VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

INFORMAČNÍ SYSTÉM S INTERPRETEM PRO ZPRACOVÁNÍ MĚŘENÝCH VELIČIN A JEJICH VIZUALIZACI

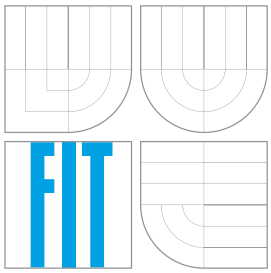
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

MARTIN ŠIFRA

BRNO 2014

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

INFORMAČNÍ SYSTÉM S INTERPRETEM PRO ZPRACOVÁNÍ MĚŘENÝCH VELIČIN A JEJICH VIZUALIZACI

INFORMATION SYSTEM WITH INTERPRETER FOR MEASURED DATA PROCESSING AND VISUALISATION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — MARTIN ŠIFRA
AUTHOR

VEDOUCÍ PRÁCE — Ing. IGOR SZŐKE, Ph.D.
SUPERVISOR

BRNO 2014

Abstrakt

Tato bakalářská práce se zabývá návrhem a implementací webového informačního systému pro správu výkonnostních testů sportovních lezců Českého horolezeckého svazu. Systém je implementován v jazyce PHP s využitím frameworků Nette a Doctrine 2 s důrazem na snadnou rozšiřitelnost a udržovatelnost. Kromě běžných operací umoňuje integrovaný interpret provádět s daty rozsáhlé výpočty. Vytvořené řešení zachovává vlastnosti tabulkového procesoru a zpřístupňuje data ze sítě Internet. Jeho použití urychluje orientaci v datech s možností vizualizace za účelem zkvalitnění tréninku závodníků.

Abstract

This thesis describes the design and implementation of web-based information system for performance tests of Czech Mountaineering Association's sport climbers management. The system is implemented in PHP using frameworks Nette and Doctrine 2 with an emphasis on easy extensibility and maintainability. Beside basic operations allows integrated interpreter to execute extensive calculations with the data. Build solution keeps features of the spreadsheet and makes data available from the Internet. Its usage accelerates orientation in data with posibility of their visualization in order to improve the training of athletes.

Klíčová slova

web, informační systém, PHP, Nette, komponenty, Doctrine, ORM, OOP, interpret, Excel, CSV

Keywords

web, information system, PHP, Nette, components, Doctrine, ORM, OOP, interpreter, Excel, CSV

Citace

Martin Šifra: Informační systém s interpretem pro zpracování měřených veličin a jejich vizualizaci, bakalářská práce, Brno, FIT VUT v Brně, 2014

Informační systém s interpretem pro zpracování měřených veličin a jejich vizualizaci

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Igora Szőkeho, PhD. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Martin Šifra
26. května 2014

Poděkování

Rád bych poděkoval panu Ing. Igoru Szőkemu, Ph.D., vedoucímu bakalářské práce, za poskytnutí odborných informací a konzultací, které přispěly ke zkvalitnění výsledné práce.

Obsah

Úvod

K porovnávání sportovních výkonů docházelo již v antickém Řecku během olympijských her. Od té doby sport urazil dlouhou cestu a z pouhé zábavy se stalo významné odvětví světového hospodářství. Vrcholově se však sportu věnuje stále jen hrstka nejobětavějších, kteří v závislosti na popularitě mezi masou získávají finance pro své sportovní odvětví. Své výkony ale srovnávají stále, ať už na stadionech nebo s tabulkami. Porovnávání výkonů sportovce ovšem nemusí být vždy jednoduché, jako například v atletice.

Sportovní lezení je mladý sport a dříve než byly objeveny objektivní techniky, existovala pro poměřování lezeckých výkonů pouze subjektivní stupnice obtížnosti. S popularizací sportovního lezení ve světě byly postupně zavedeny různé další metody měření výkonnosti a tělesných dispozic. Na rozvoji subjektivních metrik se v České republice podílí odborníci z Fakulty tělesné výchovy a sportu Univerzity Karlovy v Praze ve spolupráci s Českým horolezeckým svazem (ČHS).

ČHS využívá metrik k měření výkonnosti reprezentantů s cílem kvalitního a propracovaného tréninku tam, kde je opravdu potřeba. V případě porovnání výkonnosti s normou je možné odhalit sportovcovy slabiny a zacílit trénink na ně. K tomu jsou zapotřebí nejen dobře zpracované metody testování, ale také možnost výsledky prezentovat trenérům v pochopitelné formě. Tyto možnosti však řešení s použitím tabulky v aplikaci Microsoft Excel nenabízelo. Měření ztrácela význam a stávala se pouze akademickým pojetím sportu. Zmíněné nedostatky iniciovaly vznik této práce s cílem převést naměřené hodnoty do praxe a využít na maximum jejich potenciál.

Dostupnost informačních technologií nebyla nikdy lepší. Ke spuštění malého informačního systému není zapotřebí žádných speciálních služeb. Principy, které jsou při implementaci použity využívají stejně tak malé webové aplikace jako velké korporace v podání Facebooku, Googlu nebo Twitteru.

Cílem práce je využít naplno dostupné technologie a získat flexibilní informační systém přehledně prezentující data trenérům, ať už jsou z jakéhokoliv konce České republiky. Na druhé straně musí usnadnit práci všem, kteří se starají o testování lezců a zadávání hodnot do systému. Díky možnostem responzivního layoutů webových aplikací budou moci zadávat hodnoty již během měření z mobilních zařízení bez nutnosti tvorby nativních aplikací pro dané platformy. Aby však mohli nadobro zahodit tužku a papír musí věřit, že systém funguje, a vidět výsledky v podobě obrovské úspory času. velkou roli tak hraje i přívětivé uživatelské rozhraní, které uživatele vede systémem přesně tam, kam chce on.

Kromě vizualizace dat, musí systém zachovat možnost zpracování data na základě vzorců pro výpočet jednotlivých veličin. Součástí bude muset být interpret programovacího jazyka, jehož syntaxi pochopí během několika kroků kždý, kdo bude chtít se systémem pracovat.

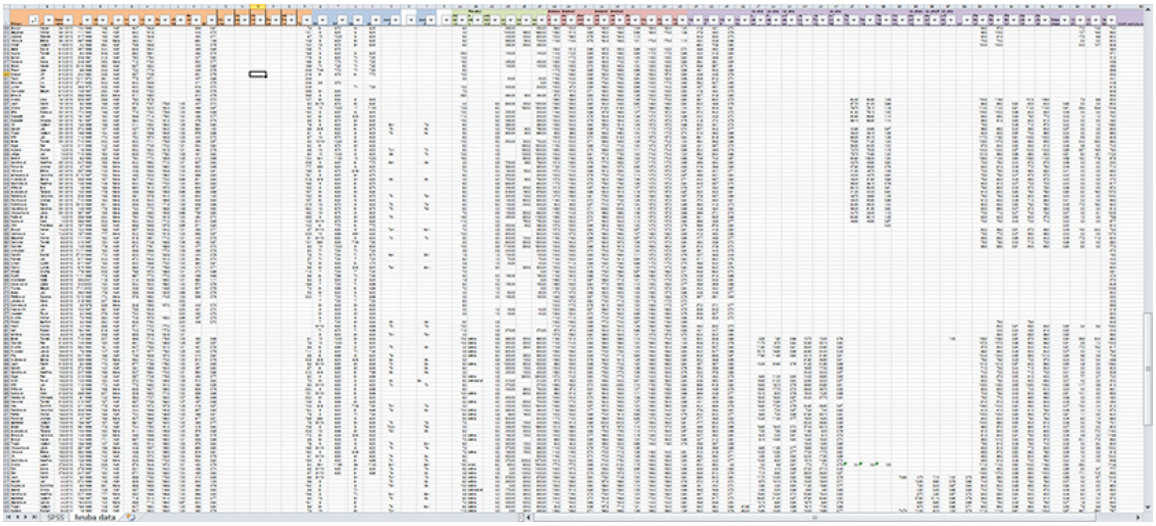
Kapitola 1

Specifikace požadavků

Na počátku procesu specifikace požadavků stojí popis stavajícího řešení používaného po celou dobu testování výkonnosti reprezentačních lezců. Popis je souhrnem informací získaných na konzultacích s uživateli a analýzou souborů s daty. Na jeho základě jsou stanoveny požadavky na služby, které by měl budoucí systém poskytovat.

1.1 Stávající řešení

Testování všech reprezentantů probíhá v laboratořích FTVS UK nebo na umělých lezeckých stěnách několikrát do roka. Každý sportovec má vyhrazen zvláštní záznamový arch, do kterého jsou postupně zapisovány jeho výkony. Údaje z archů jsou po skončení manuálně přepisovány do prostředí tabulkového procesoru Microsoft Excel. Všechny hodnoty jsou shromažovány v jediném souboru, který obsahuje od roku 2008 více jak 500 záznamů jednotlivých lezců, každý až s padesáti hodnotami. Obrázek 1.1 představuje část souboru se všemi měřenými veličinami (sloupce) a zlomek záznamů lezců (řádky) v něm uložených. Je rozdělen na tři listy. První obsahuje výsledné hodnoty (bez vzorců) všech testů vypočítané na základě těch naměřených, které jsou součástí druhého listu.



Obrázek 1.1: Naměřené hodnoty v prostředí Microsoft Excel

Vzorce jso často jedinou metodou, kterou běžní uživatelé znají, pro výpočet hodnot. Namísto využití přehledných maker psaných v jazyce VBA[1] používají vzorců i k rozsáhlejším výpočtům. Zjistit význam takového vzorce může být velice obtížné:

```
=IF(AS426="";(($G426-AU426)+($G426-AT426))/(2*$G426);((AS426-AU426)+(AS426-
AT426))/(2*$G426))
```

Po náročném manuálním zpracování naměřených hodnot, trvajícím i déle jak měsíc, jsou z výsledků vybrány pouze záznamy sportovních lezců ČHS. Filtrovaná data tvoří samostatný soubor předávaný Komisi pro sportovní lezení, která jej opět rozdělí podle osob pro trenéry, rodiče nebo konkrétní lezce. Rozeslání výsledků pro více než čtyři desítky adresátů probíhá výhradně v příloze elektronické pošty v souboru s příponou *.xls*.

Vysoký podíl manuální práce s daty činí stávající řešení neefektivním s vysokou pravděpodobností chyb. Nejdéle trvajícím procesem je vkládání naměřených hodnot do počítače a následná manipulace s hodnotami pro další zpracování, protože pro minimalizaci zanesení chyb je vždy nutné vložená i výsledná data několikrát přepočítávat a pečlivě kontrolovat. Bezpečnostním rizikem stávajícího řešení je bezesporu nekontrolovatelné sdílení citlivých osobních údajů v souborech s výslednými hodnotami. Nemožnost porovnávání výkonů v čase nebo ve vztahu k normám vede bez odborných znalostí k nebezpečné interpretaci, kontraproduktivnímu trénování a v nejvážnějších případech i ke zraněním.

1.2 Definice požadavků

Definice funkčních a nefunkčních požadavků rozvíjí předchozí analýzu stávajícího řešení s ohledem na návrhy budoucích uživatelů. Podrobněji popisuje nejdůležitější vlastnosti nového systému nebo omezuje proces jeho vývoje pouze na určité části.

1.2.1 Informační systém

Celý současný proces od zaznamenávání po rozesílání výsledků měření urychlí sofistikovaný informační systém. Při jeho návrhu a implementaci bude brán ohled především na bezpečnost uchovávaných dat, jejich dostupnost napříč platformami ze sítě Internet, a modularitu pro snadný budoucí rozvoj.

Dostupnost Možnost vkládat data musí být zajištěna z jakéhokoliv místa s přístupem do sítě Internet. Přímé zadávání naměřených hodnot do systému a zobrazení výsledků v reálném čase značně urychlí celý proces vyhodnocování testů. Nebude již nutné manuálně přepisovat hodnoty z tištěných archů a přepočítávat výsledky, zda odpovídají očekávání. Zobrazení výsledků jednotlivých testů bude možné okamžitě po vložení potřebných hodnot, nikoliv až s měsíčním zpožděním.

Bezpečnost Manuální rozesílání výsledků nahradí webové rozhraní s uživatelským přístupem. Na základě rozdílných přístupových práv bude mít po přihlášení každý uživatel možnost shlédnout pouze své vlastní výsledky, nebo v případě trenérů výsledky svých svěřenců. Absolutní kontrolu nad záznamy získají pouze osoby provádějící měření nebo členové Komise sportovního lezení a samozřejmě administrátor.

[1] VBA – VisualBasic for applications

Rozšiřitelnost Vnitřní struktura systému musí vykazovat vysokou kvalitu programového kódu a umožňovat nejen snadnou udržovatelnost, ale i rozšiřitelnost o nové funkce.

1.2.2 Práce s daty

Vlastnosti nového systému musí zachovat stávající možnosti zpracování dat a přinést řadu vylepšení především v orientaci v jejich velkém množství. Naměřené hodnoty musí být možné identifikovat podle zvolené metody testování, testovaného lezce a měření, jehož jsou součástí. Měření přestavuje pohled na osoby a jejich naměřené hodnoty v určitém časovém období, například před začátkem závodní sezóny.

Zpracování Systém umožní oddělenou správu typů měřených hodnot a hodnot samotných. Mimo přímého zadávání hodnot je nutné implementovat rozhraní pro výpočet výsledků odvozených z naměřených hodnot nahrazující vzorce v prostředí tabulkového procesoru Microsoft Excel. Zadávání vzorců bude na bázi jednoduchého programovacího jazyka. Protože programování nepatří k základním dovednostem běžného uživatele počítače, bude součástí práce srozumitelná dokumentace jazyka s ukázkami základních programových konstrukcí.

Porovnávání Jednou z nejočekávanějších vlastností systému je porovnávání výsledků testů v čase. Pro práci s velkým množstvím hodnot je také nutností pokročilé filtrování zobrazovaných hodnot. Přestože je práce nebude obsahovat, jsou součástí plánu vývoje. Jednotlivé fitry bude možné ukládat a opakovaně aplikovat na výstupní data.

Vizualizace Pro lepší názornost bude možné výsledky testů porovnávat také v grafech zobrazujících opět vývoj výkonu v čase.

Import a export Návaznost na minulá testování lezců zajistí rozhraní pro import stávajících dat z externího souboru. Zatím jediným exportním rozhraním, které však nebude součástí práce, je vystavení protokolu o provedeném testování. Protokol bude zobrazovat především aktuálně naměřené hodnoty vztažené k normám výkonů v paprskových grafech, aby byly na první pohled patrné jejich odchylky od norem.

Kapitola 2

Použité technologie

V závislosti na mnoha faktorech jsou v práci použity následující technologie a externí knihovny. Se záměrem usnadnění vývoje a vysoké kvality výsledného produktu byly pečlivě vybírány na základě objektivních metrik i subjektivního posouzení autora. Podrobnému rozboru jsou však podrobeny pouze části nezbytné k pochopení fungování systému.

2.1 PHP 5

Za implementační byl zvolen jazyk PHP[1], protože jde především o nejrozšířenější (82%)[2] a nejdostupnější platformu na tvorbu webových aplikací. Díky své popularitě je v PHP vytvořena řada frameworků a nástrojů, které vývoj webové aplikace značně urychlí, usnadní a vedou vývojáře k dodržování zásad čistého zdrojového kódu [7].

2.2 Git

Efektivní týmový vývoj aplikací vyžaduje použití verzovacího systému, který slouží k uchovávání historie prováděných změn ve zdrojovém kódu a jejich šíření mezi ostatní členy týmu. Avšak i v případě práce pouze jednoho vývojáře na projektu existují nesporné výhody verzování.

Distribuovaný systém správy verzí Git klade důraz především na jednoduchost a flexibilitu používání. Absence hlavního serveru umožňuje provádění operacích v lokálním repozitáři bez nutnosti jejich šíření. Účelem jeho využití v této práci je kromě revize změn i vytvoření záložní kopie celého projektu na vzdáleném serveru, ze kterého mohou být data kdykoliv obnovena.

2.2.1 GitHub

Komunitní web GitHub[3] umožňuje vytvoření veřejných i privátních Git repozitářů. Slouží jako sociální síť pro vývojáře sdílející své projekty, ohlašující chyby nebo podávající podněty na vylepšení. Pro účely práce je využíváno především snadné spravování Git repozitářů a jejich napojení na Composer popsané v následující části 2.3.

[1] PHP – rekurzivní zkratka PHP: Hypertext Preprocessor, původně Personal Home Page

[2] Statistika serverových programovacích jazyků (17.5.2014): http://w3techs.com/technologies/overview/programming_language/all

[3] GitHub – https://github.com

2.3 Composer

Composer je důležitý nástroj pro správu závislostí v PHP. Umožňuje definovat libovolně složité závislosti na externích knihovnách a automatizuje jejich instalaci a aktualizaci. Jeho používáním zajistíme vždy aktuální verze knihoven jediným příkazem, bez nutnosti pracného stahování a kopírování do projektu:

```
$ composer update
```

Ve výchozím nastavení vyhledává Composer balíčky v centrálním repozitáři Packagist[4] a instaluje je do adresáře `vendor` uvnitř projektu. Nejsnadnějším způsobem publikování balíčků je přímé napojení Git repozitářů umístěných na serveru GitHub na službu Packagist, která zapřístupní vždy aktuální verzi Git repozitáře.

2.4 Nette Framework

Nette Framework[5] (zkráceně pouze Nette) je populární nástroj k usnadnění a zefektivnění tvorby webových aplikací vytvořený v jazyce PHP. Mezi jeho hlavní přednosti patří:

- Kvalitní objektový návrh vedoucí i nezkušené programátory k čistému návrhu s důrazem na budoucí rozšiřitelnost aplikace, postavené na architektonickém vzrou Model-View-Controller (MVC).
- Rychlá nastavení všech částí aplikace pomocí konfiguračních souborů ve formátu NEON.
- Podpora techniky vkládání závislostí (dependency injection) mezi jednotlivými komponentami aplikace.
- Excelentní šablonovací systém Latte.
- Automatické ošetření bezpečnostních děr a zabezpečení proti útokům jako např. Cross-site scripting (XSS), Cross-request forgery (CSRF), session hijacking, session fixation a dalším.
- Kvalitní ladicí nástroje pro rychlé odhalování chyb.
- Automatické načítání tříd celé aplikace.
- Vysoký výkon díky využití vyrovnávací paměti cache.

V následujících částech textu jsou rozebrány nejdůležitější vlastnosti a komponenty frameworku využité ve zbytku práce. Uvedené informace platí pro Nette Framework ve verzi 2.1.3 pro PHP 5.3 a novější uvolněné 12. května 2014.

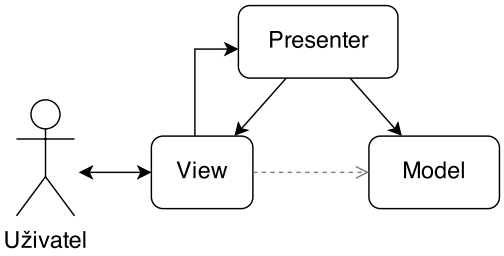
[4] Packagist – https://packagist.org
[5] Nette framework – http://nette.org

2.4.1 MVC a presentery

Webová aplikace napsaná v Nette stojí na známém architektonickém vzoru Model-View-Controller (MVC). Vzor obecně rozděluje aplikaci na tři základní vrstvy:

- **Model** – Datová a funkční vrstva poskytující svůj vnitřní stav svým vlastním rozhraním. Při správné implementaci neví modelová vrstva o existencí dalších částí systému a musí na nich fungovat zcela nezávisle. Model především spravuje přístup k databázi nebo k jiným datovým uložištím.
- **View** (pohled) – Vrstva zobrazující výsledek vykonávání aplikace. Ve frameworcích je nejčastěji reprezentována šablonovacím systémem. Nejedná se tedy pouze o výstupní kód, jak může být mylně uváděno, ale o celý mechanismus jeho generování.
- **Controller** – Výkonná vrstva zajišťující propojení zbylých dvou. Po zpracovávání a vykonání požadavků uživatele iniciuje překreslení daného pohledu.

V praxi se vyskytuje mnoho derivátů vzoru MVC. Jedním z nich je architektonický vzor Model-View-Presenter (MVP), jehož schéma je na obrázku 2.1, využívá také Nette. Presenter v něm deleguje schopnost plně zpracovávat uživatelský vstup na vrstvu View, se kteou je velice úzce spjat. Není tedy již nutná přímá vazba View na Model a všechna potřebná data jsou získávána přes Presenter.



Obrázek 2.1: Schéma architektonického vzoru MVP

2.4.2 Konfigurace a dependency injection

Ke konfiguraci Nette aplikace slouží soubory v deklarativním formátu NEON[6]. Na počátku běhu aplikace je tento soubor včetně závislostí zkompilován do PHP třídy uložené v cache. Při běhu aplikace reprezentuje tato třída *systémový kontejner*.

Nejdůležitější částí konfiguračního souboru pro tuto práci je možnost vytváření objektů služeb (services) a továren (factories). V případě služeb se jedná o objekty vytvářené dle vzoru *Sigleton* - v aplikaci existuje pouze jedna instance této třídy [4]. Naopak továrny vychází z návrhového vzoru *Factory Method* [2] a po jejím volání je vytvořena vždy nová instance požadované třídy.

Vytvořeným službám můžeme v konfiguračním souboru předávat jiné služby a parametry manuálně, praktičtější je ale využít tzv. auto-wiring[7], který umí rozpoznat požadované závislosti a při vytváření instance je automaticky vložit do konstruktoru.

[6] NEON – http://ne-on.org
[7] Auto-wiring – http://doc.nette.org/cs/2.1/configuring#toc-auto-wiring

Princip získávání závislostí v rámci systémového kontejneru nazýváme Dependency injection (DI). Jedná se o techniku, která zcela bezpečným a automatizovaným způsobem řeší dostupnost služeb ve třídě vytvořené tímto kontejnerem. Získávat závislosti lze těmito způsoby:

- **Konstruktor** – Nejčistším způsobem je předávání závislostí jako argumentů konstruktoru dané instance, ve kterém je předaná reference na instanci přiřazena do privátní proměnné.
- **Setter** – Druhým, méně vhodným způsobem je volání setteru dané instance, která zajistí vložení reference do privátní proměnné. Toto řešení však vyžaduje další podpůrný kód, který bude setter nad každou závislostí automatizovaně volat.
- **Property** – Nevhodnou metodou, která porušuje zapouzdření, je vkládání závislostí přímo do veřejných proměnných instance dané třídy.
- **Anotace** – Použitím anotace `@inject` a uvedením typu služby u privátní proměnné dané instance třídy dědící od `Nette\Application\Presenter` dosáhneme vložení požadované závislosti do této proměnné. Jedná se však pouze o úsporu času programátora. Na pozadí Nette volá settery, které závislost do proměnné vloží.

2.4.3 Přihlašování a oprávnění uživatelů

Ke všem službám se sdíleným přístupem neodmyslitelně patří správa uživetelského přístupu. Nette řeší všechny základní problémy s přihlašováním i oprávněním uživatelů implementací následujících částí popsaných detailněji v oficiální dokumentaci[8]:

- **Autentikátor** – První na řadu přijde autentikátor zajišťující autentizaci. Jedná se o proces ověření, zda je uživatel opravdu tím, za koho se vydává. Obvykle se využívá uživatelského jména a hesla. Autentikátor je třída implementující rozhraní `Nette\Security\IAuthenticator` obsahující jedinou metodu `authenticate()` ověřující údaje nejčastěji proti databázi.
- **Identita** – Výsledkem úspěšného přihlášení je instance třídy implementující rozhraní `Nette\Security\IIdentity`. Identita je udržována v session pomocí třídy `Nette\Security\User` a je dostupná buď ze systémového kontejneru nebo přímo v Presenteru voláním `$this->getUser()`. Každý uživatel disponuje jednou nebo více rolemi, umožňující přesnější řízení oprávnění nezávislé na uživatelském jméně.
- **Autorizátor** – Jde o objekt implementující rozhraní `Nette\Security\IAuthorizator` s jedinou metodou `isAllowed()`. Jejím úkolem je ověřit, zda má přihlášený uživatel oprávnění provést určitou operaci (provillege) s určitým zdrojem (source). Zdroje představují logické celky aplikace a operace činnost, kterou s nimi můžeme vykonat.

  Autorizátor využívá seznam všech rolí, zdrojů a jim povolených operací zvaný *access control list*[9] (ACL), který je možné definovat staticky nebo dynamicky. Jeho pohodlnou statickou definici provedeme v konfiguračním souboru[10]. Naopak dynamický ACL umožňuje změnu oprávnění přímo v aplikaci a potřebné údaje jsou uloženy v databázi.

[8] Přihlašování & oprávnění uživatelů – http://doc.nette.org/cs/2.1/access-control/
[9] Access control list – seznam pro řízení přístupu
[10] Statický ACL – http://pla.nette.org/cs/staticke-acl/

2.4.4 Tokenizer

Součástí knihovny `Nette\Utils` je mimo jiné nástroj `Tokenizer` sloužící k rozdělení textového řetězce v posloupnost tokenů na základě regulárních výrazů. Posloupnost tokenů uloženou v poli lze následně předat nástroji `TokenIterator`, který umožňuje jejich snadné procházení a porovnávání. Přestože se jedná o velice užitečný nástroj, je jeho dokumentace[11] umístěna pouze v repozitáři GitHub.

2.5 Doctrine 2

Práce s databází je v mnoha případech rutinní záležitostí. Její nevýhodou je však nepřirozená reprezentace dat. V relační databázi jsou data reprezentována relacemi za pomoci cizích klíčů. Ovšem v reálném světě žádné takové vazby neexistují. Tuto nevýhodu odstraňuje objektově relační mapování (ORM), které data z relační databáze převádí na objekty (entity) a umožňuje s nimi pracovat přirozeně.

Nette bohužel nemá vlastní ORM vrstvu a jeho implementaci proto zajišťuje knihovna `Kdyby\Doctrine`[12], která kromě implementace standardní Doctrine 2 přináší několik vylepšení.

Následující části popisují základy použití Doctrine 2. Blíže popisuje použití ORM vrstvy seriál na serveru Zdroják.cz[13] a samozřejmě kompletní dokumentace[14].

2.5.1 Entity

Základem celé ORM vrstvy jsou entity reprezentující objekty reálného světa. Jedná se o běžné instance tříd dědící z třídy `Kdyby\Doctrine\Entities\BaseEntity`. Chráněné proměnné takové třídy jsou poté vlastnostmi daného objektu a mohou také vytvářet vazby na jiné objekty. Specifikace jednotlivých vlastností entity je prováděna na základě anotací.

2.5.2 Entity manager

Entity samy o sobě nedisponují žádnými metodami umožňující přístup k databázi. Pouze zapouzdřují data a implementují například základní validaci vlastností. O načítání, ukládání a mazání entit z databáze se stará *entity manager* (EM). V případě Nette je EM službou dostupnou ze systémového kontejneru, kterou můžeme libovolně vkládat v podobě závislosti jiným službám.

Abychom splnili základní principy vzoru MVC, musí v systému existovat modelová vrstva, která bude zapouzdřovat metody pro práci s databází. Nejedná se ovšem pouze o jednu třídu, ale o celou řadu tříd, které jsou vkládány do presenterů a poskytují základní i složité metody pro práci s entitami.

[11] Nette\Tokenizer – https://github.com/nette/tokenizer/blob/master/readme.md
[12] `Kdyby\Doctrine` – https://github.com/kdyby/doctrine/
[13] Seriál o Doctrine 2 – http://www.zdrojak.cz/serialy/doctrine-2/
[14] Dokumentace Doctrine 2 – http://docs.doctrine-project.org/projects/doctrine-orm/en/latest/

2.6 HTML, CSS a JavaScript

2.6.1 jQuery

jQuery je JavaScriptová knihovna s podporou všech dnes používaných webových prohlížečů oddělující chování webových stránek od HTML. Díky své syntaxi usnadňuje programátorům přístup k objektům DOM. Rozšiřuje jazyk JavaScript o snadnou správu událostí a velice zjednodušuje asynchronní komunikaci se serverem na technologii AJAX.

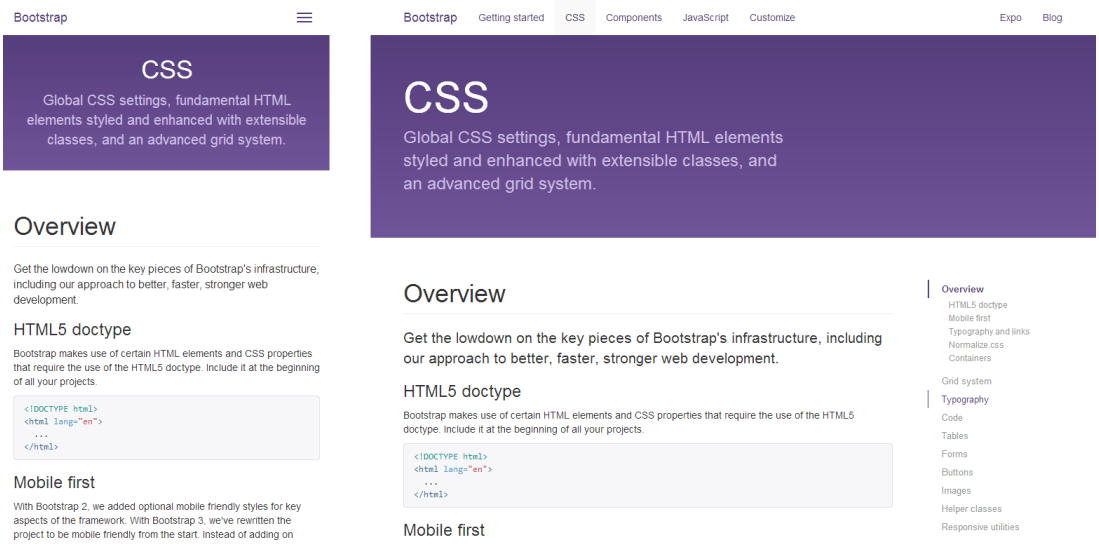
2.6.2 Bootstrap 3

Bootstrap je kolekce nástrojů k efektivní tvorbě moderních webových aplikací. Obsahuje šablony prvků založené na HTML a CSS, ze kterých snadno seskládáme šablonu výsledné stránky. Usnadňuje rozvržení stránky pomocí mřížky, tvorbu nabídek, nadpisů, formulářů a dalších komponent uživatelského rozhraní. Lze jej použít k rychlému prototypování uživatelského rozhraní i nasazení v ostrém provozu. Díky své jednoduchosti a širokému využití je knihovna velice populární a rozvíjena o další komponenty.

Responzivní layout

Se vzrůstající návštěvností webových aplikací z mobilních zařízení je nutné přizpůsobit vzhled malým displejům pro snazší ovladatelnost aplikace. Díky responzivnímu rozvržení webové stránky není nutné vytvářet speciální mobilní verzi. Na základě rozpoznaného rozlišení jsou prvky uživatelského rozhraní přeskupeny, zvětšeny nebo skryty tak, aby bylo možné aplikaci pohodlně ovládat i na malém displeji.

Součástí mřížky frameworku Bootstrap je i možnost změny šířky jednotlivých bloků umístěných v mřížce podle detekovaného rozlišení. Pouhými CSS třídami tak vytvoříme aplikaci pohodlně ovladatelnou nejen ze stolního počítače, ale i mobilního telefonu nebo tabletu. Rozdíly v zobrazení webové stránky využívající responzivní layout ukazuje orázek 2.2.



Obrázek 2.2: Zobrazení webu využívající responzivní layout na mobilním telefonu (vlevo) a stolním počítači (vpravo).

2.7 CSV

CSV[15] je jednoduchý souborový formát pro uložení tabulkových dat v prostém textu. CSV soubor se skládá z řádků, představujících řádky tabulky, a hodnot v řádcích oddělených čárkami představující sloupce. V případě, že potřebujeme v hodnotě uchovávat čárku, je nutné hodnotu uzavřít do uvozovek nebo použít jiný oddělovač, nejčastěji středník nebo tabulátor. Uvozovek je zapotřebí pro hodnoty obsahující zalomení řádku.

Následující příklad demonstruje, jak jednoduchý je převod tabulky do formátu CSV:

Celé jméno	**Výška**	**Hmotnost**
Jan Novák	189	80
Petr Kolář	175	72
Eva Svobodová	165	52

Tabulka 2.1: Data připravená pro převod do formátu CSV

Po převodu hodnot v tabulce 2.1 do formátu CSV získáme soubor s následujícím obsahem:

```
Celé jméno,Výška,Hmotnost
Jan Novák,189,80
Petr Kolář,175,72
Eva Svobodová,165,52
```

S daty uloženými ve formátu CSV můžeme snadno pracovat, avšak je nutné správně nakonfigurovat nástroj, kterým je budeme zpracovávat, aby nedošlo k chybnému převodu. Formát CSV není standardem a může se vyskytovat v mnoha podobách – s jiným separátorem nebo se záhlavím tabulky na prvním řádku.

[15] CSV – Comma-separated values, hodnoty oddělené čárkou

Kapitola 3

Návrh

Při vývoji menších a středních softwarových projektů bývá návrh často opomíjen a programátoři usedají za klávesnice bez širší koncepce budoucí práce. Věří, že lze problémy řešit až během implementace, nebo vidí rozvržení sil jako ztrátu času. Důvodů, které je vedou k nerozvážnosti, může být ale více. Všechny jsou ovšem krátkozraké a ústí ve špatná a jednostranná řešení, na která nelze v budoucnu navázat. Dostatečný návrh by měl být naopak zárukou kvalitního výstupu práce a úspory vložených prostředků.

Návrh této práce se soustředí především na modularitu informačního systému, efektivní práci s daty uloženými v databázi a uživatelsky přívětivý programovací jazyk zpracovávající uložené hodnoty.

Jedním z hlavních požadavků na výsledný produkt je dostupnost uložených dat ze sítě Internet a práce s nimi nezávislá na koncovém počítači. Proto je celý systém realizován jako webová aplikace umožňující vzdálený přístup k informacím odkudkoliv.

Pro lepší pochopení některých částí je v návrhu využito standardizovaného grafického jazyka UML[1] a jeho diagramů [1].

3.1 Informační systém

Ve středu pozornosti konceptu informačního systému jsou vždy data. Jejich sběr, zpracovávání, přenos a uchovávání za účelem prezentace uživateli udává jasný směr celému návrhu. Výsledkem je soubor prostředků a metod, které s nimi umožňují provádět zmíněné operace.

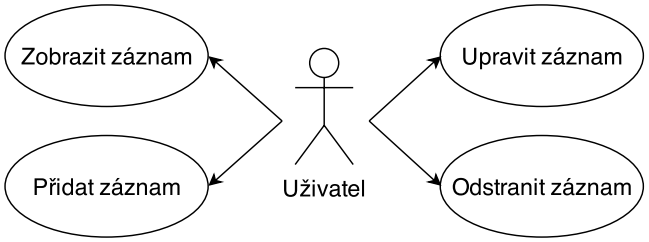
Přestože se data se často liší vnitřní strukturou, požadujeme nad nimi vykonávat stejné operace. Každá operace sestává z vizuálních komponent pro možný uživatelský vstup a metod tyto komponenty zpracovávající. V případě špatného návrhu by to znamenalo opakování aplikační logiky pro každou datovou strukturu zvlášť. Dobrým návrhem vytvoříme jednoduchou konvenci a minimalizujeme opakování zdrojového kódu. V případě nutnosti zásahu do systému provádíme změny pouze na jednom místě. Tím eliminujeme potřebné zdroje a šanci zanesení nových chyb do systému na minimum.

3.1.1 Základní operace s daty

Operace, bez kterých se žádný systém neobejde, zobrazuje diagram případů užití (use case) na obrázku 3.1. Tyto operace mohou být implementovány hned na počátku a budou vždy

[1] UML – Unified Modeling Language, jednotný jazyk pro modelování

využity. Některé z nich mohou také využívat stejné vizuální komponenty. Například formulář pro vložení nového záznamu by měl být využit i při jeho úpravě.



Obrázek 3.1: Diagram případu užití se základními operacemi s daty

Vložení Bez možnosti vytvoření nové datové struktury a jejího uložení do úložiště, by nebylo možné do systému vkládat nová data.

Zobrazení a úprava K propojení pouhé prezentace a úpravy dat nás vede především uživatelská přívětivost systému. Bez nutnosti měnit kontext okna mohou uživatelé okamžitě změnit potřebné hodnoty a uložit je.

Mazání Odstranění dat ze systému je naprosto běžnou operací. Přestože se zdá velice triviální, může nést rizika v podobě vzájemného propojení dat v úložišti. Buď mohou po smazání zůstat v úložišti struktury bez jakékoliv reference, které je nutné vyhledat a smazat dodatečně, nebo dojde k odstranění dat sdílených větším počtem jiných struktur.

3.2 Uložení dat

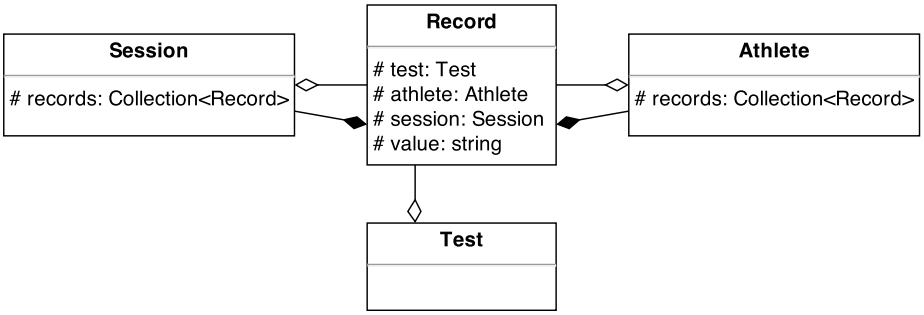
Mnohým programátorům objektově relační mapování přináší problémy s návrhem struktury dat v systému. Nedokáží se vzdát faktu, že za ORM existuje další vrstva pro uložení dat v podobě relační databáze. Na tu je v případě práce s ORM lepší úplně zapomenout a uvažovat o ni pouze jako o černé skříňce.

Díky ORM můžeme velice přesně reflektovat pořadavky na práci s daty vycházející z objektů reálného světa. Lze tak dosáhnout rychlého návrhu, bez nutnosti složitých analýz a transformací přes ERD[2] k relační databázi. Data jsou podstatou celého informačního systému, a proto je dobré věnovat značné úsilí k maximálnímu usnadnění manipulace s nimi.

3.2.1 Záznamy

Nejdůležitějším nositelem informace v systému je entita `Record` (Záznam) představující jednu buňku tabulky stávajícího řešení. Vztah s entitou `Session` (Měření) zajistí možnost výběru záznamů podle konkrétního měření. Vazba na entitu `Athlete` (Sportovec) identifikuje měřenou osobu a entita `Test` určuje měřenou veličinu. Obrázek 3.2.

[2] ERD – Entity-relationship diagram, ER diagram



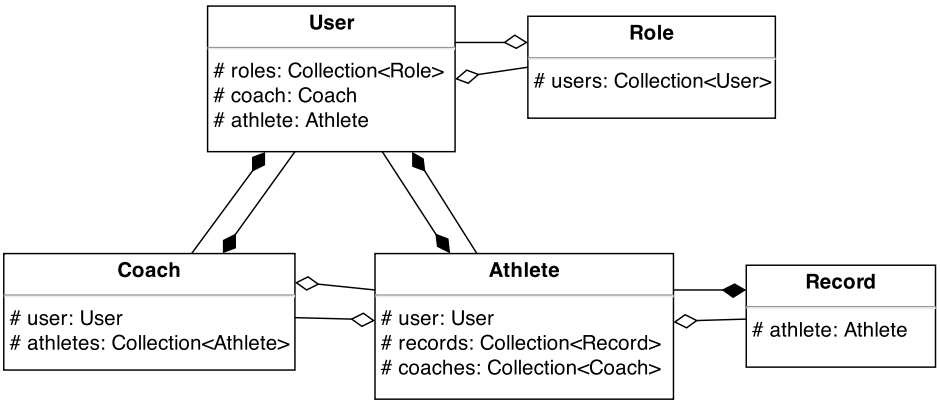
Obrázek 3.2: Minimální diagram tříd uloženého záznamu

3.2.2 Uživatelé

S využitím entit Doctrine 2 lze snadno navrhnout univerzální strukturu pro správu uživatelských účtů a rolí. Ve středu stojí entita `User` (Uživatel) obsahující všechny důležité údaje o uživateli systému (přihlašovací jméno, heslo, e-mailovou adresu atd.) vázaná na entitu `Role` vazbou *ManyToMany*. Jeden uživatel tak může zastávat více rolí.

Pro další role existují samostatné entity umožňující uchovávat rozšiřující informace o uživateli. Tyto entity jsou na entitu navázány obousměrnou vazbou *OneToOne*. Protože je vazba ve směru z entity `User` využívána zřídka, je její načítání optimalizováno nastavením `fetch = "LAZY"`. Naopak vazba z druhé strany je využívána velice často. Díky nastavení vazby `fetch = "EAGER"` proto dojde v každém SQL[3] dotazu k automatickému propojení relací pomocí syntaktické konstrukce `JOIN`.

Těchto výhod využívají entity `Athlete` a `Coach`. `Athlete` představuje závodníka a `Coach` trenéra s vazbou na své svěřence – entity `Athlete`. Propojení entit nutných pro správu uživatelského přístupu popisuje diagram tří na obrázku 3.3.



Obrázek 3.3: Minimální diagram tříd propojení entit typů uživatelů

[3] SQL – Structured Query Language, strukturovaný dotazovací jazyk

3.3 Interpret

Tvorba velice specifického interpretu s sebou přináší řadu otázek již na počátku jeho návrhu. Ze specifikace požadavků v kapitole 1 je patrné, že se jedná o klíčovou součást, bez níž by byl systém pouze skladištěm naměřených hodnot. Důležitým faktem je nutnost zachovat rozsáhlé možnosti tabulkového procesoru Microsoft Excel a zpřehlednit uživatelům zadávání vzorců, obsahujících často rozsáhlé větvení.

Návrh tvoří odpovědi na rozhodují otázky z následujícího seznamu a popisuje návrh základních částí interpretu:

- Běh interpretu zajistí server nebo klient?
- Jakým způsobem vykonávat zdrojový kód?
- Jakou zvolit syntaxi, aby byla blízká běžným uživatelům?

3.3.1 Serverová aplikace

Na prvním místě stojí rozhodnutí, zda by měl interpret pracovat na straně klienta nebo serveru. Klientská aplikace s sebou přináší moderní řešení cloudových služeb, kdy je na klienta přenesen zdrojový kód (nejčastěji JavaScript) s daty, který následně asynchronně komunikuje se serverem. Při bližší analýze však zjistíme, že je takovéto řešení mimo rozsah této práce.

Pro pohodlnou práci s větším objemem dat potřebujeme na straně klienta rozhraní (v ideálním případě MVC framework), které zpřístupní data jinak, než prostým procházením DOM[4] výsledného HTML dokumentu. Musíme se tedy zabývat nejenom jeho nastudováním, ale také napojením na existující aplikaci vytvořenou především v PHP.

Velkou roli u aplikací běžících ve webovém prohlížeči hraje výkon a podpora JavaScriptu. Je nutné zajisti vysokou míru optimalizace, především z důvodu požadavku běhu na pomalejších mobilních zařízeních.

Na základě výše uvedených důvodů je interpret implementován jako samostatná PHP knihovna využívající naplno výkon PHP a možností ukládání výsledků interpretace do sdílené cache.

3.3.2 Interpretace

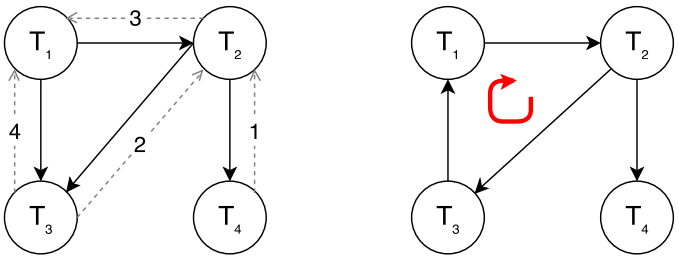
Dlouhou dobu přetrvávala myšlenka, že bude nutné zdrojový kód nejen analyzovat, ale také interpretovat. Jednalo by se však o interpretování interpretovaným PHP, což se jeví jako značné mrhání výkonem. Ideálním řešením je však převod zdrojového kódu na nativní PHP kód již během analýzy, kterého využívá také Nette při tvorbě DI kontejneru z konfiguračního souboru ve formátu NEON popsaného v sekci 2.4.2. Aby nebylo nutné provádět opakovaně analýzu nezměněného zdrojového kódu, je výsledek analýzy ukládán do paměti cache.

Specifickou částí, která již není součástí knihovny interpretu, ale služby začleňující interpret do systému, je předávání kontextu okolních hodnot do právě zpracovávaného kódu. Nejprve je nutné zajistit, že kód neobsahuje cyklické závislosti. Stejnou kontrolu provádí i tabulkové procesory, aby nedošlo k zacyklení výpočtu. V teoretické rovině lze tuto problematiku popsat orientovaným grafem závislostí jednotlivých vzorců.

[4] DOM – Document Object Model, objektový model dokumentu

V případě acyklického grafu nalezneme řešení problému postupným výpočtem hodnot, jejichž vzorce jsou závislé pouze samy na sobě, směrem k těm, které využívají závislostí více.

Cyklus v orientovaném grafu znázorněný na obrázku 3.4 lze odhalit prohledáváním grafu do hloubky. Nejjednodušším algoritmem je postupné označování bodů, které jsme při prohledávání navštívili. Pokud některý bod navštívíme znovu, nalezli jsme cyklus. Implementací takového algoritmu může být jednoduchý zásobník v podobě pole `array` a funkce `in_array($stack)`, která najde již uloženou hodnotu.



Obrázek 3.4: Graf závislostí s náznakem řešení (vlevo) a cyklický graf (vpravo).

3.3.3 Jazyk M

Většina dnes používaných procedurálních jazyků používá syntaxi podobnou jazyku C. Je snadno čitelná a především zažitá. Snad každý, kdo přišel do styku s programováním, využil jazyk podobné syntaxe. Naopak pro uživatele, který nikdy neviděl programový kód, bude z počátku velkou neznámou jakýkoliv programovací jazyk.

Syntaxe jazyka pro zpracování měřených veličin s označením M vychází původně ze skriptovacího jazyka Python, který je známý svým úsporným zápisem umožnujícím rychlé psaní zdrojových kódů. Stejné výhody nabízí i jazyk M. Kromě odsazení řádků, rozlišujícího jednotlivé bloky kódu, není programátor nucen psát zbytečné závorky a terminální znaky, které jsou v konečném důsledku vždy nahrazeny zalomením řádku. Výhodou vnitřní podobnosti syntaxe procedurálních jazyků je i snadná transformace do jazyku jiného. Syntaxe je tak ovlněna podobou jazyka PHP, C i Python zároveň.

Jazyk je dynamicky typovaný a umožňuje práci se speciálními konstantami, plněnými hodnotami až před vykonáním kódu. Konstant je v této situaci využito pro předání kontextu hodnot ostatních veličin, vždy pouze pro konkrétní osobu.

Určitý vliv na podobu jazyka má i deklarativní SQL. V rozhraní PDO[5] jazyka PHP máme možnost vkládat do SQL parametry z aktuálního kontextu PHP pomocí neterminálů začínajících dvojtečkou. Stejnou podobu mají i speciální konstanty.

Srovnáním podoby vzorců a nového jazyka M získáme jasný názor na jeho přehlednost. První fragment následujícího kódu provádí triviální výpočet relativní veličiny *Handgrip* (úchop) na základě naměřených hodnot pro pravou a levou ruku a hmotnost:

```
=((AP423+AQ423)/2)/G423
```

```
return ((:handgrip-l + :handgrip-r) / 2 ) / :hmotnost
```

[5] PDO – PHP Data Objects

Druhý fragment provádí přepočet na relativní veličinu otevřeného úchopu:

```
=IF(AS423=""; (($G423-AU423)+($G423-AT423))/(2*$G423);((AS423-AU423)+
(AS423-AT423))/(2*$G423))
```

```
if :hmotnost-s-vestou
  return  (2*:hmotnost - :otevreny-l - :otevreny-r)/2*:hmotnost
else
  return  (2*:hmotnost-s-vestou - :otevreny-l - :otevreny-r)/2*:hmotnost
```

Gramatika

Pro názorné vyjádření bezkontextové gramatiky jazyka M je vhodné použít rozvinutou Backusovu-Naurovu formu[6] (EBNF) [3]. Z velké části vychází gramatika ze výše zmíněných jazyků a liší se jen minimálně.

Lexikální analýza

S využitím nástroje `Tokenizer` z části 2.4.4 není zapotřebí vytvářet rozsáhlý konečný automat pro náčítání jednotlivých znaků. Na základě regulárních výrazů jsou pomocí něj rozpoznávány rovnou celé terminály i neterminály.

Syntaktická analýza

Proces syntaktické analýzy provádí kontrolu zdrojového kódu vůči předem dané gramatice. Srdcem celého interpretu je syntaktický analyzátor. Jeho volání probíhá rekurzivně nad polem tokenů získaných nástrojem `Tokenizer` a kromě samotné analýzy provádí generování výsledného PHP kódu [6].

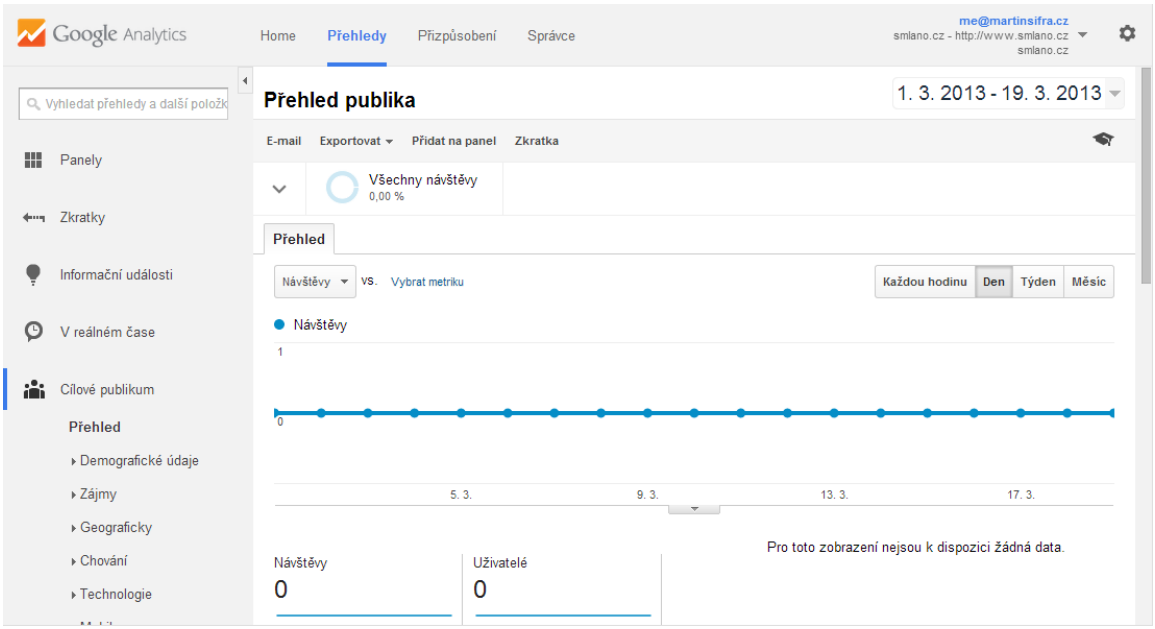
3.4 Uživatelské rozhraní

Uživatelské rozhraní je jediná část celé práce, kterou má možnost vidět koncový uživatel. Přes všechnu snahu, kvalitní objektově orientovanou implementaci a ORM pro dokonalé zpracování dat, může špatně navržené uživatelské rozhraní odradit uživatele od používání systému. Ten se rád vrátí ke svému stávajícímu řešení, které je sice neefektivní, ale dobře jej zná, a ví, jak s ním pracovat.

Klišé v podobě intuitivních a inovativních řešení uživatelských rozhraní slyšel snad každý. Málo kdy se ale opravdu podaří vyvinout systém, který uživateli padne. Většinou za takovým systémem stojí velké korporace, které mají prostředky nejen na vývoj, ale především dlouhodobé testování a ladění výsledného rozhraní.

Při vývoji menšího systému je vhodné vycházet ze zaběhnutých struktur grafického rozhraní a nenutit uživatele při ovládání přemýšlet [5]. Inspirací pro grafické rozhraní této práce jsou systémy, které denně používají miliony lidí na celém světě – aplikace společnosti Google, jejichž vzhled dnes určité části webového designu udává směr vývoje. Obrázek 3.5 zobrazuje jednoduchý vzhled aplikace Google Analytics pro měření návštěvnosti webových stránek:

[6] EBNF – Extended Backus-Naur Form, rozvinutou Backusova-Naurova forma



Obrázek 3.5: Grafické uživatelské rozhraní aplikac Google Analytics

Základem celého rozhraní jsou tři části:

- **Horní lišta** – Obsahuje v levé části logo s odkazem pro přechod na domovskou stránku aplikace následované odkazy na důležité části aplikace. V části u pravého okraje se nachází menu pro správu vlastního uživatelského účtu a jeho nastavení.
- **Levé menu** – U levého okraje stránky se nachází kompletní nabídka s odkazy ve stromové struktuře.
- **Obsah** – Okno s obsahem dává uživateli díky jasnému nadpisu přesně vědět, kde se nachází. Polohu lze často doplnit drobečkovou navigací. Poloha uživatele v aplikaci je jedním z důležitých faktorů kvality uživatelského rozhraní. [5] Uživatel se nikdy nesmí cítit v systému ztracen.

  Nadpis následuje často lišta s tlačítky pro vyvolání akcí nad zobrazenými daty a samozřejmě samotný obsah v podobě dat.

3.4.1 Odvolání provedené akce

Uživatelé neradi přemýšlí a často bez rozvahy provedou operaci, kterou nemohou vrátit zpět. Dřívější koncept, jak uživatele uchránit před provedením nevratných akcí, bylo potvrzení každé takové akce. Výsledek takového přístupu je však zcela opačný. Uživatel totiž bez přemýšlení potvrdí cokoliv, co mu vyskočí na obrazovce.

Moderní aplikace nabízí možnost odvolání skoro jakékoliv provedené akce. Lze vrátit zpět smazaný článek nebo omylem odeslaný e-mail. Stejný přístup musí implementovat i tento informační systém, aby nerozvážní uživatelé nepřicházeli o data a nebyli při tom obtěžováni potvrzovacími vyskakovacími okny.

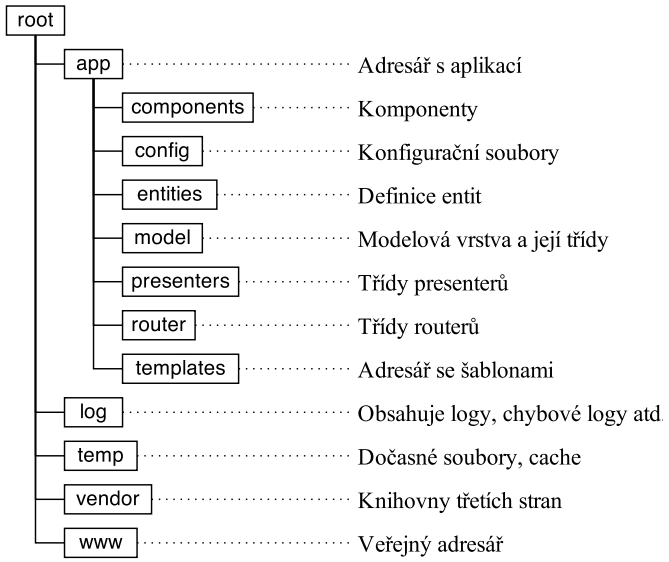
Kapitola 4

Implementace

Následující kapitola popisuje implementaci nejdůležitějších částí celé práce na základě specifikace požadavků a předchozího návrhu. Vývoj probíhal na osobním počítači s operačním systémem *Microsoft Windows 8.1* ve vývojovém prostředí *NetBeans 8.0*[1]. Serverová část aplikace je naprogramována ve skriptovacím jazyce *PHP 5.4* za využití frameworku *Nette 2.1.2*. Běhovým prostředím je server *Apache* nakonfigurovaný spolu s databázovým systémem *MySQL* balíkem *EasyPHP*[2] ve verzi 13.1.

4.1 Struktura informačního systému

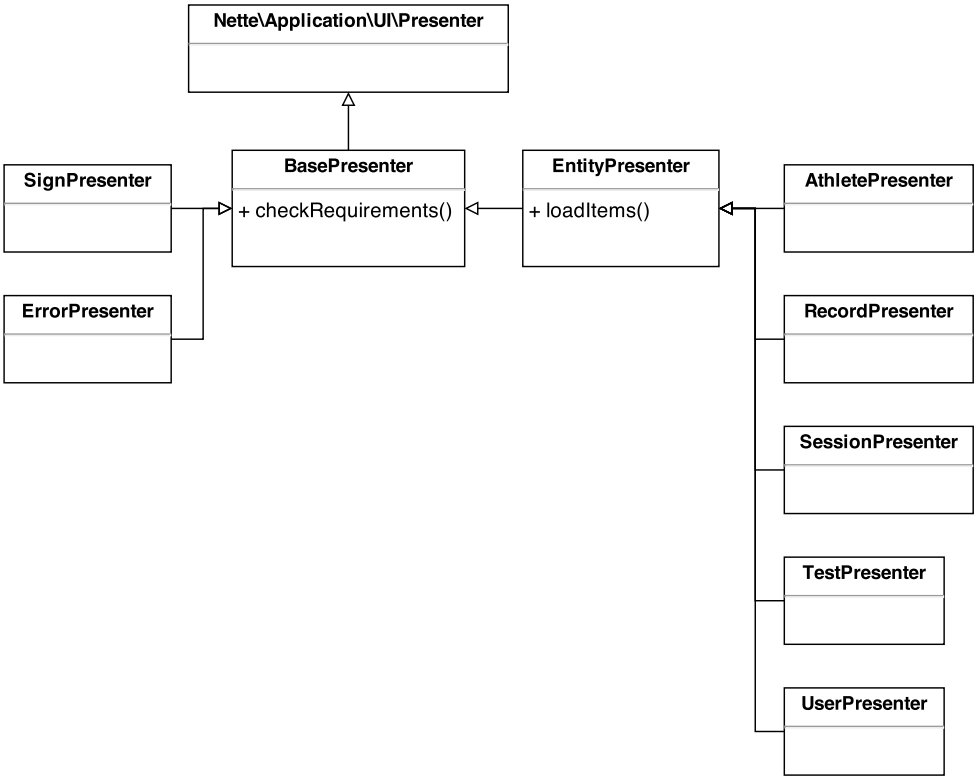
Základní hierarchie jmenných prostorů a tříd vychází především ze základní struktury použité v Nette[3] a z části kopíruje strukturu adresářů na disku z obrázku 4.1.



Obrázek 4.1: Adresářová struktura aplikace

[1] NetBeans – https://netbeans.org
[2] EasyPHP – http://www.easyphp.org
[3] Adresářová struktura Nette – http://doc.nette.org/cs/2.1/presenters#toc-adresarova-struktura

Presentery tvoří jednoduchou strukturu, protože se jedná o systém s jediným účelem a potřeba vytvářet zanořené moduly. Diagram tříd na obrázku 4.2 zobrazuje základní hierarchii, vytvořenou díky dědičnosti. Pro přehlednost nezahrnuje metody pro práci s daty (akce a signály presenterů). Pokud není uvedeno jinak, jsou presentery součástí jmenného prostoru `App\Presenters`.



Obrázek 4.2: Minimální diagram tříd presenterů informačního systému

4.2 Práce s entitami

Na základě návrhu, popsaného v sekci 3.1, bylo nutné zaměřit se při implementaci především na kvalitní provedení částí systému pracující s daty. Základ koncepce tvoří presentery, pro každou entitu jeden (viz obrázek 4.2), zpracovávající požadavky na základní operace s daty z části 3.1.1.

Konvence, vzniklá refaktorováním[4] postupně rozvíjejícího se systému, přinesla v první fázi přesun zpracování formulářů pro práci s daty z presenterů do komponent. Dále byly tyto komponenty sloučeny do jediné. Některá logika však musela zůstat v presenteru, a proto vznikl abstraktní `EntityPresenter` – rodič všech presenterů pro práci s entitami. Výsledek umožňuje na základě šablony vytvořit uživatelské rozhraní pro práci s jakoukoliv entitou během chvíle bez nutnosti udržovat duplicitní kód.

[4] Refaktorování – Proces provádění změn ve zdrojovém kódu takovým způsobem, že nemají vliv na vnější chování, ale vylepšují jeho vnitřní strukturu s minimálním rizikem vnášení chyb.

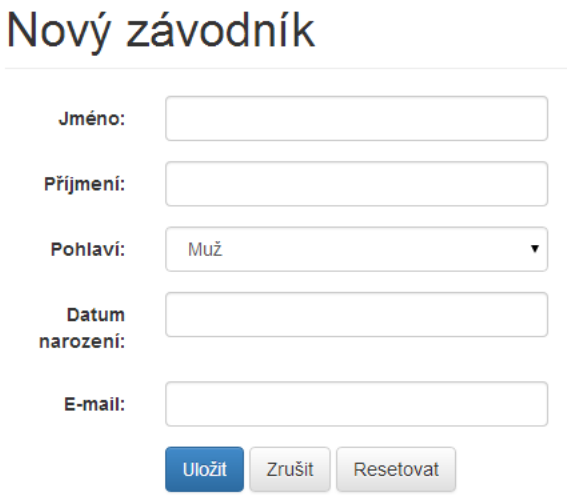
4.2.1 Komponenty jako služby

Komponenty pro práci s daty vyžadují často předání velkého počtu závislostí. V případě, že bychom komponenty připojovali k aplikaci přímo v presenteru, museli bychom tyto závislosti předat nejříve jemu a poté až do komponenty v její továrnničce.

S použitím auto-wiringu stačí pouze zaregistrovat tovární rozhraní jako službu v DI kontejneru. Odtud je komponenta vložena do presenteru jako jediná závislost. V její továrničce v presenteru poté zavoláme jedinou metodu rozhraní komponenty – `create()` a zbytek práce odvede Nette. Na základě anotace `@return` továrního rozhraní komponenty rozpozná, kterou komponentu má vytvořit, vytvoří ji a připojí k presenteru.

4.2.2 Komponenta `EntityControl`

Jmenný prostor komponent je rozdělen na podprostory na základě názvů presenterů, které tyto komponenty, potažmo entity, využívají. Součástí každého takového podprostoru je komponenta `EntityControl` zpracovávající požadavky na vložení nových dat nebo jejich editaci. Komponentu tvoří především formulář s poli korespondujícími s vlastnostmi entity. Formulář pro vložení a úpravu informací o závodníkovi ukazuje obrázek 4.3.



Obrázek 4.3: Formulář pro vložení a úpravu vlastností entity `Athlete`

Při použití komponenty `EntityControl` je do presenteru vložena pouze jediná závislost v podobě jejího rozhraní. Její továrnička obsahuje také pouze jediné volání metody `create()` (viz 4.2.1). Vše ostatní je zapouzdřeno v komponentě – na jiném místě zdrojového kódu mimo presenter.

Komponenty jsou také vzdálenými potomky třídy `Nette\Object`, která využívá přetížení metody `__call()` a lze tak volat setter pouhým přiřazení do veřejné proměnné. Protože `EntityControl` je potomkem třídy `Base\EntityControl` využijeme tohoto usnadnění a v případě, že chceme entitu upravovat, provedeme volání setteru `setEntity()` jehož parametrem je právě tato entita. Celá entita (reference na ni) je jí předávána z důvodu úspory počtu dotazů do databáze. Entitu, kterou editujeme, je totiž zapotřebí načíst již během kontroly požadavku, zda vůbec existuje. V setteru se také automaticky provede načtení výchozích hodnot do formuláře.

Zkrácený zápis v presenteru je poté s využitím metody `loadItem()` pro kontrolu existence entity v databázi velice úsporný a přehledný:

```
1 public function actionDetail($id)
2 {
3     // Načtení entity z databáze - loadItem(model, id)
4     $record = $this->loadItem($this->records, $id);
5     // Předání entity šabloně pro vykreslení nadpisů stránky apod.
6     $this->template->record = $record;
7     // Předání entity komponentě EntityControl
8     $this['entity']->entity = $record;
9 }
```

Zdrojový kód 4.1: Předání entity komponentě

Výhodou existence předka všech komponent pro práci s entitami je také možnost umístit do něj metodu `render()` obsahující kód pro vykreslení šablony. Šablona je v případě těchto komponent vždy stejná a obsahuje jediné makro {`control form`}, které způsobí vykreslení formuláře.

4.3 Autorizace

Jak již bylo zmíněno v části 2.4.3, Nette poskytuje pohodlné rozhraní pro správu uživatelských oprávnění. Jedinou částí, která zůstává v režii programátora, je volání metody `isAllowed()` nad správným zdrojem a prováděnou operací.

Díky rozšíření jazyka PHP, které Nette implemetuje[5] můžeme k rozpoznávání zdrojů a operací pohodlně využít anotací ve stylu phpDocumentor[6], jejchž použití je následující:

```
1 /**
2  * @secured
3  * @resource('record')
4  * @privilege('detail')
5  */
6 public function actionDetail($id)
7 {
8     ...
9 }
```

Zdrojový kód 4.2: Použití anotací phpDocumentor

Výskyt anotace `@secured` samotné umožňuje přístup pouze přihlašenému uživateli nazávisle na jeho roli a prováděné operaci.

Třída `Nette\Application\UI\Presenter`, ze které dědí `BasePresenter` – rodič všech presenterů aplikace – nabízí metodu `checkRequirements()`, která je volána na počátku běhu presenteru, kdy je kontrola oprávnění nejvýhodnější. Metodu `checkRequirements()` je však nutné přepsat a upravit, aby využívala anotace zmíněné ve zdrojovém kódu 4.2. Samotné načtení anotací je zcela intuitivní:

[5] Rozšíření jazyka PHP – http://doc.nette.org/cs/2.1/php-language-enhancements/
[6] phpDocumentor – http://www.phpdoc.org

```php
public function checkRequirements($element)
{
    $secured = $element->getAnnotation('secured');
    $resource = $element->getAnnotation('resource');
    $privilege = $element->getAnnotation('privilege');
    ...
}
```

Zdrojový kód 4.3: Načtení anotací při autorizaci

V případě nežádoucího přístupu je vyvolána výjimka `Nette\Application\ForbiddenRequestException`, která je v produkčním režimu odchycena a proběhne přesměrování uživatele na chybovou stránku s kódem 403.

4.4 Uživatelské rozhraní

Následující sekce zahrnuje nejen výslednou implementaci důležitých vizuálních částí systému, ale popisuje také řešení častých chyb vývojářů webových systémů. Kvalitní grafický vzhled nemusí zákonitě znamenat uživatelsky přívětivý systém, pokud bude pomalý nebo nelogicky sestavený.

4.4.1 Rychlost načítání stránky

Rychlost zobrazení webové aplikace je jedním z důležitých kritérií hodnocení kvality služby. Pro vykreslení celé stránky jsou často zapotřebí externí soubory s kaskádovými styly (CSS) a klientským JavaScriptem, které mohou vykreslení zpomalovat. Při velkém počtu externích zdrojů dochází k dlouhé prodlevě mezi obdržením odpovědi v podobě HTML stránky ze serveru a plným vykreslením stránky.

Protože prohlížeče zpracovávají HTML soubor postupně shora dolů, je jednou z významných technik, jak vykreslování urychlit, vhodné rozmístění odkazů na externí zdroje.

Umístěním CSS souborů do značky `<head>` upřednostníme jejich načtení, aby uživatel viděl přibližný vzhled stránky již od počátku a mohl jím již procházet. Pokud dojde ke zpoždění načítání souborů určujících vzhled stránky, může uživatel čekat až několik vteřin před stránkou bez viditelných informací.

Přesunutím všech souborů s JavaScriptem ze značky `<head>` na úplný konec HTML souboru do značky `<script>` zpozdíme jejich zpracování až na dobu, kdy je stránka vykreslena. V takovém případě je nutné skripty spustit až po načtení stránky v metodě `onLoad()`, v případě použití knihovny jQuery v metodě `$(document).ready()` nebo zkráceně `$()`.

WebLoader

Externí soubory jsou načítány dalšími HTTP požadavky a jejich vysoký počet také zpomaluje vykreslování požadované stránky. Při použití externích knihoven počet souborů, na kterých je webová aplikace závislá, prudce roste. V takovém případě je vhodné soubory sloučit a *minifikovat* – odstranit z nich bílé znaky nebo dokonce přejmenovat JS proměnné na kratší názvy.

Rozšíření WebLoader automatizuje tento proces a generuje sloučené a minifikované soubory podle konfigurace rozdělené na oddíly pro CSS a JS soubory. Oddíly jsou dále děleny balíčky představujícími vždy jeden výsledný soubor.

```
css:
    default:
        files:
            - screen.css
            - bootstrap.css
        filters:
            - @wlCssFilter
            - @cssMinFilter
        fileFilters:
            - @lessFilter

 js:
    default:
        files:
            - jquery.min.js
            - bootstrap.min.js
            - nette.ajax.js
            - netteForms.js
```

Výsledkem je minimální počet externích souborů navázaných na výslednou HTML stránku. Generování nových souborů probíhá pouze v případě změny v některém z původních.

Přestože je WebLoader užitečným nástrojem s mnoha možnostmi, je jeho implementace v případě základní konfigurace zbytečně složitá. Pro každý balíček zpracovávající soubory je nutné vytvořit zvláštní komponentu v `BasePreseteru`. Balíčky se vkládají do šablony s pomocí makra `cotrol`.

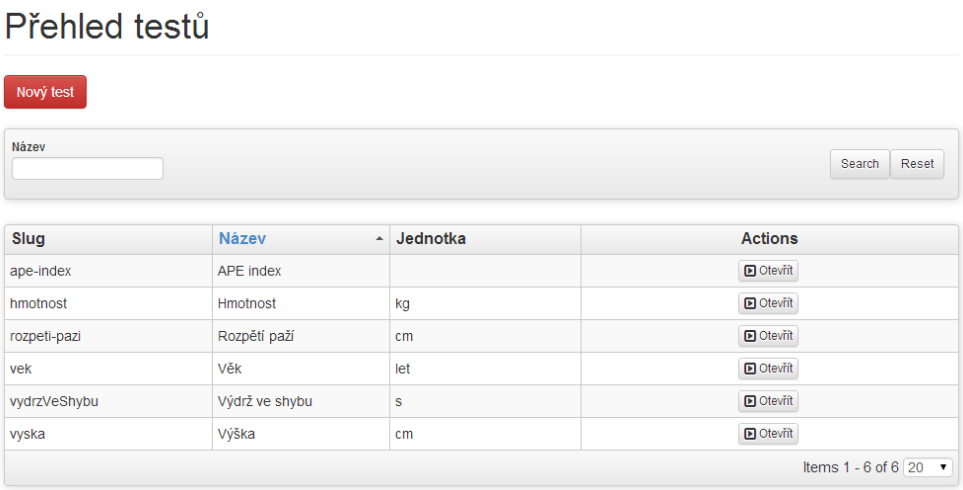
4.4.2 Grido

Základní vizualizace dat v tabulkách je nejdůležitější částí celého uživatelského rozhraní informačního systému. V podstatě se jedná vždy o stejnou tabulku na plněnou pouze jinými daty. Abychom při implementaci neduplikovali zdrojový kód, je využito externí knihovny Grido[7], která velice usnadní právě vytváření pohledů nad daty. Mezi její přednosti patří:

- Snadná konfigurace a napojení na model, podpora Doctrine 2.
- Řazení a stránkování výsledků pouze na základě dodaného modelu.
- Rychlé vytvoření filtrů nad daty s podporou našeptávání.

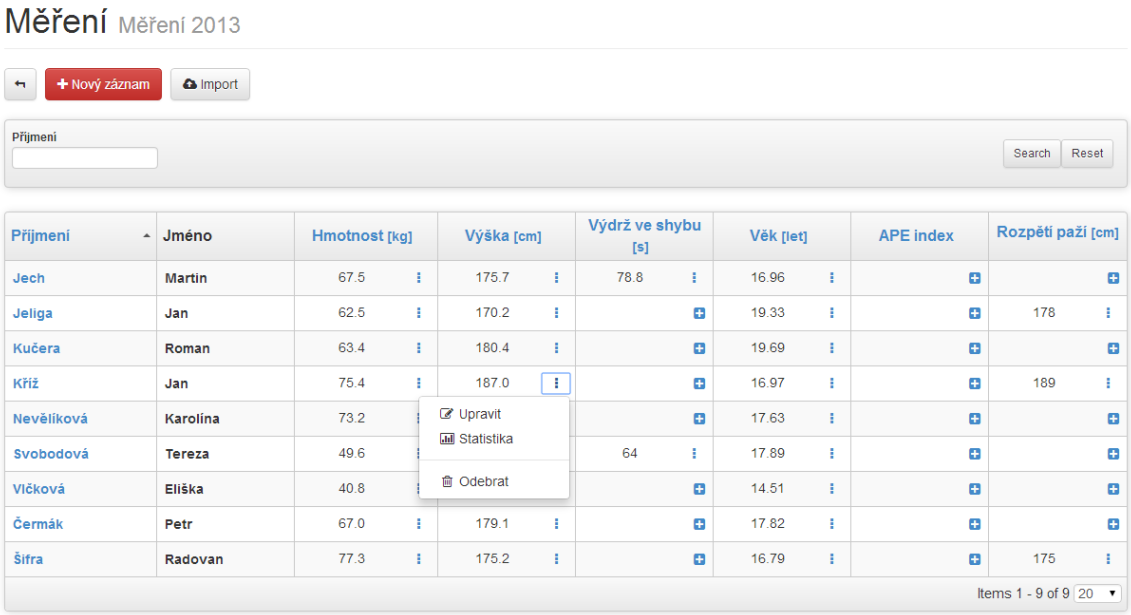
Dle zavedené konvence v systému jsou pro jednotlivé tabulky s entitami vytvořeny samostatné komponenty `GridControl`. Každá taková komponenta je samořejmě službou a své závisloti získává díky auto-wiringu a DI kontejneru. Stejně jako komponenty `EntityControl` jsou pro snadnou orientaci umístěny v jmenných prostorách vyhrazených pro komponenty dané entity. Stránka s tabulkou včetně bloku s filtry obsahující entity `Test` je na obrázku 4.4.

[7] Grido – http://o5.github.io/grido-sandbox/



Obrázek 4.4: Tabulka komponenty Grido s entitami `Test`

Komponenta umoňuje získávat data i z jiných zdrojů. Při vytváření pohledu velice podobnému stávajícímu řešení v podobě tabulky z tabulkového procesoru Microsoft Excel, je použito běžného pole `array`. Celá tabulka je poté složitě skládána z několika entit. Sloupce tvoří entity `Test`, řádky entity `Athlete` a buňky entity `Record`. Z důvodů co nejnižšího počtu dotazů do databáze je celý proces pečlivě optimalizován. Výsledkem je přehledná vizualizace dat v tabulce na obrázku 4.5.



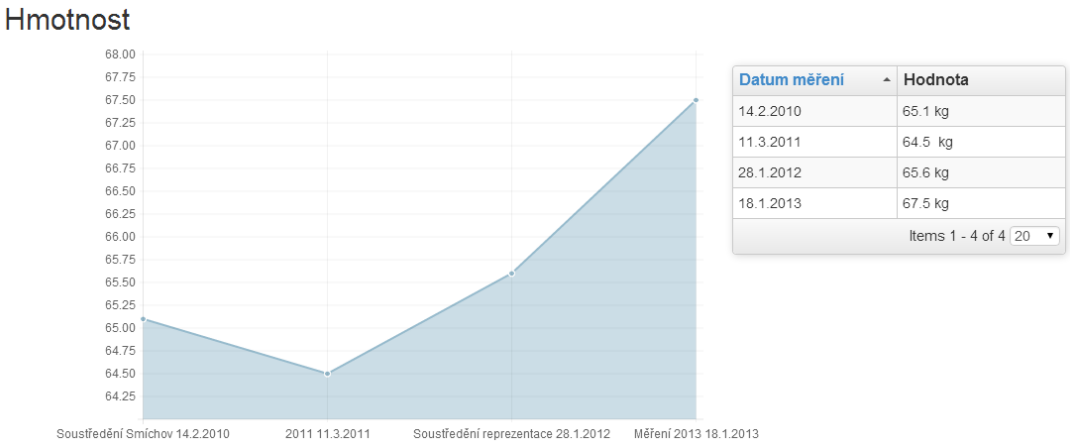
Obrázek 4.5: Pohled na tabulku zobrazující data jednoho měření

Do tabulky lze hodnoty rychle vkládat pomocí tlačítek +. Po rozbalení menu tlačítkem ⋮ lze naopak již vložené hodnoty upravovat, editovat či zobrazit jejich vývoj v čase.

4.4.3 Grafy

O vizualizaci hodnot v grafech se stará externí JavaScriptová knihovna Chart.js[8] vykreslující grafy do jediné HTML 5 značky `<canvas>`. JS kódu, který graf inicializuje, jsou přímo v šabloně předávána data k zobrazení. Následující fragment JS kódu se poté interpretován jako graf na obrázku 4.6.

```
var data = {
    labels : [ "20.5.2010" ,   "28.1.2012" ,   "18.1.2013"  ],
    datasets : [
        {
            data : [ "64.5 " ,   "65.6" ,   "67.5"  ]
        }
    ]
}
```



Obrázek 4.6: Vizualizace dat v grafu a tabulce zároveň

4.5 Importní rozhraní

Pro svou jednoduchost a možnost okamžitého exportu z prostředí Microsoft Excel byl pro import zvolen formát CSV podrobněji popsaný v části 2.7. Protože PHP nedisponuje nástroji pro práci s CSV, je k jeho zpracování použita externí knihovna EasyCSV [viz příloha XXXX]. Import dat je možné provádět vždy pouze do jednoho vybraného měření.

Zprostředkovatelem importu je komponenta `ImportControl` ve jmenném prostoru `App\Components\Session` s formulářem pro načtení dat z lokálního disku počítače. Komponenta je vytvářena stejným způsobem jako komponenty pro práci z entitami z části 4.2.1, proto je jí nutné předat pouze existující entitu měření `Session`. Všechny ostatní závislosti získává z DI kontejneru.

Proces importu je řízen načítáním jednotlivých řádků CSV souboru. Pro každý řádek je nejprve provedena kontrola existence lezce v databázi podle jména, přijímení a data narození. Pokud není taková osoba nalezena, je vložena nová. Následuje vytvoření entity `Record`

[8] Chart.js – http://www.chartjs.org

a uložení jednotlivých hodnot měření. Mapování typu hodnoty na entitu `Test` probíhá přes název sloupce uvedený na prvním řádku vstupního souboru a atribut `slug`, které se musí shodovat. Neznámé hodnoty jsou ignorovány.

4.6 Interpret

Interpret tvoří knihovna nezávislá na informačním systému, nad kterou však bylo nutné vystavět obslužnou třídu na úrovni modelové vrstvy architektonického vzoru MVP. Knihovna je rozdělena na tři základní třídy `Lexer`, `Parser` a `Interpreter`.

4.6.1 Lexikální analýza

Přesně podle návrhu v části 3.3.3 byla s pomocí nástroje `Tokenizer` implementována lexikální analýza. Neopírá se o konečný automat, ale o pole důmyslně uspořádaných vzorků regulárních výrazů jednotlivých terminálů a neterminálů. Výrazy jsou v `Tokenizeru` vyhodnocovány pomocí PHP funkce `preg_match_all()` jejímž výsledkem je vždy první shoda se vzorkem. Jazyk podporuje nejzákladnější konstrukce potřebné pro tvorbu vzorců pro výpočem hodnot na základě měřených veličin. Celé pole regulárních výrazů pro rozpoznávání tokenů obsahuje zdrojový kód 4.4.

```
1  private static $patterns = [
2      self::T_EOL => '\v+',
3      self::T_COMMENT => '\h+#.*[^\v]',
4      self::T_INDENT => '[␣\t]+',
5      self::T_IF => 'if\h+',
6      self::T_ELSE => 'else\h*',
7      self::T_ELSEIF => 'elif\h+',
8      self::T_WHILE => 'while\h+',
9      self::T_RETURN => 'return\h+',
10     self::T_DNUMBER => '\d+',
11     self::T_STRING => '".*"',
12     self::T_PARAMETER => ':[a-zA-Z][a-zA-Z0-9_-]*',
13     self::T_VARIABLE => '[a-zA-Z_][a-zA-Z0-9_-]*',
14     self::T_SYMBOL => '[,=:\-\><\'"\.\+/\?\()%]',
15 ];
```

Zdrojový kód 4.4: Pole vzorů pro rozpoznávání tokenů

4.6.2 Syntaktická analýza

Implementace syntaktického analyzátoru využívá techniky syntaxí řízeného překladu, kdy syntaktický analyzátor načítá posloupnost vstupních tokenů a zároveň generuje výstupní PHP kód. Je využito rekurzivního sestupu shora dolů, avšak výrazy nejsou a ani nemohou být kontrolovány pomocí precedenční analýzy. V případě, že analyzátor narazí na výraz, který je nutné vyhodnotit, přeloží triviálním způsobem výraz do PHP, a pomocí funkce `eval()` provede jeho kontrolu. Aby nebylo možné zasáhnout z funkce `eval()` do kontextu právě prováděného kódu, je volána v lambda funkci, pomocí které dosáhneme odstínění kontextu. Již během triviálního překladu, jsou odchytávány tokeny, které výrazy nemohou obsahovat.

Součástí syntaktické kontroly je také kontrola odsazení bloků. V místě, kde nový blok očekáváme je právě rekurzivně volána hlavní metoda syntaktického analyzátoru `parseBlock()`.

Následuje kontrola zda jsme otevřeli nový blok a uložení velikosti nového odsazení na zásobník. Ve chvíli, kdy blok uzavíráme, je proveden rozbor zásobníku, abychom zjistili, kolik bloků bylo uzavřeno. Tato hodnota je návratovou hodnotou metody `parse()`. V kódu, ze kterého jsme rekurzi zavolali, je poté nutné provést vložení uzavíracích závorek do výsledného PHP kódu v počtu uzavíraných bloků. Základní strukturu parseru zobrazuje zdrojový kód 4.5.

```
1  private function parseBlock($newBlock = FALSE)
2  {
3      $closedBlocks = 0;
4
5      while ($this->iterator->nextToken()) {
6          if ($tType == Scanner::T_IF) {
7              while ($this->iterator->nextToken(Scanner::T_ELSEIF)) {
8                  // 0-N ELSEIF bloků
9              }
10
11             if ($this->iterator->nextToken(Scanner::T_ELSE)) {
12                 // 0-1 ELSE blok
13             }
14         } elseif ($tType == Scanner::T_EOL) {
15             // Konec řádku
16         } elseif ($tType == Scanner::T_INDENT) {
17             // Chybné odsazení
18         } elseif ($tType == Scanner::T_RETURN) {
19
20         } elseif ($tType == Scanner::T_PARAMETER) {
21
22         } elseif ($tType == Scanner::T_VARIABLE) {
23
24         } elseif ($tType == Scanner::T_COMMENT) {
25
26         } else {
27             // Syntaktická chyba
28         }
29     }
30
31     return $closedBlocks;
32 }
```

Zdrojový kód 4.5: Základní struktura syntaktického analyzátoru

Parametry nalezené ve zdrojovém kódu v jazyce M jsou ve výsledném PHP reprezentovány polem `$_params`, kde názvy parametrů odpovídají klíči do pole. Přestože jazyk disponuje konstrukcí `return`, je výstup z bezpečnostních důvodů ukládán do proměnné `$_result`.

4.6.3 Interpretace

PHP kód získaný syntaxí řízeným překladem je v modelové vrstvě zajišťující běh interpretu ukládán do paměti cache, aby mohl být opakovaně volán bez nutnosti opakovaného překladu. Invalidace cache je prováděna při změně zdrojového kódu v entitě `Test`.

Než může být PHP kód vykonán, musí být naplněno pole `$_params` hodnotami parametrů potřebnými k výpočtu. Jak již bylo zmíněno v části 3.3.2, obnáší s sebou získávání hodnot kontrolu cyklických závislostí. Problematiku řeší triviální přístup, kdy se pokoušíme

postupně vyčíslit každou z nich. V případě, že je závislá na další pokračujeme na ni, dokud nenalezneme hodnotu zcela nezávislou. Musí být proto ukládána posloupnost testů, které jsme se již pokoušeli řešit a v případě, že některý test řešíme opakovaně, nalezli jsme cyklus. Řešení odpovídá teoretickému procházení grafu, avšak graf není nikdy fyzicky sestaven.

Pole `$_params` je sdíleno mezi jednotlivými výpočty a není tak zapotřebí počítat hodnoty opakovaně, ale pouze jednou. Jakmile ke klíči existuje hodnota, nic nepočítáme.

Závěr

Cílem práce bylo vytvořit informační systém, který však v sobě skrýval druhý, zcela nezávislý projekt v podobě interpretu naprosto nového jazyka v prostředí interpretovaného PHP. Bylo se nutné zaměřit hned na několik oblastí vývoje software.

Při vývoji informačního systému jsem zhodnotil znalostí z oblasti softwarového inženýrství a tvorby uživatelských rozhraní. Dlouhá a pečlivá počáteční analýza tvořila sice skoro polovinu času tvorby informačního systému, ale výsledkem je velice sofistikovaná struktura, která umoňuje nasazení nad jakýmikoliv daty během chvíle. Zcela zásadní je využití dependency injection, které de facto tvoří kostru celého návrhu. Implementovaný systém tedy splňuje počáteční požadavky a s budoucími vylepšeními může být základem rozsáhlejšího frameworku pro tvorbu webových aplikací nejen pro správu dat.

Druhou částí práce je interpret, kde jsem využil znalostí z oblasti bezkonetextových gramatik a jazyků. Jeho návrh byl z počátku zcela přímočarý a vycházel ze zkušeností a nové teorie nastudované hlavně z literatury. Nevýhodu však byla jeho unikátnost, díky které jsem některé části velice špatně navrhnul a doplatil na to během implementace. Lexikální analýza i parsování rekurzivním sestupem je jasná a přímočará. Problémem je až následné zpracování výrazů obsahující hodnoty, které jsou závislé na výpočtu jiných hodnot. Z důvodu nedostatku času bylo nutné analýzu rychle přehodnotit. Výsledkem je fungující, ale lehce zmatečné a do jisté míry i nebzpečné řešení. Zhodnotím-li syntaxi, umožňuje velice rychlé psaní zdrojových kódů stejně jako zmíněný jazyk Python. Obecnou nevýhodu je malé povědomí o programování, přes kterou je však tvorba základních konstrukcí pochopitelná většině schopnějších uživatelů.

Spojení obou částí však vytváří očekávaný celek, umožňující oproti předchozímu řešení pohodlnou správu dat v systému. Přestože je již systém v testování ČHS, nejsem schopen blíže určit jeho slabá a silná místa v reálném provozu. Nová měření od doby nasazení neproběhla a práci se starými daty zatím vyzkoušela jen hrstka těch, kteří se mnou konzultovali vývoj. Další zásadního testování systému přijde až vdobě dalšího měření, kdy se s největší pravděpodobností potvrdí jeho rychlost a použitelnost.

4.7 Budoucí rozvoj

Než bude informační systém spuštěn v ostrém provozu přijde na řadu především revize interpretu s návrhem lepšího řešení, pokud však nějaké za daných podmínek vůbec exisuje. Dalším vylepšením může být editor zdrojových kódů jazyka M se zvýrazňováním syntaxe, pro lepší přehlednost.

Protože ČHS nedisponuje žádným webovým systémem pro správu výsledků závodů a závodníků bude v případě dostatku prostředků informační systém rozšířen o další možnosti v podobě správy profilů závodníků a on-line zpracování výsledků.

Princip komponent, na kterém stojí informační systém bude v brzké době využit při tvorbě internetových obchodů, menších webů nebo jiných informačních systémů.

Literatura

[1] Arlow, J.: *UML a unifikovaný proces vývoje aplikací*. Brno: Computer Press, první vydání, 2003, ISBN 978-80-251-1519-0.

[2] Böhmer, M.: *Návrhové vzory v PHP*. V Brně: Computer Press, první vydání, 2012, ISBN 978-80-251-3338-5.

[3] Chytil, M.: *Automaty a gramatiky*. Praha: SNTL, první vydání, 1984.

[4] Gutmans, A.: *Mistrovství v PHP 5*. Brno: Computer Press, druhé vydání, 2007, ISBN 978-80-251-1519-0.

[5] Krug, S.: *Nenuťte uživatele přemýšlet!* Brno: Computer Press, první vydání, 2010, ISBN 978-80-251-2923-4.

[6] Louden, K.: *Compiler construction: principles and practice*. Computer Science Series, PWS-KENT Publishing Company, 1997, ISBN 9780534939724.

[7] Martin, R. C.: *Čistý kód*. Brno: Computer Press, první vydání, 2009, ISBN 978-80-251-2285-3.

Příloha A

Obsah přiloženého CD

/doc – adresář s textem bakalářské práce

/source – adresář se zdrojovými kódy informačního systému i interpretu

/readme.txt – návod na instalaci