Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství (PEF)

Bakalářská práce

Tvorba webové aplikace pro podporu plánování projektů ve firmě

Jan Štrunc

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Jan Štrunc

Informatika

Název práce

Tvorba webové aplikace pro podporu plánování projektů ve firmě

Název anglicky

Development of web application for company project planning support

---

Cíle práce

Hlavním cílem práce je vytvořit webovou aplikaci, která bude sloužit jako podpůrný prostředek pro plánování projektů ve firmě. Dílčím cílem práce je popsat použitelné technologie a přístupy při tvorbě webových aplikací.

Metodika

Metodika práce je založena na analytickém a syntetickém přístupu. Bude provedena analýza odborných informačních zdrojů. Na základě syntézy zjištěných poznatků bude zpracován popis výchozího stavu problematiky, popsány nástroje a technologie pro tvorbu webových aplikací a budou vybrány technologie pro využití při tvorbě aplikace.

Dále bude proveden návrh a implementace dané webové aplikace s využitím vybraných technologií. Aplikace bude dále nasazena a otestována. Budou shrnuty poznatky a zkušenosti získané při její tvorbě a nastíněny možnosti dalšího rozvoje aplikace.

Doporučený rozsah práce

35-40 stran

Klíčová slova

webová aplikace, PHP, databáze, projekt, plánování, management

---

Doporučené zdroje informací

DANEL, Roman. Informační systémy: Texty pro výuku, odkazy a otázky ke zkouškám. VŠB – Technická univerzita Ostrava: INNET [online]. 4.5.2011 [cit. 2015-03-29]. Dostupné z: http://homel.vsb.cz/~dan11/rd_is_skripta.htm

KOSEK, Jiří a FAWCETT. PHP a XML. 1. vydání. Praha: Grada Publishing, a.s, 2009. ISBN 978-80-247-1116-4.

LECKY-THOMSON, Ed a Steven D. NOWICKI. PHP 6: Programujeme profesionálně. 1. vydání. Brno: Computer Press, a.s, 2010. ISBN 978-80-251-3127-5.

ZAKAS, Nicholas. JavaScript pro webové vývojáře: Programujeme profesionálně. 1. vydání. Brno: Computer Press, a.s, 2009. ISBN 978-80-251-2509-0.

ZAKAS, Nicholas, Jeremy MCPEAK a Joe FAWCETT. Ajax: profesionálně. 1. vydání. Brno: Computer Press, a.s, 2007. ISBN 978-80-86815-77-0.

Předběžný termín obhajoby

2016/17 ZS – PEF

Vedoucí práce

Ing. Jiří Brožek, Ph.D.

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 20. 2. 2016

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 20. 2. 2016

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 29. 11. 2016

Čestné prohlášení

Prohlašuji, že jsem svou bakalářskou práci "Tvorba webové aplikace pro podporu plánování projektů ve firmě" vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor uvedené bakalářské práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 30. 11. 2016 _____

Poděkování

Rád bych touto cestou poděkoval vedoucímu práce Ing. Jiřímu Brožkovi, Ph.D za ochotu a čas strávený na konzultacích. Dále za cenné rady a připomínky k práci.

Tvorba webové aplikace pro podporu plánování projektů ve firmě

Souhrn

Tato práce se zabývá naprogramováním webové aplikace sloužící pro podporu plánování projektů na základě dat vedené v její evidenci.

Teoretická část se zaobírá problematikou vývoje aplikace a zároveň jsou zmíněny technologie, které budou použity pro její vývoj.

Praktická část bude věnována vývoji a programování jednotlivých nástrojů a aplikace samotné. Technologie popsány v první části budou aplikovány na vývoj a řádně demonstrovány.

Klíčová slova: webová aplikace, PHP, databáze, projekt,podpora plánování

Development of web application for company project planning support

Summary

The bachelor thesis deals with programming web applications used to support project planning based on data maintained in its records.

The theoretical part deals with the issue of application development and also mentions the technology that will be used for its development.

The practical part will be devoted to the development and programming of various tools and applications themselves. The technology described in the first part will be applied to the development and well demonstrated.

Keywords: Web applications, PHP, database, project planning support

Obsah

Seznam obrázků

1 Úvod

Pojem tvorba aplikace je velmi široký pojem avšak lze ho blíže specifikovat například jazykem, ve kterém je naprogramována. Jsme schopni rozeznat 2 druhy aplikace z hlediska uživatele, a to aplikace spustitelné z operačního systému, pak se jedná o tzv. "desktopové" nebo z prohlížeče (z webového prostředí), pak se jedná o aplikace webového rázu. Hlavní rozdíl mezi těmito aplikacemi je místo spouštění. Zatím co "desktopové" aplikace se spouštějí z úložiště počítače, webové se spouštějí na serveru, kde jsou umístěny a počítač slouží jako terminál.

Jak typ aplikace ovlivňuje klientův přístup k ní? U "desktopových" aplikací probíhá veškerý výpočet a běh aplikace v počítači klienta, zatím co u webových aplikací počítač klienta není zatížen, výpočet se přesouvá na server. Webová aplikace má výhodu přístupnosti odkudkoliv, kde je dostupné internetové připojení nebo připojení na daný server. "Desktopové" aplikace jak bylo zmíněno výše, běží na lokálním úložišti daného počítače.

Pro vývoj webových aplikací je používáno dnes mnoho serverových programovacích jazyků, mezi které například patří PHP, Java, Ruby s frameworkem Rails, ASP.NET, Perl, Python. Hypertext Preprocesor ve zkratce PHP je jeden z nejpoužívanějších jazyků pro tvorbu webových aplikací. Pro problematiku na straně serveru bude používán výše zmiňované PHP, které bude řešit logiku aplikace a pro zobrazování výsledků uložených v MySQL databázi. Na straně klienta použijeme HTML a CSS.

2 Cíl práce a metodika

2.1 Cíl práce

Hlavním cílem práce je vytvořit webovou aplikaci, která bude sloužit jako podpůrný prostředek pro plánování projektů ve firmě. Dílčím cílem práce je popsat použitelné technologie a přístupy při tvorbě webových aplikací.

2.2 Metodika

Metodika práce je založena na studijním a syntetickém přístupu. Bude provedena studie odborných informačních zdrojů. Na základě syntézy zjištěných poznatků bude zpracován popis výchozího stavu problematiky a popsány nástroje a technologie pro tvorbu webových aplikací.

Dále bude proveden návrh a implementace dané webové aplikace s využitím následujících technologií: PHP, MYSQL, HTML a CSS. Aplikace bude dále nasazena a otestována. Budou shrnuty poznatky a zkušenosti získané při její tvorbě a nastíněny možnosti dalšího rozvoje aplikace.

3 Webové technologie

Internet jako takový vznikl roku 1969. Ovšem technologie, která směřovala k umožnění vývoje webových aplikací, jak je dnes známe, vznikly až o dvacet let později (roku 1989). Tim Berners-Lee definoval protokol HTTP (hyper-text transfer protokol), sloužící pro výměnu hypertextových dokumentů ve formátu HTML. Po deseti letech vznikly první webové prohlížeče a pro standardizaci webových technologií byla ustanovena organizace W3C (WWW Consortium). Během několika let byl zaznamenán masivní nárůst webových severů. V roce 1995 bylo na světě již sto tisíc serverů podporující HTTP. Tímto započala éra webových technologií. (Palas, 2001)

Obyčejné statické stránky se vyvinuly do dynamické podoby díky skriptovacím jazykům, které se objevily až koncem devadesátých let. Zpřístupnily řadu možností jak stránku upravovat na základě nějaké podmínky nebo uživatelova vstupu. Toto propojení uživatele se stránkou určilo další směr vývoje webových aplikací. Zároveň se rozšířila i škála programovacích jazyků. Jako první serverové rozšíření (technologie), které zpracovávalo klientovi požadavky bylo CGI (Common Gateway Interface). To umožňovalo spouštět programy (standardní vstup a výstup) napsané v C/C++, perlu nebo pythonu a dalších jazycích. Jedním z posledních stupňů jsou skriptovací jazyky, které umožňují rychlejší zpracování kódu. Skriptovací jazyky procházely také vývojem, a již několik let je možné v jejich logice uplatnit objektově orientovaný přístup. Vznikaly různá hotová řešení či celé frameworky, umožňující rychlejší a standardnější vývoj. Mezi skriptovací jazyky patří např. PHP, Python, Perl nebo Ruby. (Palas, 2001)

Zároveň se s jazyky rozvíjely i databáze, které se zaměřovaly na rychlost dotazovaných dat a jejich úsporu. Mezi nejznámější patří MySQL, Microsoft SQL, Oracle Database, Mongo DB a mnoho dalších. Rozdíl mezi těmito databázemi je většinou razantní v pozadí ale v syntaxi (poskládání dotazu) není rozdíl až tak markantní, což umožňuje vývojářům lehčí přeorientování při práci s jinou databází.

Dnes se webové technologie rozrostly poměrně robustně a v podstatě není možné ovládat všechno. Proto se rozdělili na backend (logika na straně serveru) a frontend (design a kódování vzhledu aplikace/stránky).

3.1 Backend

Pojem backend, neboli pozadí aplikace, je stanoveno jenom pro logiku aplikace. Všechno týkající se grafického výstupu, tedy to co vidí uživatel, spadá do frontendu. Rozdělení umožnilo jejich nezávislost, proto si lze pro aplikaci můžeme zvolit jakýkoliv programovací jazyk, aniž by to jakkoliv ovlivnilo grafický výstup. Pro účely této práce byl zvolen skriptovací jazyk PHP a jako datové úložiště databáze MySQL. Hlavními důvody byly četná komunita, sdílející informace a autorova znalost jazyka PHP a práce s MYSQL.

3.1.1 PHP

Celý název je "Hypertext Preprocessor", původně znamenal "Personal Home Page". Ve výsledku se však zkratka nezměnila. Jedná se o skriptovací jazyk určený především pro programování dynamických internetových stránek a aplikací. Byl vyvinut v roce 1995 Rasmusem Lerdorfem. Od této doby se neustále vyvíjí, poslední verze (7.0.9) byla uvolněna 21.července 2016 a pracuje se na další.

Lze v něm programovat i desktopové aplikace, proto existuje kompilovaná forma jazyka (pro svoje spuštění nepotřebuje zdrojový kód). Syntaxe je inspirovaná jazyky jako C, Perl, Pascal a Java. Umožňuje také připojení k většině databázových systémů. Jazyk PHP je dynamicky typovaný, tzn. že datový typ proměnné je vázán na hodnotu, nikoliv na proměnnou.

V dnešní době je nejrozšířenějším jazykem za čímž stojí také jeho open source model, který umožňuje komunitě ho neustále zdokonalovat. Obsahuje také řadu knihoven, které umožňují výše zmíněné připojení do databázových systémů. Od verze 5.0 je k dispozici nová knihovna PDO zajišťující toto připojení s OOP přístupem a již implementovanými ovladači pro ostatní databázové systémy jako MySQL, Microsoft SQL Server, Oracle nebo Postgre SQL. Další knihovny slouží pro práci např. s IMAP, POP3 protokolem nebo GD grafikou. (Achour, a další, 2001) (Lecky-Thomson & Steven, 2010)

3.1.1.1 OOP

Jedná se o zkratku Object-Oriented-Programing, umožňuje jiný pohled na programování. Ve své podstatě se snaží odrazit realitu do kódu, který by měl

napodobovat chování z reálného světa. Modelováním objektů na základě jejich skutečné podoby lze dosáhnout přehledného a čitelného kódu. (Lecky-Thomson & Steven, 2010)

Pro lepší představu poslouží například stavba domu, což je celek, který chceme zbudovat. Na tomto celku pracují různí subdodavatelé se společným cílem. Každý z nich se dá popsat jako objekt, který zná detaily svojí práce. Ovšem na venek o objektu víme, že je to např. jenom elektrikář nebo instalatér. (Lecky-Thomson & Steven, 2010)

V malém projektu převedení z procedurálního kódu do OOP nemá takový efekt. Oproti velkému projektu, kdy je zapotřebí čitelnost, rychlost a přehlednost je velice důležitá objektově orientovaná architektura. Pokud by někde v kódu nastala chyba a bylo jí potřeba opravit, v procedurálním kódu by tato činnost trvala o 60% více času než v objektově orientovaném kódu. (Lecky-Thomson & Steven, 2010)

Na všechny objekty si lze udělat předpis, který je nazván třídou. Například adresa je pojem určující polohu resp. umístění. Adresa má různé parametry jako město, ulice, číslo orientační a číslo popisné. Promítnutím do kódu je vytvořena třída adresa se stejnými vlastnostmi. Její instance (objekt - konkrétní adresa) dosáhne odrazu z reálného světa. Obrázek 1 popisuje situaci, kde uživatel resp. osoba z reálného světa využívající aplikaci si nastavuje svojí adresu. V programování se nelze vyhnout abstraktním situacím, kdy daný předmět v reálném světě neexistuje. V tuto chvíli pomůže představa k čemu má prvek sloužit a dle toho je daná třída zastřešena nejvhodnějším jménem. (Lecky-Thomson & Steven, 2010)

```
$user->getAddresses()[0]->setType($type);
$user->getAddresses()[0]->setStreet($st);
$user->getAddresses()[0]->setStreetNumber($stnumber);
$user->getAddresses()[0]->setCity($city);
$user->getAddresses()[0]->setZip($zip);

$this->dataManager->persist($user->getAddresses()[0]);
$this->dataManager->reflect();
```

Obrázek 1: Ukázka OOP kódu (zdroj: vlastní zpracování)

Třída a objekt není zdaleka všechno, co OOP definuje. Dále je to dědičnost, pokud mají třídy společné znaky resp. budování nové struktury(třídy) na základě staré. Polymorfismus, kdy stejná zpráva (uživatel->jméno) vrátí jiný výsledek, jelikož každý objekt nemusí být stejný. Za příklad je možno uvažovat jména lidí. Dalším principem je rozhraní definující společnou metodu pro třídy, avšak každá třída ji řeší po svém. Například metoda "nastartuj motor", činnost, která je stejná u auta i letadla, ale u každého se provádí jinak. Posledním znakem je zapouzdření. Schopnost objektu chránit před přístupem svá interní data. Důkladnější vysvětlení lze ukázat taktéž na lidech. Na venek je poznat barva vlasů, výška nebo barva pleti ale už není očividný věk či jeho rodinný stav. (Lecky-Thomson & Steven, 2010)

3.1.1.2 PDO

PHP od verze 5.0 poskytuje rozšíření PDO (PHP Database Object), které nahrazuje staré funkcionální řešení. Jedná se o připojení do databáze postavené na objektově orientované architektuře. V podstatě poskytuje abstraktní vrstvu pro komunikaci s databází. PHP od verze 5.6 tuto vrstvu obsahuje již ve výchozím nastavení, tudíž jí tam není potřeba externě dodávat. (Lecky-Thomson & Steven, 2010)

```php
self::$connection = new PDO("mysql:dbname=$database;host=$host", $user, $pass,array(PDO::ATTR_PERSISTENT => true));
self::$connection->exec("set names utf8");
//SET CHARACTER SET utf8;
if(self::$connection){
    return true;
}else{
    return false;
}
```

Obrázek 2: Příklad připojení do databáze PDO (zdroj: vlastní zpracování)

Převedením do vrstvy na objektově orientované architektuře přineslo jednoduchost a sjednocení původních knihoven. PDO zajišťuje také lepší datovou integritu při sestavování dotazů, kde se mají rozlišit datové typy. Nebo automatická ochrana při parametrizování proměnných před "MySQL injection", praktiky pro nabourávání se do databáze skrz aplikaci či vkládáním javascriptu do databáze a tím poškodit aplikaci. (Achour, a další, 2001)

```
$sql = 'SELECT name, colour, calories
    FROM fruit
    WHERE calories < :calories AND colour = :colour';
$sth = $dbh->prepare($sql, array(PDO::ATTR_CURSOR => PDO::CURSOR_FWDONLY));
$sth->execute(array(':calories' => 150, ':colour' => 'red'));
$red = $sth->fetchAll();
```

Obrázek 3: Dotaz do databáze s použitím parametrů (PDO::prepare)

Další výhodou je zvolení výstupu, který zajišťují PDO konstanty určující různé formáty výstupu. Například klasické pole záznamů, pole objektů podle námi zvolené třídy nebo výsledek funkce, která se provede na každém záznamu (parametr funkce) námi definovaného sloupečku. (Achour, a další, 2001)

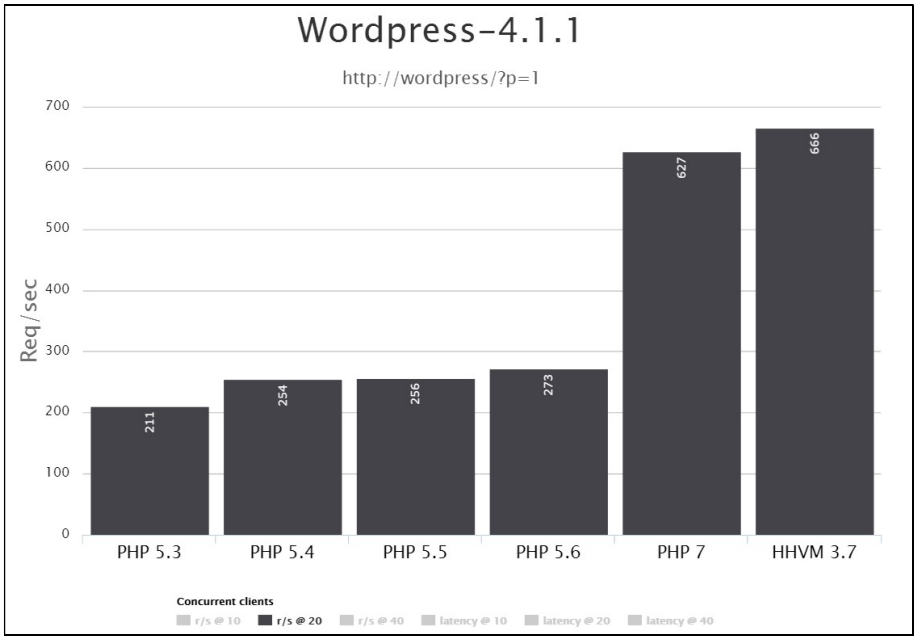
Je nutno zmínit i nevýhody oproti starší nativní funkci dotazu. Klasická funkce je schopna dodat výsledek v asociativním poli, kde klíč je ve formátu "tabulka.název sloupce". Tímto bylo jasné odkud daný záznam pochází a tím byl plně identický. PDO ovšem tento formát postrádá a výstup předává ve formátu "sloupeček". Aby nedocházelo k duplicitě, do pole zařadí první sloupeček s daným názvem a ostatní sloupečky se stejným názvem ignoruje, ačkoliv jsou z jiné tabulky. (Achour, a další, 2001)

3.1.1.3 Session

Princip session je založen na uchovávání dat během doby co je aktivní a její data zůstávají i při načtení jiné stránky v aplikaci. Nativní PHP session je souborová záležitost. Při zavolání funkce "session_start", je na serveru vytvořen soubor s názvem tzv. hashem, který odpovídá klíči v cookies v prohlížeči. Ve skriptu potom funguje jako asociativní pole "$_SESSION" do nějž jsou vkládány hodnoty odkazované klíčem. V PHP existuje "handler" hlídající zápis, smazání nebo úpravu hodnoty, nazývá se "session_set_save_handler". Tím lze přepsat metody a místo ukládání např. databáze nebo cache. (Lecky-Thomson & Steven, 2010) (Achour, a další, 2001)

3.1.2 PHP 6(5.6) -7

Poslední dvě hlavní verze PHP se dají popsat jako zlomové. Verze přímo s názvem PHP 6 oficiálně nevyšla. Jednalo se pouze o neformální název pro verzi 5.6 a minoritní vyšší verze (5.6.1 atd.) na které probíhala v podstatě příprava na verzi sedm. Jedna z větších změn proběhla v oblasti připojování do databáze. Zde byla klasická funkcionalita nahrazena abstraktní vrstvou s objektově orientovanou architekturou pod názvem PDO, již zmíněnou ve stejnojmenné kapitole. Část změny, která provází PHP verzi sedm, je definitivní odstranění starých funkcí pro komunikaci s databází a plným nahrazením právě PDO. Oproti PHP 5.6 je nová verze rychlejší a nejenom o "nějaký malý kousíček" ale o pořádný kus. Některé anonymní zdroje uvádějí až desetinásobné zrychlení, ale pro většinu aplikací postavených na známých nástrojích např. Wordpress nebo frameworcích jako Symphony, lze bezpečně uvést 100% zrychlení - tedy dvojnásobné. (Lerdorf, 2015)



Obrázek 4: Zrychlení v PHP 7 (Lerdorf, 2015)

3.1.3 MySQL

Jedná se o jednoduchou multiplatformní relační databázi. Komunikace s ní probíhá, jak již název napovídá jazykem SQL (Simple Query Language). Jedním z nejpoužívanějších datových úložišť resp. store enginům používaných pro MySQL je

InnoDB, starající se o základní soubor operací CRUD (create, read, update, delete). Zpočátku obsahovala pouze základní databázové prvky např. jednoduché dotazy. Z důvodu rychle rozvíjejících se webových aplikací a jejich nároků, byly přidány možnosti jako uložené procedury, triggery a pohledy. MySQL je oblíbená právě díky své rychlosti a jednoduchosti. Navíc lze na vlastní server k MySQL nainstalovat nástroj MyPHPAdmin, který umožňuje správu databáze přes webové rozhraní. Výchozí nástroj pro správu je též příkazový řádek. (Oracle Corporation)

```
1 SELECT user.*,address.*,project.* FROM user join project_user on user.user_id=project_user.user_id
2                                        join project on project_user.project_id = project.project_id
3                                        join address on address.user_id = user.user_id
```

Obrázek 5: Ukázka SQL dotazu (zdroj: vlastní zpracování)

Všechny informace jsou uloženy v databázi pod "information schema". Nachází se zde všechny systémové informace obsahující data o vytvořených databází tedy i tabulek a sloupců až po uživatelskou a logovací úroveň, jejíž úkolem je sledovat celé dění v databázi.

Zakladatelé firmy, která MySQL vyvinula, Michael Widenius a David Axmar jsou zároveň průkopníci v oblasti licencování, kde zavedli takzvané dvojí licencování. Umožňuje totiž jak licenci zakoupit pro komerční použití (v případě, že je databáze součástí např. prodávané aplikace) tak ji poskytuje zdarma. (Oracle Corporation)

3.1.4 ORM

Obecně známá zkratka pro Object-Relation-Mapping. Nástroj umožňující převést data z databáze (řádky) do námi určeného objektu nebo objektů, splňující určitá pravidla, která nastavují vztahy jak v databázi tak v architektuře naší aplikace. V kapitole, která se zabývala OOP byla zmíněna podstata vztahů a objektů. Přesně i tato podstata je nutná pro návrh databáze. Pokud je všude dodržena stejná logika modelování objektů a jejich vztahů stejně jako u tabulek v databázi, je možné začít budovat ORM. (Lecky-Thomson & Steven, 2010)

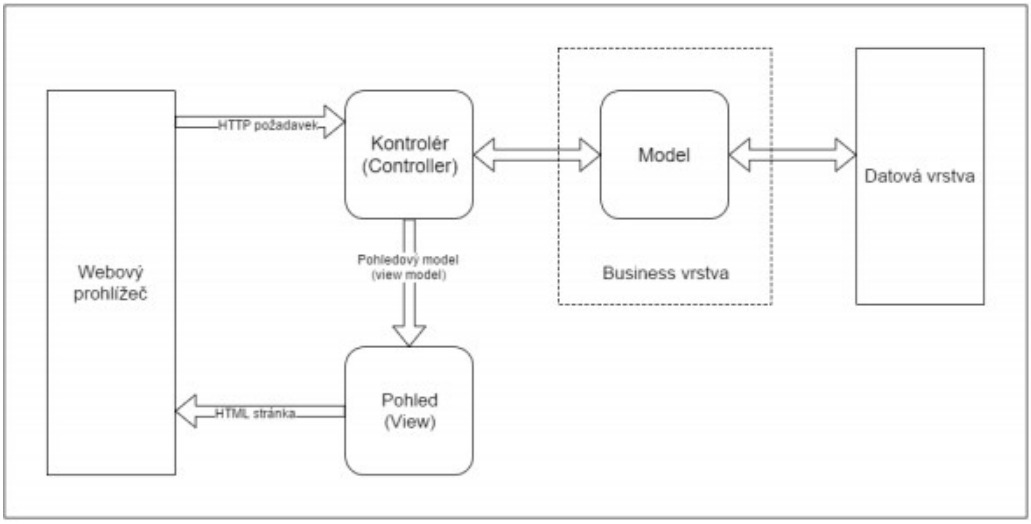
Logika mapování na jednotlivé objekty už není tak jednoduchá jak se může zdát. Ovšem v dnešní době, pokud si nechceme vytvořit vlastní ORM vrstvu,

jsou k dispozici na výběr různé frameworky, které ji mají implementovanou nebo se zabývají pouze jen touto problematikou, například Doctrine1,2 nebo Axon. Ovšem i takový ORM framework pracuje na určitém modelu (vzoru) a to buď dnes již "zastaralý" Active record nebo modernější Repository, Data Mapper a další. Ovšem nelze jednoduše hodnotit na základě použitých návrhových vzorů, neboť záleží na logice daného frameworku. (Team, 2010)

Instalace a používání hotového řešení v této oblasti nebývá vždy to nejlepší. Nejprve je potřeba si uvědomit velikost aplikace resp. její složitost a následně se rozhodnout zda je výhodné dané řešení implementovat. Každý z frameworků má svá pro a proti. Většina výhod je v syntaxi a hotových metodách, které zrychlí vývoj aplikace. Avšak je potřeba uvědomit si z kolika procent je zhruba framework využíván. Pokud je to malé procento, je nejspíše lepší se zaměřit na vlastní kód a nevyužívat tak robustní nástroj. V přirovnání by se to dalo nazvat "příliš velké kladivo na malého mravence". Z toho plyne, velká část kódu je nevyužita a je tu riziko razantního zpomalení aplikace. (Lecky-Thomson & Steven, 2010) (Team, 2010)

3.1.5 Architektura MVC

Architektura MVC je v dnešní době nejběžnější architektura používaná pro webové aplikace. Je jednoduchá, rychlá a přehledná. Oddělení logiky od designu je první část. Další část odděluje data od logiky. Architektura se skládá ze tří částí, model, view a controller a proto tedy MVC. Každý z těchto tří reprezentuje jednu část. Model se stará o získávání dat z databáze, která jsou předávána controlleru a ten je zakomponuje do view. Controller kromě toho, že dostává z modelu data, řeší logiku funkčnosti dané části aplikace. Poté předá data do view. View je pouze nakódovaný design obohacený o data předaná z controlleru, která do šablony vypisuje a poté jsou zobrazena uživateli. (Bernard, 2009) (Lecky-Thomson & Steven, 2010)



Obrázek 6: MVC Paradigma (Svačina, 2016)

3.1.5.1 Model

Jak bylo zmíněno, model je zodpovědný za získávání dat ať z databáze nebo ze souboru. Paradigma (obrázek 6) určuje pouze vztahy, jedná se o takový předpis. Logiku aplikace si pak každý řeší sám dle jeho konkrétní problematiky. Model lze obohatit již zmíněnou ORM vrstvou. Logiku aplikace to ovšem nijak nemění, přibude pouze "abstraktní vrstva" spojující datovou vrstvu a model.

```
public function getUserByAuthCreditals($id, $token){
    $query = $this->dm->newQueryBuilder();
    $user = $query->select(array("user_id","user_token"))
        ->from(array("user"),false)
        ->where("user_id=:id and user_token=:token")
        ->setParameters(array("id"=>$id,"token"=>$token))
        ->execute("User");
    return $user;
```

Obrázek 7: Model s ORM (zdroj: vlastní zpracování)

Na obrázku 7 je ukázková situace, kdy model využívá ORM, který řeší složení dotazu a vrátí výsledek jako instanci objektu "User". Tím se model stane značně jednoduchý a přehledný. Díky tomu není potřeba analyzovat a vyhodnocovat SQL dotaz, který by tu nejspíše figuroval namísto ORM. (Bernard, 2009) (Lecky-Thomson & Steven, 2010)

3.1.5.2 View

Zobrazuje výsledek na obrazovku (to co vidí uživatel). V podstatě je to nakódovaná šablona, ať už do HTML nebo jiného značkovacího jazyka, která interpretuje výstup v prohlížeči. Conroller do šablony pošle data a ty jsou následně zobrazena na patřičných místech v šabloně. Jakým způsobem jsou data do šablony poslána view (vrstva) neřeší, zabývá se pouze jejich zobrazení. Příklad je vidět na obrázku 8, kde je použita zkrácená notace cyklu vypisující možnosti elementu select v šabloně. (Bernard, 2009)

```
<select multiple name="mates">
    <?php foreach ($users as $user) : ?>
      <option value="<?= $user->getId(); ?>"><?= $user->getUserName(); ?> <?= $user->getUserSurname(); ?></option>
    <?php endforeach ?>
</select>
```

Obrázek 8: View - plnění dat (zdroj: vlastní zpracování)

3.1.5.3 Controller

Nejprve je třeba se zmínit o router controlleru. Ten směruje požadavky dle URL adresy, podle které rozeznává jaký controller, model a view má sestavit a jakou metodu zavolat. Zároveň musí validovat URL adresu, která do něj vstupuje, aby dokázal vyhodnotit neexistující požadavek. Respektive požadavek, za kterým nestojí žádný model/view/controller. Pokud požadavek projde validací, vytvoří se jemu přiřazený model a controller, který zavolá metodu v něm již obsaženou, určenou požadavkem. Dále se v controller nastaví šablona, do které se předavají data z modelu. (Bernard, 2009) (Lecky-Thomson & Steven, 2010)

```
class RouterController extends Controller
{

    protected $controller;
    protected $slugs = array(
        // "URL alias" => "Module/controller:akce?promena(podstranka)&promena &promena
        "account/profil" => "CoreModule/User?pageAction",
        "account/prihlaseni" => "CoreModule/Authentication:login",
        "account/odhlaseni" => "CoreModule/Authentication:logout",
        "projects/seznam" => "ProjectModule/ProjectList?id",
        "projects/novy" => "ProjectModule/Project:create"
    );
```

Obrázek 9: Router controller - URL (zdroj: vlastní zpracování)

Pro logičtější URL adresy je možno nahradit tzv. aliasem, ten je zastupuje a musí být jedinečný. Pokud tedy jméno controlleru není optimální, lze si tímto pomoci jak je uvedeno na obrázku 9. Logika přiřazení aliasu je popsána později v kapitole " Controller a Router Controller ". (Bernard, 2009) (Achour, a další, 2001)

3.2 Frontend

Zajímá se jenom o vizuální stránku aplikace. Nejen o statickou část ale i o dynamickou, jelikož v dnešní době jsou dostupné i animace a různé další zpestření, aby aplikace budila větší zájem, popřípadě pocitově zkrátila dobu načítání. Vizuální stránka aplikace zahrnuje nejen design ale i převedení do podoby, kterou dokáže prohlížeč interpretovat. Jednu z prvních podob frontendu na webové úrovni zajišťovalo již zmíněné HTML. Následovně bylo doplněné kaskádovými styly, jenž se do té doby psaly uvnitř struktury HTML. Tím se nejen zpřehlednil kód ale i samotné CSS (Cascading Style Sheets) bylo možné načítat externě ze souboru s příponou ".css". V dnešní době již existuje více značkovacích jazyků nežli jenom HTML, například XML (vzniklo nejdříve po html), XHTML, Polymer atd. (Lecky-Thomson & Steven, 2010) (Berners-Lee & Jaffe)

3.2.1 HTML

Jeden z nejznámějších značkovacích jazyků. HTML je zkratka pro HyperText Markup Language. Jeho historie byla zmíněna na začátku v kapitole "webové aplikace", tudíž ji dále rozebírat nebudeme. Každopádně si od jeho vzniku prošel značným vývojem a v dnešní době je oficiálně nejnovější verze HTML5 nabízející značné možnosti. Ačkoliv je tento značkovací jazyk standard, o který se stará W3C, každý prohlížeč jej může interpretovat jinak. V poslední verzi bylo umožněno vytvářet svoje atributy ve formátu "data-atribut". Bohužel ne všechny prohlížeče to akceptovali, resp. proběhla velmi zdlouhavá procedura, než bylo umožněno využívat celý potenciál HTML5. (Berners-Lee & Jaffe)

3.2.2 CSS

Svými styly doplňuje HTML, pomáhá určovat grafickou podobu zobrazenou na obrazovce resp. v prohlížeči. Jak bylo zmíněno, dříve se psaly styly v samotné struktuře HTML, později je bylo umožněno načítat z externího souboru a tato možnost je stále

nejpoužívanější. Aby byla zachována i možnost psaní v HTML, byla stylům uvnitř jeho struktury přiřazena vyšší priorita než načteným stylům ze souboru. Styly si za svůj čas prošly také velkým vývojem. Nejnovější verze je CSS3, která webové aplikace obohatila o spoustu grafických prvků jako zaoblené rohy, stíny, průhlednost apod. Ovšem ani styly neušly diferenci v zobrazování mezi prohlížeči a v některých je potřeba styly upravit či vytvořit nové pro daný prohlížeč.

Pro rozdílnost vykreslování na různých enginech, se i v této oblasti vyvinuly frameworky jako Bootstrap, který tuto rozdílnost eliminuje a nabízí rychlejší stylování díky předdefinovaným třídám či atributům. Dokonce nabízí plnou responzivitu (přizpůsobení aplikace na šířku zařízení ve kterém se zobrazuje). (Berners-Lee & Jaffe) (Sears, a další, 2011)

4 Vlastní práce

Popsané technologie ve třetí části této práce budou aplikovány na reálný vývoj webové aplikace. Na vývoj aplikace nejsou použity žádné frameworky či hotová řešení, kromě jedné výjimky a to použití vývojového nástroje TracyDebugger pro lepší orientaci při opravě chyby v kódu či sledování výkonnosti skriptů. Aplikace má sloužit jako podpora pro plánování projektů na základě analýzy vložených dat uživatelem.

Aplikace má k dispozici 2 moduly (uživatelský a projektový). Přihlášením uživatele do systému umožňuje zobrazení a editaci jeho profilu. Dále vytváření nových projektů, jejich zobrazení v seznamu a následné prohlédnutí konkrétního projektu v detailu.

4.1 Vytvoření návrhu aplikace

Nejedná se o funkcionálně bohatou aplikaci, přesto obsahuje propracovaný backend, který je nutno rozdělit na dvě části. První část se zaobírá aplikací samotnou a druhá část knihovnami. Pro vývoj bude použit program PHPStorm, který je velice oblíbený mezi PHP vývojáři. Jak bylo již zmíněno, nebude použito žádného frameworku, tím se práce stává složitější a pomalejší avšak výměnou za větší rychlosti aplikace. Je vhodné začít návrhem architektury a jak bude aplikace pracovat. Není zrovna na místě řešit detaily, tím by se celý koncept jenom zkomplikoval. Na jednotlivé detaily dojde v rámci řešení dané části. Celá aplikace bude stát na MVC architektuře propojenou s MySQL databází již přeinstalovanou na soukromém serveru. Pro propojení modelu a databáze bude vyvinuta vlastní ORM knihovna a knihovna starající se o prvky doplňující MVC architekturu.

Grafické rozhraní obsahuje dvě hlavní stránky a pět podstránek. První z hlavních stránek bude tvořena přihlašovacím formuláře a druhá bude tvořit neměnný základní interface systému (menu, vrchní lištu a pozadí). První dvě podstránky se budou týkat uživatele systému, první poskytne zobrazení profilu a druhá jeho editaci. Další podstránky jsou věnovány projektu. Zahrnují seznam všech projektů, formulář pro jejich založení a detailní zobrazení jednotlivých projektů. Z důvodu časové náročnosti této práce, bude aplikace omezena jenom na již zmíněné stránky/podstránky.

4.2 Vytvoření projektu aplikace a jeho struktury

Pro účely této práce bude využit server, na kterém je nainstalována MySQL databáze a webový server Apache. Celý server běží na operačním systému Linux a instalovaná verze PHP je 5.6. Celá aplikace bude napsána tak, aby byla kompatibilní s PHP 7.

4.2.1 Založení projektu

Na serveru bude založena nová subdoména, pro ní se automaticky vytvoří základní souborová struktura se složkou "www", která by měla obsahovat veřejně přístupné soubory.

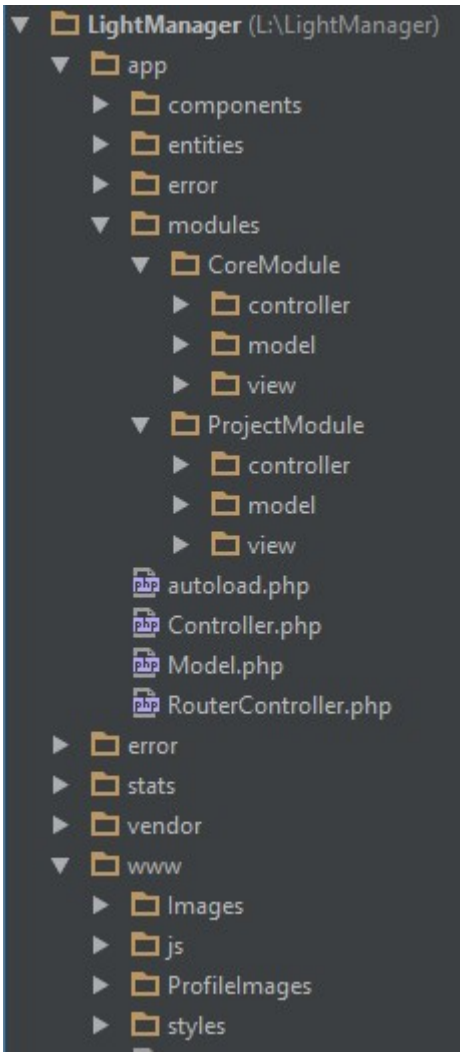
Ve vývojovém prostředí PHPStormu je po založení nové projektu třeba nastavit deployment na FTP server, kde se bude aplikace vyvíjet a testovat. V záložce "File" je možnost "New project from existing files...", která dále odkazuje na připojení k FTP, pojmenování projektu a určení jeho lokálního úložiště. Dále v je FTP nutno vybrat kořenový adresář pro projekt a tím bude složka nadřazená složce "www". Konkrétně v této práci se jedná o složku "web". Tlačítko "finish" vytvoří projekt a zkopíruje adresářovou strukturu z FTP.

V momentě kdy je projekt vytvořen, lze v záložce "Tools" zvolit možnost "Deployment", která umožňuje vytvoření a konfiguraci FTP. Ta slouží pro nahrávání souborů na server.

4.2.2 Souborová struktura

Dalším krokem je vytvoření složky s názvem "app". V té bude celá aplikace uložena. Tím se oddělí aplikace od veřejně dostupných souborů. Ve složce "app" si budou vytvořeny další složky a to "entities" a "modules". Ve složce modules budou založeny další složky s názvy modulů "CoreModule" a "ProjectModule" a v každé z nich ještě složky "model","view","controller". Oddělením souborové struktury modelu, viewu a controlleru týkající se jednotlivých modulů se zpřehlední vývoj a zároveň se moduly stanou na sobě nezávislé. Na stejné úrovni byla vytvořena složka "app", bude vytvořena ještě složka "vendor", která bude sloužit jako úložiště pro knihovny.

Složka "www" strukturuje obsah ještě na podsložky "js", "styles", "ProfileImages", "Images".



Obrázek 10: Souborová struktura projektu (zdroj: vlastní zpracování)

4.3 Vývoj Frontendu

Pro účel této práce postačí, pokud je pro každou stránku vytvořen wireframe, podle kterého se vybuduje design. Jelikož se jedná o jednoduchý design, bude nakódován při samotném vývoji. Pro převod do interpretovatelné podoby poslouží značkovací jazyk HTML a kaskádové styly.

4.3.1 Přihlašování / Odhlašování

Wireframe by měl obsahovat skicu přihlašovací obrazovky a uprostřed formulář obsahující dvě pole pro zadání emailové adresy a hesla a dále tlačítko pro potvrzení

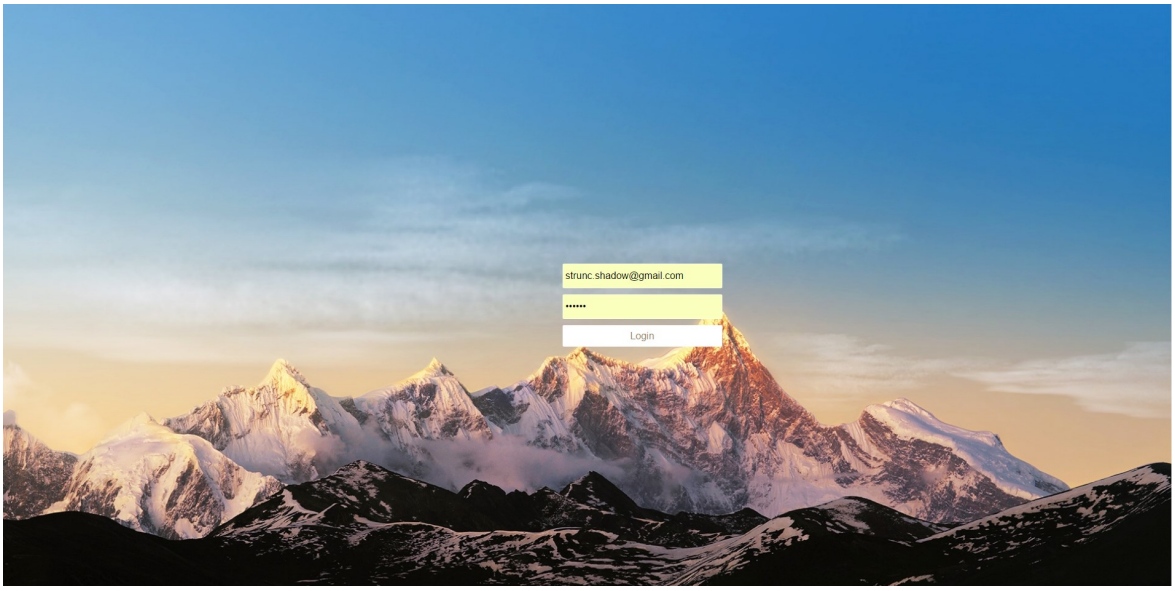
s popiskem "přihlásit se". Stejná šablona bude použita v případě nečinnosti uživatele s následkem odhlášení a přesměrování na tuto stránku se zprávou "Byli jste odhlášeni". Ve zmíněném případě je třeba označit to místo, kde se bude tato zpráva vypisovat.

Převedený wireframe do HTML souboru je pojmenován "login.phtml". Přípona "phtml" je v podstatě takovým nepsaným pravidlem, pro rozeznání zda jde o šablonu nebo skript. V nově vytvořeném souboru "app.css", budou definovány kaskádové styly (CSS) a přiřazením příslušných stylů k formuláři, aby se zobrazoval uprostřed obrazovky.

```
<body>
<div id="login-screen">
    <h3><?= $routedController->getLoginMessage() ?></h3>
    <form class="login" action="/account/prihlaseni" method="post">
        <input type="text" placeholder="username" name="user"><br>
        <input type="password" placeholder="password" name="password"><br>
        <input type="submit" value="Login">
    </form>
</div>
<script src=""></script>
</body>
```

Obrázek 11: Přihlašivací formulář v html (zdroj: vlastní zpracování)

Obohacení stránky pozadím vytvoří lepší dojem. Pro potřeby práce bude opatřen obrázek pro nekomerční použití z dostupného zdroje a vložen do složky "Images" pod názvem "login-screen.jpg". Následuje nastavení atributu stylu "background-image:" na URL obrázku "url('../Images/login-screen.jpg')". Nakonec celý soubor s css ("app.css") je umístěn do složky "styles".



Obrázek 12: Přihlašovací stránka - vzhled (zdroj: vlastní zpracování)

4.3.2 Hlavní stránka

Zde bude uplatněn stejný postup. To znamená nakreslit jednoduchý wirefram a jeho následné převedení do HTML kódu s přiřazením příslušných stylů. Soubor bude pojmenován "layout.pthml". Tato šablona se zobrazí přihlášeným uživatelům. Za zmínku stojí jeden element "div" s atributem "id" a hodnotou "container", ten bude obsahovat HTML podstránek.

4.3.3 Ostatní podstránky

Smyslem těchto podstránek je, aby se vykreslovali uvnitř hlavní stránky. Pro další postup bude vhodné použít předchozí nákres a další papír, který bude symbolizovat podstránku. Pro každou z nich bude použit stejný postup, nakreslení wireframu a převedení do html. Čítají tedy několik html souborů a to "profile.phtml" (formulář pro úpravu profilu, zobrazení profilu), "ProjectList.phtml" (seznam projektů), "NewProject.phtml" (formulář pro vytvoření projektu) a "Project.phtml" (zobrazení projektu v detailu).

4.4 Vývoj Backendu

Každý projekt potřebuje jakýsi výchozí soubor. Pro tento účel postačí vytvoření souboru "index.PHP", ze kterého bude aplikace spuštěna. Je nutné, aby byl umístěn ve složce "www", jelikož je přístupná pro všechny uživatele. Zatím zůstane soubor prázdný, bude doplněn později.

4.4.1 Vývoj MVC architektury

Dle paradigma je tvořena určitá posloupnost. Vznikne instance modelu, ta je v konstruktoru předána controlleru a ten daty naplní a vykreslí danou šablonu (view). Aby bylo zabráněno duplicitě v kódu, budou vytvořeny vzory, resp rodičovské třídy pro modely a controllery, které budou zastřešovat metody pro jejich potomky. Nakonec bude potřeba vytvořit metodu, která se bude starat o automatické načítání souborů obsahující potřebné třídy. Z tohoto důvodu se soubory, ve kterých jsou třídy umístěny budou jmenovat stejně.

4.4.1.1 Model

Dle principů OOP bude vytvořena rodičovská třída, ze které budou modely dědit případnou funkčnost. Zatím není nezbytné definovat její logiku. Soubor s třídou patří do složky "app", kde by měly být uloženy všechny MVC rodičovské třídy.

4.4.1.2 Controller a Router Controller

Stejně tak jako u modelu, je potřeba vytvořit obecnou třídu pro controllery, "Controler". Možnosti složky "app", které se zobrazí po kliknutí pravým tlačítkem, nabízejí vytvoření nového souboru "PHP Class". Pak jen zbývá vyplnit jméno souboru a třídy. Protože se jedná o obecnou třídu, bude děděním poskytovat svoje proměnné a metody ostatním controllerům, je za vhodné, aby byla abstraktní (nelze vytvářet její instance). Její vlastnosti jsou zobrazeny na obrázku 13.

```
protected $data = array();
protected $view = "";
protected $module = "";
protected $header = array('title' => '', 'keywords' => '', 'description' => '');
protected $do;
protected $components = array();
protected $dataManager;
public $mainLayout = "layout";
```

Obrázek 13: Vlastnosti třídy controller (zdroj: vlastní zpracování)

Proměnnou data reprezentuje pole, které se plní v modelu a vstupuje ve finále do view. Každý controller si může nastavit svůj view, stačí do proměnné nastavit jméno souboru, bez přípony. Proměnná "module" se bude plnit automaticky na základě

získané URL adresy. Proměnná "header" má za úkol vypisovat data do hlavičky hlavních šablon. Proměnné "dataManager" a "mainLayout", budou využity později.

Je třeba vytvořit metody potřebné pro všechny controllery, všechny jsou uvedeny na obrázku 14. Prozatím metodu construct zůstane prázdnou, její úprava bude realizována později. Další metoda "action" bude abstraktní, aby figurovala v každém controlleru a mohla být nadefinována dle potřeby. Metoda "setController" bude přijímat rodičovský controller, aby se na něj bylo možno zpětně odkazovat. Nesmí chybět logika pro vykreslování šablony, proto je zavedená metoda "render". Pomocí nativní funkce "extract" se převede pole na proměnné a vloží se soubor se šablonou. Poslední metoda "redirect", jejíž účelem je přesměrování na danou URL adresu.

```php
public function __construct($dataManager){
    $this->dataManager = $dataManager;
}

abstract function action($parameters);

public function setController($controller){
    $this->data['controller'] = $controller;
}
public function render()
{
        if ($this->view)
        {
            extract($this->data);
            require_once('../app/modules/'.$this->module.'/view/'.$this->view.'.phtml');
        }
}

public function redirect($url)
{
        header("Location: /$url");
        header("Connection: close");
        exit;
}
```

Obrázek 14:Metody rodičovské třídy controller (zdroj: vlastní zpracování)

Stejný postup pro vytvoření třídy "Controller" je aplikován na třídu "RouterController". Tato třída bude mít za úkol zpracovávat URL adresu a podle ní volat příslušné controllery. Třída "Controller" definovaná pro dědění, eliminuje duplicity proměnných a metod, jenž je využito v "RouterControlleru". Syntaxe "extends Controller" přidaná za název třídy, zajišťuje dědičnost. Dále je nadefinováno pole

"slugs". Klíče pole budou sloužit pro rozpoznání URL adresy a hodnota v klíči bude obsahovat řetězec složený z modulu, controlleru, akční metody a parametrů. Celé pole je zobrazeno na obrázku 9. Následovně musí být vytvořena zděděná metoda "action". Do ní vstupuje parametr zastupující URL adresu. Parametr projde zpracováním do provedení, všechen text za prvním lomítkem bez následných proměnných oddělené "?" a rozdělený do pole podle zbývajících lomítek. Z prvních dvou prvků v poli jsou tvořeny dvě části URL adresy do formátu "část1/část2" a dle toho jsou vyhledávány v poli "slugs" na základě klíče. Pokud existuje taková stránka jako klíč v poli "slugs", potom její hodnota různými úpravami vypovídá o jaký modul, controller a akci se jedná. Parametry se reprezentují jako zbytek URL adresy za první a druhou částí. Pokud dojde k neexistenci klíče, nenalezení metody controlleru, proběhne přesměrování na "stranka-nenalezena".

4.4.1.3 Autoload

V PHP existuje nativní funkce pro změnu načítání tříd "spl_autoload_register". Do funkce vstupuje anonymní funkce s parametrem "className". Pro získání všech potenciálních cest je vytvořeno pole "modules" osahující názvy modulů, dále pole "library" a naplním ho cestami do složek obsahující třídy MVC. Pokud by nebyla tato funkce změněna, bylo by nutné při použití jakékoliv třídy zahrnout soubor s danou třídou funkcí "include_once".

```
//INITIALIZE MODULES
for($i=0;$i<count($modules);$i++){
    $library[] = "../app/modules/".$modules[$i]."/controller/";
    $library[] = "../app/modules/".$modules[$i]."/model/";
}
```

Obrázek 15: Autoload – moduly (zdroj: vlastní zpracování)

K nalezení správné cesty je použit cyklus, kontrolující zda daná třída resp. soubor alespoň v jedné z cest existuje, pokud ano soubor se vloží. To je jeden z důvodů proč jsou pojmenovány soubory stejně jako názvy tříd.

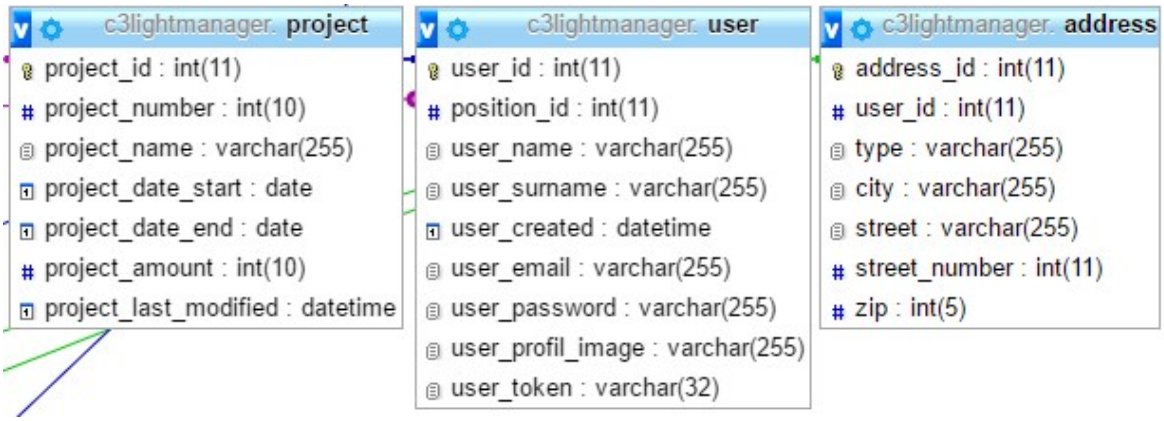
4.4.2 Instalace Tracy Debugger

Nástroj kontroluje chyby ve skriptech, zároveň sleduje výkon skriptu v milisekundách, proto není nutné na začátku a na konci každého skriptu implementovat řešení sledující výkonnost.

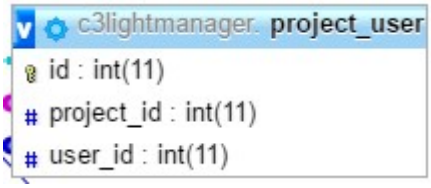
Stránka GitHub poskytne zdrojové kódy. Složka "vendor" slouží pro knihovny, proto i složka s Tracy by měla být zde. Obsahuje vlastní autoloader, tudíž není zapotřebí řešit žádné načítání.

4.4.3 Návrh a vytvoření databáze

Aplikace obsahuje třídy uživatel, uživatelova adresa, projekt. Proto se i tato struktura musí odrazit v databázi. Struktura zahrnuje tabulku "user" s atributy user_id, position_id, user_name, user_surname, user_created, user_email, user_password, user_profil_image a user_token. Dále tabulku "adress" s parametry adress_id, user_id, type, city, street, street_number, zip. Atributem user_id je tvořena reference k tabulce "user" a jejich vztah je tím definován jako "1:M", tedy jeden uživatel může mít více adres. Mezi uživatelem a projektem bude figurovat ještě jedna tabulka, která je pojmenována "project_user" a bude mezi nimi definovat vztah "M:M" s atributem "user_id" odkazující na tabulku "user" a atributem "project_id" odkazující na tabulku "project", kterou je nutno vytvořit před definováním vazby. Logika vztahu se dá vysvětlit jako, jeden uživatel se může účastnit více projektů, stejně tak projekt může být dělán více uživateli.



Obrázek 16: MySQL tabulka "user", "adress" a "project" (zdroj: vlastní zpracování)



Obrázek 17: tabulka "project_user" definující "M:M" vazbu (zdroj: vlastní zpracování)

4.4.4 Databázové řešení Session

Na serveru ve složce, kde je nainstalované PHP, je potřeba změnit v konfiguračním souboru "php.ini" způsob ukládání session ze souboru na databázi. Protože session není přímou součástí do MVC architektury, lze jí zahrnout do složky "vendor" v složce "Light", kde bude vyvíjena vlastní knihovna. Vytvořím si třídu "DBSessionHandler". Pro přepsání výchozích metod, jsou vytvořeny vlastní metody pro zápis, čtení, otevření, zničení a její údržbu. Pro přehlednost knihovny, je vhodné změnit namespace podle souborové struktury, tedy na "Light". V databázi je zapotřebí přidat tabulku, která bude reprezentovat úložiště pro session. Jelikož zatím není nikde vyvinuté připojení do databáze, zůstane prázdná a vývoj bude pokračovat později.

```
public function open($savePath, $sessionName) {
    return true;
}

public function close() {
    return true;
}

public function read($id)
{

}
public function write($id, $data) {

}

public function destroy($id) {

}

public function gc($maxlifetime) {

}
```

Obrázek 18: Session metody (zdroj: vlastní zpracování)

4.4.5 Vývoj vlastní ORM knihovny

Pro udržení konceptu adresářové struktury je potřeba vytvořit složku "ORM" vedenou pod sloužkou "Light", tím je zároveň definován namespace jako "Light\ORM". Namespace bude sloužit pro třídy, které se týkají dané problematiky a jsou vedeny pod touto složkou. Připojení do databáze bude zajišťovat třída "DatabaseWorker". Jako prevence proti opětovnému připojování bude využit návrhový vzor singleton. Jak je singletom implementován konkrétně je vidět na obrázku 19, kde ho představuje metoda "getInstance".

```
13  class DatabaseWorker
14  {
15      private static $_instance = null;
16      private static $connection = null;
17
18      public static function getInstance()
19      {
20          if (self::$_instance === null) {
21              self::$_instance = new self;
22          }
23          return self::$_instance;
24      }
```

Obrázek 19: DatabaseWorker – Singleton (zdroj: vlastní zpracování)

Hlavní účelová metoda třídy "DatabaseWorker" zahrnuje logiku připojení do databáze pomocí PDO, "ConnectDatabase". Ta bude vytvářet nové připojení a ukládat ho do vlastnosti "connection". K instanci samotného připojení je možné přistupovat další metodou "getConnection", která bude vracet soukromou vlastnost "connection", v níž bude uložena instance třídy PDO. Je potřeba pamatovat i na uzavření spojení, proto by zde neměla chybět ani metoda "CloseConnection". Jejím úkolem bude pouze nastavit vlastnost "connection" na hodnotu null.

```
self::$connection = new PDO("mysql:dbname=$database;host=$host",
                            $user, $pass,array(PDO::ATTR_PERSISTENT => true));
self::$connection->exec("set names utf8");
```

Obrázek 20: Připojení PDO (zdroj: vlastní zpracování)

Vytvoření této třídy bylo klíčové z hlediska databázových operací zajišťující funkcionalitu session. Proto je teď možné dokončení třídy "DBSessionHandler". Kontruktor zajistí připojení session do databáze. Zbývá jen dotvořit logiku předpřipravených metod. Pro správné načítání souborů ze složky "Light", je nutné

funkci starající se o autoload opatřit výjimkou, pokud se jedná o třídy, jejichž namespace začíná slovem "Light", hledej ve složce "vendor/Light". Bez této výjimky by funkce pro autoload hledala jen ve složce "tracy", kvuli implicitnímu nastavení "autoloaderu" tohoto nástroje.

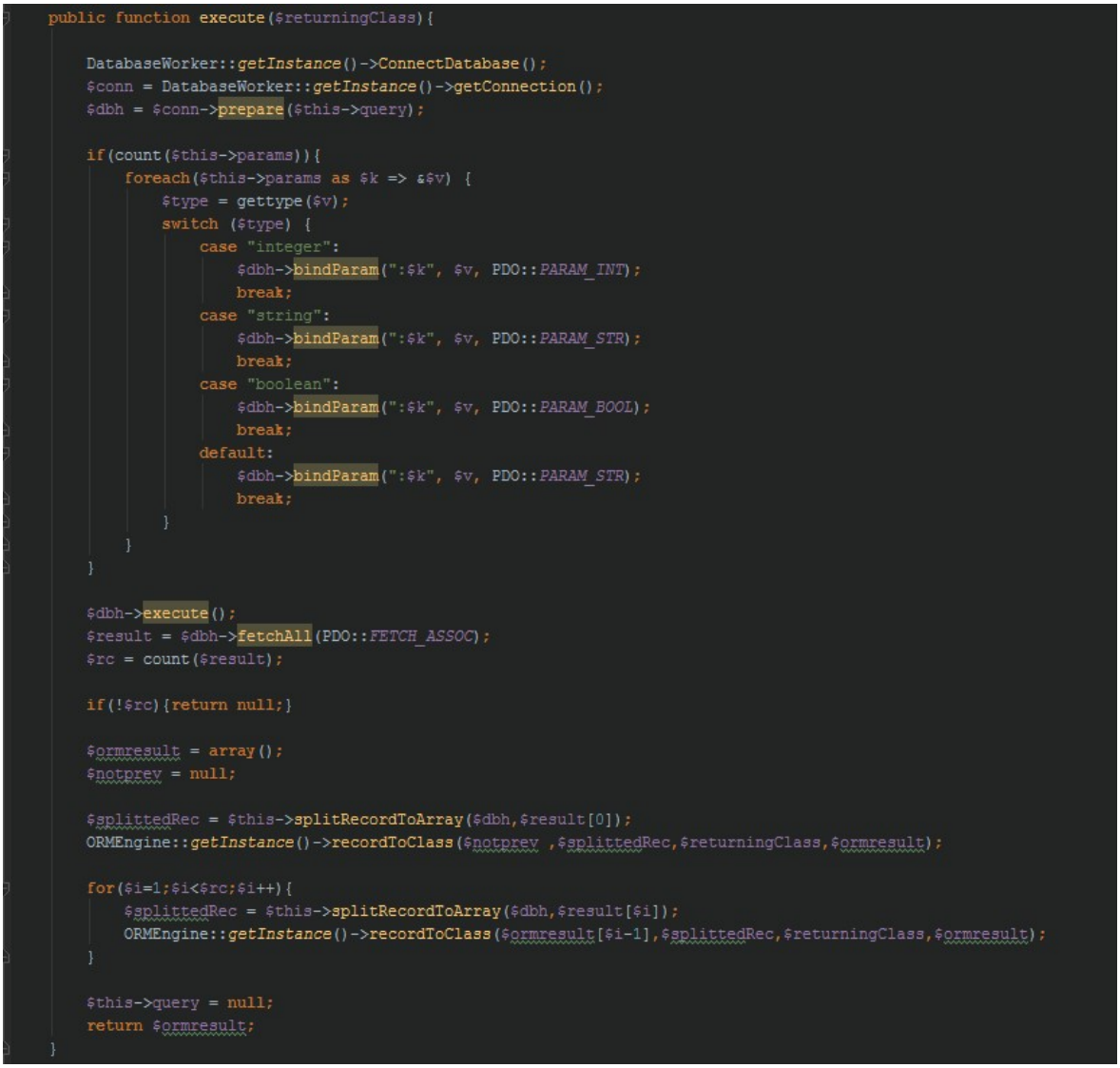
```php
class DBSessionHandler{
    private $savePath;
    private $sessionName;
    private $db = null;

    public function __construct() {
        DatabaseWorker::getInstance()->ConnectDatabase();
        $this->db = DatabaseWorker::getInstance()->getConnection();
        $this->db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
        $this->db->setAttribute(PDO::ATTR_EMULATE_PREPARES, 0);

    }
```

Obrázek 21: Session - Připojení do databáze (zdroj: vlastní zpracování)

Pro vytvoření snadno použitelného ORM, je dobré začít s logikou sestavování SQL dotazů. Třída, obsahující takovou funkcionalitu se bude řadit pod složku "ORM" s názvem "Query" podle terminologie. Metody týkající se dotazu jsou naplněny konkrétní logikou. U třídy bude fungovat tak zvané řetězení metod, kdy každá z nich bude vracet instanci, z které byla zavolána pro větší flexibilitu a úspornost kódu. Hlavní metoda stojící za odesílání SQL dotazu a vracení výsledků z databáze v objektově orientované podobě ponese název "execute". Jako parametr do ní bude vstupovat řetězec zatupující jméno třídy, která bude definovat, v jakých objektech bude výsledek reprezentován.



```
public function execute($returningClass){

    DatabaseWorker::getInstance()->ConnectDatabase();
    $conn = DatabaseWorker::getInstance()->getConnection();
    $dbh = $conn->prepare($this->query);

    if(count($this->params)){
        foreach($this->params as $k => &$v) {
            $type = gettype($v);
            switch ($type) {
                case "integer":
                    $dbh->bindParam(":$k", $v, PDO::PARAM_INT);
                    break;
                case "string":
                    $dbh->bindParam(":$k", $v, PDO::PARAM_STR);
                    break;
                case "boolean":
                    $dbh->bindParam(":$k", $v, PDO::PARAM_BOOL);
                    break;
                default:
                    $dbh->bindParam(":$k", $v, PDO::PARAM_STR);
                    break;
            }
        }
    }

    $dbh->execute();
    $result = $dbh->fetchAll(PDO::FETCH_ASSOC);
    $rc = count($result);

    if(!$rc){return null;}

    $ormresult = array();
    $notprev = null;

    $splittedRec = $this->splitRecordToArray($dbh,$result[0]);
    ORMEngine::getInstance()->recordToClass($notprev ,$splittedRec,$returningClass,$ormresult);

    for($i=1;$i<$rc;$i++){
        $splittedRec = $this->splitRecordToArray($dbh,$result[$i]);
        ORMEngine::getInstance()->recordToClass($ormresult[$i-1],$splittedRec,$returningClass,$ormresult);
    }

    $this->query = null;
    return $ormresult;
}
```

Obrázek 22: Metoda execute v třídě Query (zdroj: vlastní zpracování)

Při skládání dotazu je důležité myslet na bezpečnost, proto jakýkoliv proměnný vstup do formule je parametrizován a PDO se již o vše postará. Když jsou parametry vloženy, dotaz vrátí výsledek tvořen klasickým dvourozměrným polem, kde první index reprezentuje řádek a druhý sloupec. Jako úložiště pro nově vytvořené objekty poslouží klasické pole. Podpůrná metoda "splitRecordToArray" roztřídí výsledky z databáze podle tabulek. Do metody bude vstupovat celý řádek tedy pole s indexem 0 až X jako parametr a bude vracet dvourozměrné pole, kde první klíč bude název tabulky a druhý sloupec.

4.4.5.1 ORMEngine

Nově vytvořená třída s názvem "ORMEngine" bude zajišťovat převod výsledku z databáze ve formě pole na objekty. Využití návrhového vzoru Singleton

pro tuto třídu je spíše účelové. Není žádný důvod tvořit její instance, jen používat její metody. Klíčová je metoda "recordToClass" pro vracení výsledku v objektu. Při vymýšlení logiky je bráno v potaz, jaké případy mohou nastat pokud by se nejednalo o pouhý dotaz týkající se jedné tabulky ale o dotaz obsahující "join", který spojuje dvě tabulky. V takovém případě bude vymyšlení logiky složitější. Jde o to zjistit, v jakých formách se může výsledek z databáze objevit na to reagovat příslušnou logikou.

```php
public function recordToClass(&$prevRecord,$record, $returningClass, &$ormresult){
    $recordTables = array_keys($record);
    $table = self::toDatabaseRelatedName($returningClass);

    $is_prev = false;
    if($prevRecord==NULL){
            $returningClass = new $returningClass();
    }else{
        if(($prevRecord->getId() != $record[$table]["id"])){
            $returningClass = new $returningClass();
        }else{
            $returningClass = $prevRecord;
            $is_prev = true;
        }
    }
    foreach($recordTables as &$tablename){
        if($table!=$tablename){
            $ClassName = self::toObjectRelatedName($tablename);
            $newInstance = new $ClassName();
        }else{
            $newInstance = $returningClass;
        }
        //FILLING OBJECT
        if($table!=$tablename or !$is_prev) {
            $newInstance->loadArray($record[$tablename]);
        }
        if($table!=$tablename) {
            $fn1 = "set" . $ClassName;
            $fn2 = "add" . $ClassName;
            if (method_exists($returningClass, $fn1)) {
                $returningClass->$fn1($newInstance);
            } elseif (method_exists($returningClass, $fn2)) {
                $returningClass->$fn2($newInstance);
            }
        }
    }

    if(!$is_prev){
        $ormresult[] = $returningClass;
    }
}
```

Obrázek 23: ORM - recordToClass metoda (zdroj: vlastní zpracování)

4.4.5.2 Entity

Složka "Entities" ve složce "app" slouží pro ukládání tříd, reprezentující tabulky v databázi i s jejich vazbami. Pro účely této práce je jedná o třídy "User", "Address" a "Project". Vlastnosti jim jsou přiřazeny stejné jako atributy tabulek v databázi.

4.4.5.3 DataManager

Třída "DataManager" pomocí návrhového vzoru fasáda je schopna zastřešit celou funkčnost ORM. Uvnitř třídy jsou nadefinovány metody pro zavolání nových instancí různých tříd, například metoda volající novou instanci třídy "Query", uvedená na obrázku 24.

```
public function newQueryBuilder(){
    return new Query();
}
```

Obrázek 24: DataManager - vzor fasáda (zdroj: vlastní zpracování)

4.4.6 Řešení MVC

V tom to bodě jdou připraveny všechny prostředky k budování samotné aplikace. Soubor "index.PHP" se postará o spuštění aplikace a sledování jejího vývoje.

```
1   <?php
2   mb_internal_encoding("UTF-8");
3   include("../app/autoload.php");
4
5   include_once("../vendor/tracy/tracy/src/tracy.php");
6   use Tracy\Debugger;
7   Debugger::enable(Debugger::DEVELOPMENT);
8   Debugger::$strictMode = false;
9
10  // Create router and process current URL
11  $router = new RouterController(new Light\ORM\DataManager());
12  $router->action(array($_SERVER["REQUEST_URI"]));
13  // Render router - layout.phtml
14  $router->render();
```

Obrázek 25: Index.PHP a spuštění aplikace (zdroj: vlastní zpracování)

Aby byly základní třídy "controller" a "model" kompletní. Do obou je implementováno řešení, kontrolující zda je uživatel přihlášen. Ostatní potomci těchto tříd je pak mohou použít kdekoliv je to zapotřebí.

4.4.6.1 Vytvoření přihlašovací procedury

Jako první je vytvořen controller řídící přihlašovací proceduru uživatele a pokud je již přihlášen, odkáže ho na profil. Název by měl odpovídat obsahu nebo funkčnosti třídy, proto tedy "AuthenticationController". V adresářové struktuře by měl zaujmout místo ve složce "controller" pod složkou "CoreModule". Složka view v stejném kontextu jako složka controller by měla obsahovat šablony "layout.pthml" a "login.phtml". Controller vlastní příslušnou metodu pro přihlašování "actionLogin", ve které je nastavena vlastnost "mainLayout" na "login.phtml". Tím je řízena hlavní šablona, implicitně je nastavená na šablonu "layout". V této chvíli je aplikace schopna se spustit a zobrazit přihlašovací obrazovka. Zatím ještě nefunkční.

Pro získání dat poskytující controlleru je vytvořen model s názvem "AutenticationModel", v adresářové struktuře zaujímá místo ve složce model v stejném kontextu jako controller. Jeho hlavní metodou je "getUserByLoginCreditals" do, které vstupují přihlašovací údaje. Pokud nic nevrátí, uživatel nemá právo vstoupit do systému a do šablony se pošle zpráva "Chybné přihlašovací udaje". Pokud proces přihlášení projde v pořádku, uživatel je odkázán na jeho profil. Do pole slugs ve třídě "RouterController" je potřeba doplnit klíč "account/prihlaseni" s hodnotou "CoreModule/Authentication:login".

```
public function getUserByLoginCreditals($email, $password){
    $query = $this->dm->newQueryBuilder();
    $user = $query->select(array("user_id","user_email","user_password"))
          ->from(array("user"),false)
          ->where("user_email=:email and user_password=:password")
          ->setParameters(array("user_email"=>$email,"user_password"=>$password))
          ->execute("User");
    return $user[0];
}
```

Obrázek 26: AuthenticationModel – přihlášení (zdroj: vlastní zpracování)

```php
function actionLogin($dm,$parameters){
    if(!$this->isAuthenticateAccess()){
        if(count($_POST))
        {
            $email = $_POST["user"];
            $pass = $_POST["password"];
            $user = $this->model->getUserByLoginCreditals($email, $pass);
            if($user != null){
                $_SESSION["login"] = true;
                $_SESSION["uid"] = $user->getId();
                $token = md5(date("Y-m-d").time()."L1ghtC0d3");
                $_SESSION["token"] = $token;
                $user->setUserToken($token);

                $dm->persist($user);
                $dm->reflect();

                $this->redirect("account/profil");
            }
            $this->loginMessage = "Chybné přihlašovací údaje";
        }
```

Obrázek 27: AuthenticationController – actionLogin (zdroj: vlastní zpracování)

4.4.6.2 Modul uživatele

Ve stejném modulu by složka "view" měla obsahovat šablonu "profile". Konkrétní funkčnost by měl zajišťovat "UserController" starající se o akce uživatele. Z pravidla "RouterControlleru", pokud není žádná akční metoda nadefinována automaticky je volána metoda "action", která je povinná z pravidla rodičovské třídy "Controller". Vlastnost "view" je nastavena na hodnotu "profile", což je šablona do, které vstupují data o uživateli. V "RouterControlleru" je potřeba doplnit pole "slugs" klíčem "account/profil" a jemu příslušnou hodnotou.

```php
    function action($parameters)
    {
        if($parameters["pageAction"] == "upravit"){
            $this->actionEdit($parameters);
        }else{
            $user = $this->model->getUserByID($_SESSION["uid"]);

            $this->header['title'] = 'Profil';

            $this->data["display1"] = "none";
            $this->data["display2"] = "block";
            $this->fillViewVariables($user);
            $this->view = 'profile';
        }
    }
```

Obrázek 28: UserController - načtení dat a zobrazení (zdroj: vlastní zpracování)

Pokud URL adrese bude ještě prodloužena o text "/upravit", bude vstupovat do controlleru jako hodnota parametru nadefinovaného v poli "slugs".

4.4.6.3 Modul projektu

V podstatě se jedná o podobné operace jako v modulu uživatele. V aplikaci nyní stačí vytvářet jenom modely, šablony a controllery.

Pole "slugs" v RouterControlleru je doplněno o další prvky "projects/seznam" a "projects/novy", hodnoty jsou jim přiřazeny dle určitého vzoru definovaného od začátku vývoje tohoto controlleru.

```
"projects/seznam" => "ProjectModule/ProjectList?id",
"projects/novy" => "ProjectModule/Project:create"
```

Obrázek 29: Projekt modul - slugs prvky (zdroj: vlastní zpracování)

Modul bude obsahovat controllery "ProjectListController" a "ProjectController". Předem připravené šablony je třeba zakomponovat do adresářové struktury dle aktuálního modulu, tím je myšlena složka "view" v "ProjectModule". Modely získávající data k projektům jsou nazvány "ProjectListModel" s metodou "getAllProjects" a "ProjectModel" s metodou "getProjectById". Nejjednodušší logika patří vývoji vypisování seznamu projektů. Pokud tedy se uživatel dostane na adresu "projects/seznam", vypíšou se mu všechny projekty, ale pokud bude vyplněný parametr např. "projects/seznam/1", zobrazí se detail projektu, který má v databázi id "1". Pokud URL adresa bude "projects/novy", "ProjectController" zavolá metodu "create" a ta vyhodnotí, zda byly poslány nějaká data přes požadavek POST. Pokud pole POST zůstane prázdné, vykreslí se šablona s prázdným formulářem. V opačném případě se vytvoří nový projekt a přesměruje uživatele na seznam projektů.

5 Výsledky a diskuse

5.1 Zhodnocení

Vytvořená aplikace nyní umožňuje všechny základní operace ze souboru CRUD (Create, Read, Update, Delete), kromě mazání. Zahrnuje bezpečnostní prvek pro identifikaci uživatele jeho přihlášením. Její interní funkce poskytují uživateli možnost zobrazení a editace svého profilu, vytvoření projektu, jeho následné zobrazení v detailu a výpis všech projektů. Funkcionalita tvořící podporu pro plánování projektů je v základech vedení evidence projektů a analýzou jejich dat. Jakékoliv její rozšíření je otázkou dalšího vývoje. V porovnání se současnými databázovými systémy nedosahuje aplikace zdaleka takové funkčnosti avšak základní možnosti vývoje webové aplikace s použitím architektury MVC a knihovny ORM byli demonstrovány uspokojivě.

5.2 Možnosti pro další vývoj

Aplikace je ještě ve svých začátcích, tudíž je spousty možností jak aplikaci obohatit. První úpravy by mohly být zaměřeny k doladění ORM knihovny. Měla by být schopna i mazání dat v databázi a podporovat více relací typu "M:M". Další funkčnost by mohla obsahovat vyhledávání v aplikaci, stránkování v sekci "projekty" pro přehlednost v případě většího množství dat. Určitě by seznam projektů zpřehlednilo jejich filtrování dle uživatelem zadaných kritérií a řazení podle vybraného atributu. Projekt by měl zároveň obsahovat další parametry jako uživatelé, kteří se ho účastní nebo úkoly jim zadané. Samozřejmě by musela změnami projít i databáze, kde by bylo vhodné přidat příslušné tabulky a propojit je vazbami. V poslední řadě je možné uvažovat o vývoji dalších modulů.

6 Závěr

V této práci byl popsán a demonstrován vývoj webové aplikace pomocí konkrétních technologií. Byl obecně popsán programovací jazyk PHP, ve kterém aplikace byla vyvinuta. Dále byla představena technologie ORM a MVC, které v dnešní době tvoří standard v oblasti vývoje webových aplikací. Dále byla okrajově popsána technologie relační databáze MySQL a technologie pro interpretaci dat HTML a CSS. V praktické části byl zdokumentován vývoj aplikace využívající architekturu MVC doplněnou o abstraktní vrstvu ORM. Také byla ukázána tvorba několika tabulek v databázi uchovávající informace o uživateli a jeho projektech. Na základě hlavní funkce této aplikace, což bylo uchovávání a správa dat týkajících se projektů, byl splněn cíl týkající se této problematiky.

7 Seznam použitých zdrojů

Oracle Corporation. (nedatováno). *MySQL 5.7 Reference Manual*. Získáno 5. říjen 2016, z MySQL: http://dev.mySQL.com/doc/refman/5.7/en/

Achour, M., Betz, F., Dovgal, A., Lopes, N., Magnusson, H., Richter, G., a další. (2001). *PHP*. Získáno 25. září 2016, z PHP Manual: http://PHP.net/manual/en/

Bernard, B. (7. květen 2009). *Úvod do architektury MVC*. Získáno 5. říjen 2016, z Zdroják.cz: https://www.zdrojak.cz/clanky/uvod-do-architektury-MVC/

Berners-Lee, T., & Jaffe, J. (nedatováno). *World Wide Web Consortium (W3C)*. Získáno 26. září 2016, z W3C: https://www.w3.org/

Lecky-Thomson, E., & Steven, D. N. (2010). *PHP 6 Programujeme profesionálně.* Brno: Computer Press, a.s.

Lerdorf, R. (29. květen 2015). *PHP.* Získáno 16. září 2016, z PHP Presentation System: http://talks.PHP.net/velocity15#/wpbench

Palas, P. (duben 2001). *Historie Webu z pohledu vývojáře*. Získáno 10. září 2016, z Fakulta informatiky Masarykovy univerzity: http://www.fi.muni.cz/usr/jkucera/pv109/2001/xpalas.htm

PDO::prepare. (nedatováno). Získáno 17. září 2016, z PHP Manual: http://PHP.net/manual/en/pdo.prepare.PHP

Svačina, O. (20. září 2016). Tvorba webových aplikací v ASP.NET. Praha, Praha, Česká republika. Načteno z Tvorba webových aplikací v ASP.NET.

Team, D. (2010). *Doctrine 2 ORM 2 documentation*. Získáno 30. září 2016, z Doctrine: http://docs.doctrine-project.org/en/latest/

8 Přílohy

8.1 Zdrojový kód aplikace

Kompletní zdrojový kód aplikace je uveden v přiloženém CD.