007	Inteligentní systém pro řízer
			schválen	MŠMT	CZ.1.07/2.2.00/15.0463	Modernizace výukových mat
			schválen	GA AV	QH72117	Biostimulátory a indukto
			schválen	GA AV	VG20112014067	Systém hodnocení odolnosti
			schválen	GA ČR	502092-LLP-1-2009-SK-ERASMUS-EMHE	Leonardo da Vinci
			schválen	GA ČR	Xbk	Náhodný název
			schválen	GA ČR	GP 102/09/P243	Prediktivní řízení nelineárních
			schválen	GA ČR	GA 102/09/1680	Evoluční návrh řídicích algori
			schválen	MŠMT	MSW7088352102	Modelování a řízení zpracova
			schválen	MŠMT	2C06007	Inteligentní systém pro řízer
			schválen	MŠMT	1M0567	Centrum aplikované kyberne
			schválen	MŠMT	CZ.1.07/2.4.00/12.0024	Systémová podpora spoluprá
			schválen	MŠMT	CZ.1.07/2.2.00/15.0463	Modernizace výukových mat
			schválen	MPO	FI-IM5/183	Suchá fermentace biomasy a

Exportovat do xls

◦ Administrace ◦ Uživatelé ◦ Poskytovatelé ◦ Předkladatelé ◦ Měny ◦ Záloha databáze

Databáze Databáze1 | Přihlášen editor | odhlásit

Obr. 12 Správa projektů

Agenda výzkumných projektů Databáze1

◦ Správa projektů ◦ Moje projekty ◦ Nový návrh projektu

Moje projekty

Sloupce k zobrazení Filtr hodnot Základní nastavení

Položek: 5

				Stav projektu ↓ ↑	Poskytovatel finančních prostředků ↓ ↑	Registrační číslo ↓ ↑	Název projektu ↓ ↑
				schválen	GA ČR		Nějaký název
				schválen	MŠMT	1W0567	Centrum aplikované kybernetiky
				schválen	GA ČR	GP 102/09/P243	Prediktivní řízení nelineárních sy
				schválen	MPO	FI-IM/195	Vývoj technologií a produktů mi
				schválen	MPO	TA5/041	Audiový detekční systém pro urč

Exportovat do xls

◦ Administrace ◦ Uživatelé ◦ Poskytovatelé ◦ Předkladatelé ◦ Měny ◦ Záloha databáze

Databáze Databáze1 | Přihlášen editor | odhlásit

Obr. 13 Moje projekty

Horní část obsahu tvoří tři tlačítka pro nastavení zobrazované tabulky. První dvě - ‚Sloupce k zobrazení‘ a ‚Filtr hodnot‘ slouží k nastavení vlastností tabulky, ‚Základní nastavení‘ vrací hodnoty do úvodního stavu. Těsně nad tabulkou je zobrazen celkový počet položek v celé sestavě, může se lišit s počtem zobrazeným na stránce, potom je tedy zobrazena i navigace na další stránky. Tabulka projektů se dá řadit podle jednotlivých sloupců a to buď vzestupně, nebo sestupně. Před sloupci s informacemi o projektech jsou ještě sloupce ikon, které slouží pro práci s projektem:

- zobrazení projektu,
- exportování projektu do formuláře návrhu výzkumného projektu ve formátu PDF,
- editace projektu,
- smazání projektu,
- sloupec se stavem projektu obsahuje odkaz na změnu stavu.



Obr. 14 Ikony projektu

Tyto tlačítka jsou pro roli zadavatele omezeny na první dvě, protože ke zbylým nemá oprávnění přístupu.

Tlačítko „Sloupce k zobrazení“ nabízí tabulku se seznamem sloupců, které je možné si nechat v tabulce zobrazit. Jednoduchým zaškrtnutím požadované volby a potvrzením formuláře dojde ke znovunačtení stránky s projekty a zvolenou sestavou sloupců. Pod tabulkou je tlačítko „Exportovat do xls“ po jehož stisku je zobrazovaná sestava projektů bez ohledu na stránkování nabídnuta uživateli ke stažení ve formátu tabulky excelového dokumentu.

Volba sloupců	
Zadavatel	<input type="checkbox"/>
Stav projektu	<input checked="" type="checkbox"/>
Datum vložení	<input type="checkbox"/>
Poskytovatel finančních prostředků	<input checked="" type="checkbox"/>
Název výzvy	<input type="checkbox"/>
Registrační číslo	<input checked="" type="checkbox"/>
Název projektu	<input checked="" type="checkbox"/>
Anotace	<input type="checkbox"/>
Jméno předkladatele	<input type="checkbox"/>
Délka řešení	<input type="checkbox"/>
Termín zahájení	<input type="checkbox"/>
Právě probíhá	<input type="checkbox"/>
Měna	<input type="checkbox"/>
Celkové náklady	<input type="checkbox"/>
Neinvestiční	<input type="checkbox"/>
Kapitálové	<input type="checkbox"/>
Spoluúčast celková neinvestiční	<input type="checkbox"/>
Spoluúčast celková kapitálové	<input type="checkbox"/>
Režijní celkové náklady	<input type="checkbox"/>
	Nastavit
Spoluúčast neinvestiční 1. rok	<input type="checkbox"/>
Spoluúčast neinvestiční 2. rok	<input type="checkbox"/>
Spoluúčast neinvestiční 3. rok	<input type="checkbox"/>
Spoluúčast neinvestiční 4. rok	<input type="checkbox"/>
Spoluúčast neinvestiční 5. rok	<input type="checkbox"/>

Obr. 15 Sloupce k zobrazení

Tlačítko „Filtr hodnot“ zobrazuje nabídku pro možnost zvolení konkrétních hodnot konkrétního sloupce. Pro zvolené hodnoty se dá navíc zvolit podmínka v jakém vztahu je daná hodnota s hodnotami v tabulce, tedy nemusí jít o rovnost ale třeba o nerovnost či vyloučení. Podle typu hodnot pro daný sloupec je nabízena konkrétní sada podmínek:

- výběr z číselníku – možnost omezení zda ve výstupní tabulce budou jen záznamy tuto hodnotu obsahující nebo právě neobsahující,
- datum nebo číselná hodnota – stejně jako u předchozího, doplněno o nerovnost tedy zda je hodnota větší nebo menší případně ostrá nerovnost zda je větší nebo rovna nebo zda je menší nebo rovna,
- textové hodnoty – mohou být testovány na rovnost se zadaným textem nebo zda textem začínají, končí, nebo jej obsahují.

Filtry lze jakkoliv nastavovat a poté zpětně jednotlivě mazat, mohou se různě krýt – vytvářet více omezení pro hodnoty jednoho sloupce.

Filtr hodnot ✕

Zadavatel		editor
Stav projektu		čeká na schválení
Datum vložení		
Poskytovatel finančních prostředků		EU
Název výzvy		
Registrační číslo		
Název projektu		
Anotace		
Jméno předkladatele		Franta Liška
Délka řešení		
Termín zahájení		
Právě probíhá		ano
Měna		1 €
Celkové náklady		
Neinvestiční		
Kapitálové		
Spoluúčast celková neinvestiční		
Spoluúčast celková kapitálové		
Režijní celkové náklady		
		Nastavit
Spoluúčast neinvestiční 1. rok		
Spoluúčast neinvestiční 2. rok		
Spoluúčast neinvestiční 3. rok		
Spoluúčast neinvestiční 4. rok		
Spoluúčast neinvestiční 5. rok		

Obr. 16 Filtr hodnot

Pokud bude aktivován aspoň jeden filtr, tak se nad tabulkou s projekty zobrazí seznam filtrů odkud je možno jednotlivé filtry smazat nebo je smazat všechny zároveň.

Aktivní filtry

Stav projektu je 'schválen'	
Poskytovatel finančních prostředků je 'GA ČR'	
Měna není '1 €'	
celkem: 3	

Obr. 17 Aktivní filtry

7.3 Projekt

Informace o projektu si můžeme zobrazit ve třech různých podobách:

- zobrazení všech dostupných informací o projektu,
- editace údajů projektu,
- exportování do formuláře návrhu výzkumného projektu ve formátu PDF.

7.3.1 Zobrazení projektu

Na této stránce jsou zobrazeny všechny dostupné informace o projektu. Jsou rozděleny na informativní část a údaje o projektu. Informace obsahují stav projektu, uživatele, který do systému zadal projekt, a datum vložení do systému. Údaje o projektu nabízí kompletní pohled na data o projektu ve formě tabulky. Nad údaji jsou tlačítka pro práci s projektem (změna stavu, export do PDF, editace a smazání). Pod těmito informacemi je možno k projektu přiložit přílohy pro možnost uchování dat o projektu na jediném místě, zde budou také zobrazovány.

Projekt

Změnit stav projektu
 Změnit zadavatele
 Exportovat do PDF
 Editovat projekt
 Smazat projekt

Informace

Stav projektu	schválen
Zadavatel	editor4
Datum vložení	4.3.2011

Údaje

Poskytovatel finančních prostředků	MŠMT		
Název výzvy	Y		
Registrační číslo	MSM7088352102		
Název projektu	Modelování a řízení zpracovatelských procesů přírodních a syntetických polymerů		
Anotace	Anotace		
Jméno předkladatele	Štěpán Krátký		
Délka řešení	6		
Termín zahájení	4.3.2011		
Měna	1000 Kč		
Celkové náklady	100		
- z toho náklady	Neinvestiční		Kapitalové
	60		40
- spoluúčast	1. rok	0	0
Plánované režijní náklady	1. rok	0	

Přílohy

priloha.doc		
Nová příloha	<input type="text"/>	<input type="button" value="Procházet..."/> <input type="button" value="Odeslat"/>

Obr. 18 Projekt

7.3.3 Editace projektu

Formulář návrhu výzkumného projektu přístupný pro editaci. Je možné měnit údaje formuláře v plném rozsahu. Hodnoty formuláře jsou kontrolovány, tak aby nemohli být vloženy nepřipustné hodnoty pro daný údaj.

Editace projektu

Poskytovatel finančních prostředků	MŠMT		
Název výzvy	Y		
Registrační číslo	MSM7088352102		
Název projektu	Modelování a řízení zpracovatelských procesů př		
Anotace	Anotace		
Předkladatel	Štěpán Krátký		
Délka řešení projektu	6		
Termín plánovaného zahájení	4.3.2011		
Měna a násobek	1000 Kč		
Celkové plánované náklady	100		
- z toho náklady	Neinvestiční	Kapitalové	
	60	40	
- spoluúčast	1. rok	0	0
Plánované režijní náklady	1. rok	0	
<input type="button" value="Uložit"/>			

Obr. 20 Editace projektu

7.3.4 Stav projektu

Možnost změnit stav projekt v jednoduchém formuláři.

Aktuální stav	schválen
Nový stav projektu	schválen ▼
Změnit	

Obr. 21 Změna stavu projektu

7.3.5 Zadavatel projektu

Možnost přiřadit projekt jinému zadavateli než jej do systému vložil.

Aktuální zadavatel	editor4
Nový zadavatel	editor4 ▼
Změnit	

Obr. 22 Změna zadavatele projektu

7.4 Administrace

Tato sekce je přístupná pouze uživateli s právy editora. Nabídka s odkazy do jednotlivých sekcí je pod obsahem stránky, tvoří část patičky. Administrace nabízí správu číselníků, nastavení uživatelů a zálohování databáze.

7.4.1 Uživatelé

Stránka je určená k přidání, smazání a správě účtů uživatelů – zadavatelů. Obsahuje výpis všech uživatelů v systému se počtem projektů, které do systému vložili, nastavením role a možností zakázat přihlášení do systému.

[+ Přidat nového uživatele](#)

Uživatelské účty

	jméno	projekty	práva	přihlášení	
	editor	5	 editor		
	editor1	1	 editor		
	editor2	2	 editor		
	editor3	1	 editor		
	editor4	1	 editor		
	editor5	1	 editor		
	zadavatel	2	 zadavatel		
	zadavatel1	3	 zadavatel		
	zadavatel10	1	 zadavatel		
	zadavatel11	1	 zadavatel		
	zadavatel12	0	 zadavatel		
	zadavatel13	0	 zadavatel		
	zadavatel2	3	 zadavatel		
	zadavatel3	1	 zadavatel		
	zadavatel4	1	 zadavatel		
	zadavatel5	3	 zadavatel		
	zadavatel6	2	 zadavatel		
	zadavatel7	2	 zadavatel		
	zadavatel8	0	 zadavatel		
	zadavatel9	2	 zadavatel		

Obr. 23 Uživatelské účty

Stiskem tlačítka ‚Přidat nového uživatele‘ nebo editací existujícího uživatele se navíc zobrazí formulář pro změnu údajů a vlastností.

Nový uživatel

Jméno	<input type="text"/>
Heslo	<input type="password"/>
Heslo znovu	<input type="password"/>
Práva	<input type="radio"/> zadavatel <input type="radio"/> editor
	<input type="checkbox"/> zakázat přihlášení
	Vytvořit uživatele

Obr. 24 Přidání nového uživatele

7.4.2 Poskytovatelé finančních prostředků

Číselník obsahující výpis všech poskytovatelů finančních prostředků v systému včetně počtu projektů, které na daný údaj číselníku odkazují.

[+ Přidat nového poskytovatele](#)

Poskytovatelé finančních prostředků

		jmeno	projekty
		EU	2
		GA AV	3
		GA ČR	11
		MPO	4
		MŠMT	12
		TAČR	0

Obr. 25 Poskytovatelé finančních prostředků

Stiskem tlačítka ‚Přidat nového poskytovatele‘ nebo editací existujícího poskytovatele se navíc zobrazí formulář pro tento úkon.

Nový poskytovatel

Jméno:	<input type="text"/>
	Vytvořit poskytovatele

Obr. 26 Nový poskytovatel

7.4.3 Předkladatelé

Číselník obsahující výpis všech předkladatelů výzkumných projektů v systému včetně počtu projektů, které na daný údaj číselníku odkazují.

+ Přidat nového předkladatele

Předkladatelé

		jméno	projekty
		Franta Liška	7
		Ivan Pečený	1
		Jan Novák	2
		Jan Štědrý	1
		Josef Liška	6
		Martin Dlouhý	2
		Petr Novák	3
		Petra Nováková	3
		Přemysl Oráč	4
		Radomír Bohatý	1
		Štěpán Krátký	2

Obr. 27 Předkladatelé

Stiskem tlačítka „Přidat nového předkladatele“ nebo editací existujícího předkladatele se navíc zobrazí formulář pro tento úkon.

Nový předkladatel

Jméno:

Vytvořit předkladatele





Obr. 28 Nový předkladatel

7.4.4 Měny

Číselník obsahující výpis všech měn v systému včetně počtu projektů, které na daný údaj číselníku odkazují. Kromě názvu měny se zadává koeficient násobení, tak aby se v projektu nemuseli uvádět hodnoty nákladů vůči základní jednotce dané měny, ale jejímu násobku.

+ Přidat novou měnu

Měny

		jmeno	projekty
		1000 Kč	30
		1 €	2

Obr. 29 Měny

Stiskem tlačítka ‚Přidat novou měnu‘ nebo editací existující měny se navíc zobrazí formulář pro tento úkon.

Nová měna

Měna:	
Koeficient:	
	Vytvořit měnu

Obr. 30 Nová měna

7.4.5 Záloha databáze

Tento odkaz nezobrazuje stránku, ale vygeneruje soubor se zálohou databáze a nabídne jej uživateli k uložení do jeho počítače.

ZÁVĚR

Cílem diplomové práce bylo vytvořit databázový systém pro správu výzkumných projektů od úvodní analýzy problematiky, návrh modelu systému až po vytvoření funkční aplikace včetně uživatelské podpory v podobě příručky.

K jednotlivým částem práce jsem se snažil přistupovat odborně, tak aby byl patrný postupný vývoj od teoretického základu po realizaci pomocí konkrétních technologií. Na počátku práce byl požadavek na vytvoření databázového systému. Zjistil jsem aktuální situaci, kdy neexistuje žádný používaný informační systém usnadňující práci, a nastínil jsem způsob možného řešení. Pro aplikaci byla zvolena třívrstvá architektura klient – server pro možnost vzdáleného přístupu s centrálním úložištěm dat v podobě databázového systému. Pro potřeby systému, tedy vhodně zobrazovat data a zadávat do systému informace podobné tištěným formulářům, byla zvolena webová aplikace, která má minimální nároky na klientské vybavení a nabízí dobrou podporu práce s databázovým systémem.

Návrh databáze byl proveden pomocí metodiky mající tři úrovně, kdy úvodní pohled na informace pro systém byl od uživatelů a to jak v podobě konzultací, tak tištěných formulářů a tabulek, které by měl systém obsahovat. Po fázi sběru informací došlo k vytvoření konceptu databáze. Byl vytvořen návrh datových struktur a jeho vazby, následně interní vrstva definovala samotný systém řízení báze dat v podobě serverové aplikace MySQL. Společně s návrhem databáze jsem také popsal princip relačních databází opět s pohledem od modelu dat až po tvorbu databáze včetně užívaného názvosloví v tomto oboru.

Kromě databázového systému MySQL jsem uvedl i další požadavky na systém v podobě skriptovacích jazyků PHP a JavaScript, ten ovšem není striktně vyžadován. Aplikační logika systému je celá napsaná v jazyce PHP, který obstarává na webovém serveru požadavky od klientského prohlížeče, komunikuje s databázovým serverem a převádí data do formy vhodné pro uživatele. JavaScript pracuje na straně klienta a slouží ke zpříjemnění pracovního prostředí informačního systému. Pro tvorbu aplikace je využita knihovna funkcí Nette Framework, která zjednodušuje některé časté operace a podporuje architekturu tvorby aplikace Model–View–Presenter. Nette Framework má také úzkou vazbu na databázovou knihovnu Dibi, přes kterou jsou využívány služby databáze.

Praktická část popisuje výsledné schéma databáze, jednotlivé tabulky a jejich provázanost. Aplikační část je popsána rozdělením podle MVP architektury a využití Dibi vrstvy. Struktura popisu kódu aplikace odpovídá také členění a logice struktury samotné aplikace, tedy aplikace je tvořena funkčně podobnými celky, které odpovídají celkům vytvořeného informačního systému. Při tvorbě systému jsem také dbal na bezpečnost přístupu k datům, aplikace obsahuje řízení přístupu k datům na základě uživatelských rolí. Vytvořený kód aplikace je okomentován a vytvořen pomocí objektů, jeho struktura je jednouchá na pochopení. Informační systém splňuje požadavky na jeho funkce a navíc nabízí rozšířenou funkcionalitu v podobě podpory více databázových systémů. Další z kapitol praktické části obsahuje návod jak systém nainstalovat a nastavit, tedy použití z pohledu správce aplikace. Poslední kapitola obsahuje příručku informačního systému pro správu výzkumných projektů z pohledu uživatele.

ZÁVĚR V ANGLIČTINĚ

The aim of this thesis was to create a database system for managing research projects from initial problem analysis, design of model systems to create functional applications, including user support in the form of a manual.

For each part I have tried to approach scientifically, so as to reveal the gradual development of a theoretical basis for the realization of using specific technologies. The early work was a requirement for a database system. I found the current situation where there is no information system used to facilitate the work and the way I outlined a possible solution. To apply the three-layer architecture was chosen client - server for remote access to central data storage in the form of database system. For the needs of the system, thus properly display data and enter information into the system similar to printed forms, has been elected a web application that has minimal client software and offers good support to work with the database system.

Database design methodology was carried out by having three levels, the initial view of information for users of the system both in terms of consultation, and printed forms and tables that the system should contain. After the stage of collecting information was to create a database concept. It was created by design of data structures and its links, then the internal layer to define the actual database management system in the form of the MySQL server applications. Together with the database design, I also described the principle of relational databases, looking back from the data model to the creation of databases, including terminology used in this field.

In addition to the MySQL database system, I observed other system requirements in terms of scripting languages PHP and JavaScript, that is not strictly required. Application logic is a system written in PHP, which caters to the web server requests the client browser communicates with a database server and converts the data into a form suitable for users. Javascript works on the client side and serve to make your work environment information system. To build an application is used by library functions Nette Framework, which simplifies some common operations and supports the creation of application architecture Model-View-Presenter. Nette Framework is also closely linked to the database library Dibi, through which services are used by the database.

The practical part describes the resulting database schema, the tables and their interdependence. Application of division is described according to MVP architecture and use Dibi layer. The structure of the code corresponds to the application structure and logic structure of the application itself, so the application is made up of functionally similar units, which correspond to units set up an information system. In developing the system, I also look after the safety of access to data, applications include access control data based on user roles. Created application code is commented and created by using objects, its structure is simple to understand. Information system meets its functionality and also offers extended functionality in the form of support for multiple database systems. Another of the chapters contains practical guidance on how to install and configure the system, then use the view controller application. The last chapter contains a manual system to manage research projects from a user's perspective.

SEZNAM POUŽITÉ LITERATURY

Monografie:

- [1] CASTAGNETTO, Jesus. PHP : Programujeme profesionálně. Přeložil Luděk Roubíček. 2., opr. a aktualiz. vyd. Praha : Computer Press, 2002. 656 s. ISBN 80-251-0046-4.
- [2] CONOLLY, Thomas. Mistrovství - databáze : profesionální průvodce tvorbou efektivních databází. Přeložil Vilém Gutfreund. Vyd. 1. Brno : Computer Press, 2009. 584 s. ISBN 978-80-251-2328-7.
- [3] RESIG, John. JavaScript a Ajax : Moderní programování webových aplikací. Přeložil Ondřej Baše. Vyd. 1. Brno : Computer Press, 2007. 360 s. ISBN 978-80-251-1824-5.

Internetové zdroje:

- [4] PHP [online]. 2011 [cit. 2011-01-31]. Manual. Dostupné z WWW: [www.php.net/manual/en/].
- [5] MySQL 5.1 [online]. 2005 [cit. 2011-01-31]. Reference Manual. Dostupné z WWW: [dev.mysql.com/doc/refman/5.1/en/].
- [6] Nette Framework [online]. 2008 [cit. 2011-01-31]. Dokumentace. Dostupné z WWW: [nette.org/cs/dokumentace].
- [7] Dibi database layer [online]. 2008 [cit. 2011-05-01]. Dokumentace. Dostupné z WWW: [dibiphp.com].

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

AJAX	Asynchronous JavaScript and XML
ASP	Active Server Pages, programovací jazyk
HTML	HyperText Markup Language
MVC	Model-View-Controller, softwarová architektura
MVP	Model-View-Presenter, softwarová architektura
PDF	Portable Document Format
PHP	Hypertext Preprocessor, programovací jazyk
SQL	Structured Query Language
UI	User Interface
XHTML	Extensible HyperText Markup Language
XLS	Microsoft Excel file format
XML	Extensible Markup Language