VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

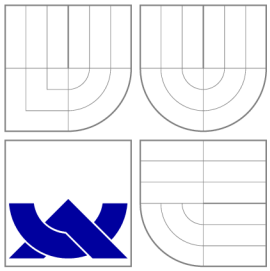
DATABÁZOVÝ SYSTÉM PRO SPRÁVU BIOLOGICKÝCH DAT

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

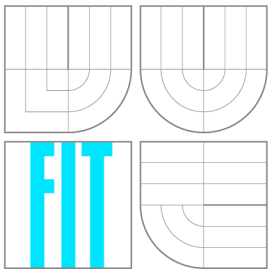
AUTOR PRÁCE
AUTHOR

Bc. RADOVAN DRLÍK

BRNO 2010



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

DATABÁZOVÝ SYSTÉM PRO SPRÁVU BIOLOGICKÝCH DAT

DATABASE SYSTEM FOR BIOLOGICAL DATA MANAGEMENT

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. RADOVAN DRLÍK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. PETR JAŠA

BRNO 2010

Zadání diplomové práce/10849/2009/xdrlik03

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2009/2010

Zadání diplomové práce

Řešitel: **Drlík Radovan, Bc.**
Obor: Informační systémy
Téma: **Databázový systém pro správu biologických dat**
Database System for Biological Data Management
Kategorie: Databáze

Pokyny:

1. Seznamte se s požadavky skupiny proteinových inženýrů Loschmidtových laboratoří na sestavení databázového systému pro správu biologických dat, převážně dat týkajících se enzymů Haloalkane Dehalogenases.
2. Seznamte se se současným stavem tohoto projektu označovaného jako HADES (Haloalkane Dehalogenases Database) a s jeho naimplementovanými částmi.
3. Na základě prvních dvou bodů rozhodněte, zda je vhodné využít stávající rozpracovanou verzi databáze, provést její úpravy a rozšířit ji nebo vytvořit kompletně nový návrh. Dále navrhněte jaké technologie budou pro tento projekt nejvhodnější.
4. Po konzultaci s vedoucím navrhněte pomocí vhodných nástrojů strukturu databáze a aplikaci, která bude klást důraz na přehledné a srozumitelné zobrazování dat a jednoduché vkládání. U vkládání dat se také zaměřte na vytvoření rozhraní pro snadný import dat z programu Microsoft Excel.
5. Implementujte navrženou databázi a aplikaci.
6. Nasaďte aplikaci do provozu. Zhodnoťte, jak vyhovuje proteinovým inženýrům. Navrhněte další úpravy pro zlepšení aplikace.

Literatura:

- Kofler, M.: Mistrovství v MySQL 5 - Kompletní průvodce webového vývojáře, Computer Press, 2007, ISBN: 978-80-251-1502-2
- Schneider R. D.: MySQL - Oficiální průvodce tvorbou, správou a laděním databází, Grada, 2006, ISBN: 8024715163

Při obhajobě semestrální části diplomového projektu je požadováno:

- Body 1 až 3.

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci ročníkového a semestrálního projektu (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Jaša Petr, Ing.**, UIFS FIT VUT
Datum zadání: 21. září 2009
Datum odevzdání: 26. května 2010

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2

doc. Dr. Ing. Dušan Kolář
vedoucí ústavu

Abstrakt

Práce se zabývá problematikou uchovávání a správy biologických dat, především enzymů halogenalkan dehalogenáz. Dále se zaměřuje na projekt HADES (HAloalkane DEhalogenase databaSe) iniciovaný proteinovými inženýry Loschmidtových laboratoří Masarykovy univerzity v Brně. Jedná se o projekt, jehož hlavním cílem je jednoduše ukládat, uchovávat a zobrazovat různorodá data o proteinech. Výsledkem práce je flexibilní databázový systém umožňující snadnou rozšiřitelnost a udržovatelnost, který je postavený na technologiích Apache, PostgreSQL a PHP s využitím Zend Frameworku.

Abstract

This thesis describes the problems of storage and management of biological data, particularly of Haloalkane Dehalogenase enzymes. Furthermore, the thesis aims at project HADES (HAloalkane DEhalogenase databaSe) initiated by protein engineering group of Loschmidt Laboratories, Masaryk University in Brno. This is a project whose main goal is simply to store, preserve and display a wide variety of proteins data. The result of this work is a flexible database system allowing easy extensibility and maintainability, which is built on technologies Apache, PostgreSQL and PHP using the Zend Framework.

Klíčová slova

bioinformatika, molekulární biologie, proteiny, enzymy, halogenalkan dehalogenázy, Zend Framework, MySQL, PostgreSQL, Office Open XML, Entity–Attribute–Value model

Keywords

bioinformatics, molecular biology, proteins, enzymes, Haloalkane Dehalogenases, Zend Framework, MySQL, PostgreSQL, Office Open XML, Entity–Attribute–Value model

Citace

Radovan Drlík: Databázový systém pro správu biologických dat, diplomová práce, Brno, FIT VUT v Brně, 2010

Databázový systém pro správu biologických dat

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Petra Jaši. Další informace mi poskytli členové B–týmu Loschmidtových laboratoří Masarykovy univerzity v Brně, jmenovitě Mgr. Eva Chovancová a Mgr. Antonín Pavelka. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Radovan Drlík
26. května 2010

Poděkování

Na tomto místě bych rád poděkoval vedoucímu práce Ing. Petru Jašovi za jeho odborné vedení, poskytnuté konzultace a podnětné náměty ve všech fázích práce. Dále bych chtěl poděkovat členům B–týmu Loschmidtových laboratoří, především Mgr. Antonínovi Pavelkovi a Mgr. Evě Chovancové, za poskytnuté konzultace a užitečné návrhy a připomínky.

Obsah

Seznam obrázků

Kapitola 1

Úvod

Oblast molekulární biologie a s ní související informatická vědní disciplína bioinformatika prochází fází dynamického rozvoje a nabízí tak široké možnosti bádání, získávání nových poznatků a vytváření potřebných nástrojů sloužících k uchovávání, analýze a prezentaci biologických dat. Se zkvalitňováním technické vybavenosti výzkumných laboratoří dochází k neustálé produkci stále většího množství dat, jež se musí dále zpracovávat nebo uchovávat pro pozdější účely. Je třeba vzít v úvahu, že získaná data nemusí být neměnná a mohou být v průběhu času zpřesňována, případně doplňována nově zjištěnými informacemi. Proto musí vznikající informační systémy a jejich datová úložiště poskytovat dostatečnou flexibilitu a prostor k dalšímu rozšiřování.

Tato práce se zabývá analýzou, návrhem a implementací databázového systému pro správu biologických dat, převážně dat týkajících se enzymů Haloalkane Dehalogenases. Práce vznikla ve spolupráci s Loschmidtovými laboratořemi Masarykovy univerzity v Brně pod názvem HADES (Haloalkane Dehalogenases Database). Skupina proteinového inženýrství Masarykovy univerzity v Brně studuje základní principy enzymové katalýzy a vyvíjí zlepšené enzymy pro environmentální, chemické a biomedicínské využití [23].

V průběhu specifikace požadavků se ukázalo, že systém nebude využíván pouze proteinovými inženýry Loschmidtových laboratoří, ale že cílem je vytvořit systém, který by mohly využít také jiné laboratoře. Z tohoto důvodu je požadavek flexibility, jednoduché rozšiřitelnosti a snadné udržovatelnosti systému mimořádně důležitý.

Cílem práce je navrhnout a vytvořit systém pro ukládání a prezentaci dat enzymů, jejichž výzkumem se zabývají skupiny proteinových inženýrů různých laboratoří. Unikátnost systému tkví v jeho snadné rozšiřitelnosti a v možnosti sloučení teoretických i experimentálních dat uložených proteinů. Výsledkem je aplikace umožňující proteinovým inženýrům jednoduché vkládání a editaci dat a následně vytvoření jednoduchého portálu pro veřejnost, umožňujícího prohledávání databáze.

1.1 Členění práce

Práce je rozdělena do devíti kapitol. V kapitole 2 jsou vysvětleny základní pojmy z oblasti molekulární biologie, které uvedou čtenáře do problematiky a usnadní pochopení dat poskytovaných proteinovými inženýry. Na tuto kapitolu navazuje kapitola 3 věnující se bioinformatice, oboru využívajícímu informačních technologií v oblasti biologie. Představeny jsou některé veřejně dostupné bioinformatické databázové systémy a základní formáty ukládání biologických dat.

Praktická část práce se zabývá specifikací požadavků kladených proteinovými inženýry na vznikající systém. Tyto požadavky jsou popsány v kapitole 4. Dále je v kapitole 5 provedena analýza dostupných řešení. Jedná se o diplomovou práci Ivy Urbanové: *Sbírka dat z oblasti proteinové a enzymové biologie* [26], u které je zkoumán návrh databáze, funkcionalita aplikace i její zdrojový kód. Na základě analýzy je navržen další postup řešení. Následně jsou popsány možnosti využití XML k uchovávání biologických dat a *Entity–Attribute–Value* model pro ukládání dat. Ten je analyzován podrobněji a jsou pro něj provedeny také testy výkonnosti.

Po úvodní fázi analýzy je přistoupeno k návrhu vlastního systému, kterému se věnuje kapitola 6. Protože jedním ze základních požadavků na systém je jeho flexibilita a jednoduchá rozšiřitelnost, je navržen univerzální systém pro modelování dat, který je reprezentován systémem entit a atributů.

V následující kapitole 7 jsou představeny vybrané technologie, které jsou využity při realizaci aplikace. Jedná se především o databázový systém *PostgreSQL* a *MySQL*, PHP framework *Zend Framework* a formát pro ukládání dokumentů *Office Open XML*.

Kapitola 8 uvádí popis implementace aplikace a fáze jejího nasazení do provozu a testování.

Závěr pak shrnuje průběh a dosažené výsledky diplomové práce a navrhuje možnosti dalšího pokračování v projektu.

Práce navazuje na výsledky semestrálního projektu, který se zabýval prvními třemi body zadání. Jeho výsledky odpovídají přibližně kapitolám 2 a 3 věnujícím se teoretické oblasti molekulární biologie a bioinformatiky, kapitole 4 popisující požadavky proteinových inženýrů kladené na systém a kapitolám 5 a 7, ve kterých byla analyzována dostupná řešení ukládání biologických dat a technologie, jež jsou pro tento účel nejvhodnější.

Kapitola 2

Molekulární biologie

Cílem kapitoly je uvést čtenáře do problematiky molekulární biologie a vysvětlit především pojmy protein a enzym, kterých se práce týká. Protože molekulární biologie úzce souvisí také s chemií, je třeba definovat některé pojmy z této oblasti.

Účelem není podat kompletní přehled oboru, ale stručně popsat vybrané pasáže, které umožní pochopení významu poskytovaných dat a jejich vztahů. Čtenář znalý problematiky nebo zajímající se pouze o informatickou část práce může tuto kapitolu přeskočit.

Molekulární biologie je podle B. Albertse a kolektivu [8] vědní disciplína, která se zabývá studiem biologických procesů na molekulární úrovni. Ve svém přístupu slučuje biologická, chemická, fyzikální a genetická hlediska.

2.1 Pojmy z oblasti chemie

Podkapitola stručně vysvětluje pojmy z oblasti chemie, které jsou dále použity v textu a jejichž význam by nemusel být zcela jasný. Více můžete nalézt v odborné literatuře, například v [27], ze které definice jednotlivých pojmů vycházejí.

Hydrolýza je chemická rozkladná reakce, při které dochází k rozkladu látky za působení molekuly vody.

Katalyzátor je látka, která vstupuje do reakce, urychluje ji a vystupuje z ní v nezměněné podobě.

Aby mohla reakce proběhnout, je třeba, aby molekuly měly určitou energii — takzvanou *aktivační energii*[1]. Pokud je aktivační energie příliš vysoká, může reagovat málo molekul a reakce tak probíhá pomalu [8]. Katalyzátor snižuje aktivační energii a umožní tím rychlejší průběh reakce.

Substrát představuje látku vstupující do reakce (tzv. reaktant).

Aminokyselina v chemii představuje molekulu obsahující *karboxylovou* (—COOH) a *aminovou* (—NH_2) skupinu [8]. V molekulární biologii aminokyselinami chápeme pouze ty aminokyseliny, které jsou tzv. *proteinogenní* — tedy ty aminokyseliny, které představují základní stavební složky proteinů. Takovýchto proteinogenních aminokyselin je 21[2].

[1] aktivační energie: minimální energie, která je potřebná k převedení látky do stavu schopného reakce http://www.cojeco.cz/index.php?detail=1&id_desc=1689&s_lang=2

[2] V literatuře se někdy uvádí počet 20, 21. aminokyselinou je selenocystein, který v některých enzymech nahrazuje cystein.

Registrační číslo CAS je jednoznačný numerický identifikátor, používaný v chemii pro chemické látky, polymery, biologické sekvence, směsi a slitiny.

Číslo CAS je tvořeno třemi částmi oddělenými pomlčkami. První část se skládá z proměnného počtu číslic, kterých může být až sedm, druhá ze dvou číslic a třetí z jediné číslice sloužící jako kontrolní součet správnosti registračního čísla.

2.2 Proteiny

Proteiny (také označované jako bílkoviny) jsou dle B. Albertse a kolektivu [8] organické sloučeniny tvořené aminokyselinami uspořádanými do lineárního řetězce a složené do kulovitého tvaru. Každý typ proteinu má jedinečné pořadí aminokyselin, které je u všech molekul tohoto proteinu stejné. Tvar proteinu je dán pořadím jednotlivých aminokyselin, kterými je tvořen.

Proteiny plní celou řadu funkcí, na jejichž základě je lze rozdělit do kategorií: strukturní, katalyzátory reakcí (enzymy), transportní, pohybové, zásobní, signální, receptorové, regulační v genové expresi a další specializované proteiny [8].

Každý protein je složen do vlastní trojrozměrné struktury, která je dána pořadím aminokyselin v jeho řetězci. Prostorové uspořádání je vytvořeno tak, že obsah volné energie proteinu je minimální. Pokud protein reaguje s jinými molekulami, může dojít ke změně uspořádání, což může vést také ke změně funkce proteinu [8].

Primární struktura proteinu představuje sekvenci aminokyselin, kterými je protein tvořen. Určuje chemické vlastnosti proteinu a také předurčuje strukturu sekundární.

Protože lineární řetězec představující sekvenci aminokyselin je možné číst ze dvou stran, a tedy je možné získat dvě odlišné sekvence, bylo třeba určit pravidla pro způsob získání tohoto řetězce. Jde o konvenci čtení řetězce od tzv. *N–konce* k *C–konci*[3] [16].

Sekundární struktura proteinu řeší prostorové uspořádání řetězce aminokyselin. V případě sekundární struktury se většinou jedná o lokální uspořádání, tedy pouze o části proteinového řetězce.

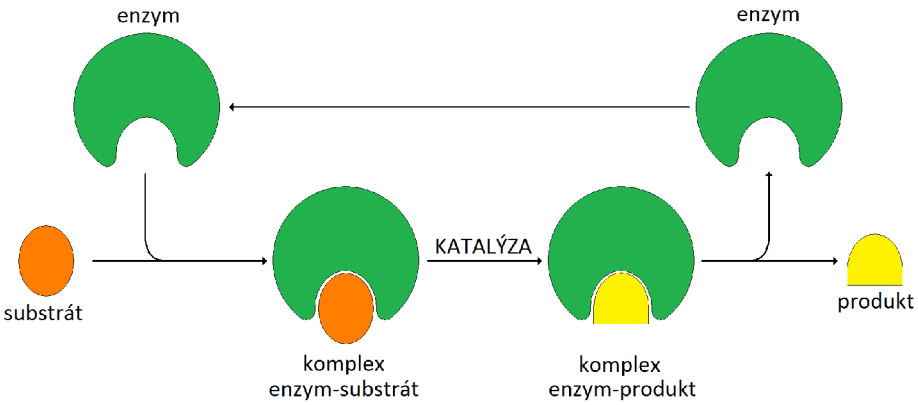
Terciální a kvartérní struktura proteinů představují uspořádání celého proteinového řetězce, případně uspořádání podjednotek v komplexech více proteinů. Díky znalosti prostorové struktury je možné určit pravděpodobnou funkci proteinu.

2.3 Enzymy

Enzymy jsou proteiny, které katalyzují chemické reakce. Patří mezi nejúčinnější známé katalyzátory a umožňují tak děje, které by za běžných teplot vůbec neprobíhaly [8]. Jak již bylo řečeno, enzymy jako katalyzátory snižují aktivační energii pro proběhnutí reakce a výrazně tak zvýší její rychlost.

Každý enzym má aktivní místo specifického tvaru (kapsovitého nebo žlábkovitého) a do tohoto místa padnou jen určité substráty. Proto každý enzym nějčastěji katalyzuje jen jednu určitou reakci. Na aktivní místo se váže molekula substrátu, čímž vzniká komplex *enzym–substrát*. Reakce probíhá v aktivním místě a vzniká komplex *enzym–produkt*. Následně je produkt uvolněn a enzym může na sebe vázat další molekuly substrátu. Průběh procesu katalýzy je znázorněn na obrázku 2.1 [8].

[3] Jedná se o chemickou podstatu tvorby řetězce aminokyselinami. Více v literatuře [16].



Obrázek 2.1: Průběh katalýzy

EC číslo

EC číslo (zkratka EC z anglického Enzyme Commission) představuje systém nomenklatury[4] (názvosloví) enzymů, jde o numerické označení enzymů založené na chemických reakcích, jež katalyzují. EC číslo tedy neurčuje enzym, ale reakci enzymu a katalyzátoru.

Každý kód enzymu se skládá z označení 'EC' a následně čtyř čísel oddělených tečkou. Jednotlivá čísla postupně zpřesňují klasifikaci enzymu. (Například EC 3.8.1.5: číslo 3 značí, že se jedná o *hydrolázy*[5], číslo 8 označuje vazbu na halogenid a celé číslo EC 3.8.1.5 pak označuje enzymy haloalkan dehalogenázy). Přehled EC čísel můžete nalézt například na webové adrese http://www.brenda-enzymes.org/ v sekci „EC Explorer“.

2.4 Halogenalkan dehalogenázy

Halogenalkan dehalogenázy — Haloalkane Dehalogenases (EC 3.8.1.5) jsou podle informací uvedených skupinou proteinových inženýrů Loschmidtových laboratoří [23] bakteriální enzymy katalyzující hydrolytické štěpení vazby mezi uhlíkem a halogenem v haloorganických sloučeninách. Při reakci z haloalkanu a vody vzniká primární alkohol a halogenidový iont.

Tyto enzymy jsou využívány například pro čištění podzemní vody kontaminované halogenovými sloučeninami, odstranění vedlejších produktů chemické syntézy nebo pro biologickou detekci kontaminace prostředí halogeny [23].

[4]Nomenklatura je definována jako systém pojmenování a klasifikace určitých objektů jakožto prvků dané kategorie. http://cs.wikipedia.org/wiki/Nomenklatura

[5]Hydroláza je enzym, který katalyzuje hydrolýzu. [27]

Kapitola 3

Bioinformatika

Kapitola seznamuje čtenáře s pojmem bioinformatika a dále nabízí přehled databázových systémů a formátů uložení biologických dat.

Bioinformatiku můžeme definovat jako výpočetní oblast molekulární biologie [16].

Podle J. M. Clavericha a kolektivu [16] je bioinformatika vědní disciplína, která využívá informačních technologií v oblasti biologie, zejména v oblasti molekulární biologie. Zabývá se metodami shromažďování, analýzy a vizualizace rozsáhlých souborů biologických dat. Zahrnuje vytváření, úpravy a vyhledávání v biologických databázích, porovnávání sekvencí, zkoumání struktur proteinů a obecně řečeno řešení biologických a biomedicínských otázek za použití počítače.

3.1 Databáze biologických dat

V současné době již existuje velké množství databází biologických dat. Většina z nich slučuje data z různých databází nebo se na ně odkazuje a vytváří nad nimi vlastní uživatelské rozhraní. V přehledu se zaměřím pouze na nejvýznamnější a nejčastěji používané databáze proteinů a v závěru také na databázi referenční literatury, která bývá součástí každého záznamu o proteinu.

3.1.1 ExPASy

ExPASy (*Expert Protein Analysis System*) je světově známý a používaný zdroj informací o proteinech [16]. Poskytuje také velké množství nástrojů pro analýzu sekvencí a odkazů na jiné servery.

Server ExPASy http://www.expasy.ch/ udržuje organizace *Swiss Institute of Bioinformatics*[1]. Podle E. Gasteigerové [17] spojuje ExPASy velké množství databází specializujících se na různé oblasti zkoumání proteinů. Jedná se zejména o databáze:

- **UniProtKB** – databáze proteinových sekvencí a anotací (bude dále popsána)
- **PROSITE** – databáze rodin proteinů, jež umožňuje určit, do které rodiny proteinů patří zadaná sekvence

[1] http://www.isb-sib.ch/

- **ENZYME** – databáze uchovávající informace o zařazení enzymů (viz EC číslo)
- **SWISS–MODEL Repository** – databáze modelů 3D struktur proteinů
- a další, podrobnosti na http://www.expasy.ch/

3.1.2 UniProt

Universal Protein Resource (zkráceně UniProt) je zdrojem informací o proteinových sekvencích a anotacích. Hlavní část tvoří databáze *UniProt Knowledgebase* (UniProtKB). Informační portál je možné najít na webové adrese http://www.uniprot.org/.

UniProtKB se skládá ze dvou sekcí, a to sekce *UniProtKB/Swiss-Prot*, která obsahuje ověřené záznamy, a dále sekce *UniProtKB/TrEMBL*, která je tvořena automaticky, bez kontroly [17]. Záznamy UniProtKB se skládají z informací o názvu a původu proteinu, z obecných informací (funkce, vlastnosti, příbuzné sekvence, atd.), odkazů na literaturu a především z vlastní sekvence.

3.1.3 Protein Data Bank

Protein Data Bank (zkratka PDB) představuje informační portál určený veřejnosti, především výzkumným pracovníkům a studentům, kteří se zajímají o molekulární biologii. Podle H. M. Bermana a kolektivu [10] představuje PDB archiv mezinárodní úložiště dat o 3D struktuře biologických makromolekul. Molekuly pocházejí z různých organismů (bakterií, rostlin, zvířat i lidí). Poskytovaná data jsou veřejně přístupná a jejich formát je dobře zdokumentovaný. Bližší popis formátu PDB je uveden dále v kapitole 3.2.3.

Informační portál, který je dostupný na webové adrese http://www.pdb.org/, umožňuje vyhledávání na základě klíčového slova, identifikátoru PDB (PDB ID), autora nebo chemického názvu.

3.1.4 MEDLINE, PubMed

MEDLINE (*Medical Literature Analysis and Retrieval System Online*) je bibliografická databáze medicínské literatury. Na základě informací uvedených U. S. National Library of Medicine [1] obsahuje databáze přes 16 milionů odkazů, především z oblasti biomedicíny.

Databáze MEDLINE je základem systému PubMed, udržovaného organizací *National Center for Biotechnology Information*[2] (NCBI) [1]. PubMed poskytuje uživatelské rozhraní pro přístup k databázi a umožňuje vyhledávání citací a abstraktů biomedicínských odborných článků.

3.2 Formáty uložení biologických dat

Pro ukládání biologických dat existuje celá řada formátů, zaměřím se však pouze na nejčastěji používané formáty pro uložení sekvencí a prostorového uspořádání proteinů.

3.2.1 RAW

RAW formát představuje nejjednodušší možný způsob uložení sekvencí. Jedná se o prosté uložení jednopísmenných kódů dané sekvence bez možnosti specifikace dalších informací. Je určen pouze pro uchování jedné sekvence (pro každou sekvenci existuje jeden soubor).

[2] http://www.ncbi.nlm.nih.gov/

3.2.2 FASTA

FASTA formát je určen pro uchovávání sekvencí DNA nebo proteinu. Sekvence mají předepsaný tvar: sekvenci uvozuje řádek začínající znakem '>', za kterým následuje jednoznačný identifikátor sekvence, případně i krátký popis sekvence. Následující řádky obsahují vlastní sekvenci proteinu (nebo DNA), která je zapsána jednopísmenným kódem proteinu (DNA).

FASTA formát umožňuje uchování více sekvencí v jednom souboru (začátek nové sekvence je rozpoznán na základě znaku '>').

Ukázka souboru ve formátu FASTA, obsahujícího dvě sekvence proteinů:

```
>3GOU:A|PDBID|CHAIN|SEQUENCE
VLSPADKTNIKSTWDKIGGHAGDYGGEALDRTFQSFPTTKTYFPHFDLSPGSAQVKAHGKKVADALTTAVAHLDDLPGAL
SALSDLHAYKLRVDPVNFKLLSHCLLVTLACHHPTEFTPAVHASLDKFFAAVSTVLTSKYR
>3EKN:A|PDBID|CHAIN|SEQUENCE
GVFPSSVYVPDEWEVSREKITLLRELGQGSFGMVYEGNARDIIKGEAETRVAVKTVNESASLRERIEFLNEASVMKGFTC
HHVVRLLGVVSKGQPTLVVMELMAHGDLKSYLRSLRPEAENNPGRPPPTLQEMIQMAAEIADGMAYLNAKKFVHRNLAAR
NCMVAHDFTVKIGDFGMTRDIYETDYYRKGGKGLLPVRWMAPESLKDGVFTTSSDMWSFGVVLWEITSLAEQPYQGLSNE
QVLKFVMDGGYLDQPDNCPERVTDLMRMCWQFNPNMRPTFLEIVNLLKDDLHPSFPEVSFFHSEENK
```

Díky jednoduchosti formátu, umožňující snadné strojové zpracování, je formát FASTA velmi oblíbený a nejčastěji používaný formát pro uchovávání sekvencí [16].

3.2.3 PDB

PDB formát je využíván především pro uchovávání informací o 3D struktuře proteinu. Soubor v PDB formátu je textový soubor, ve kterém je každý řádek tvořen přesně 80 znaky. Na začátku každého řádku je specifikován název typu záznamu. Tento název musí odpovídat některému z definovaných klíčových slov [10].

Struktura souboru v PDB formátu je již mnohem komplikovanější než tomu bylo u formátu FASTA. Uchovávány jsou informace o zařazení záznamu, datu jeho pořízení, identifikátoru záznamu, klíčových slovech vystihujících záznam, o autorovi, organismu, ze kterého byl záznam pořízen, použité metodě, sekvenci aminokyselin a informace o pozici jednotlivých atomů proteinu.

Níže je uvedena ukázka části PDB souboru. Informace o typech záznamů a další podrobnosti o formátu PDB lze nalézt v příručce popisující PDB formát [10].

```
HEADER    OXYGEN TRANSPORT                        20-MAR-09   3GOU
TITLE     CRYSTAL STRUCTURE OF DOG (CANIS FAMILIARIS) HEMOGLOBIN
                                                :
SOURCE   2 ORGANISM_SCIENTIFIC: CANIS FAMILIARIS;
SOURCE   3 ORGANISM_COMMON: DOG;
                                                :
EXPDTA    X-RAY DIFFRACTION
                                                :
SEQRES   1 A  141  VAL LEU SER PRO ALA ASP LYS THR ASN ILE LYS SER THR
SEQRES   2 A  141  TRP ASP LYS ILE GLY GLY HIS ALA GLY ASP TYR GLY GLY
                                                :
ATOM     956  N   LYS A 127     -26.156  -9.235  -1.714  1.00 17.63           N
ATOM     957  CA  LYS A 127     -25.997 -10.441  -0.923  1.00 17.12           C
ATOM     958  C   LYS A 127     -24.727 -11.172  -1.277  1.00 16.50           C
ATOM     959  O   LYS A 127     -24.037 -11.677  -0.403  1.00 16.59           O
ATOM     960  CB  LYS A 127     -27.206 -11.336  -1.076  1.00 17.04           C
                                                :
END
```

Kapitola 4

Specifikace požadavků

Cílem této kapitoly je prezentovat požadavky skupiny proteinových inženýrů Loschmidtových laboratoří na sestavení databázového systému pro správu biologických dat.

Protože původní zadání práce bylo pouze rámcové, bez udání podrobností, bylo třeba přistoupit k získání detailnějších informací o požadavcích proteinových inženýrů na vznikající systém. Za účelem zpřesňování požadavků a debaty nad návrhem řešení jsou pořádány schůzky B–týmu Loschmidtových laboratoří, na kterých jeho členové prezentují aktuální stav projektů, určují cíle příští schůzky, případně se seznamují s pojmy z oblasti bioinformatiky a proteinového inženýrství.

Současný stav

V současné době není v Loschmidtových laboratořích používán jednotný systém pro uchovávání dat zkoumaných enzymů. Zjištěná data o proteinech jsou uchovávána v příslušných speciálních souborech. Formát souboru závisí na typu dat — PDB pro 3D strukturu, FASTA pro sekvenční strukturu proteinu. Pro naměřená data je nejčastěji použit sešit programu Microsoft Excel. Některá data, která budou do databáze ukládána, nejsou prozatím k dispozici.

V minulosti proběhl pokus o vytvoření databáze HADES — jedná se o diplomovou práci Ivy Urbanové: *Sbírka dat z oblasti proteinové a enzymové biologie* [26], výsledná aplikace však nebyla nasazena do provozu. Je třeba určit, zda je možné práce dále využít, nebo je třeba vytvořit nový návrh.

Požadavky na databázi

Databáze bude obsahovat experimentální i teoretická data o proteinech. Při návrhu je třeba vzít v úvahu, že vkládaná data nemusí být stejného charakteru, uchovávány budou rozličné údaje o proteinech, jako například údaje o 3D struktuře, různé naměřené hodnoty, sekvenční struktura proteinů a podobně.

Struktura databáze nebude pevná, musí být umožněna její jednoduchá modifikace, a to i bez znalosti databázových dotazovacích jazyků. Za tímto účelem je třeba navrhnout způsob řešení, který bude umožňovat dostatečnou flexibilitu a rozšiřitelnost. Přitom je třeba brát ohled na to, že úpravy bude provádět specialista z oboru proteinového inženýrství, ne programátor.

Funkční požadavky

Jedním z nejdůležitějších požadavků je možnost snadného rozšiřování databáze o nové entity a vlastnosti. Protože se předpokládá, že systém budou využívat i jiné laboratoře, nejen Loschmidtovy laboratoře, je požadováno, aby každá laboratoř měla možnost si zvolit strukturu a podobu ukládaných dat. Tato struktura se může v průběhu užívání systému měnit a rozšiřovat, na což je třeba při návrhu a realizaci aplikace brát ohled.

Hlavní část aplikace bude tvořit administrace, která umožní uživatelům základní manipulaci s daty (zobrazení, vkládání, editaci a mazání). Administrační část aplikace bude nabízet také definici struktury databáze, která bude určovat podobu ukládaných dat. Struktura databáze může být modifikována dynamicky, v průběhu využívání systému. Je třeba, aby úpravy mohli provádět samotní proteinoví inženýři, bez nutnosti zásahu programátora.

Protože s aplikací budou pracovat především proteinoví inženýři, nesmí je aplikace zatěžovat technickými detaily, ale musí poskytovat uživatelsky přívětivé prostředí pro snadné zadávání a editaci záznamů. Dále by měla být nabídnuta automatická kontrola formátu vstupních dat. Jelikož je v současné době pro uchovávání dat používán program Microsoft Excel, bude vhodné umožnit export a import dat z tohoto formátu.

Do administrační části aplikace bude moci vstoupit pouze přihlášený uživatel, stejně tak pro provádění některých operací bude vyžadováno oprávnění daného uživatele. Za tímto účelem je třeba, aby aplikace poskytovala správu uživatelských účtů a definici jejich oprávnění.

Aplikace bude dále nabízet jednoduchý portál určený veřejnosti, který bude sloužit k prezentaci informací o uložených enzymech.

Kapitola 5

Analýza dostupných řešení

Před návrhem a realizací vlastního systému bylo věnováno značné úsilí průzkumu dostupných návrhů a řešení systémů pro správu biologických dat a systémů pro uchovávání dat, které mohou být snadno rozšiřovány a udržovány neprogramátory.

V této kapitole je nejdříve proveden rozbor diplomové práce Ivy Urbanové — *Sbírka dat z oblasti proteinové a enzymové biologie* a je určeno, zda bude na tuto práci navázáno nebo bude vytvořen nový návrh aplikace. Dále jsou popsána nejzajímavější řešení, která by mohla být použita při realizaci aplikace. Jedná se především o řešení EAV (*Entity–Attribute–Value*), podrobněji popsané v kapitole 5.3. Na závěr kapitoly je proveden test tradičního a EAV přístupu uchování dat, tyto přístupy jsou porovnány a je zvolen přístup, který bude dále využit pro návrh systému HADES.

5.1 Analýza práce *Sbírka dat z oblasti proteinové a enzymové biologie*

Po úvodní fázi specifikace požadavků proteinových inženýrů na databázový systém byla analyzována diplomová práce Ivy Urbanové — *Sbírka dat z oblasti proteinové a enzymové biologie* [26] z roku 2006, jejímž cílem byla elektronická organizace sbírky dat centra National Centre for Biomolecular Research při Přírodovědecké fakultě Masarykovy univerzity v Brně (projekt HADES).

Hlavním cílem analýzy je zjistit, zda je vhodné využít stávající rozpracovanou verzi databáze a provést její úpravy, nebo vytvořit kompletně nový návrh.

5.1.1 Rozbor návrhu systému

Původní návrh databázového systému Ivy Urbanové [26] postupně prošel dvěma nevyhovujícími verzemi, než vznikl výsledný návrh řešení. Za účelem možného poučení se z chyb a vyvarování se jich jsou v této kapitole stručně rozebrány i oba neúspěšné návrhy. Třetí, výsledný návrh databáze je analyzován podrobněji, aby bylo možné určit, zda je vyhovující a bude použit pro pokračování v projektu, nebo zda je třeba vytvořit nový.

Pevně daná struktura databáze

První verze počítala s pevnou strukturou databázových tabulek obsahujících nejdůležitější informace o proteinech. Dále byla pro každou sekci (*Molecular properties*, *Structural proper-*

ties, *Evolution*, atd.) vytvořena tabulka s pevně danými sloupci představujícími vlastnosti sekce.

Vzhledem k tomu, že data jsou klasifikována do předem určených sekcí a zadavatelé mají jasnou představu, do které sekce data patří, byl tento návrh nejjednodušším řešením, které by bylo dostačující v případě, že by data byla již neměnná, nedocházelo by k vytváření nových sekcí a atributy sekce by byly pevně dané. Tato skutečnost je však v oblasti výzkumu nereálná. Navíc není možné očekávat, že proteinoví inženýři budou provádět odborné zásahy a změny ve schématu databáze nebo v samotném zdrojovém kódu aplikace. A pokud ano, lze předpokládat, že taková činnost bude pro proteinového inženýra neúměrně pracná, časově náročná a zbytečně zatěžující.

Na základě výše zmíněných důvodů byl návrh zamítnut a dále se ubíral směrem větší flexibility, která umožní svobodné vytváření nových sekcí, případně dovolí přidávat nové vlastnosti sekce.

Úplná obecnost databázových tabulek

Druhá verze návrhu spočívala v nahrazení programu Microsoft Excel webovou aplikací vzhledově připomínající tabulkový editor, ale s omezenou funkčností, která by dostačovala potřebám proteinových inženýrů pro vkládání a editaci dat.

Návrh umožňuje libovolné vytváření tabulek (obdoba listu aplikace Microsoft Excel) a spočívá v zavedení tří hlavních databázových tabulek:

- `HTable`, která uchovává informace o tabulkách,
- `HCols`, uchovávající informace o jednotlivých sloupcích pro danou tabulku,
- `HCell`, která obsahuje skutečná data pro daný sloupec a řádek tabulky.

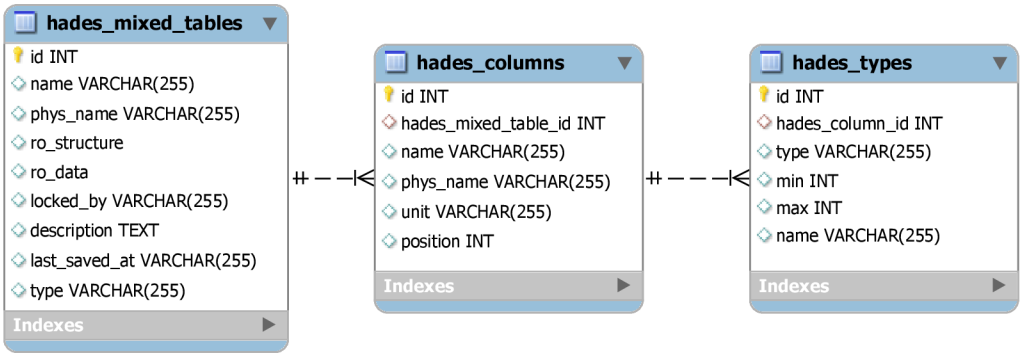
Tento způsob řešení byl pro proteinové inženýry použitelný a dostačující, ale podle I. Urbanové [26] se ukázalo, že problémem je neefektivnost přístupu k datům.

Výsledný návrh

Z důvodu zvýšení efektivity přístupu k datům byla omezena obecnost některých tabulek. Jednalo se o specifické vlastnosti, jako například sekvenci, strukturu nebo reference. Tyto vlastnosti jsou uloženy do specializovaných tabulek, se kterými tak mohou být prováděny požadované operace bez zatěžování tabulek obecných. Myšlenka nahrazení souborů Microsoft Excel webovou aplikací byla zachována. Veškerá data jsou ukládána do databáze, včetně speciálních formátů dat, která byla uchovávána v souborech.

Návrh spočívá ve vytvoření tří metadatových tabulek, které jsou znázorněny na obrázku 5.1. Jedná se o tabulky:

- `tables`, která uchovává název tabulky zadaný uživatelem a fyzický název tabulky vytvořené v databázi,
- `columns`, která popisuje sloupec tabulky a obsahuje název zadaný uživatelem a fyzický název sloupce v databázové tabulce,
- `types` definující datový typ sloupce, případně omezení rozsahu a podobně.



Obrázek 5.1: Metadatové tabulky výsledného návrhu

5.1.2 Analýza aplikace

Výsledná aplikace byla implementována v interpretovaném skriptovacím jazyce *Ruby*[1] s využitím frameworku *Ruby on Rails*[2] a relačního databázového systému *PostgreSQL*[3].

Protože systém nebyl nasazen do provozu, bylo třeba pro jeho vyzkoušení zprovoznit webový server a interpret jazyka Ruby, včetně instalace frameworku Ruby on Rails. Za tímto účelem byl využit webový server *WEBrick*[4], který je standardně nabízen frameworkem *Ruby on Rails* pro testování během vývoje aplikace. Vzhledem k tomu, že aplikace vznikala v roce 2005–2006, objevilo se několik problémů způsobených rozdíly v použitých verzích frameworku. Problém byl odstraněn nahrazením již nepodporovaných metod metodami novými a po těchto úpravách se již mohlo přistoupit k vlastnímu testování funkcionality aplikace.

Implementovaná funkcionalita

Funkčnost systému je rozdělena do dvou oblastí, a to portálu určeného pro veřejnost a administrační části, do které mají přístup pouze oprávnění uživatelé.

Administrační část aplikace umožňuje především manipulaci s tabulkami, které odpovídají jednotlivým sekcím (obdobně jako tomu bylo při využívání programu Microsoft Excel). Jedná se o vytváření nových tabulek, definici jejich sloupců, editaci hodnot, řazení a mazání řádků a zobrazování obsahu datových tabulek. Editace záznamů vzhledově připomíná tabulkový editor, poskytuje však pouze základní funkčnost potřebnou pro zadávání hodnot.

Grafické uživatelské rozhraní editace dat je tvořeno jednoduchým tabulkovým systémem, který vzhledově připomíná list sešitu programu Microsoft Excel. Sloupce tabulky představují jednotlivé vlastnosti sekce a řádky záznamy dané sekce. Tento způsob editace odpovídá požadavkům proteinových inženýrů na jednoduchost a přehlednost editace dat.

Jedním z požadavků bylo umožnění importu dat ze sešitů programu Microsoft Excel. Protože v roce 2006 ještě nebyla dostupná a používaná specifikace *Office Open XML*, navržená společností Microsoft, byl pro získání dat z Excelu použit mezikrok spočívající v uložení dat programem Open Office, který vytvořil XML soubor s požadovanými daty.

[1] http://www.ruby-lang.org/
[2] http://rubyonrails.org/
[3] http://www.postgresql.org/
[4] http://en.wikipedia.org/wiki/WEBrick

Pro procházení struktury XML souboru byla využita knihovna jazyka Ruby. Určitým omezením je také skutečnost, že importní soubor musí být podle I. Urbanové [26] vytvořen v Open Office verze 1, novější verze programu Open Office přešly na novou vnitřní strukturu ukládaného XML souboru, kterou aplikace není schopna zpracovat.

Veřejná část aplikace poskytuje informace o databázi HADES a umožňuje především vyhledávání údajů v databázi. Implementována byla pouze základní funkcionalita, a to rychlé vyhledávání zadaného klíčového slova.

Vyhledávání je možné na základě obecné identifikace proteinu (kód proteinu nebo jeho název), případně prostřednictvím specifické vlastnosti proteinu (molekulární váha, registrační číslo CAS, hodnota optimálního pH atd.). Vyhledávání na základě obecné identifikace proteinu probíhalo bez problémů, pokus o vyhledávání vlastnosti proteinu byl však neúspěšný.

Zdrojový kód

Díky použití frameworku Ruby on Rails jsou zdrojové soubory organizovány do určených složek, což po nastudování konvencí přináší výrazně rychlejší orientaci v množství souborů zdrojového kódu aplikace. Stejně tak architektura *Model–View–Controller* vnáší do kódu vyšší přehlednost díky oddělení datového modelu od řízení aplikace a prezentace dat. Při implementaci editace hodnot tabulek byly použity technologie *JavaScript* a *AJAX*.

Zdrojový kód aplikace není příliš rozsáhlý, velké množství kódu (především datové modely) je vygenerováno prostřednictvím frameworku. Většina kódu se vztahuje k administrační části aplikace, veřejná část poskytuje pouze základní vyhledávání proteinů.

Za značnou nevýhodu považuji nedostatečně (minimálně nebo vůbec) komentovaný zdrojový kód, což znesnadňuje orientaci a pochopení kódu. Nepoužíváním standardizovaných zápisů komentářů je znemožněno automatické generování programové dokumentace.

5.1.3 Závěr analýzy

Podrobně byl popsán návrh, funkcionalita i zdrojový kód existující aplikace. Vychází se především z práce Ivy Urbanové [26] a z dostupného zdrojového kódu výsledné aplikace, která však nebyla nasazena do provozu. Důvodem může být fakt, že vyvíjená aplikace prošla třemi, značně odlišnými, verzemi, což bylo časově příliš náročné a nebylo tak dosaženo požadovaného výsledku.

Cílem analýzy bylo určení směru, kterým by se měl budoucí návrh ubírat. Zjistilo se, že je třeba poskytnout odpovídající flexibilitu a jednoduchou rozšiřitelnost a přitom zajistit dostatečnou rychlost a efektivitu přístupu k datům.

Po podrobné analýze současného řešení (popsaného v této kapitole) a následné dohodě s vedoucím práce a členy B–týmu Loschmidtových laboratoří se dospělo k závěru, že bude navržena nová databáze HADES. Klíčovými body v rozhodnutí byl poměrně malý rozsah aplikace, využití jazyka Ruby, který není tolik známý (jako např. jazyky PHP nebo Java) a také špatně komentovaný nebo již nefunkční zdrojový kód aplikace, což by ztěžovalo pokračování ve vývoji aplikace.

5.2 Uchovávání dat v XML formátu

Obecné značkovací jazyky (jako například XML) definují logickou strukturu dokumentu a jejich obsah. Není předdefinována jejich pevná struktura a názvy prvků, struktura i názvy mohou být vytvářeny libovolně. To s sebou přináší výhody nezávislosti použití modelovaných dat a také nezávislost dat na způsobu jejich prezentace [14].

Použití obecných značkovacích jazyků umožňuje modelování struktury dat v podobě vhodné pro aplikační zpracování a s názvy strukturních prvků, které odpovídají kontextu aplikace [14].

Jak uvádí J. Chen a A. S. Sidhu [14], je díky formalismu nabízenému obecnými značkovacími jazyky možné specifikovat komplexní struktury. Navíc je dovoleno využívat volitelných prvků. Volitelné prvky představují výjimky, které se v datech vyskytují pouze v některých případech. Modelování volitelných prvků v databázích bývá často komplikované.

Podpora XML v databázových systémech

Protože uchování dat ve formátu XML by přineslo dostatečnou flexibilitu a rozšiřitelnost, která je od vznikajícího systému vyžadována, byla prozkoumána situace v podpoře XML formátu některými databázovými systémy. Zejména je třeba se zaměřit na možnost vyhledávání v XML řetězci, případně i indexování těchto hodnot.

Databázový systém MySQL nabízí funkce `ExtractValue()` a `UpdateXML()`, které získají nebo aktualizují hodnotu v textovém sloupci databázové tabulky, jehož obsahem je XML řetězec [3]. Protože MySQL nenabízí pro uchování XML řetězce speciální datový typ a tento řetězec je uložen v textovém poli bez dalších úprav, není možné jej indexovat a zrychlit tak vyhledávání na základě hodnot prvků XML struktury [3].

Databázový systém PostgreSQL nabízí datový typ `xml`, který je určen k ukládání dat ve formátu XML [4]. Podle dokumentace databázového systému PostgreSQL [4] je jeho výhodou oproti ukládání XML řetězce jako textu kontrola správného formátování vstupu a také možnost využití některých funkcí pro manipulaci s XML. Přímé vyhledávání na základě hodnoty prvku XML řetězce ale není možné, protože datový typ `xml` neposkytuje žádné operátory porovnání. Pro porovnání hodnot je doporučeno je nejprve pomocí poskytovaných funkcí převést na řetězce [4]. Protože datový typ `xml` neposkytuje operátory porovnání, není možné databázové sloupce tohoto datového typu indexovat [4].

Přestože popsané databázové systémy umožňují uchovávání libovolných dokumentů a jejich částí ve formátu XML, nejsou kvůli neexistenci dobře definovaného a univerzálního algoritmu pro porovnání dat XML dokumentu vhodné pro vyhledávání hodnot v XML datech.[5] XML data jsou v těchto databázových systémech využívána především pro uchovávání libovolných dat v dobře definované a lehce rozšiřitelné struktuře.

[5] http://www.postgresql.org/docs/current/static/datatype-xml.html

5.3 *Entity–Attribute–Value* model pro ukládání dat

Entity–Attribute–Value (EAV) model představuje netradiční způsob uložení dat, která jsou široce variabilní a často také neúplná nebo neznámá. Typickým příkladem jsou vědecká data, kdy se pro zkoumaný problém sleduje velké množství vlastností, z nichž jsou definovány pouze některé (většinou pouze malá část).

Konkrétním příkladem může být databáze pacientů a jejich nemocí, kdy každá nemoc je provázena jinými příznaky a pro různé diagnózy pacienta jsou tedy sledovány různé příznaky (například u zápalu plic bude zkoumána teplota pacienta a poslechový nález, kdežto při zlomeninách bude sledován typ zlomeniny, případně tendence srůstu).

Při tradičním přístupu k modelování databáze by bylo nutné vytvořit pro každý příznak nový sloupec v databázové tabulce, což by při komplexním systému vedlo k extrémnímu množství sloupců.

Jinou možností by bylo jednotlivé příznaky vkládat do jednoho databázového textového sloupce a vzájemně je oddělovat definovaným oddělovačem. Přitom je třeba zajistit, aby stejný oddělovač nebyl použit také v hodnotě příznaku — to je možné využitím tzv. *escape sekvencí*[6]. Protože se však dá předpokládat, že bude třeba s jednotlivými příznaky dále pracovat (například je prohledávat nebo porovnávat), není takový způsob uložení jednotlivých příznaků vhodný.

Problém různorodosti dat je možné pozorovat také v každodenním světě, ne jen u klinických a vědeckých dat. Příkladem může být databáze produktů v supermarketu, kdy každý typ produktu má své charakteristické vlastnosti, se kterými je třeba dále pracovat.

Princip EAV

Princip EAV spočívá v modelování vlastností objektu jako jednotlivých řádků v databázi. Každý řádek v tabulce reprezentuje jediný fakt (na rozdíl od konvenčního modelování, kdy pro každou vlastnost je vytvořen sloupec databázové tabulky a řádek pak obsahuje více faktů – jeden pro každý sloupec) [12].

Jak již z názvu *Entity–Attribute–Value* vyplývá, jsou data modelována jako uspořádané trojice (*entity*, *attribute*, *value*), kde *entity* představuje prvek z množiny entit (definovaných objektů), *attribute* je prvek z množiny atributů entity a *value* představuje hodnotu tohoto atributu.

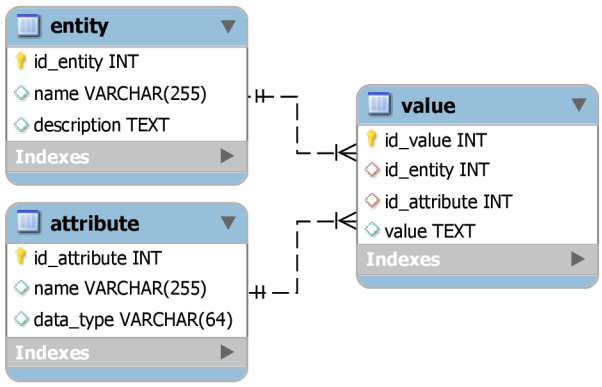
Důležitou roli v tomto přístupu hrají *metadata*[7]. Nezbytnou součástí EAV je metadatová tabulka popisující dostupné atributy entity. Na obrázku 5.2 je znázorněno databázové schéma jednoduchého EAV systému. Hodnoty atributů jsou ukládány do jediné datové tabulky se sloupcem textového charakteru, a tedy všechny ukládané hodnoty musí být převedeny na řetězce.

Databázová tabulka `entity` představuje úložiště pro vlastní entitu (mohlo by se jednat např. o pacienta), tabulka `attribute` představuje metadatovou tabulku, která udržuje přehled o dostupných atributech pro entitu, a do tabulky `value` jsou ukládány konkrétní hodnoty atributů entity.

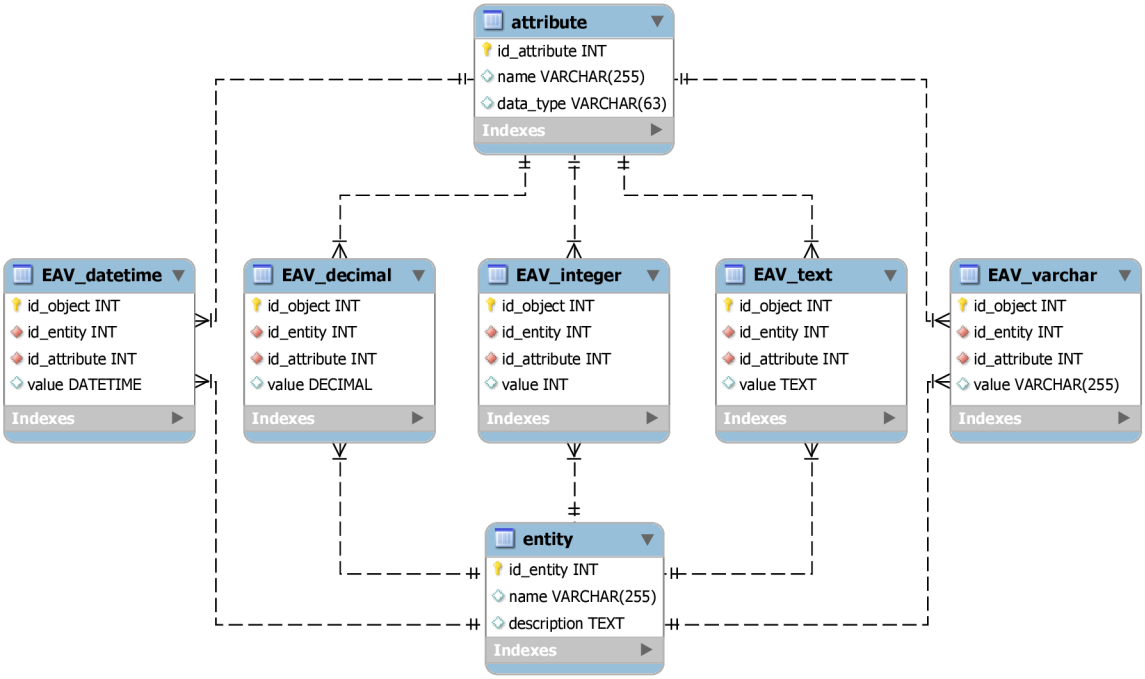
Častěji však jsou hodnoty uloženy ve specializovaných datových tabulkách (viz obrázek 5.3), které se určují na základě datového typu atributu [12]. Využití specifičtějších databázových tabulek umožňuje lepší podporu indexování a řazení hodnot databázovým systémem.

[6] http://en.wikipedia.org/wiki/Escape_sequence

[7] metadata představují strukturovaná data o vlastních datech



Obrázek 5.2: Databázové schéma jednoduchého systému EAV



Obrázek 5.3: Databázové schéma rozšířeného systému EAV

Vlastnosti EAV

Výhodou EAV přístupu je, že není třeba ukládat prázdné nebo nedefinované hodnoty, ale je možné ukládat pouze ty, které jsou pro danou entitu podstatné. Dalším přínosem je jednoduchost přidávání nových vlastností, což je jedním ze základních požadavků při vývoji informačního systému vědeckých pracovišť. EAV umožňuje přidávání nových atributů, a to bez změny databázového schématu. Stejně tak může být přínosné umožnit uložení více hodnot jedné vlastnosti.

Flexibilita EAV modelu však s sebou přináší některé nevýhody, a to zejména jeho složitost, protože logický model dat je značně odlišný od fyzického uložení v databázi. Z důvodu

této odlišnosti není EAV systém vhodný pro analytické operace [12].

Přestože jsou hodnoty jednotlivývh atributů uloženy v databázi jako řádky, musí uživatelské rozhraní vytvořit iluzi konvenčního uložení dat – tak, jak jsou na něj koncoví uživatelé zvyklí [21], to znamená podstatně náročnější návrh i vývoj systému.

Často se kombinuje přístup konvenčního ukládání dat typických pro entitu s využitím EAV pouze pro tzv. *řídká data* [21]. Teprve na základě typu atributu se určuje přístup k němu (uložení atributu ve sloupci nebo na řádku databázové tabulky), čímž se stává implementace systému ještě náročnější.

5.3.1 Varianta EAV/CR

Možnou variantou EAV je tzv. *EAV/CR* (*Entity–Attribute–Value with Classes and Relationships*), která představuje rozšíření EAV o definice tříd (různé typy entit) a umožňuje dále definovat vazby mezi jednotlivými třídami [21].

Přínosem jsou třídy představující typy entit a umožňující přiřazovat atributy k typům entit. Díky tomu je možné vytvářet různé varianty entit, případně vytvořit entity zcela odlišné. Atributy jsou vázány na třídy, což může být výhodné při používání EAV systému pro více různých typů entit, protože dochází k určení, které atributy jsou dostupné pro daný typ entity.

5.3.2 Srovnání klasického a EAV přístupu

Entity–Attribute–Value model pro reprezentaci dat poskytuje jednoduché a flexibilní databázové schéma pro ukládání různorodých biomedicínských dat, avšak získávání dat tohoto modelu je méně efektivní, než je tomu v případě tradičního způsobu modelování dat [15].

Aby bylo možné určit rozdíly v jednotlivých přístupech modelování rozšiřujících se a univerzálních dat, byly provedeny základní testy zabývající se dobou trvání SQL dotazů.

Způsob testování

Za účelem testování byly vytvořeny dvě databáze — první pro testování klasického přístupu a druhá pro testování EAV přístupu. Obě databáze byly vytvořeny v databázovém systému MySQL a testovány na nevytíženém lokálním serveru. Konfigurace serveru je popsána v tabulce 5.1.

Operační systém	Ubuntu 9.10 (Linux 2.6.31-20-generic)
Verze MySQL	5.1.37
Procesor	Intel(R) Core(TM)2 Duo CPU E6750 @ 2.66GHz
Fyzická paměť	2.0 GiB

Tabulka 5.1: Konfigurace testovacího serveru

Pro test byl uvažován příklad, kdy entitu tvoří 35 atributů — 10 atributů typu celé číslo (`INT`), 10 atributů typu řetězec (`VARCHAR`), 5 atributů typu desetinné číslo (`DECIMAL`), 5 atributů typu text (`TEXT`) a 5 atributů typu datum (`DATETIME`). Vytvářené a vyhledávané hodnoty atributů byly vybírány náhodně.

Pro testování klasického přístupu je vytvořena jediná databázová tabulka, která je tvořena 35 sloupci požadovaných datových typů. Dalšími sloupci jsou identifikátor entity, její název a popis.

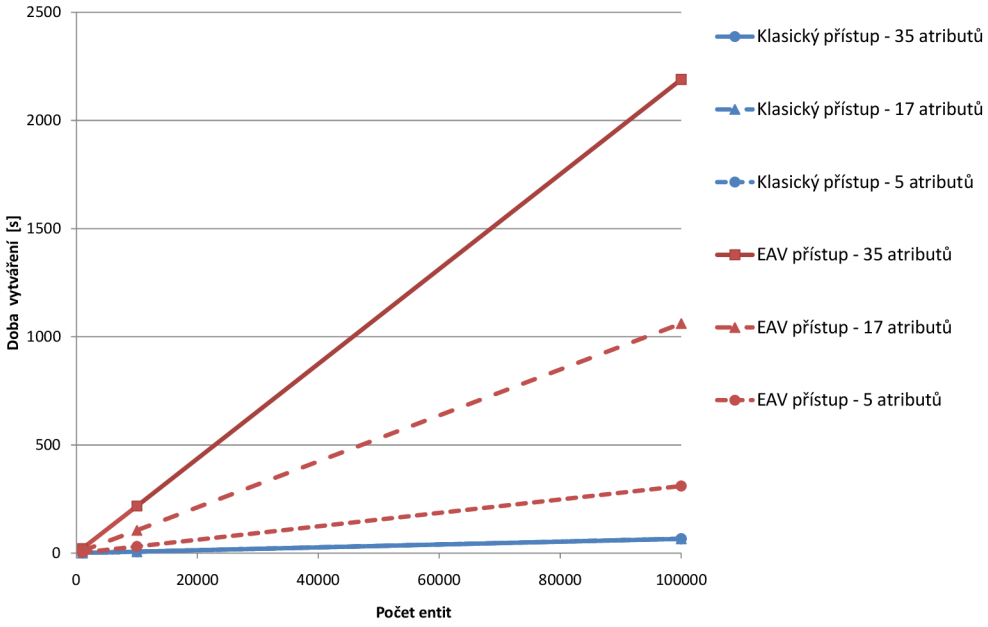
Pro testování EAV přístupu jsou vytvořeny metadatové tabulky `entity` a `attribute` a dále datové tabulky `eav_int`, `eav_varchar`, `eav_decimal`, `eav_text`, `eav_date`.

Test vytváření entit

Testem vytváření entit byla zkoumána doba potřebná pro vytvoření záznamu entity. Zkoumány byly doby potřebné pouze k vytvoření dat entity (tedy bez metadat).

Pro klasický přístup je pro vložení jednoho záznamu využit jeden SQL příkaz, kdežto pro EAV přístup je každá hodnota uložena jako řádek v příslušné datové tabulce, a tedy pokud jsou pro entitu známy všechny hodnoty atributů, je provedeno 35 SQL příkazů `INSERT`.

Byly testovány také případy, kdy jsou definovány hodnoty pouze pro část atributů, a to 17 hodnot z 35 nebo 5 hodnot z 35. Výsledky testu jsou znázorněny v grafu na obrázku 5.4 a naměřené hodnoty připojeny v příloze A.



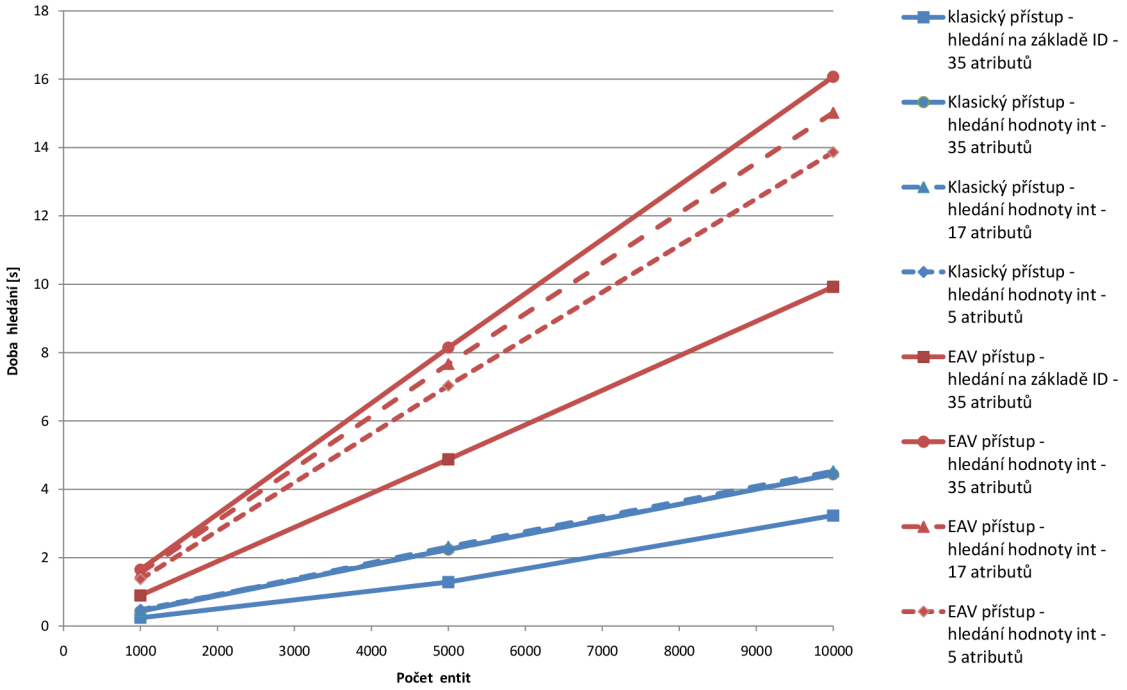
Obrázek 5.4: Graf doby trvání vytváření entit — srovnání EAV a klasického přístupu

Test vyhledávání entit

Cílem testu vyhledávání entit bylo určení času potřebného k získání dat entity. Vyhledávání bylo provedeno na základě hodnoty primárního klíče entity (ID entity), na základě číselné hodnoty v neindexovaném sloupci a na základě číselné hodnoty v indexovaném sloupci, pro indexování je využita datová struktura B–stromu.

Při vyhledávání jsou získávána všechna data entity, a tedy pro přístup EAV bylo třeba nejprve nalézt požadovanou hodnotu, na základě vyhledávání určit požadované entity a pro ty následně získat data.

Výsledky testu jsou znázorněny v grafu na obrázku 5.5 a naměřené hodnoty připojeny v příloze A.



Obrázek 5.5: Graf doby trvání vyhledávání entit — srovnání EAV a klasického přístupu

Zhodnocení testu

Výsledek testu ukazuje výkonnostní problémy systému uchovávání dat EAV. Test vytváření entit ukázal značnou slabinu EAV přístupu, a to pomalost vkládání nových dat. Z grafu znázorněného na obrázku 5.4 a naměřených dat prezentovaných v příloze A lze vypozorovat, že vytváření entit pro EAV přístup je asi 30krát pomalejší, pokud jsou známy hodnoty pro všech 35 atributů, 15krát pomalejší, pokud jsou zadány hodnoty 17 atributů, a 4krát pomalejší, pokud jsou známy hodnoty 5 atributů.

Doba pro vytvoření entity je pro klasický přístup nezávislá na počtu ukládaných hodnot (doba je konstantní), kdežto pro EAV přístup je čas vytvoření entity lineárně závislý na počtu definovaných hodnot. To odpovídá skutečnosti, že pro klasický přístup je proveden vždy jeden SQL příkaz `INSERT`, kdežto pro EAV přístup je počet SQL příkazů závislý na počtu definovaných hodnot (jeden příkaz pro každou hodnotu).

Jak je znázorněno v grafu zobrazeném na obrázku 5.5, trvalo vyhledávání entity pro EAV přístup 3krát až 4krát déle, než tomu bylo při klasickém přístupu uložení dat. To je dáno skutečností, že je třeba nejprve provést vyhledávání hodnoty v datové tabulce, určit požadované entity a pro ty provést získání hodnot ze všech datových tabulek.

Aby byla výkonnost EAV systému zvýšena, je možné využit některé optimalizace. Jedná se především o využití vyrovnávací paměti, ve které jsou uchovávány datové hodnoty, čímž se snižuje počet databázových vyhledávání, nebo datovou strukturu EAV pravidelně převádět do klasické formy [15].

Test výkonnosti EAV systému popsaného v [15] ukazuje, že výkonnost reprezentace EAV oproti klasickému přístupu je asi třikrát až pětkrát menší. Rozdíly v efektivitě dotazů se projevovaly více při rozšiřování databáze. Práce [15] poukázala, že opakované jednoduché databázové dotazy vykonávané v dávkách byly efektivnější než složené dlouhé SQL dotazy.

Kapitola 6

Návrh řešení

Kapitola se věnuje návrhu flexibilního systému pro správu biologických dat. Návrh vychází z analýzy možných řešení, která byla popisována v kapitole 5, zejména z modelu *Entity–Attribute–Value* (viz kapitola 5.3), popisuje navržený systém entit a atributů, charakteristiku navrhované aplikace a způsob mapování datových modelů na databázové objekty.

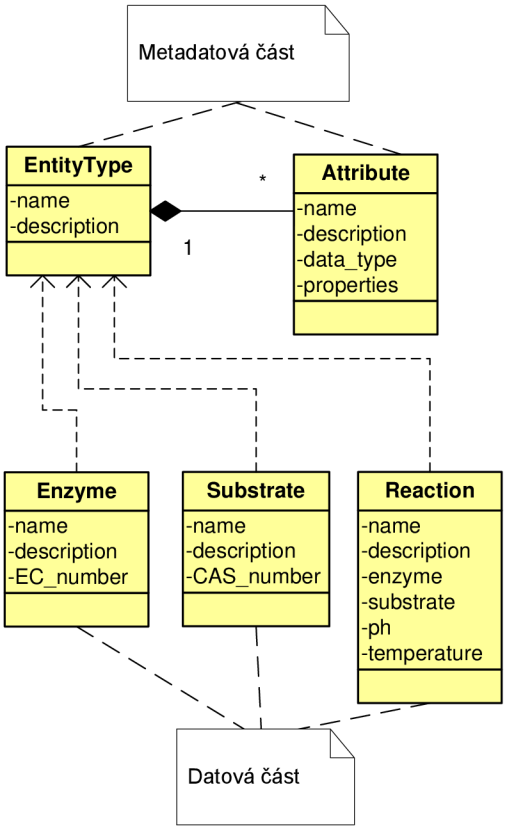
6.1 Univerzální systém pro modelování dat

Vzhledem k základnímu požadavku flexibility a jednoduché rozšiřitelnosti systému bylo třeba navrhnout flexibilní a univerzální způsob modelování a uchovávání dat. Navržený systém vychází z *Entity–Attribute–Value* modelu pro ukládání dat popsaného v kapitole 5.3, avšak reflektuje výkonnostní potíže popsané v [15] a ověřené testem porovnávajícím dobu trvání SQL dotazů pro EAV přístup a klasický přístup (viz kapitola 5.3.2).

Pro poskytnutí dostatečné volnosti při specifikaci datového modelu budou k dispozici základní prvky, a to:

- **typy entit**, představující dostupné typy objektů, které mají společnou charakteristiku (např. typ entity enzym, substrát, reakce atd.). V terminologii objektově orientovaného programování jde o třídy.
- **atributy**, představující vlastnosti, které jsou charakteristické pro daný typ entity (např. název, EC číslo, teplota apod.). V terminologii objektově orientovaného programování se jedná o vlastnosti třídy.
- **entity**, které reprezentují konkrétní výskyt typu entity s konkrétními hodnotami atributů (např. enzym trypsin, pepsin apod.). V terminologii objektově orientovaného programování jde o instance tříd.

Navržený systém umožní libovolné vytváření typů entit a jejich atributů a pro tyto definované typy entit bude možné následně vytvářet konkrétní entity. Entity představují vlastní data, pro která jsou definována metadata v podobě typů entit a atributů. Situace je znázorněna na obrázku 6.1, který zobrazuje diagram aplikačních tříd entit a atributů. Třídy `EntityType` a `Attribute` představují metadatovou část, která pro daný typ entity definuje dostupné atributy. Datová část je reprezentována konkrétními třídami `Enzyme`, `Substrate` a `Reaction`. Tyto aplikační třídy jsou vytvářeny dynamicky na základě dostupných metadat. Podrobnějším popisem a návrhem univerzálního systému entit a atributů se zabývá následující podkapitola.



Obrázek 6.1: Diagram tříd entit a atributů

6.2 Systém entit a atributů

Zejména kvůli potřebě jednoduché rozšiřitelnosti systému a umožnění provádět změny ukládaných dat samotnými proteinovými inženýry je navržen systém entit a atributů, který dovoluje rozšiřování datového modelu na základě aktuálních potřeb.

Uživatelé systému mohou specifikovat dostupné typy entit a jejich atributy, modifikovat je, organizovat do skupin a následně pro definované typy entit vytvářet konkrétní entity představující vlastní modelovaná data.

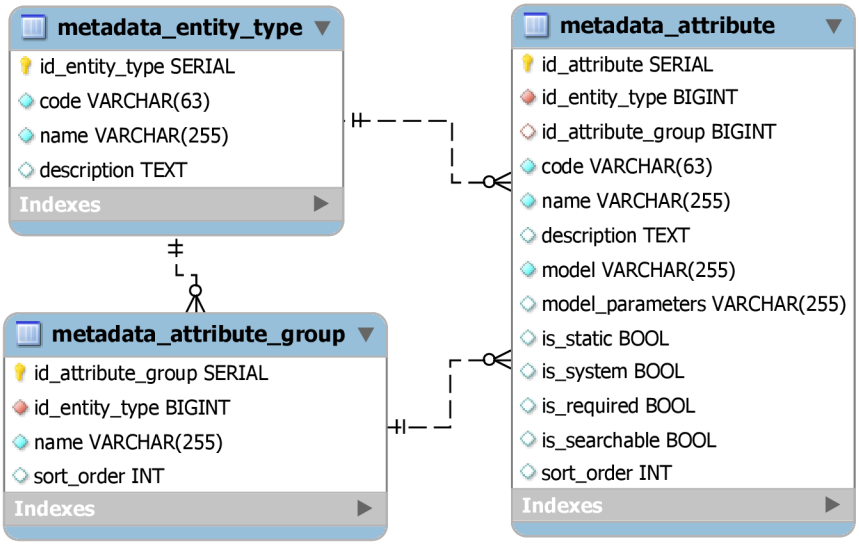
6.2.1 Ukládání metadat o entitách

Přestože systémy pro řízení bází dat (DBMS) obsahují a zpracovávají metadata o dostupných databázových tabulkách, jejich sloupcích, případně indexech, jsou navrženy ještě vlastní metadatové tabulky, které popisují jednotlivé typy entit a jejich atributy.

K udržování vlastních metadatových tabulek se přistoupilo především z důvodu nedostačujícího oboru hodnot datového typu databázového sloupce, podle kterého by se měly určovat vlastnosti atributu. Například do databázového sloupce textového charakteru mohou být uloženy hodnoty různých typů atributů (prostý text, sekvence proteinu, hypertextový odkaz a podobně). Pokud by určení typu atributu probíhalo pouze na základě datového typu databázového sloupce, představovaly by všechny hodnoty těchto atributů prostý text bez možnosti dalšího formátování či přiřazení sémantických vlastností.

Budováním metadat bude také umožněn jednodušší přechod na smíšený přístup k ukládání hodnot atributů (přístup, kdy jsou některé hodnoty atributů uloženy jako sloupce v entitní tabulce a některé hodnoty atributů jsou uloženy jako řádky v datových tabulkách EAV) a nebo bude možné metadata využívat k automatickému vytváření formulářů uživatelského rozhraní aplikace pro editaci dat.

Metadatové tabulky vychází z modelu *Entity–Attribute–Value*, ovšem jsou přizpůsobeny potřebám dynamické tvorby entit. Navržená podoba metadatových tabulek je znázorněna na obrázku 6.2.



Obrázek 6.2: Databázové schéma metadatových tabulek

Typy entit

Do metadatové tabulky `metadata_entity_type` se ukládají dostupné typy entit a informace o nich (např. typ entity enzym, substrát, produkt atd.). U typů entit prozatím postačují informace o kódu, názvu a podrobnějším popisu typu entity. Významnou roli zde hraje kód (sloupec `code`), který musí být unikátní, protože je dále používán k identifikaci typu entity a využívá se především při dynamické tvorbě entitních tabulek. Na základě kódu typu entity je vytvořen název entitní databázové tabulky.

Atributy entity

Tabulka `metadata_attribute` uchovává informace o dostupných atributech pro daný typ entity. Kromě základních vlastností o atributech, jako jsou název a podrobný popis, jsou důležité především informace o modelu atributu a jeho vlastnostech (databázový sloupec `model` a `model_parameters`). Model atributu představuje aplikační třídu, která zajišťuje manipulaci s hodnotou atributu (formátování, validace nebo způsob editace hodnoty). Důležitou roli hraje také kód atributu, který je využit pro stanovení názvu sloupce v entitní databázové tabulce určeného pro uložení hodnoty atributu.

Skupiny atributů

Zejména pro rozsáhlejší databáze může být výhodné jednotlivé atributy dále seskupovat do uživatelsky definovaných skupin, což umožní lepší orientaci ve velkém množství atributů. Tyto skupiny mohou být využity jak v administrační části aplikace, tak i v části prezentační při zobrazování informací o entitě. K uchovávání dostupných skupin atributů je využita databázová tabulka `metadata_attribute_group`, která slouží také pro určení pořadí skupin atributů používané pro řazení.

6.2.2 Dynamická tvorba entitních tabulek

Na základě výsledků testu, který porovnával klasický přístup s univerzálním přístupem *Entity–Attribute–Value* modelu, bylo rozhodnuto, že bude vhodnější zvolit klasický přístup. Aby ovšem nebyl omezen požadavek flexibility a rozšiřitelnosti systému, byla navržena dynamická tvorba entitních databázových tabulek a jejich atributů (databázových sloupců), která spojuje výhody rychlosti klasického přístupu s možností jednoduché rozšiřitelnosti databáze.

Do těchto entitních tabulek budou následně ukládána data entit, pro které jsou dostupné pouze ty atributy (databázové sloupce), které jsou pro daný typ entity definovány.

Tvorba typů entit

Pro každý typ entity je vytvořena databázová tabulka, která má název odvozený z kódu typu entity (`code` z databázové tabulky `metadata_entity_type`). Tento kód musí být unikátní a v případě, že není uživatelem zadán, je vhodné jej pro zjednodušení práce se systémem automaticky vytvářet na základě názvu typu entity. Vytváření databázových tabulek a jejich modifikace je možná prostřednictvím SQL příkazů `CREATE TABLE` a `ALTER TABLE`.

Aby byla databázová struktura přehlednější, je názvům entitních tabulek přidáván prefix, na jehož základě lze jednoznačně určit, že se jedná o dynamicky vytvářenou entitní tabulku. Navíc se zabrání situacím, kdy by název entitní tabulky byl shodný s názvem již existující databázové tabulky systému. Zvoleným prefixem je označení „`entity_`“.

Tvorba atributů entity

Aby byla zajištěna rozšiřitelnost datového modelu, je entitní tabulka modifikována na základě definice atributů. Pro každý atribut dochází k definici jednoho sloupce v entitní tabulce. Název sloupce je odvozen od kódu atributu (databázový sloupec `code` tabulky `metadata_attribute`). Vytváření a modifikace sloupců entitní tabulky lze provést SQL příkazem `ALTER TABLE`.

Součástí každé vytvářené entity je její identifikátor, název a podrobný popis, proto jsou tyto atributy (a tedy i sloupce) vytvářeny automaticky již při vytváření entitní tabulky. Tyto tři atributy jsou systémové (`is_system`), a proto je není možné odebrat ani změnit jejich kód.

6.2.3 Systém atributů entity

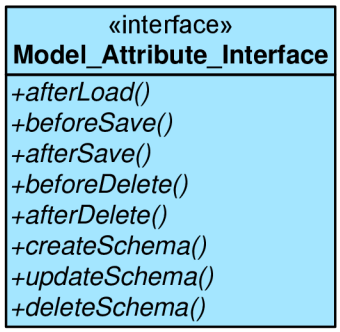
V procesu rozšiřitelnosti aplikace hrají atributy entity velice důležitou roli. Prostřednictvím atributů je možné definovat specifické vlastnosti jednotlivých typů entit a na základě typu atributu lze určit obor hodnot tohoto atributu, případně určit další vlastnosti jako způsob zobrazení hodnoty atributu nebo způsob její editace.

Základní typy atributů mohou být odvozeny od základních datových typů databázového systému. Pro uživatele však nemusí být tento systém dostačující a pravděpodobně brzy by vznikl požadavek na přiřazení sémantických vlastností pro daný typ atributu. Příkladem může být atribut `sequence`, který má představovat sekvenci proteinu a jehož základním datovým typem by byl řetězec. Takovýto atribut by byl zpracováván jako text a v této podobě by byl také prezentován. Z uživatelského hlediska je ovšem výhodnější přistupovat k sekvencím odlišným způsobem — zejména v prezentační části je možné sekvenci nezobrazovat pouze jako prostý text, ale využít některého z nástrojů, který by kromě zobrazení sekvence například znázornil její sekundární strukturu, případně při editaci hodnoty by byla ověřena platnost zadaných dat.

Za účelem přiřazení sémantických vlastností atributu je definován model atributu, který charakterizuje vlastnosti a chování tohoto atributu.

Model atributu

Chování každého typu atributu je definováno aplikační třídou, která implementuje předepsané rozhraní, viz obrázek 6.3. Je třeba, aby třída zprostředkovávala vytvoření a modifikace databázového sloupce entitní tabulky, určovala způsob editace hodnoty atributu, zajišťovala ověření správnosti hodnoty (validaci) a stanovila způsob prezentace hodnoty ve veřejné části aplikace.



Obrázek 6.3: Rozhraní modelu atributu

Modifikaci databázového schématu zajišťují metody modelu atributu `createSchema`, `updateSchema` a `deleteSchema`, které realizují vytváření či úpravu databázového sloupce entitní tabulky pro uchování hodnot atributu.

Aby byl přístup k atributům co nejuniverzálnější, musí třída implementovat metody, které jsou volány v průběhu práce s entitou. Jedná se o metodu `afterLoad()`, která je pro každý atribut volána po načtení entity, metody `beforeSave()` a `afterSave()`, které jsou volány před respektive po uložení entity, a dále metody volané při odstraňování entity `beforeDelete()` a `afterDelete()`. Tyto metody mohou být prázdné, nebo je lze využít pro rozšiřování funkcionality atributu (například metoda `afterLoad` může zajistit požadovanou modifikaci hodnoty, metoda `beforeSave` ověřit platnost ukládaných dat a podobně).

Třída implementující chování atributu je určena na základě hodnoty v databázovém sloupci `model` metadatové tabulky `metadata_attribute`. Různé parametry (například minimální a maximální délka hodnoty atributu) pak mohou být pro atribut uchovávány ve sloupci `model_parameters`.

6.2.4 Prostor k ukládání dat do datových tabulek EAV

Přestože se v současné době zdá, že je výhodnější zvolit přístup dynamické tvorby entitních databázových tabulek, je třeba počítat se skutečností, že s budoucím růstem databáze bude přibývat počet atributů, které budou známy pouze pro některé entity, nebo budou zadávány vícenásobné hodnoty pro jeden atribut. Pro tyto situace by byl výhodnější přístup EAV, který uchovává data jako řádky v datových tabulkách. Návrh aplikace umožňuje budoucí rozšíření na smíšený přístup k ukládání hodnot atributů (do databázových sloupců nebo datových tabulek EAV).

EAV přístup je vhodný také v situacích, kdy počet atributů (a tedy i databázových sloupců) překročí obvyklé meze. Například databázový systém MySQL a jeho datové úložiště InnoDB je schopno zpracovat maximálně 1 000 databázových sloupců (tento počet se v závislosti na objemu ukládaných dat může snížit) [3]. Pro databázový systém PostgreSQL je maximální počet sloupců pro databázovou tabulku 250–1 600 (v závislosti na datových typech sloupců) [4].

Pro zjednodušení případného rozšíření aplikace o uchovávání dat v datových tabulkách EAV (ne pouze v dynamicky vytvářených entitních tabulkách) je součástí metadatových tabulek také příznak `is_static`, který určuje datové úložiště atributu. Pro zvolený dynamický přístup, kdy jsou vytvářeny sloupce pro každý atribut, nabývá příznak hodnoty `true`. V případě rozšíření systému o možnost ukládání dat do datových tabulek EAV bude atribut s hodnotou příznaku `is_static=false` značit, že hodnota tohoto atributu je uchovávána v datové tabulce EAV.

6.3 Návrh a charakteristika aplikace

Aplikace bude implementována ve skriptovacím jazyce PHP verze 5, který umožňuje plně využít principy objektově orientovaného programování. Pro ukládání dat bude použit databázový systém PostgreSQL a nebo MySQL. Výběr použitého databázového systému bude záviset na počátečním nastavení při instalaci aplikace.

Za účelem urychlení vývoje a také zpřehlednění zdrojového kódu aplikace bude využit Zend Framework, který nabízí velké množství knihoven poskytujících řešení různých problémů, umožňuje vytvoření aplikace s využitím architektury MVC a nebo definuje sadu doporučených pravidel pro tvorbu zdrojového kódu. Podrobnější informace o Zend Frameworku a technologiích vhodných pro realizaci systému jsou podány v kapitole 7.

Aplikaci tvoří dvě části, a to:

- **administrace**, která bude sloužit přihlášeným uživatelům především k editaci dat uložených v databázi nebo ke specifikaci podoby těchto dat, a dále
- **veřejná část** určená pro zobrazování uložených dat, případně jejich vyhledávání.

6.3.1 Administrační část

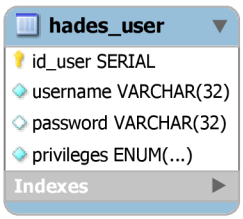
Do administrační části aplikace budou mít přístup pouze oprávnění uživatelé. Kontrola oprávnění probíhá na základě znalosti uživatelského jména a hesla. Dostupné uživatelské účty budou uchovávány v databázi, a to v databázové tabulce `hades_user`, která je znázorněna na obrázku 6.4.

Z bezpečnostních důvodů budou hesla v databázi uchována v nečitelné podobě. Pro zabezpečení hesel je využit alogitmus `MD5`. Protože je tento algoritmus jednocestný (lze

ho využít pouze pro převod hesla do nečitelné podoby, a nikoliv zpět), je třeba při kontrole správnosti hesla na něj nejprve tento algoritmus aplikovat a výsledek (hash) porovnat s hodnotou uloženou v databázi.

Přístup k administračním funkcím systému je omezen také úrovní oprávnění přihlášeného uživatele. Za účelem řízení přístupu k funkcím administrace jsou definovány tři úrovně oprávnění:

- **základní oprávnění**, které poskytuje základní funkce systému (především modifikace hodnot entit a vytváření nových entit),
- **vedoucí**, které umožňuje provést akce základního oprávnění a navíc je rozšířené o možnosti editace typů entit a jejich atributů (metadat) a dále o možnost exportu a importu entitních dat,
- **administrátorské**, pro které jsou dostupné veškeré funkce (funkce nižších oprávnění a možnost definovat či upravovat uživatelské účty a práva uživatelů).



Obrázek 6.4: Databázová tabulka pro uložení uživatelských účtů

6.3.2 Veřejná část

Veřejná část aplikace bude tvořena jednoduchým portálem, který bude umožňovat základní vyhledávání uložených entit a zobrazení podrobností o nich.

Přístup k portálu nebude omezen a bude dostupný pro všechny uživatele. Na titulní straně portálu budou obsaženy informace o aplikaci a dále bude umožněno jednoduché prohledávání databáze.

Jak již bylo řečeno, při zobrazování dat entity budou využity skupiny atributů, které zajistí sloučení atributů entity do pojmenovaných skupin, čímž dojde k značnému zpřehlednění informací o prezentovaných entitách. Formát prezentovaných hodnot atributů entity je dán využitým modelem atributu. Systém atributů byl popsán v kapitole 6.2.3.

6.3.3 Grafické uživatelské rozhraní

Protože systém budou využívat především proteinoví inženýři, kteří nemají čas zabývat se složitostí systému, je třeba, aby grafické uživatelské rozhraní aplikace bylo co nejvíce jednotné a práce s ním byla pokud možno intuitivní. V opačném případě by se dalo předpokládat, že bude nadále využíván program Microsoft Excel, který umožňuje jednoduchou editaci dat, přestože následná správa těchto dat je obtížnější.

Jelikož proteinovým inženýrům způsob editace dat v programu Microsoft Excel vyhovoval, bude vhodné tento princip editace dat zachovat. Aplikace bude tedy umožňovat editaci dat ve formě tabulky, kde jednotlivé sloupce tabulky budou představovat atributy (vlastnosti) typu entity a jednotlivé řádky vlastní entity a jejich hodnoty.

6.4 Mapování datových modelů do databáze

Ještě před zahájením fáze implementace vlastního systému bylo třeba přistoupit k určení způsobu ukládání dat do databáze. Je navržen způsob mapování datových modelů systému (tedy objektů) do databáze (databázových tabulek) a je zpětně umožněno získání těchto objektů z databáze.

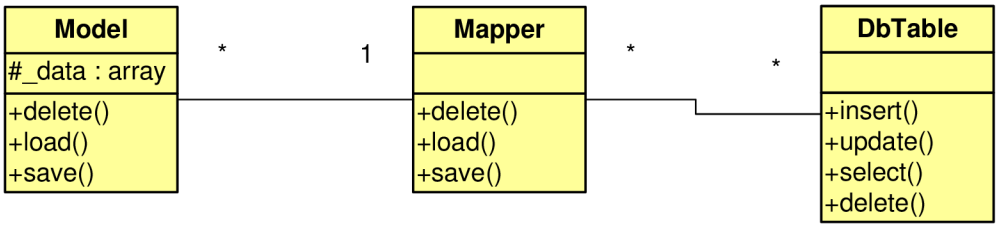
Cílem je, aby datový model poskytoval různé operace, které budou využívat databázi, přitom aby realizace těchto operací probíhala mimo třídu modelu. Datový model tak je zcela nezávislý na použitém databázovém systému a databázových objektech.

Za tímto účelem je využita struktura *Model–Mapper–DbTable*, která vychází z doporučení vývojářů Zend Frameworku [2] a která zajišťuje požadované oddělení datového modelu od databázových objektů.

Aplikační třída `Model` konceptu *Model–Mapper–DbTable* představuje model reprezentující data, se kterými pracuje aplikace. Přestože data modelu mohou být ukládána do databáze, je třída `Model` zcela nezávislá na použitém databázovém systému a k databázi nepřistupuje tato třída přímo.

Za účelem poskytnutí rozhraní pro přístup k databázovým objektům jsou vytvořeny třídy `DbTable`, které reprezentují databázové tabulky a nabízejí operace, jež je možné s nimi provádět.

Jelikož se předpokládá, že mohou nastat situace, kdy se bude datový model mapovat do více databázových tabulek, je zavedena aplikační třída `Mapper`, která realizuje mapování datového modelu na databázové tabulky (objekty tříd `DbTable`), zprostředkovává přístup k databázi a implementuje operace nad databází pro třídu modelu. Diagram tříd *Model–Mapper–DbTable* je znázorněn na obrázku 6.5.



Obrázek 6.5: Mapování datového modelu do databáze

Kapitola 7

Použité technologie

Na základě požadavků a návrhu systému byly zvoleny technologie, které jsou pro implementaci aplikace nejvhodnější, případně které fázi implementace zjednoduší či urychlí. Tato kapitola popisuje technologie využité při realizaci aplikace.

Protože většina technologií je čtenářům pravděpodobně známá, zaměřím se pouze na části, které jsou pro práci důležité a které nemusí být čtenáři známy. Kapitola je zaměřena především na druhy úložišť databázového systému MySQL, databázový systém PostgreSQL, popis Zend Frameworku, jeho porovnání s některými dalšími PHP frameworky a formát Open Office XML používaný pro uchovávání dokumentů. Nakonec jsou stručně charakterizovány další použité technologie.

7.1 MySQL

MySQL je relační systém řízení báze dat s otevřeným zdrojovým kódem, který je oblíbený u velkého množství uživatelů. Hlavní výhodu systému MySQL představuje jeho rychlost [18]. Přehled vlastností MySQL je možné nalézt v literatuře [18], [24] nebo na oficiálních stránkách http://www.mysql.com/.

Datová úložiště MySQL

MySQL nabízí několik typů databázových úložišť (tzv. *engine*). Každý typ má své specifické vlastnosti a chování, které ovlivňují především výkonnost databáze. Proto je třeba se při tvorbě databáze zamyslet nad budoucím využitím jednotlivých tabulek a zvolit správný druh úložiště.

- **MyISAM** představuje velmi rychlé úložiště, používané v případech, kdy není třeba zajistit integritu transakcí [24]. Díky možnosti komprese dat dochází k úspoře diskového prostoru. Výhodou je také možnost fulltextového vyhledávání pro sloupce datového typu `CHAR`, `VARCHAR` nebo `TEXT`. Většinou je MyISAM nastaven jako výchozí úložiště pro vytváření nových tabulek.
- **InnoDB** úložiště se využívá především kvůli možnosti transakčního zpracování, což rychlejší MyISAM nenabízí. InnoDB nabízí také zamykání jednotlivých řádků tabulek nebo definici cizích klíčů. InnoDB se využívá v případech, kdy je třeba vykonávat kritické transakce, které provádějí změny ve více tabulkách najednou, nebo v případech, kdy jsou ukládány velmi rozsáhlé soubory dat.

- **MEMORY** úložiště využívá tabulek uložených v paměti, což přináší výrazné zvýšení rychlosti přístupu k datům. Nevýhodou je, že data uložená v tabulkách typu MEMORY jsou dostupná pouze po dobu běhu MySQL serveru. Tabulky typu MEMORY jsou proto používány především k meziprocesové komunikaci [24].
- **ARCHIVE** úložiště je určeno k uchování velmi objemných dat, která nejsou často aktualizována. Tato data jsou komprimována, čímž dochází k úspoře diskového místa. Vložená data již není možné měnit (pro tento druh úložiště nejsou dostupné SQL příkazy `UPDATE` a `DELETE`), lze pouze přidávat další řádky (`INSERT`). Protože tabulka není indexována, dochází při výběru dat prostřednictvím SQL příkazu `SELECT` k sekvenčnímu prohledávání celé tabulky, což je velice náročné [24].

7.2 PostgreSQL

PostgreSQL je objektově-relační databázový systém s volně dostupným zdrojovým kódem, jež podle B. Momjiana [19] mezi nekomerčními databázovými systémy vyniká především svou podporou transakcí, složenými datovými typy a také spolehlivostí. Systém podporuje velkou část SQL standardu a nabízí spoustu moderních rysů, jako například komplexní dotazy, cizí klíče, triggery, pohledy, transakční integritu a podobně [4]. Podrobnější informace o databázovém systému PostgreSQL lze nalézt v literatuře [19], [4] nebo na oficiálních stránkách http://www.postgresql.org/

Fulltextové vyhledávání

Stejně jako některé ostatní databázové systémy nabízí PostgreSQL operátory `LIKE`, případně `ILIKE` k vyhledávání hodnot v textu. Ovšem vyhledávání na základě využití regulárních výrazů neposkytuje dostatečnou jazykovou podporu (například není možné jednoduše použít odvozená slova), nelze řadit výsledky na základě relevance a takovéto vyhledávání je také pomalejší, protože nevyužívá indexů [4].

Podle dokumentace databázového systému PostgreSQL [4] je pro fulltextové vyhledávání třeba nejprve dokument předzpracovat a vytvořit index pro pozdější využití při vyhledávání. Fáze předzpracování představuje rozčlenění dokumentu na jednotlivé části, vytvoření lexémů[1] a uložení předzpracovaného dokumentu optimalizovaného pro vyhledávání.

Pro ukládání předzpracovaných dokumentů je poskytován datový typ `tsvector` a typ `tsquery` pro reprezentaci zpracovaných dotazů. Pro fulltextové vyhledávání hraje v PostgreSQL důležitou roli operátor shody `@@`, jehož návratová hodnota představuje, zda daný `tsvector` (dokument) odpovídá danému `tsquery` (dotazu). Fulltextové vyhledávání může být zrychleno využitím indexů. Podrobnější informace o fulltextovém vyhledávání a definici indexů naleznete v dokumentaci [4].

Podpora transakcí

Na základě informací uvedených v [5] je hlavní výhodou PostgreSQL podpora transakcí i pro DDL (*Data Definition Language*). Tento návrh umožňuje zrušení (*rollback*) rozsáhlých změn DDL, jako je například tvorba databázových tabulek nebo definice jejich sloupců. Pouze zotavení se z tvorby databáze nebo prostoru pro fyzické uložení tabulek není možné, ostatní operace s katalogem jsou odvolatelné [5].

[1] lexém: normalizované slovo umožňující vyhledávání různých variant téhož slova

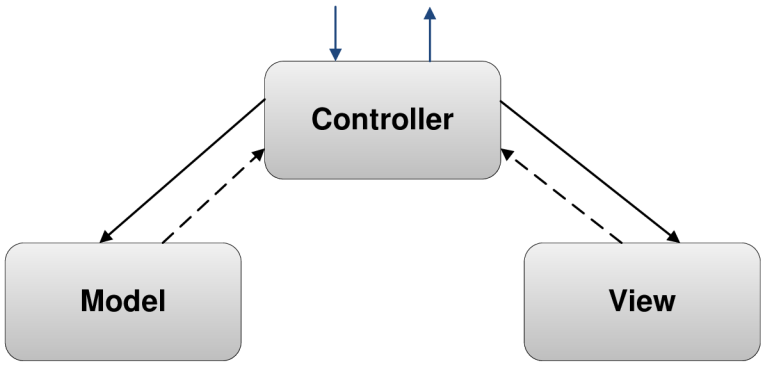
Databázový systém MySQL doposud (ve verzi 5.1) nenabízí podporu transakčního zpracování DDL. Dokonce pro datové úložiště MyISAM není transakční zpracování nabízeno vůbec, pro InnoDB je transakční zpracování DML podoprováno, ale při zpracování DDL dojde k automatickému potvrzování provedených změn (tzv. *auto-commit*) a odvolání změn tedy není možné [5].

Podrobnější informace o problému transakčního zpracování DDL a chování různých databázových systémů naleznete v [5].

7.3 Model–View–Controller (MVC)

Model–View–Controller (MVC) představuje návrhový vzor, který definuje architekturu vytvářeného systému. Principem je rozdělení aplikační a prezentační logiky vytvářené aplikace [20]. Systém je rozdělen do tří částí, viz 7.1:

- **datový model (Model)**, který reprezentuje uchovávané informace a manipulaci s nimi (tzv. *business logic*)
- **uživatelské rozhraní (View)**, které prezentuje data modelu do podoby vhodné k zobrazení uživateli
- **řídící logika aplikace (Controller)**, která zajišťuje komunikaci mezi modelem a uživatelským rozhraním, reaguje na vznikající události a zpracovává uživatelský vstup



Obrázek 7.1: Architektura Model–View–Controller

Aplikace navržená s využitím principů MVC tak rozděluje zdrojový kód do více souborů. Každý soubor má svou specifickou funkci, což zjednodušuje orientaci v kódu, případně budoucí změny kódu.

7.4 Zend Framework

Zend Framework představuje *framework*[2] s otevřeným zdrojovým kódem, který je určen pro vývoj webových aplikací v programovacím jazyce PHP. Důraz je kladen na jednoduchost vývoje, správy a rozšiřitelnosti vyvíjené aplikace.

Vlastnosti Zend Frameworku

Zend Framework je implementován v PHP 5 a veškerý jeho kód je objektově orientovaný. Nabídnuta je také velice kvalitní dokumentace [2] nebo diskuzní fóra s poměrně rozšířenou a aktivní uživatelskou základnou. Na vývoji a dokumentaci se podílejí stovky vývojářů, přesto je podle informací uvedených v dokumentaci [2] kód před uveřejněním důkladně testován a prověřován.

Zend Framework nabízí celou řadu knihoven a rozšíření, která poskytují vývojářům řešení často opakujících se problémů. Jedná se především o implementaci MVC komponenty, která zajišťuje rozdělení aplikační a prezentační logiky aplikace na základě principů MVC.

Nabídnuty jsou knihovny pro práci s databází. Podporováno je několik databázových systémů: IBM DB2, MySQL, Microsoft SQL Server, Oracle, PostgreSQL, SQLite [2]. Přitom pro přechod na jiný databázový systém postačí změnit konfigurační soubor s nastavením databáze, bez nutnosti dalších úprav v kódu aplikace (při dodržení zásad pro zápis SQL dotazů a využití knihovny `Zend_Db`).

Dále jsou dostupné knihovny pro autentizaci, lokalizační a jazykové komponenty, komponenty pro komunikaci prostřednictvím HTTP protokolu, webové služby, správu vyrovnávací paměti, správu sezení (session), formuláře a jejich validaci a další komponenty, které jsou rozděleny do modulů [2].

Zend Framework je značně modulární, což umožňuje využít pouze ty komponenty, které jsou pro danou aplikaci potřeba. Díky modularitě a snaze o minimální vzájemnou závislost jednotlivých modulů je možné naučit se používat Zend Framework v poměrně krátké době. Není třeba znát, jak je daná komponenta implementována, stačí vědět, co umožňuje. Díky kvalitní a rozsáhlé uživatelské dokumentaci je možné tyto informace rychle a přesně zjistit.

Pro Zend Framework jsou také nabídnuta pravidla pro psaní zdrojového kódu (způsob odsazování, pojmenování proměnných, metod, funkcí, způsob zápisu řídicích struktur, maximální délka řádků a další doporučené konvence). Doporučená pravidla vnášejí do kódu jednotnost a přehlednost, což umožní zejména rychlejší orientaci v kódu. Popisovaná pravidla jsou využita také pro veškerý zdrojový kód Zend Frameworku. Přehled pravidel lze nalézt v referenční příručce Zend Frameworku [2], v sekci „*Zend Framework Coding Standard for PHP*“.

Princip zpracování HTTP požadavku

Na obrázku 7.2 je znázorněn způsob zpracování HTTP požadavku na zobrazení dané webové strany, určenou akcí kontroleru [9].

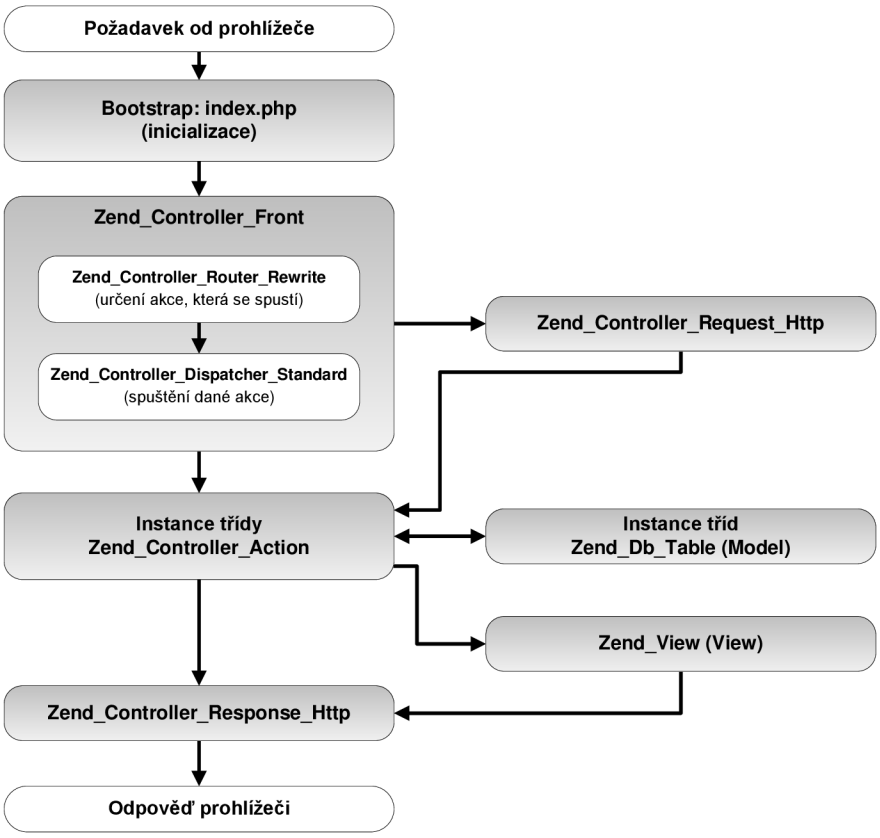
Veškeré požadavky se směřují na skript `index.php`, který zajistí vytvoření aplikace Zend Frameworku a provede její inicializaci. Na základě požadavku (vyžádané URL adresy) dojde k určení kontroleru a akce, která se má vykonat. Za tímto účelem je vytvořen `Zend_Controller_Front`, který centralizuje všechny požadavky do jediného souboru.

[2] framework: soubor knihoven a rozhraní, které nabízí řešení často opakujících se problémů vznikajících při vývoji aplikací

Proces směrování (tzv. *routing*) představuje odvození kontroleru a jeho akce k uspokojení požadavku. To je realizováno prostřednictvím tříd `Zend_Controller_Router_Rewrite` zajišťujících rozčlenění URL na části, které jsou následně rozděleny do parametrů.

Proces zpracování je realizován třídou `Zend_Controller_Dispatcher_Standard`, která zprostředkovává volání požadované metody ve správné třídě (na základě parametrů — kontroler, akce, případně i modul).

Konktétní třída kontroleru odvozená od abstraktní třídy `Zend_Controller_Action` obsahuje metody implementovaných akcí kontroleru a realizuje požadovanou funkcionalitu. Manipuluje přitom se třídami modelu a předává data do view, které je zpracováno a výstup předán prohlížeči.



Obrázek 7.2: Zpracování požadavku na zobrazení strany v Zend Frameworku

7.4.1 Srovnání s ostatními PHP frameworky

Před rozhodnutím použít v projektu Zend Framework jsem přistoupil k průzkumu dostupných PHP frameworků, z nichž za zmínku stojí především Symfony a nepříliš známý framework Yii.

Symfony

PHP framework Symfony je vytvořený za účelem zrychlení vývoje webových aplikací. Jeho hlavním přínosem je implementace MVC a velké množství knihoven ulehčující práci při

řešení často opakujících se problémů.

Symfony je implementováno v PHP 5 a plně tak využívá možnosti objektového programování. Podporovány jsou databázové systémy MySQL, PostgreSQL, SQLite, Oracle, Microsoft SQL Server. Pro objektově relační mapování jsou využity nástroje *Propel*[3] nebo *Doctrine*[4]. [6]

Důležitou roli při vývoji aplikace s využitím frameworku Symfony hrají konfigurační soubory, které jsou zapisovány především ve formátu *YAML*[5]. Pomocí konfiguračních souborů dochází k nastavení projektu, aplikace, modulů, databázového spojení nebo struktury databáze.

Manipulace s projektem probíhá spuštěním skriptu z příkazového řádku. Umožněno je velké množství operací, jako například vytvoření projektu, modulu, konfigurace databáze, vytvoření databáze na základě konfiguračního souboru nebo také automatické generování kódu aplikace na základě struktury databáze.

Stejně jako Zend Framework nabízí Symfony kvalitní uživatelskou dokumentaci. Počet uživatelů Symfony je taktéž značný. Na základě informací získaných na [6] není počet vývojářů tak velký jako u Zend Frameworku a aktualizace verzí není tak častá , přesto si myslím, že je dostatečná. Za hlavní nevýhodu (pro nové uživatele Symfony) považuji nutnost dodržovat velké množství pravidel a velké množství konfiguračních souborů ovlivňujících chování aplikace, což je pro začínajícího uživatele nepřehledné a zažití všech pravidel prodlužuje dobu potřebnou k naučení se pracovat s frameworkem. Na rozdíl od Zend Frameworku, kde jsou pravidla definována pouze jako doporučená, je třeba pravidla Symfony striktně dodržovat. Určitou výhodou, zejména pro menší a rychle vyvíjené projekty, může být možnost automatického generování kódu aplikace, včetně administrační části aplikace.

Yii

PHP framework Yii (čte se [ji:]) vznikl za účelem zjednodušení a urychlení vývoje rozsáhlých webových aplikací. Hlavní důraz klade na maximální znovupoužitelnost kódu. [28] Stejně jako předchozí frameworky je Yii implementováno v PHP 5 a představuje čistě objektově orientovaný PHP framework.

Za vznikem frameworku stojí Qiang Xue, který se podílel také na vývoji PHP frameworku PRADO, z něhož je převzato velké množství myšlenek (především rozdělení aplikace do komponent a událostmi řízené programování). Na vývoji se podílí celkem šest vývojářů, z nichž každý má předchozí zkušenosti s vývojem frameworků. [28]

Funkčnost, kterou framework nabízí, je rozdělena do komponent (komponenty pro manipulaci s databází a formuláři, zpracování HTTP požadavků, cache, autorizaci, správu souborů, zpracování neošetřených chyb a další). [28] Stejně jako u Symfony je manipulace s projektem prováděna pomocí spuštění skriptu v příkazovém řádku operačního systému. Umožněno je také generování zdrojového kódu aplikace na základě struktury databáze.

Za hlavní nevýhodu považuji fakt, že framework Yii je příliš nový a neznámý, na vývoji se podílí pouze několik vývojářů, a tak je budoucnost dalšího vývoje nejistá. S ohledem na velikost projektu je dokumentace překvapivě rozsáhlá a podrobná. Svým rozsahem Yii nedosahuje kvalit robustních frameworků Symfony nebo Zend Frameworku, přesto nabízí kvalitní podporu při vývoji webových aplikací. Díky své jednoduchosti a kvalitě zpracování se doba potřebná k naučení se frameworku minimalizuje.

[3] http://propel.phpdb.org/
[4] http://www.doctrine-project.org/
[5] http://yaml.org/

7.5 Microsoft Excel a Office Open XML

Microsoft Excel je tabulkový editor určený k vytváření a formátování tabulek a analýze informací v nich. Nabízí velké množství funkcí (jako například vytváření grafů z dat v tabulce, třídění a filtrování dat, případně sdílení dat s ostatními uživateli). Microsoft Excel je produkt společnosti *Microsoft* a prodává se především jako součást kancelářského balíku *Microsoft Office*[6].

Office Open XML

Office Open XML představuje otevřený (volně dostupný) formát pro ukládání dokumentů kancelářských aplikací (textových a tabulkových dokumentů, prezentací). Formát byl navržen společností Microsoft a organizací *Ecma International*[7] přijat jako standard ukládání dokumentů [22].

Poprvé byl formát použit v balíku kancelářských programů Microsoft Office 2007. Office Open XML tak nahradil původní uzavřený formát, který byl ukládán v binární podobě. Protože šlo podle informací, které uvádí T. Ngo v přehledu formátu Office Open XML [22], o přímou serializaci[8] datových struktur aplikací Microsoft Office a formáty těchto struktur nebyly veřejně dostupné, byl obsah souboru s dokumentem nečitelný a implementace zpracovávající takovýto soubor příliš náročná, případně nespolehlivá. Díky otevřenosti formátu již existuje velké množství knihoven pro práci s ním, a to v různých programovacích jazycích.

Office Open XML definuje formáty pro textové dokumenty, prezentace a tabulkové dokumenty. Každý typ dokumentu je tvořen pomocí značkovacího jazyka:

- **WordprocessingML** určeného k uchování textových dokumentů
- **PresentationML** určeného k uchování prezentací
- **SpreadsheetML** určeného k uchování dokumentu tabulkového editoru

Dokument ve formátu Office Open XML představuje *ZIP*[9] archiv, který obsahuje soubory reprezentující dokument (XML, metadata, obrázky). ZIP archiv obsahuje soubor `[Content_Types].xml`, který udržuje obsah archivu (typ a umístění jednotlivých souborů tvořících dokument). [22]

Na nejvyšší úrovni je dokument tabulkového editoru tvořen souborem `workbook.xml`, který obsahuje definice jednotlivých listů dokumentu.

```
<sheets>
 <sheet name="Sheet1" sheetId="1" r:id="rId1" />
 <sheet name="Sheet2" sheetId="2" r:id="rId2" />
</sheets>
```

Jednotlivé vztahy částí dokumentu jsou ukládány do souboru `_rels/*.rels`. Ukázka souboru `_rels/workbook.xml.rels`:

```
<Relationships
 xmlns="http://schemas.openxmlformats.org/package/2006/relationships">
```

[6] http://www.microsoft.com/cze/office/default.mspx
[7] http://www.ecma-international.org/
[8] serializace: převedení objektu nebo datové struktury do podoby, která může být uložena a později převedena do původní podoby
[9] ZIP: souborový formát pro kompresi a archivaci dat

```
 <Relationship Id="rId1" Type="worksheet" Target="worksheets/sheet1.xml" />
 <Relationship Id="rId2" Type="worksheet" Target="worksheets/sheet2.xml" />
 <Relationship Id="rId6" Type="sharedStrings" Target="sharedStrings.xml" />
</Relationships>
```

Vlastní list tabulky je tvořen kořenovým elementem `worksheet`, který obsahuje element `sheetData` s definicemi jednotlivých řádků `row`, sloupců `c` a jejich hodnot `v`.

```
<worksheet xml:space="preserve"
 xmlns="http://schemas.microsoft.com/office/excel/2006/2">
 <sheetData>
  <row r="1" spans="1:2" customFormat="1">
   <c r="A1" t="s">
    <v>0</v>
   </c>
   <c r="B1">
    <v>111</v>
   </c>
  </row>
 </sheetData>
</worksheet>
```

Podrobnou specifikaci Office Open XML, včetně SpreadsheetML pro definici dokumentů tabulkových editorů je možné nalézt na [22].

7.6 Další technologie

Další použité technologie jsou natolik využívané a známé, že je podán pouze stručný popis těchto technologií. V případě zájmu lze podrobnější informace vyhledat v odborné literatuře.

PHP je skriptovací programovací jazyk s otevřeným zdrojovým kódem, který je určen pro tvorbu dynamických webových stránek. Jazyk PHP je nezávislý na použité platformě (jak na hardwaru, tak i na operačním systému). Syntaxí se podobá ostatním programovacím jazykům (jako například C, Perl nebo Java). Více v literatuře [13] nebo na webové adrese http://php.net/.

Apache HTTP Server představuje webový server s otevřeným zdrojovým kódem, jehož cílem je nabídnout bezpečný, efektivní a rozšiřitelný server poskytující HTTP služby. Rozšiřitelnosti je dosaženo díky možnosti využití různých modulů (například modulů pro podporu programovacích jazyků PHP, Python, Perl a podobně). Více na webové adrese http://httpd.apache.org/.

JavaScript je programovací jazyk, který je interpretován na straně klienta a je tak závislý na webovém prohlížeči klienta. JavaScript rozšiřuje možnosti jazyka HTML a umožňuje úkony, které by s využitím statického HTML nebyly možné (například validaci údajů zadaných do formuláře, manipulaci s objektovým modelem dokumentu — DOM nebo animace a různé efekty). Podrobnější informace naleznete v literatuře [29].

XML (*eXtensible Markup Language*) představuje rozšiřitelný značkovací jazyk, který je určen zejména k uchování nebo výměně dat mezi aplikacemi. Jeho rozšiřitelnost spočívá v možnosti libovolného vytváření nových značek.

Jazyk nedefinuje sémantiku značek, ale pouze jejich syntaxi. Sémantika je dána až interpretací jednotlivými aplikacemi. Správně strukturovaný XML dokument musí splňovat definovaná pravidla, což ulehčuje strojové zpracování dokumentu. Důležité je zdůraznit, že

XML je tzv. *case sensitive* (rozlišuje velikost písmen). [11]

XHTML (*eXtensible HyperText Markup Language*) je značkovací jazyk určený k tvorbě webových hypertextových dokumentů. XHTML vychází z jazyka HTML (HyperText Markup Language), který dává značkám jejich sémantiku, a řídí se principy XML. Díky tomu je zpracování dokumentu prohlížečem jednodušší a přitom je při dodržení některých doporučení zpětně kompatibilní s HTML. Více v literatuře [11].

AJAX (*Asynchronous JavaScript And XML*) představuje spojení technologií JavaScript a XML za účelem změny části objektového modelu dokumentu (DOM), a to bez nutnosti nového načtení celé webové stránky. Díky možnosti modifikace pouze části stránky může dojít ke snížení zátěže serveru, zmenšení objemu přenášených dat a tedy k celkovému nárůstu výkonnosti aplikace. Pro asynchronní komunikaci s webovým serverem se využívá objektu `XMLHttpRequest`, přičemž je uživateli umožněno dále pracovat s uživatelským rozhraním aplikace.

jQuery představuje JavaScriptovou knihovnu s otevřeným zdrojovým kódem, která poskytuje interakci mezi JavaScriptem a HTML, přitom klade důraz na funkčnost v různých prohlížečích [7]. Knihovna přináší zlehčení práce s objektovým modelem dokumentu (DOM), obsluhu událostí, rozšiřující elementy uživatelského rozhraní a nebo efekty. Díky své nenáročnosti vzniká velké množství rozšíření, které využívají vlastností jQuery [7].

Kapitola 8

Popis implementace systému

Kapitola si klade za cíl popsat nejdůležitější implementační detaily a problémy, které během implementace aplikace nastaly. Implementace vychází z návrhu prezentovaného v kapitole 6, a proto jsou popsány pouze její detaily s odkazem na jednotlivé části návrhu.

Nejprve je předložen způsob realizace aplikace, techniky využité pro minimalizaci opakování se zdrojového kódu, adresářová struktura aplikace a její rozdělení do modulů, princip poskytnutí podpory více databázových systémů, uživatelské rozhraní aplikace a dále způsob realizace systému entit a atributů, který představuje klíčovou část celého systému ukládání a prezentace dat. Se systémem entit také úzce souvisí import a export entitních dat. Na závěr kapitoly jsou podány podrobnosti o způsobu testování aplikace a jejím nasazení do provozu.

Jelikož byly při implementaci ve velké míře využity komponenty Zend Frameworku, představují všechny třídy, jejichž název začíná předponou „`Zend_`“, třídy, které jsou součástí Zend Frameworku. Ostatní popisované třídy představují aplikační třídy (pokud není uvedeno jinak). V příloze C je potom uveden přehled implementovaných modulů a použitých komponent.

8.1 Struktura aplikace

Aplikace je implementována ve skriptovacím jazyce PHP a využívá PHP frameworku Zend Framework, který poskytuje velké množství komponent, jež napomáhají k zjednodušení a urychlení vývoje webových aplikací. Základní charakteristika Zend Frameworku byla podána v kapitole 7.4.

Vlastní aplikaci reprezentuje třída `Application`, která zajišťuje vytvoření a nastavení `Zend_Application`. Hlavní roli zde hraje statická metoda *`run()`* třídy `Application`, prostřednictvím které dochází ke spuštění aplikace.

Pro spuštění aplikace se využívá PHP skript umístěný v souboru `index.php`, na který jsou směrovány veškeré požadavky na zobrazení stran. Způsob zpracování požadavku byl popsán v kapitole 7.4.

8.1.1 Prostředí pro testování a ladění aplikace

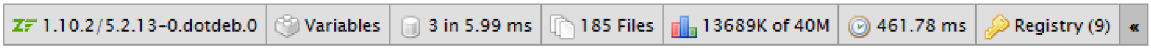
Prostřednictvím konfiguračního souboru je možno nastavit chování systému pro různá prostředí (produkční prostředí určené pro ostrý provoz na produkčním serveru a nebo testovací prostředí určené k testování na serveru vyhrazeném pro vývoj). Jedná se o konfigurační soubor `application.ini`, který bude podrobněji popsán v kapitole 8.3.

V produkčním prostředí jsou potlačeny výpisy chybových hlášení uživateli, nejsou zobrazována varování, nevyužívají se nástroje pro sledování výkonu nebo vykonaných SQL dotazů a podobně.

Protože pro vývojáře může být vhodné těchto nástrojů využít i v produkčním prostředí (například pro ověření funkcionality nebo při řešení konktétního problému), a to bez ovlivnění chování aplikace pro běžné uživatele, byl navržen způsob, který určuje, zda aplikace běží v běžném uživatelském prostředí nebo v ladicím. Princip spočívá v nastavení příznaku `_isDebugMode` třídy `Application`, který ovlivňuje chování spuštěné aplikace.

Pro spuštění aplikace v ladicím prostředí je využit skript `index_dev.php`, který zajistí nastavení statické vlastnosti `_isDebugMode` třídy `Application` na hodnotu `true` a následně provede spuštění aplikace voláním statické metody *`run()`*.

Přínosem ladicího prostředí je především úplné zobrazování chyb a varování, včetně jejich popisu a dále využití knihovny `ZFDebug` pro zobrazení lišty s informacemi o běhu aplikace Zend Frameworku. Tato lišta nabízí zobrazení verze Zend Frameworku, verze PHP, hodnot některých proměnných Zend Frameworku, vykonaných databázových dotazů včetně jejich doby trvání, přehledu načtených PHP souborů, paměťové náročnosti aplikace, doby trvání od zaslání požadavku na požadovanou stranu do získání jejího obsahu a nebo hodnot uložených v `Zend_Registry`. Podoba lišty je znázorněna na obrázku 8.1.



Obrázek 8.1: Lišta `ZFDebug` s informacemi o běhu aplikace

8.1.2 Abstraktní třídy

Při vývoji aplikace je kladen důraz na co největší přehlednost kódu a zároveň také na minimalizaci opakování se kódu.

Aby nedocházelo k přebytečnému opakování se stejného kódu, čímž by se podstatně zhoršila udržovatelnost kódu, je společné chování jednotlivých částí systému implementováno v abstraktních třídách. Tento přístup umožňuje také jednoduché přidání další funkcionality, která je společná všem odvozeným třídám.

`Bootstrap`

Jedná se o abstraktní třídu vycházející ze třídy `Zend_Module_Bootstrap`, jejíž veškeré metody (jejichž název začíná `_init`) jsou volány na počátku spuštění aplikace.

Protože abstraktní třída `Bootstrap` tvoří základ ostatních modulových tříd `Bootstrap`, obsahuje pouze metody provádějící akce, které jsou společné pro všechny moduly. Například načtení konfigurace modulu, zpracování uživatelského menu daného modulu a podobně.

`Controller`

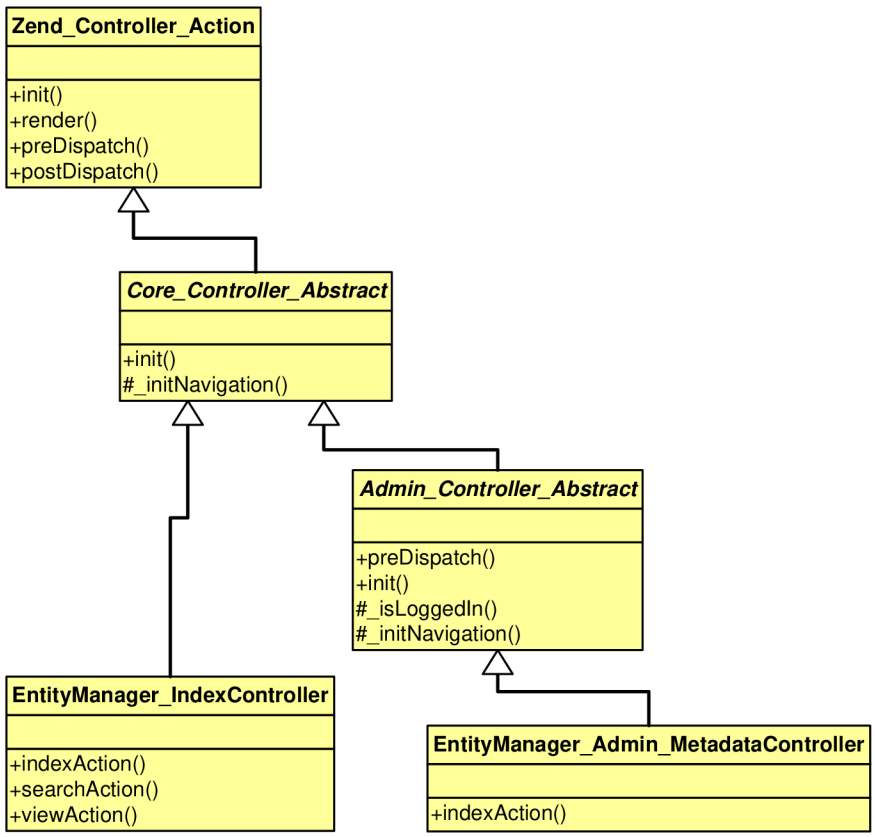
Řídicí logika aplikace, která je definována v kontrolerech, je pro každý kontroler odlišná, přesto mohou být některé akce všem kontrolerům společné. Jedná se především o akce spojené s inicializací kontroleru, například ověření oprávnění přístupu ke kontroleru, nastavení navigace pro daný kontroler nebo provedení dalších společných akcí.

Aplikační kontroler je definován v abstraktní třídě `Core_Controller_Abstract` a tvoří základ veškerých aplikačních kontrolerů, pro které je tak dostupná stejná funkcionalita.

Za účelem vytvoření kontroleru pro administrační část aplikace je definována abstraktní třída `Admin_Controller_Abstract`, která vychází z třídy `Core_Controller_Abstract` a dále rozšiřuje její funkčnost o automatickou kontrolu oprávnění uživatele k přístupu ke kontroleru. Tato kontrola je provedena ještě před zpracováním stránky — v metodě `preDispatch()`.

Protože třída `Admin_Controller_Abstract` představuje rodičovskou třídu pro všechny administrační kontrolery, nemusí v nich být již prováděna kontrola oprávnění uživatele. Také při změně způsobu ověření uživatele postačí úprava v abstraktní třídě a tato funkcionalita se na základě dědičnosti projeví i ve třídách odvozených.

Situace je znázorněna na diagramu tříd na obrázku 8.2, kde jsou zobrazeny abstraktní třídy kontrolerů a ukázka konkrétních aplikačních tříd kontrolerů.



Obrázek 8.2: Abstraktní třídy kontrolerů

`Model`

Také v datových modelech reprezentujících data lze nalézt společné rysy, které bylo výhodné implementovat v abstraktní třídě `Core_Model_Abstract`. Tato třída představuje základ jednotlivých datových modelů aplikace.

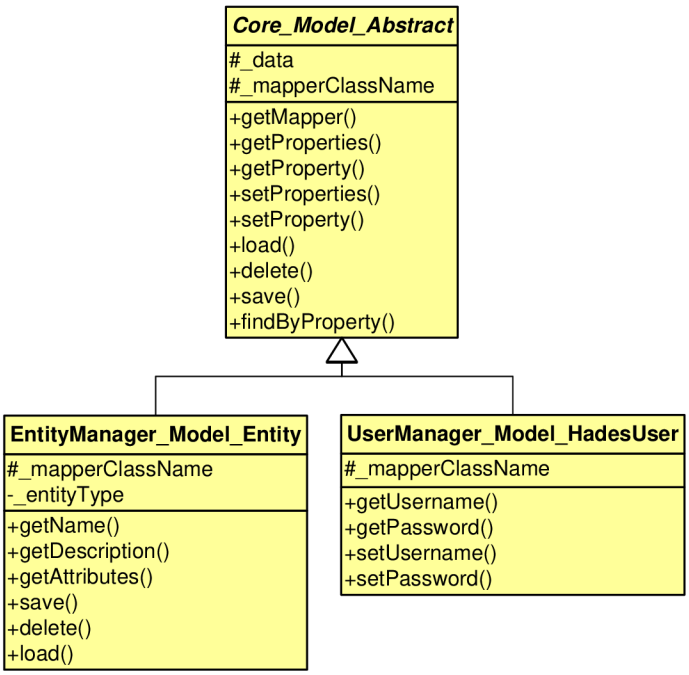
Modelům jsou společné například metody pro ukládání, mazání nebo získání hodnot na základě vlastností. Protože modely nepřistupují k databázi přímo, ale k přístupu využívají

třídy `Mapper`, mohou zůstat tyto metody v dceřinných třídách v nezměněné podobě. Způsob mapování *Model–Mapper–DbTable* je popsán v kapitole 6.4

Součástí modelu jsou data reprezentující model. Aby bylo možné k datům jednoduše přistupovat a zároveň aby měly různé modely podobné vlastnosti, ke kterým je možné přistupovat jednotným způsobem, nejsou tato data uložena jako vlastnosti třídy, ale jsou uložena do asociativního pole vlastnosti `$_data`, kde klíčem do tohoto pole je název vlastnosti datového modelu.

Názvy vlastností mohou odpovídat názvům sloupců databázové tabulky, do které je model mapován, a pokud jsou data modelu mapována pouze do jedné databázové tabulky, lze tyto hodnoty předat metodě `insert()` nebo `update()` třídy `Zend_Db_Table`, a to bez dalších úprav (pomineme-li validaci dat zadaných uživatelem).

Na obrázku 8.3 je znázorněna abstraktní třída `Core_Model_Abstract`, její vlastnosti a metody, které jsou společné pro všechny modely konceptu *Model–Mapper–DbTable*.



Obrázek 8.3: Abstraktní třída `Model`

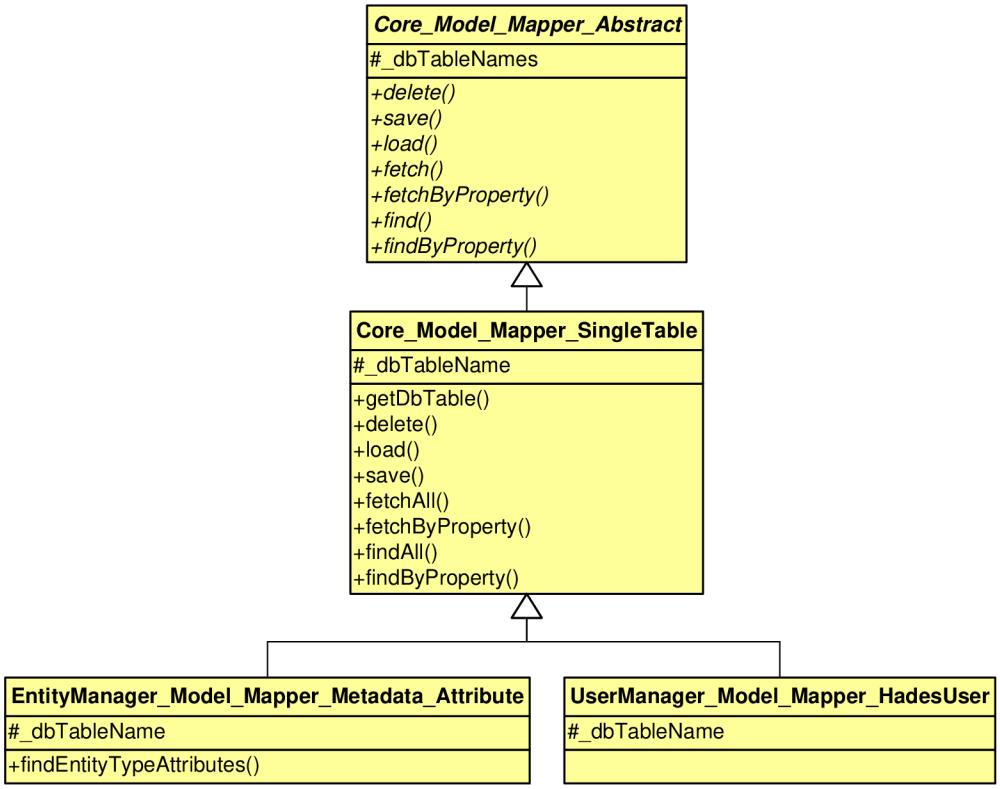
`Mapper`

Třída `Mapper` představuje prostředníka v přístupu datového modelu k rozhraní databáze. Způsob mapování modelu byl popsán v kapitole 6.4. Třída `Core_Model_Mapper_Abstract` obsahuje definice abstraktních metod, kterých mohou využívat třídy modelu a které musí být v odvozených třídách implementovány.

Pokud je celý datový model mapován pouze do jediné databázové tabulky se sloupci shodnými s vlastnostmi datového modelu, je značná část operací pro takovéto třídy `Mapper` stejná. V těchto případech lze využít společných metod pro ukládání dat modelu do databáze, případně jejich získávání, a to pouze na základě znalosti názvu mapované databázové tabulky.

Za účelem zjednodušení tříd `Mapper` a neopakování obdobného zdrojového kódu byla vytvořena třída `Core_Model_Mapper_SingleTable`, která implementuje abstraktní metody třídy `Core_Model_Mapper_Abstract` a na základě znalosti mapované databázové tabulky poskytuje požadované operace.

Abstraktní třída `Core_Model_Mapper_Abstract`, odvozená třída pro mapování modelu do jedné databázové tabulky `Core_Model_Mapper_SingleTable` a jejich dostupné vlastnosti a metody jsou znázorněny na obrázku 8.4.



Obrázek 8.4: Abstraktní třída `Mapper`

`DbTable`

Třída `Zend_Db_Table` poskytuje objektově orientované rozhraní pro přístup k databázovým tabulkám a nabízí metody pro většinu běžně prováděných operací nad databázovými tabulkami [2]. Podle dokumentace Zend Frameworku [2] implementuje třída `Zend_Db_Table` návrhový vzor *Table Data Gateway*[1] a *Row Data Gateway*[2].

Díky tomu, že třída `Zend_Db_Table` využívá `Zend_Db_Adapter`, je umožněno provádět operace s databázovými tabulkami jednotným způsobem, nezávisle na použitém databázovém systému.

[1] http://www.martinfowler.com/eaaCatalog/tableDataGateway.html
[2] http://www.martinfowler.com/eaaCatalog/rowDataGateway.html

8.1.3 Adresářová struktura aplikace

Aby byl systém do budoucna snadno rozšiřitelný a přehledný pro další vývojáře, byla stanovena pravidla pro pojmenování a umisťování souborů a tříd aplikace. Tato pravidla je třeba striktně dodržovat.

Dodržováním pravidel dochází nejen k zpřehlednění způsobu umístění jednotlivých zdrojových souborů aplikace, ale díky nim je možné využít některých výhod, které Zend Framework nabízí. Jedná se především o mapování názvů tříd do adresářové struktury.

Adresářová struktura aplikace je znázorněna na obrázku B.1 umístěného v příloze B.

Dynamické načítání tříd

Zend Framework definuje konvence pro pojmenování tříd a jejich umístění v souborech a určuje způsob odvození cesty k souboru s definicí třídy na základě názvu třídy. Díky tomu, že z názvu třídy lze odvodit název souboru, ve kterém je definována, je možné využít automatického načtení kódu třídy (tzv. *class autoload*).

Automatické načítání tříd dovoluje použití tříd, které ještě nebyly načteny (klasicky bývají třídy načítány prostřednictvím PHP příkazu `require_once`), a může tak značně snížit objem zpracovávaných souborů, protože jsou načteny pouze ty třídy, které jsou reálně využity [2].

Základním pravidlem pro pojmenování tříd je, že název třídy je tvořen na základě cesty k souboru s definicí této třídy. Přičemž každý znak „_“ v názvu třídy je nahrazen konstantou `DIRECTORY_SEPARATOR` (tedy znakem „/“ pro aplikaci spuštěnou v operačním systému UNIX). Část za posledním symbolem „_“ odpovídá názvu souboru (s přidáním přípony `.php`). Tedy například třída `Zend_Db_Table_Abstract` je definována v souboru s cestou `Zend/Db/Table/Abstract.php`.

Umístění souborů Model–View–Controller

Pro pojmenování tříd platí výše uvedená pravidla, avšak pro lepší orientaci v adresářové struktuře MVC aplikace jsou definována některá další pravidla. Jedná se především o drobné změny v mapování názvů tříd na adresářovou strukturu.

Datové modely jsou uchovávány ve složce `models`, přičemž v názvu třídy je uvedeno `Model` (např. třída `Model_User` je definována v souboru `models/User.php`).

Řídicí logika aplikace je definována v kontrolerech umístěných ve složce `controllers`. V názvu třídy je obsaženo slovo `Controller`, které je připojeno za vlastní název kontroleru (např. třída kontroleru `User` bude mít název `UserController` a její definice bude umístěna v souboru `controllers/UserController.php`).

V kontroleru jsou obsaženy metody představující jednotlivé akce, které je možno provést. Název metody se skládá z vlastního názvu a připojeného klíčového slova `Action` (např. `InfoAction()`).

Akce kontroleru je vyvolána na základě zaslání HTTP požadavku, jehož adresa je ve tvaru `controller/action`. Pro ukázkový příklad se jedná o adresu `user/info`, která s využitím kontroleru `UserController` vyvolá akci definovanou prostřednictvím metody `InfoAction()`.

Grafická uživatelská rozhraní aplikace jsou ukládána do složky `views`. V tomto adresáři mohou být dále podadresáře `scripts` určené pro definice vzhledu stran nebo jejich částí. Soubory vzhledu mají často název odvozený z názvu kontroleru a akce, která byla vyvolána (např. definice uživatelského rozhraní pro zobrazení informací o uživateli, jehož

obsluhu zajišťuje kontroler `UserController` s akcí `InfoAction`, bude definována v souboru `views/scripts/user/info.phtml`).

Ukázka adresářové struktury MVC aplikace je znázorněna na obrázku B.1.

8.1.4 Rozdělení aplikace do modulů

Za účelem vyšší přehlednosti a jednodušší rozšiřitelnosti nebo modifikace je aplikace rozčleněna na jednotlivé tematické moduly. Každý modul poskytuje požadovanou funkčnost, přičemž jeho závislost na ostatních modulech je minimalizována. Výjimku tvoří moduly jádra aplikace (`core` a `admin`), případně situace, kdy modul rozšiřuje funkčnost modulu, na kterém je závislý. Přehled implementovaných aplikačních modulů je uveden v příloze C.

Základ aplikace tvoří moduly `core` a `admin`, které jsou pro celkový chod aplikace nezbytné. Modul `core` zajišťuje základní chování aplikace, především obsahuje definice abstraktních tříd kontrolerů, modelů a mapperů, což je výhodné pro vykonávání opakujících se akcí. Modul `admin` zajišťuje podporu administrační části aplikace.

Každý modul má definovanou třídu `Bootstrap`, která je umístěna v souboru s názvem `Bootstrap.php`. V této třídě jsou definovány metody, jejichž název začíná předponou `_init`. Tyto metody jsou automaticky vyvolány na počátku spuštění aplikace. Díky tomu je možné provést úkony, které následně ovlivní chod aplikace (načtení konfiguračních souborů a podobně).

Adresářová struktura modulu odpovídá dříve definovaným pravidlům. Přitom každý modul může obsahovat adresáře pro uložení MVC souborů. Dále každý název třídy modulu obsahuje předponu (*prefix*) odvozenou od názvu modulu (prefix `UserManagement_` pro modul s názvem `user-management`).

Na obrázku B.1 je znázorněna adresářová struktura aplikace, která obsahuje moduly `admin`, `core` a `user-manager`.

8.2 Zajištění funkčnosti pro různé databázové systémy

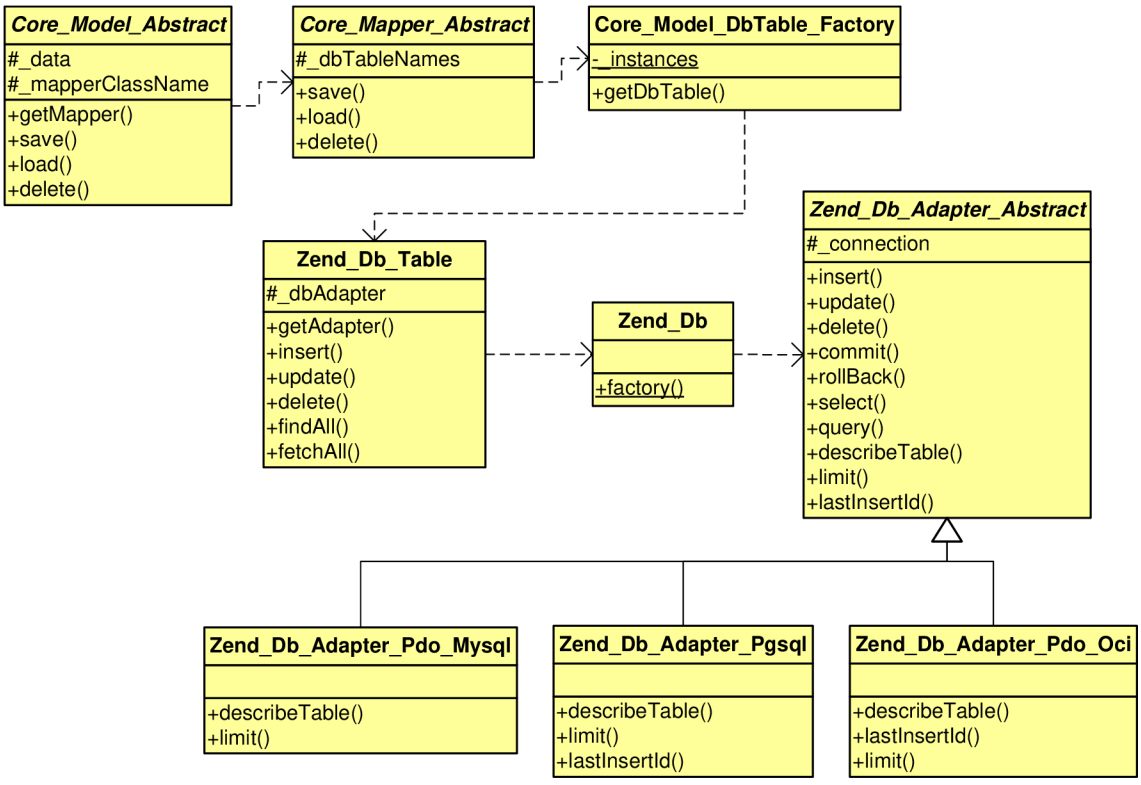
Protože se předpokládá využití systému v různých podmínkách, je vhodné, aby poskytoval podporu více databázových systémů, případně aby bylo umožněno dodatečné přidávání dalších databázových systémů.

Datový model (`Model`) zůstává neměnný pro jakýkoliv databázový systém, a to z důvodu, že vlastní model nepřistupuje přímo k databázi, nýbrž využívá za tímto účelem třídy `Mapper`.

V aplikačních třídách `Mapper` by již mohly být očekávané rozdíly pro různé databázové systémy. Protože však `Mapper` pro přístup k databázovým tabulkám využívá `Zend_Db_Table` a neprovádí přístup k databázi samostatně, není třeba vytvářet třídy mapperů pro různé databázové systémy.

`Zend_Db_Table` pro přístup k databázi využívá objektu `Zend_Db_Adapter`, který na základě konfigurace přistupuje k databázovému adaptéru požadovaného databázového systému (např. `Zend_Db_Adapter_Pdo_Mysql` nebo `Zend_Db_Adapter_Pdo_Pgsql`). Více o dostupných databázových adaptérech naleznete v [2].

Situace zajištění databázové nezávislosti datového modelu a poskytnutí funkčnosti pro různé databázové systémy je znázorněna na diagramu tříd na obrázku 8.5, který je kvůli svému skutečnému rozsahu zjednodušen.



Obrázek 8.5: Využití `Zend_Db_Adapter` pro přístup k rozhraní databáze

8.3 Konfigurace aplikace

Hlavní konfigurace aplikace se provádí prostřednictvím konfiguračního souboru s názvem `application.ini`, který je umístěn v adresáři `configs`. Zpracování konfiguračního souboru probíhá ve třídě `Bootstrap`, která je načtena před spuštěním aplikace. Pro manipulaci s konfiguračními soubory jsou využity třídy Zend Frameworku `Zend_Config_Ini` a `Zend_Config_Xml`.

Součástí hlavního konfiguračního souboru je nastavení databázového připojení, použitého databázového adaptéru, nastavení vyrovnávací paměti cache a cest k různým komponentám systému.

Konfigurace může být upravena pro různá prostředí (`APPLICATION_ENV`) — například pro produkční prostředí nebo testovací. Na jeho základě je možné provést odlišná nastavení (např. různé názvy databázového serveru nebo databází, nastavení úrovně výpisu chyb a podobně). Přitom jedno prostředí může dědit vlastnosti od jiného a bude tak pouze některé konfigurační proměnné přepisovat, nikoliv je definovat znovu.

Kromě hlavního konfiguračního souboru jsou použity další konfigurační soubory, které ovlivňují chování nebo vzhled aplikace. Především se jedná o modulové konfigurační soubory, které jsou definovány v adresáři `configs` v adresáři modulu. Příkladem mohou být konfigurační soubory definující strukturu menu aplikace a nebo konfigurační soubory definující dostupné typy atributů, které jsou v rámci modulu implementovány.

8.4 Administrační část aplikace

Administrační část aplikace slouží přihlášeným uživatelům k modifikaci stavu systému, zejména k editaci dat a nastavení vlastností systému.

8.4.1 Zabezpečení administrační části

Přihlášení uživatele a ověření jeho uživatelského jména a hesla (autentizace) je realizováno s využitím komponenty Zend Frameworku `Zend_Auth`. Po úspěšném přihlášení je do PHP session vložena informace o přihlášeném uživateli (to je třeba z důvodu bezestavového charakteru HTTP protokolu). Z důvodu bezpečnosti má PHP session nastavenou vhodnou dobu platnosti a soubor cookie s identifikátorem session platí pouze po dobu relace (po dobu, kdy je prohlížeč otevřen).

Přihlášení a odhlášení uživatele realizuje aplikační třída `Admin_LoginController`. Jak již bylo řečeno, abstraktní třída `Admin_Controller_Abstract` zajišťuje pro odvozené třídy kontrolerů automatické ověření přihlášení uživatele.

Modul pro správu uživatelských účtů a jejich oprávnění

Aby bylo možné definovat nové uživatelské účty a jejich oprávnění, byl vytvořen modul s názvem `user-manager`, který zprostředkovává přidávání a editaci uživatelských účtů prostřednictvím administrace aplikace. Každý uživatelský účet je reprezentován třídou `UserManager_Model_HadesUser`, která poskytuje požadované operace.

Za účelem povolení či odepření přístupu k funkcím administrace je využita komponenta `Zend_Acl`, která implementuje tzv. *access controll list* [2].

8.4.2 Administrační modul, oddělení administrace od prezentační části

Za účelem administrace je vytvořen modul s názvem `admin`, který poskytuje základní funkce administrace. Jedná se především o zprostředkování přihlášení a odhlášení uživatele a vytvoření základního vzhledu administrace.

Protože je třeba mít dostupnou administraci pro různé moduly (pravděpodobně pro většinu z modulů), byl navržen způsob umístění administračních kontrolerů.

Možným řešením bylo umisťovat administrační kontrolery do modulu `admin`. Řešení však není vhodné z několika důvodů. Prvním z nich je skutečnost, že tímto způsobem by se administrační modul stal závislým na ostatních modulech, což je v rozporu s předpokladem maximální nezávislosti modulů. Dalším problémem je snížení přehlednosti v rámci administračního modulu, jelikož by se zde nacházely kontrolery z různých modulů.

Jako vhodnější řešení se ukázalo uchovávat kontrolery určené pro administraci v modulu, se kterým souvisí. Problémem pak bylo pouze pojmenování a umístění takových kontrolerů, protože je třeba odlišit kontroler určený pro veřejnou část od kontroleru určeného pro část administrační.

Pro označení administračního kontroleru je v názvu třídy uvedeno slovo `Admin`. Bylo však třeba zabránit tomu, aby se všechny kontrolery nacházely v jednom adresáři, protože by to mohlo mít za následek nepřehlednost a složitější orientaci ve zdrojových souborech kódu aplikace. Pro administrační kontrolery je tedy vytvořen vlastní adresář (viz obrázek B.1).

Přitom bylo vhodné pro uživatele odlišit administrační část tím způsobem, že v URL adrese je obsaženo slovo `admin`. Přepisovací pravidla pro mapování adresy na modul, kontroler a akci jsou vytvořena pomocí `Zend_Router`.

8.5 Uživatelské rozhraní aplikace

Protože proteinoví inženýři chtějí soustředit svou pozornost na vlastní práci (zadávání a editaci dat), ne se zabývat složitostí systému, je realizované grafické uživatelské rozhraní aplikace tvořeno jednotným vzhledem a jednotným způsobem editace dat.

Kvůli rozdělení aplikace do modulů a snaze o maximální nezávislost mezi jednotlivými moduly bylo třeba také vyřešit způsob vytváření uživatelských nabídek (menu), které jsou prezentovány jako celek a přitom jsou složeny z částí charakteristických pro jednotlivé moduly.

8.5.1 Vzhled aplikace

Za účelem vytvoření jednotného vzhledu stran je použita třída `Zend_Layout`, která implementuje návrhový vzor *Two Step View*[3], jenž umožňuje vlastní obsah výstupu aplikace obalit jiným vzhledem, reprezentovaným šablonou vzhledu [2].

Aplikační šablony vzhledu jsou uloženy v adresáři `layouts` v kořenovém adresáři aplikace. V této aplikaci je vytvořena šablona umístěná v souboru `layout.phtml`, která představuje jednotný vzhled stran aplikace.

V současné době je vzhled uživatelské i administrační části aplikace stejný, v případě potřeby je možné vytvořit novou šablonu (například pro administrační část) a tuto šablonu pak zaregistrovat. Pokud by byl vzhled použit jednotně pro všechny části administrace, je možné registraci provést v abstraktní třídě `Admin_Controller_Abstract`, která tvoří základ všech administračních kontrolerů.

8.5.2 Uživatelské menu

Protože každý modul nabízí různé akce a protože nelze dopředu určit, jak se bude systém dále rozvíjet, byl navržen způsob tvorby uživatelské nabídky tak, aby vyhovoval dříve popisované modularitě systému a umožňoval definovat menu odděleně pro jednotlivé moduly.

Pro definici menu se využívá již dostupných komponent Zend Frameworku — třídy `Zend_Navigation`, která umožňuje tvorbu uživatelských nabídek.

Struktura menu je uložena v modulových konfiguračních souborech s názvem `menu.xml` pro uživatelské menu veřejné části aplikace a v souborech `admin_menu.xml` pro administrační část aplikace. Pro vytvoření konečné podoby menu jsou jednotlivé konfigurační soubory spojeny, což je možné s využitím metody `merge()` třídy `Zend_Config`.

Jelikož tvorba menu modulu patří k základní funkčnosti aplikace, která je společná pro všechny moduly, je tato funkčnost přidána do abstraktní třídy `BootstrapAbstract`. Vytvořená metoda `_initMenu` zajišťuje načtení konfiguračního souboru s definicí menu a jeho připojení k již existující definici menu. Pokud konfigurační soubor s definicí uživatelské nabídky pro daný modul neexistuje, není existující definice menu modifikována a pokračuje se zpracováním dalších konfiguračních souborů.

K uchování struktury menu je použit `Zend_Registry`, do kterého je vložena proměnná `menu` pro menu veřejné části a `admin_menu` pro menu administrační části aplikace.

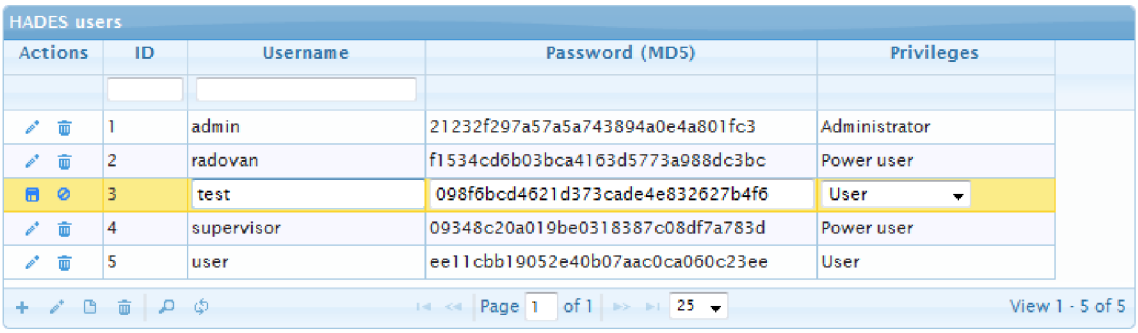
[3] http://martinfowler.com/eaaCatalog/twoStepView.html

Navržený způsob realizace uživatelské nabídky bude dostačující i v případě požadavku rozšíření menu na základě hodnot načtených z databáze (například při implementaci modulu pro zobrazení článků). Na základě načtených hodnot bude třeba vytvořit pole se strukturou menu, které bude odpovídat požadavkům `Zend_Navigation` a které bude spojeno s již vytvořenou strukturou menu uloženou v `Zend_Registry`.

8.5.3 Jednotný způsob editace dat

Aby byl systém co nejjednodušší a jeho ovládání intuitivní, bylo třeba vytvořit jednotný způsob pro editaci dat. Jelikož proteinoví inženýři Loschmidtových laboratoří pro uchovávání dat využívali především program Microsoft Excel, který jim pro tyto účely postačoval, byl zvolen tabulkový způsob editace dat.

Za účelem editace dat v administrační části aplikace je využita komponenta *jQuery Grid Plugin* (*jqGrid*)[4], která nabízí řešení pro prezentaci a editaci tabulkových dat. Data jsou načítána a je s nimi manipulováno prostřednictvím technologie AJAX, což snižuje objem přenášených dat. Podoba tabulkového editoru *jqGrid* je znázorněna na obrázku 8.6.



Obrázek 8.6: Jednotný způsob editace dat — *jqGrid*

Pro realizaci práce s tabulkou je třeba nejprve specifikovat vlastnosti datové tabulky a vytvořit modely jednotlivých sloupců. Pro sloupce jsou udávány údaje o požadované šířce, datovém typu, způsobu formátování hodnoty, způsobu editace hodnoty, možnosti vyhledávání a řazení a také názvu sloupce.

Po vytvoření datové tabulky a načtení zbytku stránky jsou prostřednictvím technologie AJAX načtena data. Data je možné předávat ve formátu XML, prostřednictvím JSON[5] a nebo pomocí JavaScriptového pole [25].

Aby byla práce se systémem co nejjednodušší, je uživateli umožněno editovat hodnoty dat přímo v datové tabulce. To je možné dvojitým kliknutím na požadovaný řádek, případně kliknutím na ikonu tužky, čímž dojde k vytvoření editačních polí na daném řádku. Editace řádků byla realizována vytvořením obsluhy události `ondblClickRow` a vytvořením editačních polí na řádku.

Práci s tabulkovým editorem *jqGrid* zajišťuje třída `Hades_JQGrid`, která realizuje potřebné operace s komponentou *jqGrid*. Podrobnou dokumentaci komponenty *jqGrid* lze nalézt v [25].

[4] http://www.trirand.com/blog/
[5] JSON: *JavaScript Object Notation* — http://www.json.org/

8.6 Modul pro práci s entitami

Zásadní část systému tvoří modul, jehož cílem je poskytnutí přístupu a manipulace s entitami. Entitou se chápe definovaný objekt, který je tvořen určitými vlastnostmi — atributy. Jelikož je tato funkcionalita poměrně specifická, je pro ni vytvořen modul s názvem `entity-manager`.

Dle návrhu popsaném v kapitole 6.2.2 jsou pro uchovávání dat využity dynamicky vytvářené entitní tabulky se sloupci odpovídajícími atributům daného typu entity.

8.6.1 Tvorba typů entit

Pro každý typ entity je při jeho vzniku vytvořena také entitní tabulka, která slouží k uchovávání dat entit daného typu.

Protože Zend Framework v současné době (verze frameworku 1.10) neumožňuje modifikace databázového schématu, bylo nutné za tímto účelem využít standardních SQL příkazů. Pro tvorbu entitních tabulek se jedná o příkaz `CREATE TABLE`.

Protože se název databázové tabulky odvíjí od kódu entity, je nutné při změně tohoto kódu také přejmenovat databázovou tabulku. Pro přejmenování se používá SQL příkaz `ALTER TABLE` *`old_name`* `RENAME TO` *`new_name`*, kde *`old_name`* představuje původní název tabulky a *`new_name`* název nový.

Model typu entity představuje třída `EntityManager_Model_Metadata_EntityType`, jež zajišťuje zprostředkování operací pro práci s typem entity a vytváření entitních datových tabulek. Tyto operace poskytuje třída `EntityManager_Model_Mapper_Metadata_EntityType`. Třída kontroleru `EntityManager_Admin_MetadataController` zajišťuje správu typů entit i jejich atributů.

8.6.2 Tvorba atributů entity

Aby byla zajištěna rozšiřitelnost datového modelu, je entitní tabulka modifikována na základě definice atributů. Pro každý atribut je definován jeden sloupec entitní tabulky. Název sloupce je opět odvozen od kódu atributu.

Součástí každé vytvářené entity je její identifikátor, název a podrobný popis, proto jsou tyto sloupce vytvářeny automaticky již při vzniku entitní tabulky. Protože jsou tyto tři sloupce systémové, není možné je odebrat ani přejmenovat.

K vytvoření úložiště pro hodnoty nového atributu se používá SQL příkaz `ALTER TABLE` *`table_name`* `ADD` *`column_name column_definition`*, kde *`table_name`* představuje název modifikované tabulky, *`column_name`* název přidávaného sloupce a *`column_definition`* definici sloupce. Pro modifikaci sloupce je využit SQL příkaz `ALTER TABLE` *`table_name`* `MODIFY` *`column_name column_type`*.

Při odstranění atributu je třeba odebrat také sloupec z entitní tabulky, využívá se příkazu `ALTER TABLE` *`table_name`* `DROP COLUMN` *`column_name`*.

Při změně kódu atributu je třeba také přejmenovat odpovídající databázový sloupec. Za tímto účelem se využívá SQL příkaz `ALTER TABLE` *`table_name`* `REMANE COLUMN` *`old_name new_name`*.

Model atributu představuje třída `EntityManager_Model_Metadata_Attribute`, která poskytuje základní operace s atributy a zprostředkovává popisované modifikace entitní databázové tabulky.

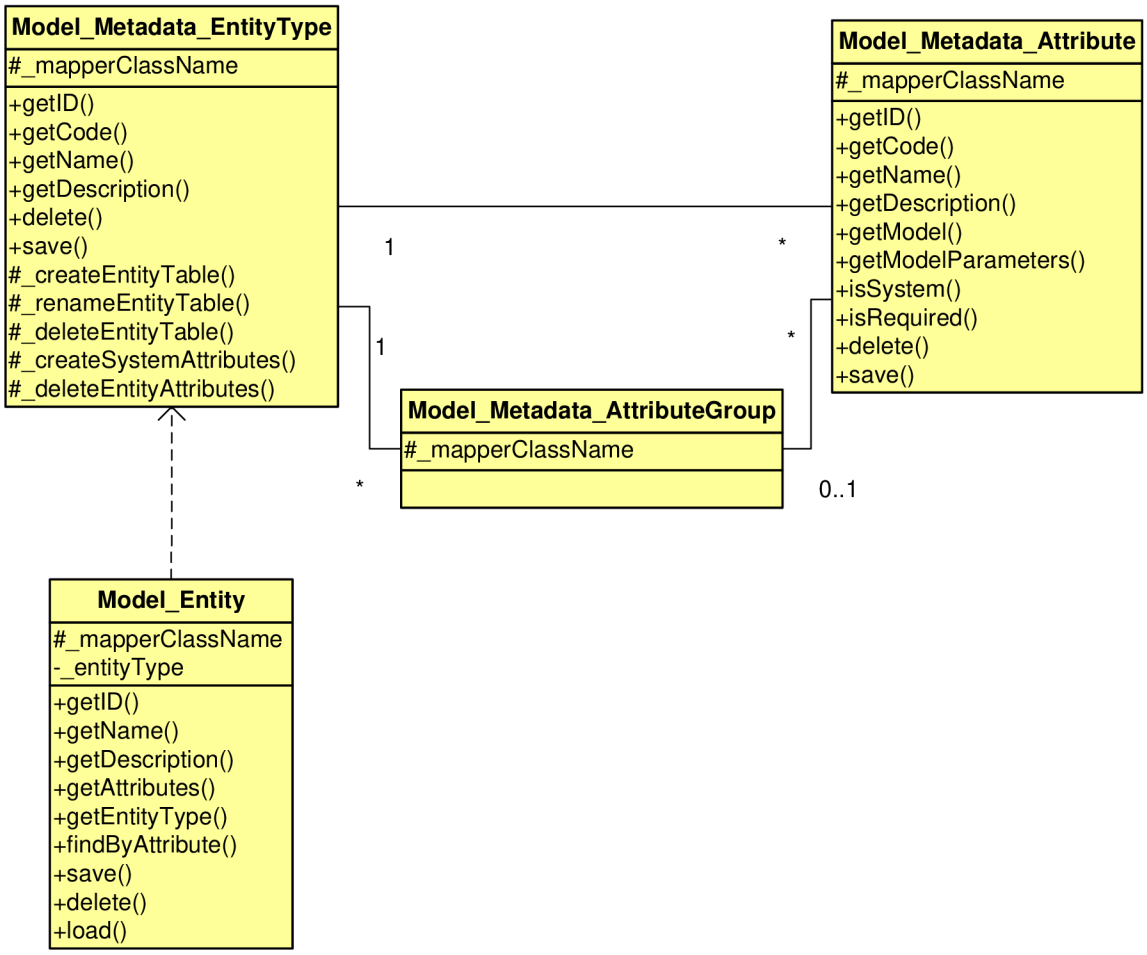
Přestože se předchozí popis zaměřoval pouze na tvorbu entitních tabulek a jejich sloupců, je důležité připomenout, že se současně vytváří také záznamy do metadatových tabulek `metadata_entity_type` a `metadata_attribute`, popsaných v kapitole 6.2.

8.6.3 Práce s entitami

V situaci, kdy jsou vytvořeny požadované entitní tabulky, reprezentující typy entit, s příslušnými sloupci odpovídajícími atributům typu entit, je možné s těmito entitními tabulkami pracovat klasickým způsobem.

Model entity představuje třída `EntityManager_Model_Entity`, která poskytuje dostupné operace s entitou. Protože struktura dat entity je odvozena od metadat reprezentovaných třídami `EntityManager_Model_Metadata_EntityType` představujícími typy entit a třídami `EntityManager_Model_Metadata_Attribute` představujícími atributy typu entity, je třeba, aby pro třídu entity byla tato metadata dostupná. Součástí metadat je také třída skupin atributů `EntityManager_Model_Metadata_AttributeGroup`.

Na obrázku 8.7 je znázorněn diagram tříd implementovaného systému entit a atributů. Diagram obsahuje nejdůležitější vlastnosti a metody, pro zjednodušení je u názvů tříd vynechána předpona `EntityManager_` označující aplikační modul.



Obrázek 8.7: Diagram tříd systému entit a atributů

Díky tomu, že jednotlivé vlastnosti entity jsou uchovávány v poli `$_data` (viz kapitola 8.1.2), je umožněno ukládání hodnot libovolných atributů entity a přitom je zachována identická podoba modelu entity.

Správu dat entity zajišťuje třída kontroleru `EntityManager_Admin_EntityController`, prezentaci dat entity a vyhledávání entit třída `EntityManager_IndexController`.

8.6.4 Dostupné typy atributů

Aby bylo možné zjistit, které typy atributů jsou dostupné, a aby bylo možné při administraci zvolit jeden z dostupných typů, je třeba uchovávat jejich seznam. Protože se předpokládá budoucí růst počtu dostupných atributů, bylo vhodné umožnit snadné přidávání nových typů atributů a jejich definování v různých modulech, protože mohou vznikat atributy, které úzce souvisí s daným modulem.

Za účelem vytvoření seznamu atributů bylo využito úložiště `Zend_Registry`, které je dostupné v rámci celé aplikace. Do registru se vloží pojmenovaná proměnná, jejíž hodnotou je pole s informacemi o dostupných typech atributů. Registrace dostupných typů atributů pak probíhá ve třídách `Bootstrap`, které jsou definovány pro každý modul a jsou volány ještě před spuštěním aplikace.

Pro větší komfort při přidávání nových typů atributů je možné využít modulových konfiguračních souborů `attributes.ini`, uložených v adresáři `configs` modulu, do kterých se ukládají informace o dostupných typech atributů, ostatní činnost je provedena automaticky — v případě, že v modulu je dostupný konfigurační soubor s definicí typů atributů, proběhne jejich automatické zaregistrování.

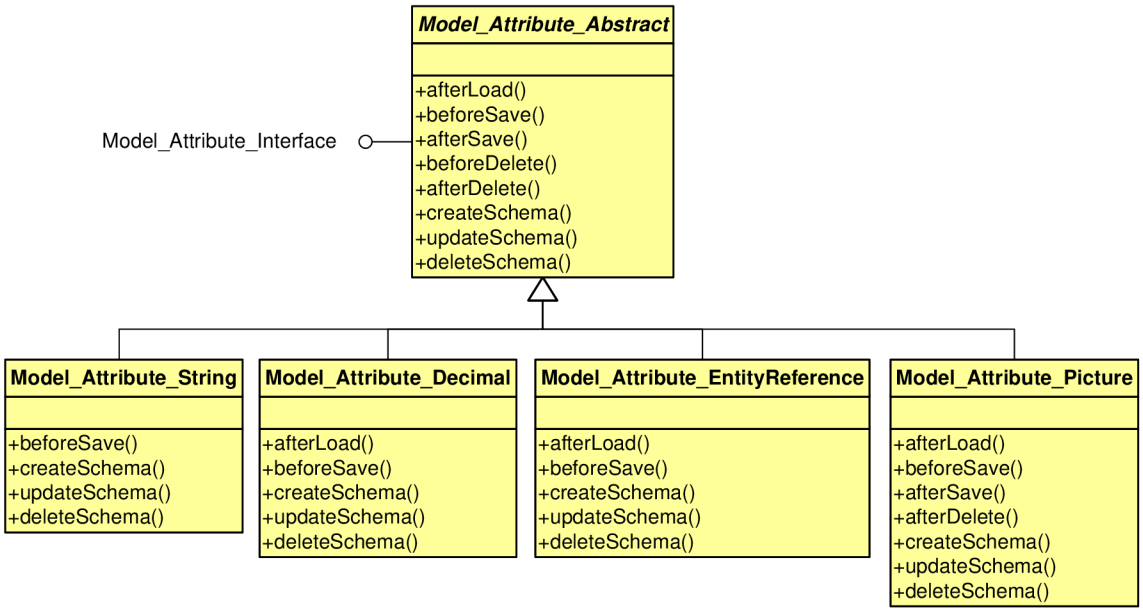
Na tomto místě je třeba poznamenat, že přidávání nových typů atributů provádí programátor, koncový uživatel pouze vybírá z dostupných typů a způsob realizace systému atributů je od něho zcela odstíněn.

Modul `entity-manager` implementuje následující typy atributů, jejichž hodnoty mohou mít definována tato omezení:

- **datum** (realizovaný modelem atributu `EntityManager_Model_Attribute_Date`) – tento typ atributu může být omezen minimálním a maximálním datem.
- **desetinné číslo** (`EntityManager_Model_Attribute_Decimal`) – možná omezení představuje počet desetinných míst a minimální nebo maximální hodnota.
- **celé číslo** (`EntityManager_Model_Attribute_Integer`) – omezeními pro tento typ atributu mohou být minimální a maximální hodnota.
- **řetězec** (`EntityManager_Model_Attribute_String`) – typ atributu umožňuje určit minimální a maximální délku řetězce.
- **text** (`EntityManager_Model_Attribute_Text`) – pro tento typ atributu není umožněno specifikovat žádná omezení.
- **odkaz** (`EntityManager_Model_Attribute_Link`) – ani pro typ atributu odkaz nelze určit omezení hodnot.
- **obrázek** (`EntityManager_Model_Attribute_Image`) – omezení pro tento typ atributu nejsou možná.

- **odkaz na entitu** (`EntityManager_Model_Attribute_EntityReference`) – tento typ atributu představuje rozšíření systému entit a atributů o možnost definovat vazby mezi jednotlivými entitami, což je pro systém velmi přínosné díky možné interakci mezi jednotlivými entitami. Omezením pro tento typ atributu je určení typu entity.
- **sekvence proteinu** (`EntityManager_Model_Attribute_Sequence`) – pro tento typ jsou automaticky všechny symboly udávající označení proteinu převedeny na velká písmena.

Funkčnost a vlastnosti těchto typů atributů jsou určeny modelem atributu, který implementuje rozhraní `EntityManager_Model_Attribute_Interface`. Diagram tříd pro vybrané modely atributů je znázorněn na obrázku 8.8 (u názvů tříd je vynechána předpona označující modul — `EntityManager_`).



Obrázek 8.8: Diagram tříd modelu atributu

8.7 Import a export dat

Protože proteinoví inženýři Loschmidtových laboratoří doposud používali k ukládání dat soubory programu Microsoft Excel, bylo vhodné umožnit import již existujících dat, případně pro plynulejší přechod na nový systém dovolit souběžně definovat data prostřednictvím aplikace a nebo je nejprve ukládat do importních souborů a tato data následně importovat. Export dat umožňuje vytváření záloh entitních dat, případně jejich přenos, následnou editaci a opětovné importování do systému.

Veškerá funkcionalita importu a exportu je implementována v modulu `entity-exchange`, který zajišťuje vlastní import entitních dat z různých souborových formátů a nebo export dat z těchto formátů.

Hlavním požadavkem na import bylo umožnit použití importních souborů programu Microsoft Excel. Formát Office Open XML, který používá balík kancelářských programů

Microsoft Office, byl popsán v kapitole 7.5. Přestože je formát dobře zdokumentovaný, byla by implementace práce se soubory Microsoft Excel příliš náročná, a protože lze předpokládat, že existují hotová řešení umožňující čtení souborů Microsoft Excel, přistoupilo se k hledání volně dostupných knihoven, které tuto funkcionalitu nabízí.

Jako velice kvalitní se jeví knihovna **PHPExcel** (http://www.phpexcel.net), která umožňuje pracovat s různými formáty vstupních a výstupních souborů.

Import dat

Import dat zprostředkovává vkládání či editaci dat prostřednictvím importního souboru. Třída aplikačního kontroleru `EntityExchange_Admin_ImportController` realizuje požadovaný import dat.

Pro čtení importního souboru je využita knihovna `PHPExcel`. Podporované jsou formáty programu Microsoft Excel (a to jak programu Microsoft Excel 2007, který podporuje ukládání v Office Open XML — přípona souboru `.xlsx`, tak i starší formáty programu — přípona souboru `.xls`), programu Open Office Calc (soubory s příponou `.ods`) a nebo soubory ve formátu CSV (soubory s příponou `.csv`).

Pro import dat je třeba zvolit typ entity, pro který jsou importní data určena, a následně vybrat atributy, které se mají importovat. Výběr atributů je prováděn, aby bylo možné použít importní soubor, který obsahuje různá data o entitě, avšak aby byla importována pouze požadovaná data (hodnoty atributů).

Přiřazení importní hodnoty ke správnému atributu entity probíhá s využitím kódu atributu, který je uveden na prvním řádku importního souboru pro každý sloupec datových hodnot.

Před vlastním importem je zobrazen náhled dat, která budou uložena. Jsou zobrazovány také atributy, které importovány nejsou. Pokud je vytvářena entita nová, jsou hodnoty těchto atributů prázdné, pokud však je prováděna aktualizace entity, jsou zobrazeny veškeré hodnoty entity. Sloupce s daty, která budou importována, jsou od ostatních sloupců barevně odlišena.

Export dat

Export dat slouží k vytvoření záloh entit uložených v databázi a nebo pro vytvoření přenositelných souborů, které mohou být editovány a následně pak opětovně importovány do systému. Za účelem vytvoření exportních souborů je využita knihovna `PHPExcel`. Umožněn je export entitních dat do souborů formátů Microsoft Excel (`.xlsx` nebo `.xls`), CSV souboru a nebo do HTML.

Pro export dat je třeba vybrat typ entity, jejíž data se mají exportovat, a následně zvolit požadované atributy. Před vlastním exportem dat je nejdříve zobrazen přehled exportovaných dat.

Export dat realizuje aplikační kontroler `EntityExchange_Admin_ExportController`.

8.8 Testování a nasazení aplikace do provozu

8.8.1 Testování aplikace

Funkcionalita aplikace byla ověřena v několika prostředích, přehled použitých konfigurací je znázorněn v tabulkách 8.1. Jedná se o testování funkčnosti pro databázové systémy PostgreSQL a MySQL.

Webový server	Apache/2.2.12 (Ubuntu)	Apache/2.2.12 (Ubuntu)	Apache/2.2.9 (Debian)
Verze PHP	5.2.10	5.2.10	5.3.2
DB systém	PostgreSQL	MySQL	MySQL
Verze DB systému	8.4.2	5.1.37	5.0.7

Tabulka 8.1: Charakteristika testovacích prostředí

Jelikož je přístup k systému prováděn prostřednictvím webového prohlížeče, byla aplikace vyvíjena tak, aby byla funkční pro různé prohlížeče. Především díky využití JavaScript frameworku jQuery byly potíže s nefunkčností JavaScriptového kódu minimalizovány.

Funkcionalita byla ověřena v prohlížečích Mozilla Firefox (verze 3.5 a 2.0), Internet Explorer (verze 8.0) a Opera (verze 10.51).

8.8.2 Způsob instalace aplikace

Nejprve je třeba zvolit databázový systém, který bude použit pro uchovávání dat. Aplikace podporuje databázové systémy PostgreSQL a MySQL. V prostředí požadovaného databázového systému je třeba vytvořit databázi (případně i uživatelské účty), do které bude importováno základní databázové schéma, které aplikace vyžaduje. Jedná se o skript `install_PostgreSQL.sql` pro databázový systém PostgreSQL a skript `install_MySQL.slq` pro databázový systém MySQL.

Po vytvoření datazáze je třeba modifikovat konfigurační soubor `application.ini`, ve kterém je třeba upravit nastavení databázového připojení. Jedná se o nastavení vlastností `resources.db`, pro databázový systém PostgreSQL bude vlastnost `resources.db.adapter` nabývat hodnoty `'Pdo_Pgsql'` a pro databázový systém MySQL hodnoty `'Pdo_Mysql'`. Dále je třeba nastavit vlastnosti `resources.db.params.host` označující databázový server, `resources.db.params.username`, `resources.db.params.password` jako uživatelské jméno a heslo použité pro připojení a `resources.db.params.dbname` s názvem použité databáze.

Následně je třeba zkopírovat kód aplikace na webový server (je třeba zkopírovat celý obsah složky `src`). V případě, že aplikace běží na vlastní doméně a skript `index.php` je umístěn v kořenovém adresáři domény, je instalace dokončena. V opačném případě je třeba ještě modifikovat soubor `.htaccess` a nastavit v něm vlastnost `RewriteBase` na hodnotu, která odpovídá cestě k souboru `index.php`.

Kapitola 9

Závěr

Náplní této diplomové práce bylo seznámit se s požadavky proteinových inženýrů Loschmidtových laboratoří Masarykovy univerzity v Brně na sestavení databázového systému pro správu biologických dat, analyzovat současný stav projektu HADES, případně dalších systémů pro uchovávání biologických dat a na základě analýzy navrhnout další postup.

Jelikož se uchovávaná data týkají enzymů, převážně enzymů halogenalkan dehalogenáz, bylo třeba nejprve proniknout do oblasti molekulární biologie a pochopit některé z pojmů proteinového inženýrství. Následně byly prozkoumány a popsány základní formáty pro uložení informací o proteinech a některé bioinformatické databáze proteinů.

Jedním z hlavních požadavků proteinových inženýrů byla dostatečná flexibilita a rozšiřitelnost systému. Protože je plánováno umožnit využití systému také pro jiné laboratoře, bylo nezbytné, aby jednotlivé laboratoře mohly definovat, jaká data budou uchovávána, případně aby v průběhu používání systému mohly datové schéma rozšiřovat nebo modifikovat, a to bez zásahu programátora.

Na základě analýzy současného stavu projektu HADES a rozboru existující aplikace bylo zjištěno, že z důvodu malého rozsahu a špatně čitelného zdrojového kódu aplikace bude vhodnější využít jiného řešení, které by přineslo požadovanou flexibilitu a rozšiřitelnost.

Jelikož nebyl nalezen žádný z existujících systémů pro uchování proteinů nebo enzymů, který by vyhovoval potřebám Loschmidtových laboratoří, bylo rozhodnuto, že bude navržen a implementován systém nový.

Navržena byla struktura databáze a aplikace, která proteinovým inženýrům nabízí dostatečnou flexibilitu v uchovávání informací o proteinech, umožňuje jednoduchou rozšiřitelnost a vkládání dat. Za tímto účelem byl analyzován model *Entity–Attribute–Value*, který však kvůli své náročnosti a výkonnostním potížím ověřených testem nebyl využit. Byl navržen systém entit a atributů, který z tohoto modelu vychází a uchovává data entit v dynamicky vytvářených entitních tabulkách.

Navržená aplikace byla implementována v programovacím jazyce PHP s využitím frameworku Zend Framework. Pro uchování dat může být využita databáze PostgreSQL a nebo MySQL.

Aplikace byla důkladně testována v různých prostředích, s využitím databázového systému PostgreSQL i MySQL. Následně byla uvedena do provozu a představena proteinovým inženýrům. Jelikož výsledný systém nabízí dostatečnou rozšiřitelnost a jednoduchou editaci dat, jeví se, že plně vyhovuje požadavkům proteinových inženýrů.

9.1 Návrh pokračování projektu

Jelikož se předpokládá, že se vzniklá aplikace bude dále rozvíjet, jsou navržena rozšíření, která by přinesla užitek nebo zkvalitnění práce se systémem.

Možným vylepšením by byla například implementace rozšířeného vyhledávání, které umožní vyhledávat entity na základě zadaných kritérií. Dále by bylo užitečné rozšíření definice přístupových práv také na typy entit, případně na konkrétní entity. Přínos by představovala i realizace hromadných operací s entitami (hromadná editace, mazání, výběr požadovaných řádků při importu nebo exportu). Možné by bylo také grafické znázornění typů entit a jejich vzájemných vazeb a atributů. Výhodné by mohlo být také zaznamenávání akcí, které jsou prováděny jednotlivými uživateli.

Literatura

[1] Fact SheetPubMed: MEDLINE Retrieval on the World Wide Web. [online], 2008-04-25 [cit. 2009-11-23].
URL http://www.nlm.nih.gov/pubs/factsheets/pubmed.html

[2] Zend Framework: Programmer's Reference Guide. [online], 2009-11-23 [cit. 2009-12-04].
URL http://framework.zend.com/manual/en/

[3] MySQL 5.5 Reference Manual. [online], 2010-03-08 [cit. 2010-03-10].
URL http://dev.mysql.com/doc/refman/5.5/en/

[4] PostgreSQL 8.4.3 Documentation. [online], 2010-03-15 [cit. 2010-04-02].
URL http://www.postgresql.org/docs/8.4/

[5] Transactional DDL in PostgreSQL: A Competitive Analysis. [online], 2010-11-23 [cit. 2010-02-24].
URL http://wiki.postgresql.org/wiki/Transactional_DDL_in_PostgreSQL:_A_Competitive_Analysis

[6] Getting Started with Symfony. [online], [cit. 2009-12-17].
URL http://www.symfony-project.org/getting-started/

[7] jQuery Documentation. [online], [cit. 2010-02-12].
URL http://docs.jquery.com/

[8] Alberts, B.; Bray, D.; Johnson, A.; aj.: *Základy buněčné biologie - Úvod do molekulární biologie buňky*. Espero Publishing, druhé vydání, 2005, ISBN 80-902906-2-0.

[9] Allen, R.; Lo, N.; Brown, S.: *Zend Framework in Action*. Manning Publications, první vydání, 2009, ISBN 978-1933988320.

[10] Berman, H. M.; Henrick, K.; Nakamura, H.: Atomic Coordinate Entry Format. [online], [cit. 2009-11-14].
URL http://www.wwpdb.org/documentation/format32/sect1.html

[11] Boumphrey, F.; Greer, C.; Raggett, D.; aj.: *XHTML Průvodce vývojáře*. Mobil Media a.s., první vydání, 2002, ISBN 80-86593-14-2.

[12] Brandt, C. A.; Morse, R.; Matthews, K.; aj.: Metadata-driven creation of data marts from an EAV-modeled clinical research database. *International Journal of Medical Informatics*, ročník 65, č. 3, 2002: s. 225 – 241, ISSN 1386-5056.

URL http://www.sciencedirect.com/science/article/B6T7S-4718DTP-1/2/d5791a194a9d65e56a40fce88d939e4c

[13] Castagnetto, J.; Rawat, H.; Schumann, S.; aj.: *PHP Programujeme profesionálně.* Computer Press, druhé vydání, 2002, ISBN 80-7226-310-2.

[14] Chen, J.; Sidhu, A. S.: *Biological Database Modeling.* Artech House, inc., první vydání, 2008, ISBN 978-1-59693-258-6.

[15] Chen, R. S.; Nadkarni, P.; Marenco, L.; aj.: Exploring Performance Issues for a Clinical Database Organized Using an Entity-Attribute-Value Representation. *Journal of the American Medical Informatics Association*, ročník 7, č. 5, 2000: s. 475 – 487, ISSN 1067-5027.
URL http://www.ncbi.nlm.nih.gov/pmc/articles/PMC79043/pdf/0070475.pdf

[16] Claverie, J.-M.; Notredame, C.: *Bioinformatics for Dummies.* Wiley Publishing, Inc., druhé vydání, 2007, ISBN 978-0-470-08985-9.

[17] Gasteiger, E.: A Quick Guide To The ExPASy proteomics server. [online], 2004-12-05 [cit. 2009-11-22].
URL http://www.embnet.org/files/shared/QuickGuides/guideExPASy.pdf

[18] Kofler, M.: *Mistrovství v MySQL 5: Kompletní průvodce webového vývojáře.* Computer Press, první vydání, 2007, ISBN 80-251-1502-2.

[19] Momjian, B.: *PostgreSQL: Praktický průvodce.* Computer Press, první vydání, 2003, ISBN 80-7226-954-2.

[20] Moore, J.: Model View Controller. [online], 2007-10-06 [cit. 2009-12-05].
URL http://www.phpwact.org/pattern/model_view_controller

[21] Nadkarni, P.: An Introduction to Entity–Attribute–Value Design for Generic Clinical Study Data Management Systems. [cit. 2010-02-05].
URL
http://ycmi.med.yale.edu/nadkarni/Introduction%20to%20EAV%20systems.htm

[22] Ngo, T.: Office Open XML Overwiev. [online], [cit. 2009-12-28].
URL http://www.ecma-international.org/news/TC45_current_work/OpenXML%20White%20Paper.pdf

[23] Richtera, L.: Protein Engeneering Group. [online], [cit. 2009-10-24].
URL http://loschmidt.chemi.muni.cz/peg/

[24] Schneider, R. D.: *MySQL: Oficiální průvodce tvorbou, správou a laděním databází.* Grada Publishing, a.s., první vydání, 2006, ISBN 80-247-1516-3.

[25] Tomov, T.: jqGrid Documentation. [online], 2009-11-28 [cit. 2010-04-19].
URL http://www.trirand.com/jqgridwiki/

[26] Urbanová, I.: *Sbírka dat z oblasti proteinové a enzymové biologie.* Diplomová práce, Fakulta informatiky Masarykovy univerzity v Brně, Brno, 2006.

[27] Vacík, J.; Barthová, J.; Pacák, J.; aj.: *Přehled středoškolské chemie.* SPN – pedagogické nakladatelství, Čtvrté vydání, 1999, ISBN 80-7235-108-7.

[28] Xue, Q.: The Definitive Guide to Yii. [online], 2009-02-17 [cit. 2009-12-18].
URL http://www.yiiframework.com/doc/guide/

[29] Škultéty, R.: *JavaScript Programujeme internetové aplikace*. Computer Press, první vydání, 2001, ISBN 80-7226-457-5.

Seznam použitých zkratek a symbolů

CSV (*Comma–Separated Values*) – jedná se o souborový formát využívaný pro uchovávání tabulkových dat, kde jsou jednotlivé sloupce hodnot odděleny čárkou.

DDL (*Data Definition Language*) – jde o formální jazyk pro definici datové struktury.

DML (*Data Manipulation Language*) – jazyk používaný pro manipulaci s daty (vkládání, aktualizaci a mazání), je určený k modifikaci dat, nikoliv databázového schématu.

DOM (*Document Object Model*) – objektový model dokumentu představuje objektově orientovanou reprezentaci dokumentu a poskytuje rozhraní umožňující přístup nebo modifikaci obsahu a struktury dokumentu nebo jeho částí.

EAV (*Entity–Attribute–Value*) – EAV model představuje datový model používaný pro ukládání dat, která jsou tvořena velkým množstvím atributů s potenciálně nedefinovanými hodnotami. Využívá se například v klinických a výzkumných oblastech.

HTTP (*HyperText Transfer Protocol*) – protokol aplikační vrstvy, který slouží k výměně hypertextových dokumentů.

MVC (*Model–View–Controller*) – návrhový vzor definující architekturu systému, který rozděluje datovou, prezentační a řídící logiku aplikace.

SQL (*Structured Query Language*) – deklarativní jazyk používaný pro komunikaci s databázovými servery.

URL (*Uniform Resource Locator*) – slouží k přesnému určení zdroje v síti Internet.

Seznam příloh

Dodatek A

Hodnoty testu výkonnosti *Entity–Attribute–Value* modelu

V této příloze jsou prezentovány naměřené hodnoty, které byly získány v průběhu testování doby trvání dotazů pro porovnání přístupu EAV s klasickým přístupem modelování dat.

Test vytváření entit

Vytvoření entit se všemi hodnotami atributů

Počet entit	Doba trvání dotazů [s]				
	1. test	2. test	3. test	4. test	5. test
1 000	0,70	0,69	0,69	0,71	0,71
10 000	6,77	6,74	6,73	6,73	6,79
100 000	65,63	66,28	66,42	66,35	66,11
Průměr na 1 entitu			0,68 ms		

Tabulka A.1: Klasický přístup — vložení hodnot všech 35 atributů

Počet entit	Doba trvání dotazů [s]				
	1. test	2. test	3. test	4. test	5. test
1 000	21,27	20,34	22,16	20,71	22,01
10 000	219,25	218,25	216,58	216,80	219,06
100 000	2217,94	2175,61	2228,64	2153,34	2174,16
Průměr na 1 entitu			21,67 ms		

Tabulka A.2: EAV přístup — vložení hodnot všech 35 atributů

Vytvoření entit s definovanými hodnotami 17 atributů

Počet entit	Doba trvání dotazů [s]				
	1. test	2. test	3. test	4. test	5. test
1 000	0,66	0,67	0,66	0,68	0,68
10 000	6,70	6,81	6,76	6,81	6,74
100 000	66,67	65,82	66,48	66,72	65,94
Průměr na 1 entitu			0,67 ms		

Tabulka A.3: Klasický přístup — vložení hodnot 17 atributů

Počet entit	Doba trvání dotazů [s]				
	1. test	2. test	3. test	4. test	5. test
1 000	10,26	10,46	10,31	10,79	10,41
10 000	106,10	107,81	104,56	106,20	103,92
100 000	1062,46	1065,70	1054,73	1060,14	1063,03
Průměr na 1 entitu			10,54 ms		

Tabulka A.4: EAV přístup — vložení hodnot 17 atributů

Vytvoření entit s definovanými hodnotami 5 atributů

Počet entit	Doba trvání dotazů [s]				
	1. test	2. test	3. test	4. test	5. test
1 000	0,70	0,68	0,69	0,66	0,73
10 000	7,13	6,74	7,18	7,02	6,99
100 000	67,35	68,11	67,17	66,07	66,68
Průměr na 1 entitu			0,69 ms		

Tabulka A.5: Klasický přístup — vložení hodnot 5 atributů

Počet entit	Doba trvání dotazů [s]				
	1. test	2. test	3. test	4. test	5. test
1 000	3,17	3,02	3,18	2,99	3,06
10 000	31,07	29,95	31,07	31,60	30,80
100 000	308,36	313,23	308,35	311,40	310,19
Průměr na 1 entitu			3,09 ms		

Tabulka A.6: EAV přístup — vložení hodnot 5 atributů

Test vyhledání entit

Veškeré testy probíhaly s počtem entit 100 000 a s počtem atributů 35 (tak, jako tomu bylo při testu vytváření entit). Testy jsou provedeny pro různé počty definovaných hodnot atributů a také pro indexované i neindexované databázové sloupce.

Vyhledání entity na základě jejího identifikátoru (ID)

Počet opakování	Doba trvání dotazů [s]				
	1. test	2. test	3. test	4. test	5. test
1 000	0,24	0,25	0,24	0,24	0,25
5 000	1,31	1,28	1,33	1,29	1,23
10 000	2,67	2,58	2,66	2,60	2,66
Průměr na 1 entitu			0,27 ms		

Tabulka A.7: Klasický přístup — vyhledání na základě ID

Počet opakování	Doba trvání dotazů [s]				
	1. test	2. test	3. test	4. test	5. test
1 000	0,91	0,90	0,92	0,88	0,87
5 000	4,99	4,80	5,03	4,82	4,75
10 000	10,09	9,89	9,77	9,92	9,95
Průměr na 1 entitu			0,95 ms		

Tabulka A.8: EAV přístup — vyhledání na základě ID, SELECT jednotlivě

Počet opakování	Doba trvání dotazů [s]				
	1. test	2. test	3. test	4. test	5. test
1 000	1,05	1,03	1,04	1,01	1,14
5 000	5,24	5,79	5,53	5,24	5,58
10 000	11,31	10,76	10,59	10,56	10,88
Průměr na 1 entitu			1,08 ms		

Tabulka A.9: EAV přístup — vyhledání na základě ID, UNION SELECT

Vyhledání entity na základě číselné hodnoty — neindexované

Počet opakování	Doba trvání dotazů [s]				
	1. test	2. test	3. test	4. test	5. test
10	8,99	8,43	8,56	8,74	8,64
100	89,15	86,51	87,19	85,94	86,17
1 000	859,33	875,74	893,42	883,69	867,90
Průměr na 1 entitu			877,71 ms		

Tabulka A.10: Klasický přístup — vyhledání na základě hodnoty čísla typu INT

Počet opakování	Doba trvání dotazů [s]				
	1. test	2. test	3. test	4. test	5. test
10	10,07	10,28	10,33	9,63	10,34
100	103,55	101,87	103,61	102,43	99,71
1 000	1037,08	1018,26	1020,67	1021,62	1016,24
Průměr na 1 entitu			1019,37 ms		

Tabulka A.11: EAV přístup — vyhledání na základě hodnoty čísla typu INT

Vyhledání entity na základě číselné hodnoty — indexované

Počet opakování	Doba trvání dotazů [s]				
	1. test	2. test	3. test	4. test	5. test
1 000	0,43	0,44	0,42	0,43	0,44
5 000	2,31	2,29	2,20	2,19	2,21
10 000	4,43	4,46	4,42	4,46	4,43
Průměr na 1 entitu			0,44 ms		

Tabulka A.12: Klasický přístup — vyhledání na základě hodnoty čísla typu INT

Počet opakování	Doba trvání dotazů [s]				
	1. test	2. test	3. test	4. test	5. test
1 000	1,63	1,61	1,63	1,78	1,61
5 000	8,35	8,06	8,03	8,42	7,88
10 000	15,98	16,17	16,10	16,11	15,99
Průměr na 1 entitu			1,63 ms		

Tabulka A.13: EAV přístup — vyhledání na základě hodnoty čísla typu INT

Vyhledání entity na základě číselné hodnoty — indexované

Počet opakování	Doba trvání dotazů [s]				
	1. test	2. test	3. test	4. test	5. test
1 000	0,44	0,43	0,43	0,44	0,47
5 000	2,35	2,27	2,28	2,34	2,38
10 000	4,54	4,76	4,46	4,50	4,41
Průměr na 1 entitu			0,45 ms		

Tabulka A.14: Klasický přístup — vyhledání na základě hodnoty čísla typu INT

Počet opakování	Doba trvání dotazů [s]				
	1. test	2. test	3. test	4. test	5. test
1 000	1,74	1,59	1,52	1,45	1,49
5 000	7,62	7,47	7,98	7,48	7,80
10 000	15,16	14,82	15,03	14,94	15,16
Průměr na 1 entitu			1,53 ms		

Tabulka A.15: EAV přístup — vyhledání na základě hodnoty čísla typu INT

Vyhledání entity na základě číselné hodnoty — indexované — 5 atributů

Počet opakování	Doba trvání dotazů [s]				
	1. test	2. test	3. test	4. test	5. test
1 000	0,48	0,48	0,49	0,47	0,46
5 000	2,28	2,18	2,22	2,25	2,26
10 000	4,32	4,44	4,52	4,49	4,45
Průměr na 1 entitu			0,46 ms		

Tabulka A.16: Klasický přístup — vyhledání na základě hodnoty čísla typu INT

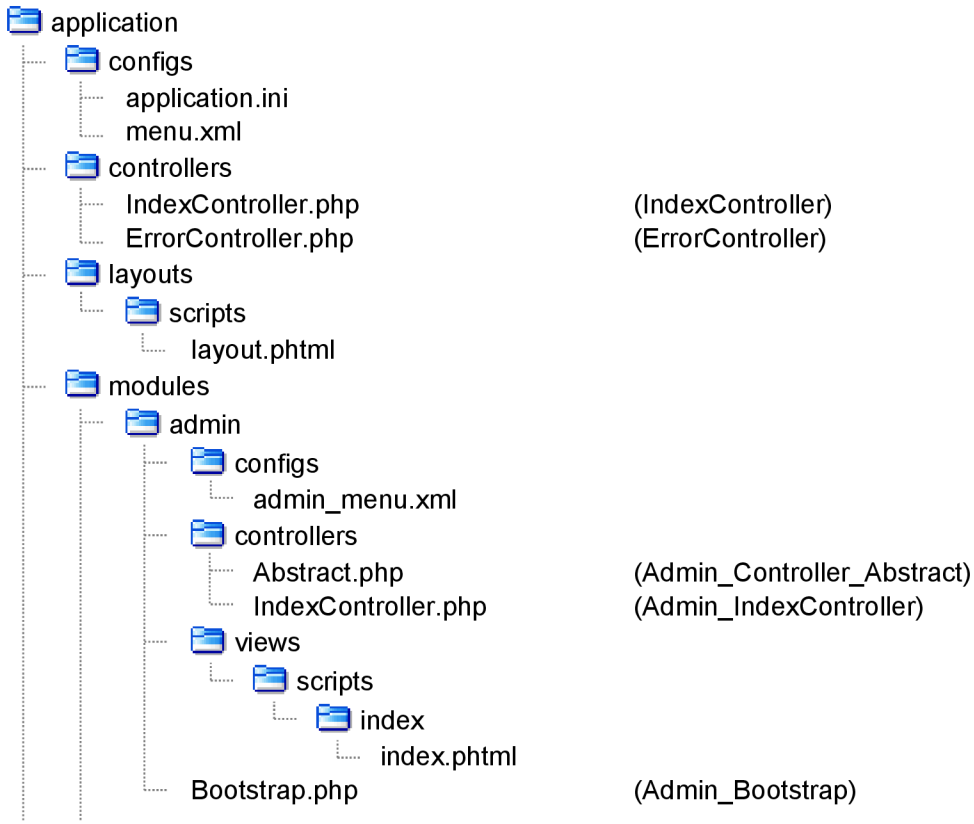
Počet opakování	Doba trvání dotazů [s]				
	1. test	2. test	3. test	4. test	5. test
1 000	1,31	1,43	1,39	1,32	1,41
5 000	6,98	7,00	6,88	7,09	7,22
10 000	13,87	13,78	13,83	13,96	13,89
Průměr na 1 entitu			1,39 ms		

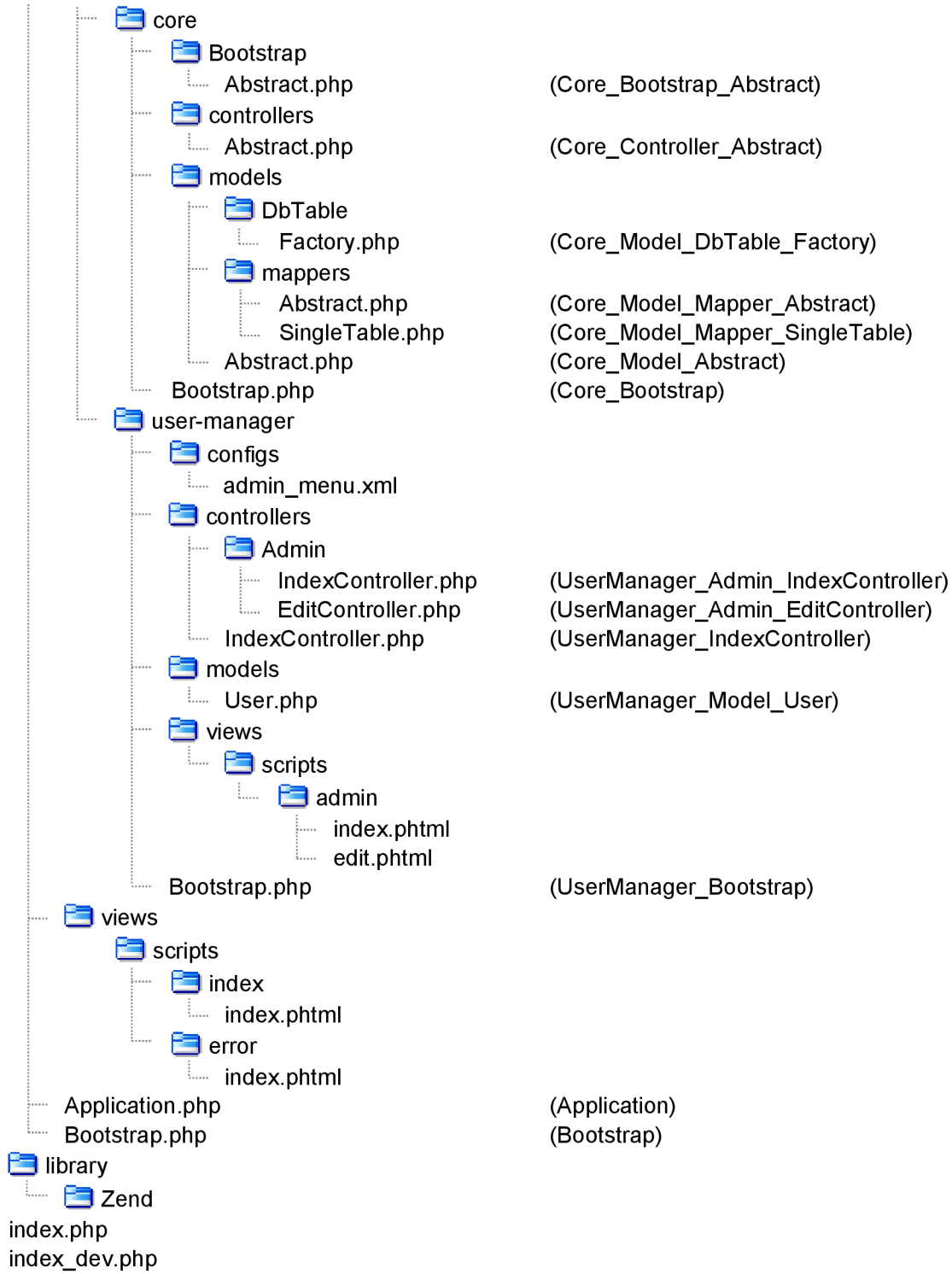
Tabulka A.17: EAV přístup — vyhledání na základě hodnoty čísla typu INT

Dodatek B

Adresářová struktura aplikace

Na obrázku B.1 je znázorněna ukázka adresářové struktury aplikace. Kvůli svému rozsahu a velkému množství souborů aplikace jsou vyobrazeny pouze důležité části (základní adresářová struktura aplikace, adresáře pro konfigurační soubory, některé moduly, abstraktní třídy, části MVC aplikace a struktura kontrolerů určených pro administraci).





Obrázek B.1: Adresářová struktura aplikace

Dodatek C

Přehled modulů a komponent aplikace

Aplikační moduly

V tabulce C.1 je uveden seznam implementovaných aplikačních modulů a jejich základní popis.

Název modulu	Popis modulu
`admin`	Modul představující administrační část aplikace. Zajišťuje přihlášení a odhlášení uživatele ze systému a poskytuje abstraktní třídu `Admin_Controller_Abstract`, která tvoří základ pro všechny admnistrační kontrolery.
`core`	Základní modul, který tvoří jádro aplikace. Obsahuje definice abstraktních tříd modelu (`Core_Model_Abstract`), mapperu (`Core_Model_Mapper_Abstract`), mapperu obsluhujícího jedinou databázovou tabulku (`Core_Model_Mapper_SingleTable`), třídy implementující návrhový vzor *factory* pro vytvoření objektů `DbTable` (`Core_Model_DbTable_Factory`), abstraktní třídy kontroleru (`Core_Controller_Abstract`) a zaváděcí třídy (`Core_Bootstrap_Abstract`). Tyto abstraktní třídy jsou základem pro třídy odvozené, které jsou v aplikaci použity.
`entity-exchange`	Modul realizující import a export entitních dat. Pro tento modul je definována pouze administrační část, veřejná část není potřeba.
`entity-manager`	Modul, který umožňuje správu rozšiřujících se a variabilních dat. Umožňuje editaci entitních dat, definici metadat (struktury dat), zobrazení uložených dat a jejich vyhledávání. Dále definuje dostupné modely atributů, které v systému entit a atributů hrají klíčovou roli.
`user-manager`	Modul zajišťující správu uživatelských účtů.
knihovna `Hades`	Knihovna, která obsahuje třídu `Hades_JQGrid` realizující práci s JavaScriptovou komponentou jQGrid a třídu `Hades_Tools` nabízející obecné metody, které aplikace využívá.

Tabulka C.1: Aplikační moduly

Ostatní použité komponenty

Přehled komponent uvedených v tabulce C.2 představuje seznam převzatých komponent, které byly při implementaci systému využity.

Název komponenty	Popis komponenty
`Zend`	Jedná se o knihovnu Zend Frameworku. Veškeré třídy, které začínají předponou „`Zend_`“ pochází z této knihovny. Jmenovitě se jedná především o třídy `Zend_Application`, `Zend_Config`, `Zend_Controller`, `Zend_Db`, `Zend_Layout`, `Zend_Navigation`, `Zend_Registry` a `Zend_View`.
`ZendX`	Jedná se o rozšíření Zend Frameworku. Z této knihovny byla využita třída `ZendX_JQuery`, která nabízí některé metody pro práci s JavaScriptovým frameworkem jQuery.
`PHPExcel`	Knihovna `PHPExcel` byla využita při importu a exportu entitních dat (v modulu `entity-exchange`). Umožňuje import dat z formátů Microsoft Excel, Open Office nebo CSV a export do formátů Microsoft Excel, CSV a HTML.
`ZFDebug`	Knihovna, která představuje rozšíření Zend Frameworku o nástroje určené pro vývoj a sledování výkonu aplikace. Zprostředkovává informace o použité verzi PHP, Zend Frameworku, proměnných předávaných prezentační části aplikace (view), vykonaných databázových dotazech a době jejich trvání, spotřebě paměti, vzniklých chybách a varováních a podobně.

Tabulka C.2: Ostatní použité komponenty

Dodatek D

Obsah přiloženého CD

- adresář `docs` – dokumentace projektu
  - adresář `api` – programová dokumentace (Javadoc)
  - adresář `example` – ukázková struktura databáze a importní data
  - adresář `install` – návod na instalaci aplikace
- adresář `src` – zdrojový kód aplikace
- adresář `thesis` – text diplomové práce
  - `thesis.pdf` – text diplomové práce ve formátu PDF
  - adresář `src` – zdrojový tvar práce ve formátu LaTeX