VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

NÁSTROJ PRO SBĚR STATISTICKÝCH DAT
TOOL FOR COLLECTING STATISTICAL DATA

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — LUKÁŠ ČERNÝ
AUTHOR

VEDOUCÍ PRÁCE — Ing. RADEK KOČÍ, Ph.D.
SUPERVISOR

BRNO 2017

Abstrakt

Tato práce se zabývá návrhem a tvorbou systému pro sledování internetových aplikací. Systém umožňuje připojit dostupné služby a data získaná z těchto služeb ukládat do databáze. Uživateli umožňuje vybírat data a provádět nad nimi agregaci. Systém umožňuje uživatelům mít všechna data z různých aplikací na jednom místě. Z nasbíraných dat je uživatelům umožněno vytvářet statistiky na základě použité agregace.

Abstract

This thesis deals with design and creation of the system for monitoring Internet applications. The system enables connecting available services and data obtained from these services to be stored in a database. It allows a user to choose and aggregate data. The system allows users to have all the data from different applications in one place. From the collected data, users are able to generate statistics based on the aggregation used.

Klíčová slova

Objektově orientovaná aplikace, PHP, sběr statistický dat, Framework, Nette

Keywords

Object oriented application, PHP, collecting statisical data, Framework, Nette

Citace

ČERNÝ, Lukáš. *Nástroj pro sběr statistických dat*. Brno, 2017. 38 s. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologii. Vedoucí práce Ing. Radek Kočí, Ph.D.

Nástroj pro sběr statistických dat

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Radka Kočího, Ph.D. Další informace mi poskytla firma Irvekon systems s.r.o. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Lukáš Černý
12. květen 2017

Poděkování

Děkuji vedoucímu bakalářské práce Ing. Radku Kočímu, Ph.D., za odborné vedení, trpělivost, vstřícný přístup a cenné rady, které mě vždy nasměrovaly správným směrem.

Obsah

1 Úvod

Cílem této práce je navrhnout a implementovat aplikaci, která umožní přidávat služby. Z těchto služeb získává data, která ukládá do databáze. Z databáze lze získat libovolná data, které si uživatel přeje zobrazit a provést nad nimi povolenou agregaci.

Na začátku této bakalářské práce jsou popsány požadavky na vyvíjenou aplikaci. Také je zde uvedeno, proč je dobré shromažďovat data a vytvářet z nich statistiky. V následující kapitole jsou popsány různé způsoby tvorby webových stránek, ze kterých je nejpodstatnější ruční programování stránek, protože pro vytvoření této aplikace byl použit tento způsob. Jelikož do vybraného způsobu zapadá tvorba stránek pomocí frameworku a také aplikace je postavena na jednom frameworku, tak je následující část kapitoly je věnována popisu frameworků, jejich výhodám a nevýhodám. Jsou zde stručně popsány a porovnány frameworky Nette, Zend Framework a Symfony2.

Zbytek práce je věnován popisu návrhu a implementace aplikace. Jsou zde popsán návrh adresářové struktury a databázového schématu. V kapitole o vývoji aplikace jsou popsány důležité části, ze kterých se aplikace skládá a také části, které jsou složitější.

2 Analýza požadavků

V této kapitole jsou popsány požadavky na aplikaci. Tyto požadavky specifikovala firma Irvekon systems s.r.o., pro kterou je aplikace vyvíjena.

2.1 Požadavky na aplikaci

Cílem aplikace a tím pádem i hlavním požadavkem na aplikaci je, že aplikace musí umožnit napojit se na vzdálené služby, shromažďovat z nich data a ty ukládat na jedno místo (do databáze). Bude poskytovat napojení pouze na takové služby, pro které bude dostupný rozšiřující modul. Z toho vyplývá, že samotná aplikace, bez rozšiřujících modulů, nebude uživateli nic poskytovat. Proto je nutné implementovat i nějaký modul. Jako první modul firma vyžadovala implementovat modul pro službu Toggl. Další požadavky jsou následující.

Ze shromážděných dat si uživatel bude moct vybrat, které si přeje zobrazit a nad zobrazenými daty bude moct provádět základní agregační funkce. Jelikož se bude do budoucna přidávat podpora pro další agregační funkce, tak aplikace musí umožnit jejich jednoduché přidávání. Například výběrem správných sloupců a provedení agregace bylo by možné získat, díky modulu Toggl, statistiky odpracovaného času na určitém projektu nebo díky modulu Sklik počet příchodů na stránku ze serveru Seznam.cz. Více o sběru dat a vytváření statistik v podkapitole 2.2.

Aplikace musí být modulární. Každá poskytována služba bude reprezentována jako samostatný modul. Aplikace musí fungovat stylem „plug and play“ tj. nové moduly se mohou přidávat, aniž by bylo nutné zasahovat do souborů již funkčního systému. Jelikož aplikace musí být modulární, tak se musí navrhnout i odpovídající logická adresářová struktura.

Aplikace musí rozlišovat oprávnění uživatelů. Uživatel aplikace v roli administrátora bude moct řídit přístupy uživatelů, kteří jsou v roli zákazníka firmy, k modulům a přidávat nové moduly. Uživatel v roli zákazníka firmy, bude moct si vybrat, ze kterých služeb bude chtít sbírat data. Také bude moct určit, které vybrané moduly bude moct vidět jeho zákazník, který je v roli zaměstnanec zákazníka. Uživatel v roli administrátora nebo zákazníka bude moct vidět všechna data všech svých zaměstnanců, ale uživatel v roli zaměstnance bude moct vidět pouze svá data.

Grafické rozhraní by mělo mít moderní a jednoduchý grafický design, který bude založený na bootstrapu. Aplikace se musí jednoduše ovládat, tj. žádné složité nastavování modulů nebo systému. Je možné také použít již existující šablonu.

2.2 Sběr informací pro statistiky

Každý, kdo má webové stránky, by se měl zajímat o to, zda a jak plní svůj účel. To lze udělat analýzou dat o počtu návštěvníků, jejich cílech a také podle toho co na stránce vyhledávali. Tyto informace lze zpracovávat dvěma způsoby, a to zpracováním logovacích souborů, které jsou uložené na serveru nebo nasadit systém, který bude potřebná data zaznamenávat a tvořit požadované statistky. Na základě nashromážděných dat lze vytvořit statistiky, ze kterých může vyplynout například: jak se změnila návštěvnost e-shopu, které produkty jsou nejvíce prohlíženy, co jaké rubriky jsou nejčtenější, jako dlouho se uživatel zdržel na stránce. Jako přiklad je v podkapitole 2.2.2 uvedena sociální síť Facebook, která velmi zajímavým způsobem zaznamenává údaje.

Na základě vyhodnocení takovéto analýzy lze říct, zda to byla dobrá investice a také jak vylepšit webovou stránku. Pro tyto účely slouží nástroj Google Analytics (více v následující podkapitole). I když tento nástroj je velmi používaný a užitečný, tak pro účely firmy byl nevyhovující. Firma požaduje aplikaci, která, jak už bylo řečeno, bude shromažďovat data a vytvářet z nich statistiky, ale tyto data lze získat ze služeb pouze přes jejich API. Tuto možnost Google Analytics neposkytuje.

2.2.1 Google Analytics

Tuto službu poskytuje společnost Google, Inc. Služba Google Analytics (dále jen GA) používá souborů cookies a krátkého javascriptového kódu. Informace, která jsou pomocí javascriptu vygenerována, jsou ukládány na serverech v USA.

Základní princip fungování této aplikace je, že na webovou stránku je umístěn krátký javascriptový kód, který se doporučuje umístit na stránku co nejdříve. Díky vložení toho kódu do stránky, dojde k „propojení“ stránky se serverem GA. Při vstoupení uživatele na stránku s tímto kódem je uživateli, který danou stránku ještě nenavštívil, vygenerován unikátní textový řetězec. Tento řetězec je uložen do souboru cookies. Díky tomuto souboru je GA schopen uživatele identifikovat a na základě toho zjistit z jaké stránky přišel.

Google bude užívat tyto informace pro účely vyhodnocování užívání stránky a vytváření zpráv o její aktivitě, určených pro jejího provozovatele, a pro poskytování dalších služeb týkajících se činností na stránce. Google může také poskytnout tyto informace osobám třetí strany, bude-li to požadováno zákonem.

2.2.2 Facebook

Facebook je jednou z největších sociálních sítí na světě. Umožňuje komunikaci mezi ostatními uživateli, sdílení multimediálních dat a mnoho dalšího. Jelikož Facebook poskytuje mnoho služeb, tak vzhledem k tématu této práce je dobré se zmínit o funkci doporučení nových přátel.

Facebook používá speciální geolokační algoritmus, díky kterému je schopen zjistit s kým se daný uživatel ve skutečném světě setkal, popř. mohl setkat. Tento algoritmus pracuje na takovém principu, že zaznamenává polohu a čas každého uživatele Facebook, a pak doporučí takové lidi, které se zdržovali ve stejnou dobu na stejném místě. Dále Facebook sbírá informace nejen na stránce Facebooku, ale také i na ostatních serverech, kde se vyskytuje tlačítko „To se mi líbí“.

3 Analýza technologií

V této kapitole jsou popsány různé metody pro tvorbu webových stránek. Jelikož vyvíjená aplikace bude nebude žádný blog ani se po aplikaci nepožaduje jednoduché zveřejňování příspěvků, tak nejvhodnější metoda je ručně aplikaci naprogramovat. Při vlastní tvorbě webové aplikace je aplikace tvořena na míru. Pro usnadnění tvorby webové aplikace lze využít tzv. frameworků. Více o frameworcích, které se používají při tvorbě webových aplikací v podkapitole 3.2. V závěru této kapitoly jsou uvedeny důležité pojmy, které se vyskytují v této práci.

3.1 Metody tvorby webových stránek

Pro vytvoření funkčních webových stránek existuje několik různých způsobů. Ty nejčastější jsou popsány v následujících podkapitolách.

3.1.1 Ruční programování webových stránek

Tato možnost je pro začátečníky trochu odstrašující, ale přesto nejlepší cesta. Pro tuto možnost je třeba se naučit základy pro programování v jazyce HTML a CSS. Každým řádkem HTML kódu je prohlížeči řečeno, co má zobrazit. Jinak řečeno, HTML jazyk pouze ovládá prvky na webu (obrázky, odstavce, tabulky apod.) a CSS říká jazyku prohlížeči, jak ty prvky zobrazit (na jakém místě, jak velké, jakou barvu, atd.).

I když bude chvíli trvat, než se vytvoří web, se kterým bude autor spokojen, tak během této doby se autor naučí vše, co bude potřebovat i pro pozdější pokročilou práci na webu. Každý, kdo se chce naučit programovat webové stránky, tak musí projít přes HTML a CSS.

3.1.2 Použití redakčního systémů

Redakční systém neboli CMS (Content Management Systém) je systém, který se vyznačuje jednoduchou instalací na hosting a snaží se uživateli maximálně zjednodušit vkládání obsahu. Jeho cílem je za autora vytvořit kostru stránky, která je snadno upravitelná. Díky tomu se autor může zabývat tím, co chce publikovat a nikoliv programováním. Mezi nejznámější redakční systémy patří Wordpress[1]a Drupal[2].

Použití redakčního systému na správný typ stránky se ušetří obrovské množství času. Všechny redakční systémy disponují velkým množstvím doplňků, kterými je možné vzhled webu přizpůsobit, ale zasahování do jejich kódu je velmi složité a je určené pouze pokročilým programátorům. Takže

[1] Více informací na https://wordpress.com/
[2] Více informaci najdete na https://www.drupal.cz/

pokud doplněk chybí nebo stránka nedělá to, co se po ní chce, tak s tím nic autor webu nic nenadělá. Redakční systémy pouze vezmou data, která se mají zobrazit a z nich vygenerují opět HTML kód. Takže i pro redakční systémy je důležité mít základy v HTML a CCS.

Hlavní výhodou redakčních systémů je, že i při nulových znalostech programovaní, se dá vytvořit mnohem hezčí web, než když se použije předchozí možnost pro tvorbu webu. Velkou nevýhodou redakčních systémů je, že vyžadují větší nároky na hosting (více paměti) a mají větší odezvu než předchozí možnost pro tvorbu webu.

3.1.3 Využití automatické tvorby webů

V dnešní domě existuje mnoho internetových služeb, které umožňují vytvoření webu zdarma. Tyto služby často neumožňují mnoho nastavení grafického designu. Bývají používány hlavně pro možnost tvorby blogů.

Pro tuto možnost není třeba rozumět programování. Lidé, tuto možnost tvorby webu obvykle zvolí, protože chtějí rychle něco sdělit a moc je nezajímá, že nemají originální design či nadstandartní funkce. Mezi nejznámější české poskytovatele této služby patří blog.cz, eblog.cz. Hlavní výhodou této možnosti je, že poskytovatel této služby se postará i o prostor na webu.

3.2 Frameworky pro vývoj webových aplikací

Asi každý programovací jazyk pro vývoj webových aplikací poskytuje přesně dané rozhraní pro oddělenou práci s daty od jejich reprezentace. V jazyce PHP je situace jiná. Neexistuje přesně dané MVC řešení. Pro ulehčení je v jazyce PHP napsáno několik desítek MVC frameworků, které ulehčují práci při psaní webových aplikací.

Framework je softwarová struktura, která slouží jako podpora při programování a vývoji jiných softwarových projektů. Může obsahovat podpůrné programy, knihovny, podporu pro návrhové vzory nebo doporučené postupy pro vývoj aplikace. Na základě toho lze je rozdělit na dvě skupiny:

- Sada skriptů/knihoven
- Skripty vytvářející jednu konkrétní webovou aplikaci

Frameworky patřící do první skupiny obsahují řadu knihoven, které pokrývají různé potřeby. Samotný framework nelze spustit, aby bylo možné webovou aplikaci spustit, tak programátor musí doplnit hlavní logiku aplikace, která může využívat knihovny frameworku. Zástupcem této skupiny je Zend Framework (více v podkapitole 3.2.3).

Frameworky patřící do druhé skupiny obsahují nejen vlastní knihovny, ale také je v nich zahrnutý funkční základ pro aplikaci. Z toho vyplývá, že ihned po stažení takového frameworku, je možné ho spustit. Do této skupiny patří například Nette, Symfony (více v podkapitole 3.2.3).

3.2.1 Výhody frameworků

Jak už bylo zmíněno, frameworky ulehčují práci, a hlavně umožňují rychlý vývoj aplikace. Poskytují dobře organizovaný, opětovně využitelný a udržovatelný zdrojový kód. Snaží se prosadit moderní webové vývojové postupy, mezi které patří nástroje OOP. Díky knihovnám, které jsou ve frameworcích zahrnuté, není třeba se zabývat programováním rutin jako připojení k databázi, generováním hezkých URL, atd.

3.2.2 Nevýhody frameworků

Každý framework má v sobě zahrnuty spousty knihoven, které často ani nejsou používané a zabírají zbytečně místo na serveru. Při nesprávném použití knihoven mohou vzniknout bezpečností díry aplikace nebo dokonce aplikace se může zhroutit.

Každý Framework má specifickou adresářovou strukturu, doporučenou strukturu tříd a také každý framework se jinak instaluje a nastavuje, proto přechod z jednoho frameworku na druhý bývá obtížný.

3.2.3 Nejznámější frameworky

V této podkapitole je popsáno několik frameworků, které se v současné době používají a také jejich klady a zápory.

3.2.3.1 Nette framework

Nette je český framework, který výrazně zjednodušuje tvorbu webových aplikací. Jeho hlavním autorem je český vývojář David Grudl. Tento framework se zatím rozšířil pouze po České republice. Zaměřuje se na eliminaci bezpečnostních rizik. V současné době vyšla verze 2.4, která podporuje php 5.6 a vyšší.

Hlavními výhodami jsou bezkonkurenční ladící nástroje, jednodušší na naučení, velmi dobré zabezpečení proti bezpečnostním dirám a česká dokumentace. Hodí se na malé i velké projekty. Hlavní nevýhoda tohoto frameworku je ta, že je rozšířený pouze v České republice, takže pokud ovládáte tento framework, tak s vysokou pravděpodobností nenaleznete mimo ČR uplatnění.

3.2.3.2 Zend Framework

Zend je komplexní a stabilní PHP framework obsahující spoustu konfiguračních voleb, proto se obvykle nedoporučuje pro menší projekty, ale je spíše vhodný na ty složitější. Používá modulární architekturu, která umožňuje vývojáři použít jen ty komponenty, které programátor potřebuje k vývoji.

Hlavní výhoda je velké množství knihoven, kvalitní dokumentace a plně testovatelný kód. To znamená, že každá třída obsahuje vlastní unit testy, které by měly zaručit nejmenší chybovost. Hlavní

nevýhodou je velmi pomalý vývoj a složitost frameworku. Při vývoji složitějších projektů nebo při nedodržení doporučené architektury aplikace, tak je potřeba podrobněji znát framework.

3.2.3.3 Symfony2

Symfony2 je PHP framework, který se celosvětově využívá. Skládá se z mnoha komponent (knihoven), které také využívá mnoho redakčních systémů jako například Drupal a Joomla nebo software phpBB.

Hlavním kladem frameworku je to, že má obrovskou komunitu vývojářů, proto se rychle vyvíjí. Dále má obsáhlou dokumentaci a mnoho rozšíření. Nevýhody Symfony2 jsou, v ČR stále není mnoho kvalitních vývojářů a také mnoho částí frameworku jsou velmi komplexní.

3.3 Důležité pojmy

V této kapitole jsou vysvětleny důležité pojmy, které se používají v tomto dokumentu.

3.3.1 URL

URL (neboli Uniform Resource Locator) je řetězec znaků s definovanou strukturou, který slouží k přesné identifikaci zdrojů informací na internetu.

Hlavní struktura je „protokol://domena:port/cesta#kotva“. Protokol se nejčastěji používá „http“ popř. „https“. Doména se skládá ze tří částí oddělených tečkami. Jsou to jméno doménového serveru, nejčastěji www, jméno domény druhého řádu a doména nejvyššího řádu. Port je nepovinná část URL. Pokud není uveden, tak automaticky se bere port 80. Cesta je adresářová cesta k soboru na serveru. Kotva je nepovinnou částí URL a může mířit na záložku v zobrazovaném dokumentu.

3.3.2 HTML

HTML (neboli HyperText Markup Language) je značkovací jazyk, který je základem pro tvorbu webových stránek. Byl vytvořen roku 1989 a vyvinul se z rozsáhlého univerzálního značkovacího jazyka SGML (Standard Generalized Markup Language). Vývoj jazyka byl ovlivněn vývojem webových prohlížečů, které zpětně ovlivňovaly definici jazyka. Poslední vydaná verze je 5.0. Jazyk HTML obsahuje párové a nepárové značky „tagy“. Na základě těchto značek je možné definovat strukturu webové stránky.

3.3.3 CSS

CSS (neboli Cascade Style Sheet) je jazyk pro popis způsobu zobrazení elementů na webových stránkách. Těmito styly lze například upravovat vzhled písma, pozadí, obrázky, barev. Tyto styly se většinou umísťují do souboru s příponou „css“, aby byl oddělený HTML kód od CSS. Díky takovému oddělení lze použít styly ve více dokumentech a mít tak webovou stránku, která má jednotné styly.

První verze CSS vznikla v roce 1996 jako reakce na chaotický se vyvíjející HTML jazyk. Poslední vydaná verze je CSS3.

3.3.4 Jazyk PHP

PHP (Hypertext Preprocesor) je interpretovaný programovací jazyk pracující na straně webového serveru. PHP je v současné době nejrozšířenější technologií pro tvorbu dynamických webových aplikací a internetových prezentací. Tento jazyk lze také použít pro tvorbu konzolových a desktopových aplikací.

3.3.5 Cookies

Cookies je malý textový soubor, který se při navštívení stránky ukládá na lokální počítač. Tyto soubory se vážou k internetovému prohlížeči, ve kterém stránka byla navštívena jinak řečeno každý webový prohlížeč má své cookies soubory. V těchto souborech se mohou uchovávat nejrůznější data. Většina internetových stránek využívá cookies pro identifikaci uživatele. Například některé internetové obchody používají cookies pro uložení současného stavu košíku a při další návštěvě obchodu jsou na základě dat v cookies vloženy položky do nákupního košíku.

3.3.6 MVC architektura

MVC je softwarová architektura, která rozděluje datový model aplikace, uživatelské rozhraní a řídicí logiku do tří nezávislých částí. Úprava některé z těchto částí má minimální vliv na ostatní části aplikace, a proto lze aplikace jednodušeji rozšiřovat. Tato architektura je součástí mnoha webových frameworků, jako např. Zend, Nette, Ruby On Rail.

Popis architektury

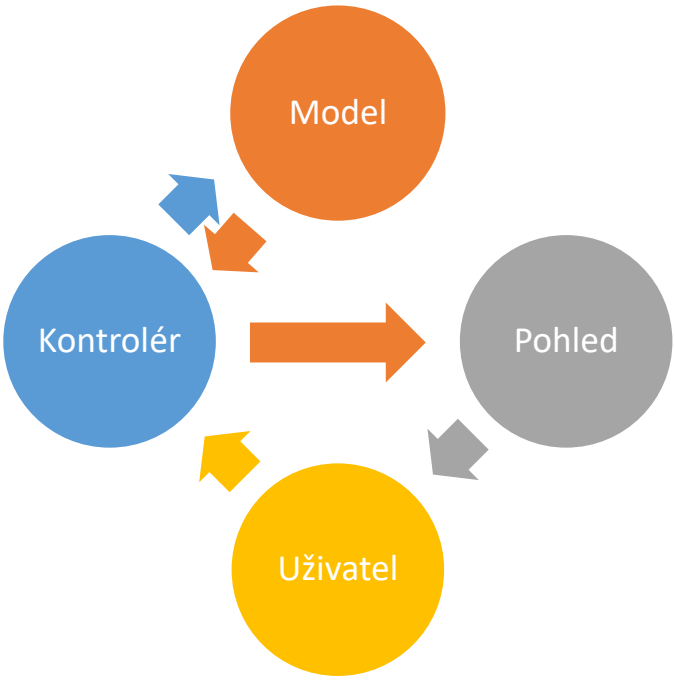
Jak už bylo zmíněno aplikace se dělí do tří část, model, pohled (view) a kontrolér (controller) z toho vznikla zkratka MVC. Na obrázku (Obr. 3.1 - MVC schéma) je zobrazeno schéma architektury.

Model obsahuje logiku a vše, co do ní spadá jako například výpočty, databázové dotazy, práce se soubory a atd. Model neví o výstupu. Jeho funkce spočívá v navrácení dat na základě předaných vstupních parametrů. Neví, odkud data v parametrech přišla a ani jak budou výstupní data formátována a vypsána.

Pohled se stará o zobrazení výstupu uživateli. Nejčastěji to bývá nějaká šablona, která obsahuje celou nebo část HTML stránky a umožňuje vypisovat proměnné, iterace a dělat podmínky. Pro šablony se často používají speciální značkovací jazyky (např. Latte).

Kontrolér je „prostředník“, se kterým komunikuje uživatel, model i pohled. Drží tedy celý systém pohromadě a komponenty propojuje. Existuje mnoho různých přístupů, nejčastěji má každá stránka vlastní kontrolér.

Poslední „částí“ je uživatel. V schématu je uveden, protože on vidí data vygenerovaná pohledem a komunikuje s kontrolérem zasíláním požadavků. Jsou to požadavky jako GET, POST apod.

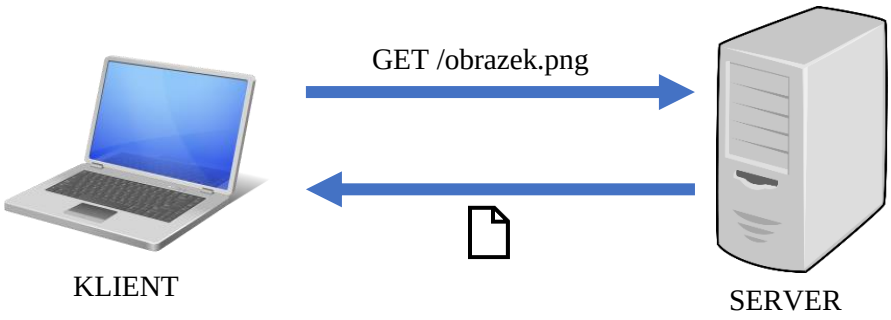


Obr. 3.1 - MVC schéma

4 Návrh aplikace

Tato kapitola je věnována návrhu aplikace. Při návrhu aplikace se vycházelo z již stanovených požadavků. Při návrhu každé aplikace se musí počítat s tím, že se aplikace bude dále rozšiřovat. Z tohoto důvodu je návrh aplikace velmi důležitý.

Aplikace se bude skládat ze dvou částí, a to z klientské části a serverové části. Klientskou část uvidí uživatel ve svém webovém prohlížeči. Návrh klientské části je popsán v podkapitole 4.1 Návrh grafického rozhraní. Serverová část přijímá požadavky od klienta a na základě vyhodnocení je uživateli vrácena odpověď (např. obrázek, kód html). Více o návrhu serverové části je v podkapitole 4.2 Návrh serverové části aplikace. Princip komunikace mezi klientem a serverem byl schematicky zobrazen na následujícím obrázku.



Obr. 4.1 - Schéma komunikace klient-server

4.1 Návrh grafického rozhraní

Grafické rozhraní nejen vytváří první dojem na aplikaci, ale také určuje, jakým způsobem bude aplikace vnímána uživateli a jak se s ní bude moct pracovat. Pro každou aplikaci je velmi důležité, aby byla dobře naprogramovaná a použitelná. Každé uživatelské rozhraní se skládá z různých grafických a textových prvků a jejích rozložení.

Aplikace se skládá z několika logických částí (dále komponent). Některé jsou jednodušší a nebylo třeba konzultovat jejich grafický návrh (například přihlašování uživatelů, přidávaní nových uživatelů), ale některé byly i složitější a bylo nutné je konzultovat se zadavatelem tématu. Během konzultacích bylo pro správné pochopení nutné vytvořit několik skic[3]. Díky skicám bylo lehce pochopitelné, jaké jsou požadavky na danou komponentu. Složitější komponenty jsou uvedeny v následujících kapitolách.

[3] Skici byly tvořeny pomocí nástroje WireframeSketcher (více informací na http://wireframesketcher.com/)

4.1.1 Komponenta pro řízení oprávnění

Tato komponenta se může zobrazit pouze uživatelům v roli zákazníka (dále jen zákazník). Zobrazí pouze maximálně 10 zaměstnanců zákazníka a pokud jich je více, tak komponenta musí umožnit těmito zaměstnanci procházet.

Vždy se budou zobrazovat jména zaměstnanců přihlášeného zákazníka. Další sloupce budou generovány na základě toho, jaké moduly aplikace bude mít zákazník aktivované. Pokud uživatel bude mít přístupová práva k danému modulu tak se musí zobrazit souhlas pomocí nějakého symbolu a naopak. Pro odebrání přístupu k modulu je třeba kliknout na symbol souhlasu. Změna se provede pomocí Ajaxu. Povolení přístupu k modulu dojde při kliknutí na symbol pro nesouhlas.

	Jméno	**Toggl**	**Sklik**
1	Jan Novák	✓	✗
2	Karel Pavlíček	✗	✗
4	Andrea Simonová	✗	✓
6	Otakar Urban	✓	✗

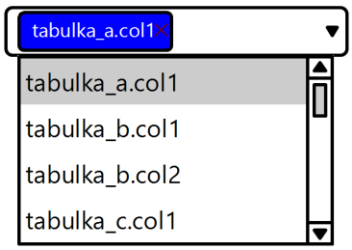
Obr. 4.2 - Náčrt komponenty pro řízení oprávnění

4.1.2 Komponenta pro agregaci dat

Komponenta pro agregaci dat musí umožnit uživateli zobrazit data, která byla získána z napojených služeb a jsou uložena v databázi. Jelikož do aplikace bude napojeno více služeb (neboli modulů), tak komponenta musí být universální, tj. musí umět pracovat s daty z libovolného modulu. Musí umožnit provádět nad zobrazovanými daty operaci řazení podle libovolného sloupce.

V neposlední řadě musí umět provést nad daty agregaci jinak řečeno podle sloupce v tabulce se provede seskupení a nad určitým sloupcem se provede umožněná operace. Sloupce, podle kterých lze provádět seskupení nebo na kterých lze provádět umožněné agregační operace, musí být specifikovány v konfiguračních souborech modulu, který komponentě předal data.

Komponenta by se měla vytvářet pomocí jednoduchého formuláře, který bude nabízet jména sloupců, která lze z databáze získat. Musí umožnit vytvořit komponentu, která se bude skládat z dat jedné databázové tabulky nebo více tabulek. Na základě toho byla vytvořena následují skica.



Obr. 4.3 - Náčrt vytváření nové komponenty

Například vybráním sloupců z tabulek `tabulka_c` (tabulka pro s uživateli) a `tabulka_a` (tabulka s odpracovanými časy uživatelů) se musí vytvořit tabulka, ve které budou zobrazeny požadované sloupce a data, která vznikla propojením těchto tabulek. Pro ukázku je uvedena následující skica.

tabulka_c.col1	tabulka_c.col2	tabulka_c.col4	tabulka_a.col2
1	Jan Novák	17-03-2017	02h 37min
3	Karel Pavlíček	25-01-2017	138h 12min
8	Andrea Simonova	01-05-2015	20h 05min
6	Otakar Urban	06-02-2011	00h 07min

Obr. 4.4 - Náčrt vygenerované komponenty

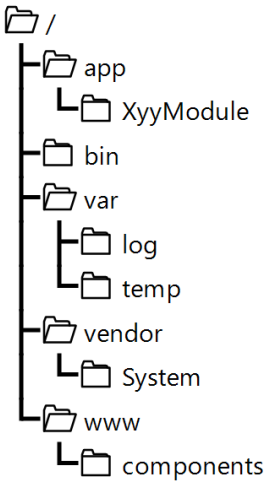
4.2 Návrh serverové části aplikace

Pro návrh se použila MVC architektura, na které je postaveno mnoho nejen webových aplikací, ale například i desktopové aplikace. Návrh serverové části aplikace se skládal z několika dílčích částí, které jsou popsány v následujících podkapitolách.

Na základě uvedených požadavků musí aplikace rozlišovat různé druhy oprávnění uživatelů. Po vytvoření use case diagramu (Příloha B - Use case diagram), byly definovány tři role uživatelů: administrátor, zákazník, uživatel. Administrátor je zaměstnanec firmy Irvekon systems s.r.o. a bude mít za úkol spravovat aplikaci. Také má nejvíce oprávnění. V roli zákazník jsou uživatelé, kteří jsou zákazníky firmy a mají v podstatě stejná oprávnění jako administrátor až na správu modulů a částečně i na správu uživatelů. Uživatel má nejméně oprávnění a je to vlastně zaměstnanec uživatele v roli zákazníka. Může pouze vytvářet agregační komponenty a provádět nad nimi agregace.

4.2.1 Návrh adresářové struktury

Při návrhu adresářové struktury se musel brát ohled na požadavek modularity aplikace. Na základě tohoto byla vytvořena základní struktura:

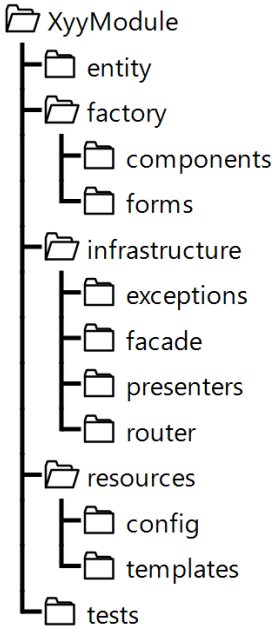


Obr. 4.5 - Základní adresářová struktura

Vysvětlení významu jednotlivých složek:

- **app** – obsahuje všechny poskytované moduly .Složky, které se nachází na stejném místě jako složka `XyyModule`, obsahuji aplikační logiku pro daný modul (službu). Název těchto složek musí odpovídat regulárnímu výrazu [A-Z][A-Za-z0-9]*Module, pokud ne, tak aplikace s takovýmto modulem nebude pracovat
- **bin** – obsahuje soubory, které lze spustit pouze přes příkazovou řádku
- **var/log** – slouží pro ukládání logovacích informací, chyb, varování a jiných chyb, které se vyskytly za běhu aplikace
- **var/temp** – obsahuje dočasné soubory např. cache
- **vendor** – obsahuje knihovny třetích stran. Tyto knihovny jsou stahovány například ze serveru github.com a týkají se serverové části aplikace. Podadresář `System/` slouží pro uložení tříd, které rozšiřují třídy třetích stran
- **www** – obsahuje soubory, které jsou dostupné a čitelné každým uživatel, který tuto aplikaci otevřel. Umisťují se zde většinou soubory, které jsou důležité pro zobrazení grafického rozhraní. Jsou to například obrázky, javascripty, kaskádové styly. Podadresář `components/` obsahuje knihovny třetích stran, které jsou důležité pro správné zobrazení grafického rozhraní

Následně bylo třeba specifikovat adresářovou strukturu modulu (viz. Obr. 4.6 - Adresářová struktura modulu)



Obr. 4.6 - Adresářová struktura modulu

Vysvětlení významu jednotlivých složek:

- **entity** – má sloužit pro ukládání entit. Každá tato entita reprezentuje jednu databázovou tabulku
- **factory/componentes** – slouží pro ukládání továren pro komponenty a samotných komponent
- **factory/forms** – slouží pro ukládání tříd, které zajišťují tvorbu a obsluhu formulářů
- **infrastructure/exceptions** – nachází se zde výjimky, které lze vyvolat v jakémkoliv modulu
- **infrastructure/facade** – jsou zde třídy které pracují s entitami ze složky entity
- **ingrastructure/presenters** – slouží pro uložení presenterů, které definují hlavní logiku aplikace
- **infrastructure/router** – obsahuje třídu, která definuje, jakým způsobem budou překládány odkazy na presentery na URL a také naopak
- **resources/config** – obsahuje hlavní konfigurační soubory pro daný modul
- **resources/templates** – obsahuje šablony pro presentery
- **tests** – pro testy modulu

4.2.2 Návrh databáze

Každá webová aplikace většinou potřebuje ukládat nějaká data. Data lze ukládat do speciálních souboru, ale tento způsob lze použít, když se pracuje s malým množstvím dat a když není moc důležitá rychlost získávání dat. Jelikož aplikace bude potřebovat pracovat s velkým objemem dat, tak byl použit

druhý způsob ukládání, a to do databáze. Pro uložení dat je nutné definovat, jaká data se budou do databáze ukládat. Na základě toho lze vytvořit schéma databáze.

Do databáze se budou ukládat data o uživatelích (jméno příjmení, email, heslo apod.), přístupové údaje pro připojení k serverům poskytovaných služeb. Hlavně se do databáze budou ukládat data, která budou stahována z různých internetových služeb, na které aplikace bude umožňovat se napojit.

Databázové tabulky se dělí do určitých skupin podle jejich prefixu, který odpovídá modulu, se kterým se tabulka pojí. Každá databázová tabulka dodržuje „snake case“ jazykovou konvenci. Tato konvence používá pro oddělení slov podtržítko.

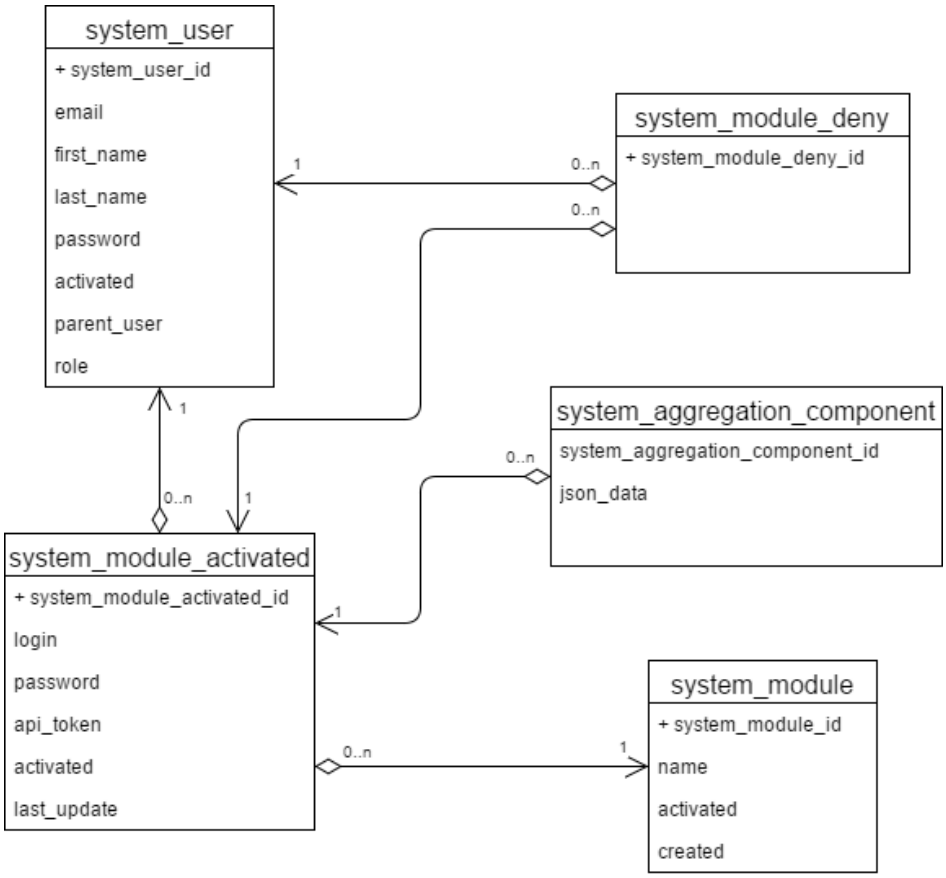
Hlavni část databáze se skládá z tabulek, které mají prefix "system_". Do databáze je třeba zaznamenávat údaje o uživatelích a modulech. Také je třeba ukládat informace jaké moduly si uživatel aktivoval a ke kterým aktivovaným modulům má uživatel povolený/odepřený přístup. Dále je třeba někam ukládat informace o vytvořených komponentách, které provádí agregaci nad daty získané z připojeného modulu. Na základě těchto informací byly vytvořeny databázové tabulky:

- system_user
- system_module
- system_module_activated
- system_module_deny
- system_aggregation_component

Poté bylo třeba specifikovat relace mezi entitami:

- uživatel může mít aktivováno více modulů
- uživatel může mít k více aktivovaným modulům odepřený přístup
- modul může být aktivován více uživateli
- uživatel může mít vytvořeno více komponent u aktivovaného modulu

Z výše uvedených informací bylo vytvořeno schéma databáze viz níže uvedený obrázek.



Obr. 4.7 - Základní schéma databáze

- **Tabulka system_aggregation_component**

Tato entita představuje jednotlivé komponenty, které provádí agregaci nad získanými daty z internetových služeb. Názvy tabulek a sloupců, ze který se bude provádět agregace, jsou zakódovány a uloženy v JSON formátu.

- **Tabulka system_user**

Účelem této entity je reprezentovat jednotlivé uživatelé, jejich přihlašovací údaje a základní informace o nich. Jsou zde zaznamenány informace jako například: jméno, heslo, datum vytvoření.

- **Tabulka system_module**

Entita `system_module` slouží pro evidenci a správu modulů. Obsahuje například informace o stavu modulu, unikátní název pro jednoznačnou identifikaci při vyhledávání modulu v souborovém systému.

- **Tabulka system_module_activated**

Účelem této entity je uchovávat informace pro autentizaci uživatele na vzdáleném serveru. Je podporována autentizace pomocí `api_token` nebo pomocí přihlašovacího jména a hesla.

- **Tabulka system_module_deny**

Tato entita obsahuje informace o uživatelích, kteří mají odepřený přístup k aktivovanému modulu.

4.2.3 Návrh rozhraní mezí aplikací a API služby Toggl

Pro implementování rozhraní bylo třeba nastudovat dokumentaci[4] této služby[5]. V současné době jsou dostupné dokumentace k APIv2 a APIv8. Na základě dokumentace požadavků zadávající firmy, byly vybraný pouze takové příkazy, které slouží pro stahování dat ze serveru Toggl.

Pro správný návrh rozhraní bylo třeba vyzkoušet jakým způsobem lze získat data ze serveru a také jaká data jsou jak propojená. Při posílání požadavků se každý uživatel musí autentizovat pomocí tzv. API TOKENu. Pro získání dat je třeba poslat na adresu rozhraní GET požadavek. Na základě získaných dat ze serveru byl vytřen návrh další části databáze.

[4] Adresa dokumentace: https://github.com/toggl/toggl_api_docs/blob/master/README.md
5 Více o nástroji na https://www.toggl.com

5 Vývoj aplikace

Tato kapitola je věnována popisu o vývoji aplikace. Jsou zde popsány části navrhované aplikace, které jsou složitější nebo jsou důležité pro funkčnost aplikace.

5.1 Vývojové prostředí

Aplikace byla vyvíjena na virtuálním stroji, na kterém byl nainstalován operační systém Linux Debian 8.7 Jessie. Pro vývoj webových aplikací bylo třeba nainstalovat webový server (Apache 2.2), databázový server (MySQL 5.6) a podporu pro jazyk, ve kterém je napsaná tato aplikace, tedy php5.6.

Pro vývoj aplikace bylo použito NetBeans IDE 8.2. Toto IDE je zcela zdarma. Je podporováno jak operačním systémem Linux, Windows ale i Mac OS. Obsahuje velmi užitečné funkce jako je podtrhovaní chyb v kódu, formátování kódu a různé pomocné funkce pro urychlení psaní kódu.

Při vývoji byl také použit verzovací nástroj Git. Repozitář byl umístěn na serveru Gitlab.com. Pro jednodušší práci s repozitářem byl použit nástroj SmartGit[6], který má jednoduché a přehledné grafické rozhraní. Tento nastroj lze použit pro komerční i nekomerční použití. Je podporován na operačních systémech Windows, Linux i Mac OS.

Dále pro instalaci balíčků, knihoven týkající se grafického rozhraní bylo třeba doinstalovat do Linuxu nástroj tzv. `bower`[7]. Tento nástroj je takový správce balíčků pro grafické rozhraní. Obstarává stahování balíčků, instalaci, aktualizaci a také správu závislostí mezi jednotlivými balíčky.

Pro instalaci knihoven třetích stran týkající se backendu bylo třeba doinstalovat nástroj tzv. `composer`[8]. Tento nástroj funguje na obdobném principu jako nástroj `bower` s jediným rozdílem, že pracuje s balíčky pro backend.

5.2 Vývoj grafického rozhraní aplikace

Pro stahování různých balíčku, javascriptu, knihoven se použil nástroj `bower`. Jelikož stažené soubory, pomocí tohoto nástroje, se ukládají do výchozího adresáře `bower_components`, která se vytvoří v současné pracovní složce (dále jen root projektu), tak bylo potřeba před jeho použitím provést konfiguraci.

Konfigurace je možná pomocí souboru `.bowerrc`. Tento soubor umožňuje konfigurovat chování nástroje pouze v současné pracovní složce, proto je uložený v rootu projektu. Hodnoty v souboru jsou uloženy ve formátu JSON. Podle návrhu adresářové struktury bylo nutné ukládat knihovny a balíčky

[6] Více informací na http://www.syntevo.com/smartgit/
7 Více o tomto nástroji na https://bower.io/
8 Více o tomto nástroji na https://getcomposer.org/

třetích stran do složky `www/components/`. Proto stačilo v tomto souboru definovat hodnotu „directory“ a dát ji hodnotu, která odpovídá cestě do potřebné složky. Výsledný soubor je uvedený na následujícím obrázku.

```
{
  "directory": "www/components/"
}
```

Obr. 5.1 - Soubor .bowerrc

Nyní bylo vše připravené pro instalaci nových knihoven. Následně bylo třeba vybrat nebo vytvořit šablonu, která by splňovala požadavky. Jelikož existuje spousta dobře zpracovaných šablon, ve které jsou doladěny do detailů, tak bylo lepší rozhodnout se pro výběr již existující šablony. Firma dostala na výběr z dvou šablon SB Admin a Admin LTE. Jak šablony vypadají lze vidět v příloze (Příloha C – Navrhnuté šablony). Na následující tabulce je možné vidět srovnání již zmíněných šablon.

	Admin LTE	**SB Admin**
Bootstrap	✓	✓
Moderní design	✓	⊗
Poutavější design	✓	⊗
Pro administraci	✓	✓
Fixní horní nabídka	✓	⊗
Komponenty instalované pomocí bower	⊗	⊗
Zdarma	✓	✓

Tabulka 5.1 - Srovnání šablon

Na základě srovnání byla vybrána šablona AdminLTE. Šablona má již v sobě vloženy všechny potřebné komponenty. Jelikož aplikace nebude potřebovat všechny komponenty, tak bylo nutné šablonu od základu předělat.

Po analýze zdrojových kódu[9] šablony byly ve složkách `dist/css` a `dist/js` nalezeny hlavní zdrojové kódy. Díky těmto souborům se AdminLTE tolik odlišuje od ostatních šablon. Soubory v těchto složkách byly přesunuty do odpovídajících složek a to `www/css` a `www/js` a přejmenovány

[9] Do zdrojových kódu lze nahlédnout na https://github.com/almasaeed2010/AdminLTE

z „app.*.css“ na „AdminLTE.*.css“. Pro dodatečné úpravy vzhledu, byly vytvořeny soubory `www/css/style.css` a `www/js/main.js`. Potřebné knihovny byly doinstalovány pomocí `bower instal %nazev_knohovny% --save`.

V následujících bodech jsou uvedeny knihovny, které byly potřeba v aplikaci:

- **Komponenta jQuery**

V aplikaci je použitá verze 3.1.1. a jedná se takový framework pro javascript, který velmi usnadňuje psaní kódu v javascriptu.

- **Komponenta jquery-slimscroll**

Tato komponenta je plugin pro komponentu jQuery. Umožňuje v rámci divu skrolovat.

- **Komponenta bootstrap**

Verze tohoto balíčku byla odvozena od použité verze v šabloně AdminLTE, tedy 3.3.7. Bootstrap je CSS framework, který je v současné době velmi populární. Obsahuje sadu předpřipravených stylů, které lze pohodlně využít.

- **Komponenta bootstrap-modal**

Jedná se o plugin pro komponentu bootstrap a umožňuje vyskakovaní boxů, které pak překrývají část původní stránky.

- **Komponenta bootstrap-table**

Také se jedná o plugin pro bootstrap, která umožňuje jednoduše a rychle upravit a nastylovat tabulku.

- **Komponenta datatables**

Jde o plugin pro komponentu jQuery. Je to velmi flexibilní nástroj pro tvorbu interaktivních tabulek. Umožňují například vyhledávání, řazení podle hodnot ve sloupci, omezení počtu řádku tabulky.

- **Komponenta nette.ajax.js**

Tato komponenta je javascriptová knihovna, která umožňuje AJAX-ovou komunikaci mezi grafickým rozhraním a serverovou částí aplikace. Pro inicializaci této komponenty bylo třeba do souboru `www/js/main.js` vložit následující kousek skriptu „`$(function () { $.nette.init(); });`“.

- **Komponenta nette-forms**

Tato komponenta slouží pro jednodušší zpracování a tvorbu formulářů. Slouží pro validaci dat jak na straně klienta a serveru.

5.2.1 Framework Latte

Tento framework byl vybrán kvůli tomu, že ve firmě je běžně používán. Framework Latte je šablonovací systém pro PHP. Tento framework pracuje se soubory příponou latte. Do těchto souboru je možné vkládat nejen standartní html kod, ale i makra a nmakra. Když šablonovací systém najde v takovém dokumentu makro/nmako, tak provede odpovídající operaci. Makra jsou zapsána ve složených závorkách a nmakra jsou

```
<ul>
<?php if ($loggedIn) : ?>
  <li>přihlášený</li>
<?php else: ?>
  <li>odhlaseny</li>
<?php endif ?>
</ul>
```

```
<ul>
{if ($loggedIn) }
  <li>přihlášený</li>
{else}
  <li>odhlaseny</li>
{/if}
</ul>
```

Obr. 5.2 - srovnání php a latte

Díky frameworku se tvoří přehledný kód. Jednou z velkých výhod tohoto frameworku je, že automaticky escapuje obsah proměnné, která je vypisována. Posledním důvod, díky kterému byl framework Latte vybrán je, že je napojený na framework Nette, který je použit v následující podkapitole.

5.3 Vývoj serverové části aplikace

V některých částech této podkapitoly jsou uváděny třídy včetně jejich jmenných prostorů. To je z toho důvodu, aby došlo k odlišení stejnojmenných tříd, které se nachází pouze v jiném jmenném prostoru.

Pro stahování různých knihoven se použil nástroj `composer`. Zadáním následujícího příkazu na příkazovou řádku `composer create-project nette/sandbox %adresar_projektu%` byla inicializována doporučená adresářová struktura pro projekty postavené na frameworku Nette. Tuto strukturu bylo třeba upravit na požadovanou adresářovou strukturu. Poté bylo nutné tento nástroj nakonfigurovat.

Konfigurace nástroje composer se provádí pomocí souboru `composer.json`. Tento soubor obsahuje data zakódovaná do JSON formátu. Při inicializaci se stáhl tento soubor vyplněný nějakými daty, které je možné měnit podle potřeb. První věcí, kterou bylo třeba změnit, je uložiště, do kterého se ukládají soubory spustitelné přes příkazovou řádku. Výchozím adresářem pro uložení těchto souborů je `vendor/bin/`. Pro změnu této cesty bylo třeba do souboru `composer.json` vložit data, která jsou na níže uvedeném obrázku.

```
"config": {
  "bin-dir": "bin"
}
```

Obr. 5.3 - Soubor composer.json, konfigurace 1

Dále bylo třeba, aby `composer` bral v úvahu i soubory, které se nachází ve složce `vendor/System/` a také, aby se třídy podadresářích `vendor/System/src/` namapovaly na jmenný prostor `System\`. Proto bylo třeba do souboru `composer.json` přidat data, která jsou uvedená na následujícím obrázku.

```
"autoload": {
  "psr-4": {
    "System\\": "vendor/system/src/"
  },
  "classmap": [
    "vendor/System/"
  ]
}
```

Obr. 5.4 - Soubor composer.json, konfigurace 2

Další krok byl výběr vhodného frameworku, na kterém bude aplikace postavena. Byl vybrán Nette framework, protože většina aplikací vyvinutých firmou Irvekon systems s.r.o. bylo postaveno právě na tomto frameworku. Jelikož framework není modulární, tak bylo potřeba udělat řadu uprav, aby tak frameworku pracoval.

Prvním krokem bylo upravit třídu `\Nette\Configurator` a rozšířit ji o funkce umožňující načítání konfigurací modulů. Více o úpravách této třídy je napsáno v podkapitole 5.3.1. Jelikož soubory s pravidly pro překlad URL adres na akci v presentru (dále jen router) se neukládá do žádné třídní proměnné Configuratoru, tak bylo třeba předělat načítání takovýchto souborů, ale vice v podkapitole 5.3.2. V tomto stavu aplikace načítá správně moduly, ale aby i zbytek aplikace mohl pracovat s těmito moduly, bylo třeba ještě upravit třídu `\Nette\Application\UI\Presenter`, více o úpravách v podkapitole 5.3.3. Díky těmto úpravám, se framework stal zcela modulární.

Dalším krokem bylo vytvoření hlavního systémového modulu. Tento modul má na starosti mnoho funkcí například přihlašování a odhlašování uživatelů, jejich správu, správu vytvořených modulů, správu přístupových práv k napojeným modulům. Více o tomto modulu v podkapitole 5.3.5. Poté následovalo napojení rozhraní pro první službu, kterou byla služba Toggl.

5.3.1 Třída \Nette\Configurator

Tato třída slouží pro načítání konfiguračních dat, která vkládá generované třídy tzv. Container. Proto je nutné již před vygenerováním třídy načíst všechny moduly a jejich konfigurační data. Jelikož php 5.6 je objektově orientovaný jazyk, tak se na úpravu třídy využily některé z jeho vlastností, polymorfismu

a dědičnosti. Byla vytvořena třída `\System\Bootstrap\Configurator`, která dědí z třídy `\Nette\Configurator`. V nové třídě byly znovu definovány některé funkce.

5.3.1.1 Funkce createConrainer()

V této funkci se vytváří Container, proto bylo nutné ji volat až po načtení dat modulů. Nejjednodušším způsobem bylo přepsat tělo této funkce. Nejdříve v ní zavolat funkce, které se starají o načítání modulů a jejich konfigurací a následně zavolat původní funkci (funkci předka). Funkce, které jsou volány pro načtení modulů a jejich konfigurací, jsou popsány v podkapitolách 5.3.1.2 a 5.3.1.3.

5.3.1.2 Funkce addModulesConfig()

Tato funkce se stará o načítání tzv. neon[10] souborů. Tento typ souborů byl vyvinutý hlavním autorem Nette a slouží pro serializaci dat. V aplikaci jsou pomocí tohoto typu souboru uloženy všechna konfigurační data.

V této funkci se prochází složky všech dostupných modulů a podle adresářové struktury jsou v adresáři `%cesta_k_modulu%/resources/config/` vyhledány všechny dostupné neon soubory. Nalezené soubory jsou předány funkci `addConfig()` na zpracování, ale o ní více v podkapitole 5.3.4.1.

5.3.1.3 Funkce addModulesRoute()

Funkce `addModulesRoute()` pracuje na stejném principu jako výše uvedená funkce, ale data nejsou ukládá do třídy `\System\Bootstrap\Configurator`, ale do statické třídy `RouterFactory`, která se nachází v systémovém modulu v adresáři pro routery. Více o této třídě v podkapitole 5.3.2.

Při procházení všech složek modulů je zavolána metoda `RouterFactory::createRouter()`, která se podle adresářové struktury nachází ve složce pro routery. Návratová hodnota z této funkce je předána metodě `RouterFactory::addList()` nacházející se v systémovém modulu.

5.3.1.4 Funkce loadContainer()

Tato funkce je volána funkcí `createContainer()`. Stará se o načtení již dříve vygenerované třídy `Container`. Pokud třída ještě nebyla vygenerovaná nebo není aktivovaný produkční režim, tak je volána funkce pro generování kontejneru více v podkapitole 5.3.1.5.

Tato funkce se vůbec nezměnila, ale musela se uvést, protože by funkci `generateContainer()` nebyl předán správný parametr.

5.3.1.5 Funkce generateContainer(DI\Compiler $compiler)

Funkce `generateContainer()` se stará o získání všech potřebných dat z třídy `Configurator` a připravuje je pro generování třídy `Container`.

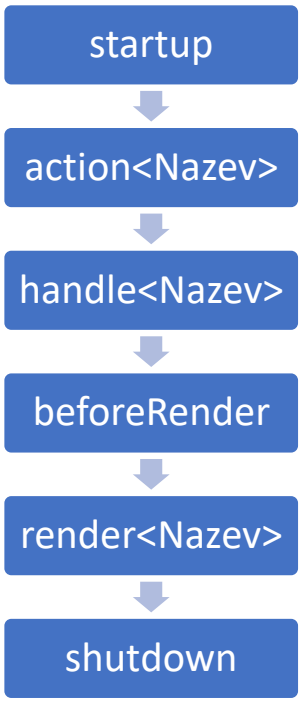
[10] Více na https://ne-on.org/

5.3.2 Práce s routery

Při spuštění celé aplikace na webu, dojde k automatickému zavolání statické metody `\App\SystemModule\Infrastructure\Router\RouterFactory::createRouter()`. Proto bylo nutné, aby se všechna pravidla pro překlad URL adres na akci presenteru a zase naopak vrátila při jejím zavoláním. Tento problém byl vyřešen pomocí návrhového vzoru Singleton. Díky tomu, že existuje pouze jedna instance třídy RouterFactory ze systémového modulu, tak všechna data, která se do instance vloží, se vždy vloží do stejné instance a jsou tak pohromadě. Při automatickém volání již zmíněné funkce jsou navrácena pravidla ze všech modulů.

5.3.3 Třída \Nette\Application\UI\Presenter

Tato třída má na starosti hlavní logiku aplikace. Jak už bylo zmíněno u třídy `\Nette\Configurator`, tak i zde bylo vhodné využit vlastností objektově orientovaných jazyků a vytvořit tak novou třídu `\System\Application\UI\Presenter` dědící od třídy `\Nette\Application\UI\Presenter`. Tato třída definuje určitý životní cyklus[11] každého presenteru.



Obr. 5.5 - Životní cyklus presenteru

Jak už životní cyklus napovídá, načítání dat z třídy `\System\Bootstrap\Configurator` bylo vhodné umístit do funkce `startup()`, ve které probíhá například inicializace proměnných. Více o této funkci v podkapitole 5.3.3.1. Jelikož došlo k velkým změnám v adresářové struktuře aplikace, bylo

[11] Více o životním cyklu na https://doc.nette.org/cs/2.4/presenters#toc-zivotni-cyklus-presenteru

nutné změnit i načítání vykreslovaných šablon. K tomu stačilo upravit funkce formatLayoutTemplateFiles() a formatTemplateFiles(), ale o nich více v podkapitolách 5.3.3.3 a 5.3.3.4.

5.3.3.1 Funkce startup()

Tato funkce bylo přepsána a přidalo se pouze volání funkce `loadModules()`, po jejím zavolání, je volána funkce předka.

5.3.3.2 Funkce loadModules()

Tato funkce se stará o zpracování dat z vygenerované třídy Container. Načtené konfigurační data modulů jsou procházena a převáděna na instance třídy `\System\Application\Module`, která reprezentuje konfigurační data každého modulu. Vytvořené instance jsou uloženy do instance třídy `\System\Application\ModulesList`, tato instance je uložena do třídní proměnné `modules`.

5.3.3.3 Funkce formatLayoutTemplateFiles()

Účelem této funkce je vrátit cesty k hlavním šablonám. V této funkci stačilo vždy vracet cestu k hlavní šabloně, která se vždy bude nacházet v hlavním modulu systému v podadresáři `resources/templates/` a vždy se bude jmenovat `@layout.latte`, ale kvůli rozšiřitelnosti aplikace, jsou vraceny i cesty k možným lokacím hlavní šablony.

5.3.3.4 Funkce formatTemplateFiles

Tato funkce má na starosti vracení cest k šablonám pohledu určitého presenteru. Jak u předchozí funkce bylo nutné změnit cestu k uloženým šablonám.

5.3.4 Třída \Nette\DI\Compiler

Tato třída slouží k zpracování dat a generovaní třídy `Container`. Pro úpravu funkcionality třídy bylo také využito polymorfismu a dědičnosti. Byla vytvořena nová třída, `\System\DI\Compiler`, která dědí od původní `\Nette\DI\Compiler.` Tato třída se musela upravit, protože je třeba trochu odlišným způsobem zpracovávat a validovat konfigurační soubory modulů.

5.3.4.1 Funkce addConfig(array $config, $moduleName)

Funkce addConfig() se stará o zpracování konfiguračních souborů. Každý modul má ve svém konfiguračním souboru vytvořenou sekci s názvem module. Tato sekce obsahuje například nastavení pro generátor agregačních komponent.

Tato funkce vezme sekci `module` a vloží ji do sekce `parameters`. Kdyby se tento přesun neuskutečnil, tak konfigurační soubor by neprošel následnou validací.

5.3.5 Systémový modul

Tento modul tak jako každý jiný modul dodržuje adresářovou strukturu. Hlavním rozdílem mezi tímto a ostatními moduly je ten, že kdyby tento modul chyběl, tak by se aplikace vůbec nespustila, protože pouze v tomto modulu je například konfigurace připojení k databázi, komponenty, které jsou využívány ve všech ostatních modulech.

Pro komunikaci mezi databází a aplikací byla použita knihovna Doctrine2[12]. Tato knihovna jednoduše řečeno mapuje řádky z databázových tabulek na instance odpovídajících tříd (entit). Jedná se o ORM (objektově relační zobrazení) knihovnu. Jelikož knihovna Doctrine2 je rozšiřující knihovna pro framework, tak bylo nutné knihovnu napojit. Pro napojení stačilo do souboru `config.neon` následující kód.

```
doctrine:
        console: Kdyby\Console\DI\ConsoleExtension
        events: Kdyby\Events\DI\EventsExtension
        annotations: Kdyby\Annotations\DI\AnnotationsExtension
        doctrine: Kdyby\Doctrine\DI\OrmExtension
```

Obr. 5.6 - Napojení Doctrine2 na framework Nette

Poté bylo třeba do souboru `config.local.neon` vložit přístupové údaje do databáze. Příklad je uveden na následujícím obrázku.

```
doctrine:
        user: root
        password: pass
        dbname: sandbox
        metadata:
                App: %appDir%
```

Obr. 5.7 - Nastavení přístupu do databáze

Po napojení a nastavení přístupu následovala tvorba entit. Každá entita odpovídá jedné databázové tabulce. Při tvorbě entit bylo postupováno podle dokumentace dostupné na http://docs.doctrine-project.org/en/latest/#mapping-objects-onto-a-database. Dále následovalo vytvoření tzv. fasád. Fasáda je třída, která provádí operace nad entitami například vyhledávání, editaci a vytváření entit a ukládání do databáze. Kvůli přehlednosti bylo pro každou entitu vytvořena fasáda, třída s názvem `NazevEntityFacade`.

Jako další krok následovalo vytvoření základní třídy `BasePresenter`, která dědí od třídy `\System\Application\UI\Presenter`. Jelikož se od této třídy bude pouze dědit a nikdy se nebude

[12] Více o nástroji na http://www.doctrine-project.org/

vytvářet přímo instance `BasePresentru`, tak třída byla implementována jako abstraktní. Pro každou stránku jako například přihlašovací stránka, hlavní stránka, správa modulů atd. se vytvořil zvláštní presenter, který dědí z BasePresenteru a obsluhuje odpovídající stránku.

Funkce `startup()` byla rozšířena o ověřování uživatelů. Jelikož do aplikace mají přístup pouze přihlášení uživatelé, tak nepřihlášení uživatelé jsou automaticky přesměrování na presenter, který obsluhuje přihlašovací stránku. Pokud uživatel je přihlášený, tak je entita uživatele předávána ve funkci `beforeRender()` frameworku Latte, kde je dostupná v proměnné `$userEntity`. V následujících podkapitolách budou podrobněji popsány složitější komponenty a algoritmy.

5.3.5.1 Komponenta aggregationComponentGenerate

Tato komponenta slouží, jak už název napovídá, pro generování agregačních komponent. Úzce spolupracuje s komponentou `aggregationComponentAdd`.

Pro vytvoření této komponenty je nutné jí předat data, která se budou zpracovávat pro vykreslení. Získání dat do agregační komponenty má na starosti presenter, na kterém se agregační komponenta vykresluje, například v modulu Toggl se o to stará funkce `getAll()` viz 5.3.6.2.

Vstupní data předaná komponentě jsou zpracována na základě konfigurace modulu, ve kterém je aplikace vytvářena. Každý modul má své konfigurační údaje pro tuto komponentu uvedeny v `%ModuleName%/resources/config/config.neon`. Zpracování je také ovlivněno tím, zda v nastavení komponenty je definováno seskupování podle sloupce a typ agregační funkce na daném sloupci. O vykreslení datových tabulek se stará komponenta datatables.

5.3.5.2 Základní fasáda pro fasády modulů

Jedná se o abstraktní třídu, která obsahuje základní funkcionalitu pro fasády. Důležitá funkce je `save($objEntity, array $data)`, která slouží pro usnadnění ukládání dat do databáze. Funkce využívá reflexe, díky které lze zjistit informace o určitém objektu za běhu aplikace.

Funkce prochází třídní proměnné, když neexistuje v anotaci `Column` atribut `name`, tak je ve vstupních datech vyhledána hodnota podle klíče, který je stejný jako jméno třídní proměnné, ale kdyby daný atribut existoval, tak je hodnota vyhledána podle klíče, který je stejný jako hodnota atributu `name`. Hodnota je následně předána setteru pro třídní proměnnou. Díky této funkci došlo k velké úspoře kódu zejména u entit, které jsou složeny z mnoha sloupců.

5.3.6 Modul Toggl

Jako první služba, která se bude napojovat byla vybrána služba Toggl[13], která umožňuje uživatelům zaznamenávat odpracovaný čas na různých projektech. Napojení služby na vyvíjenou aplikaci

[13] Více o této službě na https://www.toggl.com/

umožňuje vytvořený modul Toggl. Pro získání dat ze služby se zasílají požadavky na adresu rozhraní (API). Modul pracuje s APIv8 a částečně s APIv2.

Jednou z nejvíce časově náročných funkcí byla analýza dat, která API vrací. Na základě analýzy byla navrhnuta další část databáze. Databázové tabulky v této části mají prefix `toggl_`. Pro analýzu byly posílány následující typy dotazů na API s adresou https://www.toggl.com/api/v8/.

Dotaz	Co dotaz vrací
GET /clients	Všechny klienty
GET /workspaces	Všechna pracoviště
GET /workspaces/{id}	Detail pracoviště
GET /workspaces/{id}/users	Všechny uživatelé v pracovišti
GET /workspaces/{id}/clients	Všechny klienti v pracovišti
GET /workspaces/{id}/projects	Všechny projekty v pracovišti
GET /projects/{id}	Detail projektu
GET /projects/{id}/project_users	Uživatelé podílející se na projektu
GET /time_entries/{id}	Detail času
GET /me	Detail autorizovaného uživatele

Tabulka 5.2 - Přehled použitých požadavků na Toggl APIv8

Jelikož žádný z výše uvedených příkazu neumožňuje vrátit časy všech uživatelů, kteří se podílejí na projektu, ale pouze časy, které zaznamenal uživatel posílající požadavek na server, tak bylo nutné rozšířit modul a částečně implementovat podporu rozhraní v2. Zasláním požadavku `https://www.toggl.com/reports/api/v2/details?workspace_id={id}&user_agent={agen}&since={od}&until={do}` jsou získány časy všech uživatelů, kteří jsou členy pracoviště a probíhaly v době od do.

Jelikož se na serveru Toggl neustále objevují nové údaje, tak je třeba neustále je udržovat aktuální. Toho bylo docíleno pomocí knihovny `Cronner`[14]. Tato knihovna umožňuje jednoduchou zprávu cronnových úloh. Díky této knihovně stačí v anotaci libovolné funkce uvést nastavení cronové úlohy a knihovna cornner se sama postará o to, aby docházelo k pravidelnému volání funkce.

5.3.6.1 Funkce DataPresenter::getAll(array $tables, array $cols)

Tato funkce slouží pro získávání potřebných dat z databáze. Získaná data jsou určena pro generování agregačních komponent. Na základě jmen tabulek, které jsou funkci předány v parametru, jsou

[14] Více o této komponentě na https://componette.com/stekycz/cronner/

propojeny odpovídající tabulky pomocí cizích klíčů a spojovacích tabulek. Jsou vybrány takové sloupce, které jsou uvedeny v parametru `cols`.

5.3.6.2 Funkce DataPresenter::actionDefault()

Funkce je volána před samotným vykreslováním. Získá všechny agregační komponenty k vykreslení, na základě dat, které požaduje daná agregační komponenta jsou z databáze vybrány potřebná data. Pokud komponenta vyžaduje data z více tabulek, tak je volána funkce `getAll.`

5.4 Testování aplikace

Následující kapitola je věnována testování výsledné implementace a testování dílčích části aplikace. Aplikace byla testována dvěma způsoby.

První způsob bylo testování výsledné implementace. Při tomto způsobu došlo k ověření funkčnosti grafického rozhraní a formulářů. Testování probíhalo tak, že se do formulářů zkoušely zadávat náhodná data a sledovala se reakce aplikace na vstupní data. Pro tento způsob testování byli oslovení někteří kolegové a klienti.

Druhým způsobem testováním bylo použitím jednotkových testů. Při těchto testech dochází k otestování samotné třídy nebo funkcí. Testují se návratové hodnoty metod v závislosti na vstupních parametrech. Pro testování byl vybrán nástroj PHPUnit[15]. Hlavním úkolem bylo otestovat modularitu systému, na které celá aplikace stojí a bez funkčnosti této části by aplikace nemohla správně fungovat. Pro spuštění všech testů stačí zadat příkaz `/bin/systém --test`, ale před spuštěním je třeba smazat cache a pokud se aplikace testuje na lokálním počítači, tak je nutné zapnout produkční režim v souboru `bootstrap.php`.

5.4.1 Testování modularity

Pro ověření správné funkčnosti modularity systému byly vytvořeny tři testovací třídy. Tyto třídy se nacházejí ve složce `/vendor/System/tests/Application/`. Bez ověření funkčnosti testovaných tříd by mohla aplikace například přestat pracovat.

Třida HelpersTest

V této třídě se testují funkce třídy `\System\Application\Helpers`. Testovanou třídu bylo nutné testovat, protože na její funkčnosti závisí správné načítaní modulů. Obsahuje například metodu pro získání názvu modulu, která se v aplikaci používá pro identifikaci správného modulu.

[15] Více o nástroji na https://phpunit.de/

Třída HelpersTest

Zde je testována funkčnost třídy `\System\Application\Module`. Jelikož každá konfigurace modulu je ve skutečnosti reprezentována touto třídou, tak bylo nezbytné pro práci s moduly, ověřit její funkčnost. Jsou zde testy na vytváření modulů jak se správnými daty, tak i s neočekávanými. Byly provedeny testy na správné vytvoření třídy a také jestli třída vrací správná data.

Třída ModuleListTest

Tato třída testuje správnou funkčnost třídy `\System\Application\ModuleList`. Testovací třída pro uložení tříd `\System\Application\Module`, nad kterými provádí operace jako je vyhledávání konfigurace pro požadovaný modul. Jsou zde testy příklad na přidávání nové konfigurace pod správným klíčem do pole, vyhledávání konfigurace podle jména modulu.

5.4.2 Testování systémového modulu

Jelikož systémový modul je základní části aplikace a bez její správné funkčnosti by nemusela aplikace vůbec běžet, tak bylo nutné otestovat důležité části tohoto modulu. To zajišťuje sedm testovacích tříd. Testy jsou umístěny v adresáři `/app/SystemModule/tests/`.

Třída DashboardTest

Testovací třída pro `\App\SystemModule\Infrastrucure\Presenters\DashboardPresenter`. Je zde test na ověření správného spuštění aplikace. To se testovalo tak, že se poslal požadavek GET na uvedený presenter a ten nesměl vrátit chybu typu 50X nebo 40X. Další test se opět týkal ověření modularity, ale zde se testovalo, zda presenter má ke konfiguracím modulů přístup.

Testy pro komponentu AggregationComponentAdd

Zde bylo nutné otestovat, zda komponenta generuje správné a povolené možnosti. Tyto možnosti jsou předávány do selectboxu a na základě nich si lze vybrat data (jména sloupců), ze kterých bude tvořena vygenerovaná tabulka. Kdyby toto generování bylo chybné, tak by se by výběr dat z databáze mohl skončit chybou nebo by data byla v nesprávných sloupcích. Další skupina testů byla na vyhledávání ve vygenerovaných možnostech. Vyhledávání v možnostech se využívá při ověření správného vstupu. Při nesprávné funkčnosti by uživatel mohl podvrhnout vstupní data a získat tak přístup databázové tabulky s přihlašovacími údaji.

Testy pro komponentu AggregationComponentGenerate

Jelikož tato komponenta je jednou z nejsložitějších komponent aplikace a je využívána všemi moduly, tak bylo třeba zajistit její správnou funkčnost. Komponentě byly podstrkovány různá data od jednoduchých, na kterých se neprováděla žádná agregace, až po složitější, na kterých se testovalo správné provedení seskupení a provádění implementovaných agregačních funkcí.

5.5 Problémy při vývoji

V této podkapitole jsou popsány komplikace, které se během vývoje aplikace vyskytly a také je zde uvedeno jejich řešení.

Při vývoji aplikace docházelo často k změnám v databázi, ale jelikož aplikace používá cache, tak bylo nutné ji mazat. Proces mazání byl následující. Bylo nutné smazat celý obsah adresáře `/var/temp/`, k tomu bylo zapotřebí administrátorské oprávnění a následně vytvořit složku `/var/temp/sessions/` a nastavit přístupová práva k těmto adresářům. Aby se toto všechno provedlo, tak bylo zapotřebí do příkazové řádky napsat tři příkazy. V rámci zjednodušení a optimalizace byl vytvořen skript `/bin/system` spustitelný přes příkazovou řádku. Tento skript při spuštění s parametrem `--cache-clean` automaticky provede již výše zmíněné operace pro smazání cache.

Během vývoje se také objevily další komplikace. Pro změny v databázi se se využívá nástroje `phinx`[16]. Tento nástroj umožňuje přenositelnost databázového schématu na jiné platformy například z MySQL na Oracle. Pro instalaci komponent týkající se grafického rozhraní se používá nástroj bower, pro instalaci knihoven týkající se serverové části se používá nástroj composer a další nástroje. Na základě velkého množství nástrojů byl skript `/bin/system` rozšířen o další parametry.

Parametr	Popis
--cache-clean	Smazání cache
--db-migrate	Provedení migrace databáze
--db-rollback	Vrácení poslední migrace zpět
--install-all	Instalace knihoven, komponent a nastavení přístupových práv
--install	Instalace knihoven a komponent
--install-composer	Instalace knihoven
--install-bower	Instalace komponent
--test	Spuštění testů

Tabulka 5.3 - Přehled příkazů skriptu /bin/system

[16] Více o nástroji na https://phinx.org/

6 Závěr

Tato práce popisuje vývoj aplikace, zadané společností Irvekon systems s.r.o., která se zabývá programováním webových aplikací a informačních systémů.

Vytvořená aplikace umožňuje přidávat dostupné služby, ze kterých získává data a ukládá je do databáze. Z databáze si lze získat libovolná data. Vytvořená aplikace vyhovuje požadavkům stanovené zadavatelem a všemi vytvořenými testy úspěšně projde.

Základem pro aplikaci byl vybrán framework Nette, protože většina aplikací vyvíjených firmou je na tomto frameworku postavena. Jelikož framework není modulární a po aplikaci se modularita požadovala, tak došlo k značným úpravám, které změnily framework na modulární. V době návrhu aplikace docházelo k častým konzultacím se zadávající firmou, která většinou upřesňovala své požadavky.

Do budoucna se plánuje rozšířit komponentu pro agregaci o generování grafů, také rozšířit aplikaci o podporu dalších služeb například Sklik, Mailchimp. Kromě vytvoření nových modulů se plánuje přidat podporu pro notifikace, které budou sloužit například pro upozornění klientů na nově přidané moduly.

7 Literatura

[1] *HTML5 a CSS3: názorný průvodce tvorbou WWW stránek*. Brno: Computer Press, 2012, 440 s. ISBN 978-80-251-3733-8.

[2] Web Services Architecture. *W3C* [online]. W3C, 2004 [cit. 2017-04-13]. Dostupné z: https://www.w3.org/TR/ws-arch/

[3] DANĚK, Petr. Seriál Velký test PHP frameworků. In: *ROOT.CZ* [online]. Praha: Krčmář, 2008 [cit. 2017-04-25]. Dostupné z: https://www.root.cz/serialy/velky-test-php-frameworku/

[4] DOČEKAL, Daniel. Facebook doporučuje přátele i dle geolokace. Jak je vlastně najde? In: *Lupa.cz* [online]. Praha: Slížek, 2016 [cit. 2017-05-02]. Dostupné z: https://www.lupa.cz/clanky/facebook-doporucuje-pratele-i-dle-geolokace-jak-je-vlastne-najde/

[5] KOBELKA, Petr. Jak na Nette s funkčními moduly. In: *Egoblog* [online]. Kobelka, 2014 [cit. 2017-05-07]. Dostupné z: http://www.egoblog.cz/jak-na-nette-s-funkcnimi-moduly/

[6] BERNARD, Bořek. Seriál MVC a další prezentační vzory. In: *Zdroják* [online]. Praha: Hassman, 2009 [cit. 2017-05-02]. Dostupné z: https://www.zdrojak.cz/serialy/mvc-a-dalsi-prezentacni-vzory/

[7] ČERNÝ, Honza. Jak začít a propojit Doctrine a Nette Framework. In: *Honza Černý: Blog* [online]. Černý, 2015 [cit. 2017-05-04]. Dostupné z: http://blog.honzacerny.com/post/3-jak-zacit-a-propojit-doctrine-a-nette-framework

[8] *Doctrine Project: Welcome to Doctrine 2 ORM's documentation!* [online]. Berlin: Doctrine, 2010 [cit. 2017-05-07]. Dostupné z: http://docs.doctrine-project.org/en/latest/

[9] Toggl API Documentation. *GitHub* [online]. Fonó, 2013 [cit. 2017-05-07]. Dostupné z: https://github.com/toggl/toggl_api_docs

[10] LECKY-THOMPSON, Ed a Steven D. NOWICKI. *PHP 6: programujeme profesionálně*. Brno: Computer Press, 2010. Programujeme profesionálně. ISBN 978-802-5131-275.

[11] *Nette* [online]. Praha: Nette Foundation, 2016 [cit. 2017-05-10]. Dostupné z: nette.org/

[12] Cookies and User Identification. *Google Developers* [online]. USA, Kalifornie: Google, 2016 [cit. 2017-05-10]. Dostupné z: https://developers.google.com/analytics/devguides/collection/analyticsjs/cookies-user-id

Zdroje obrázků

[1] http://www.freeiconspng.com/img/3709

[2] http://pngwebicons.com/upload/small/laptop_PNG8919.png

Seznam obrázků

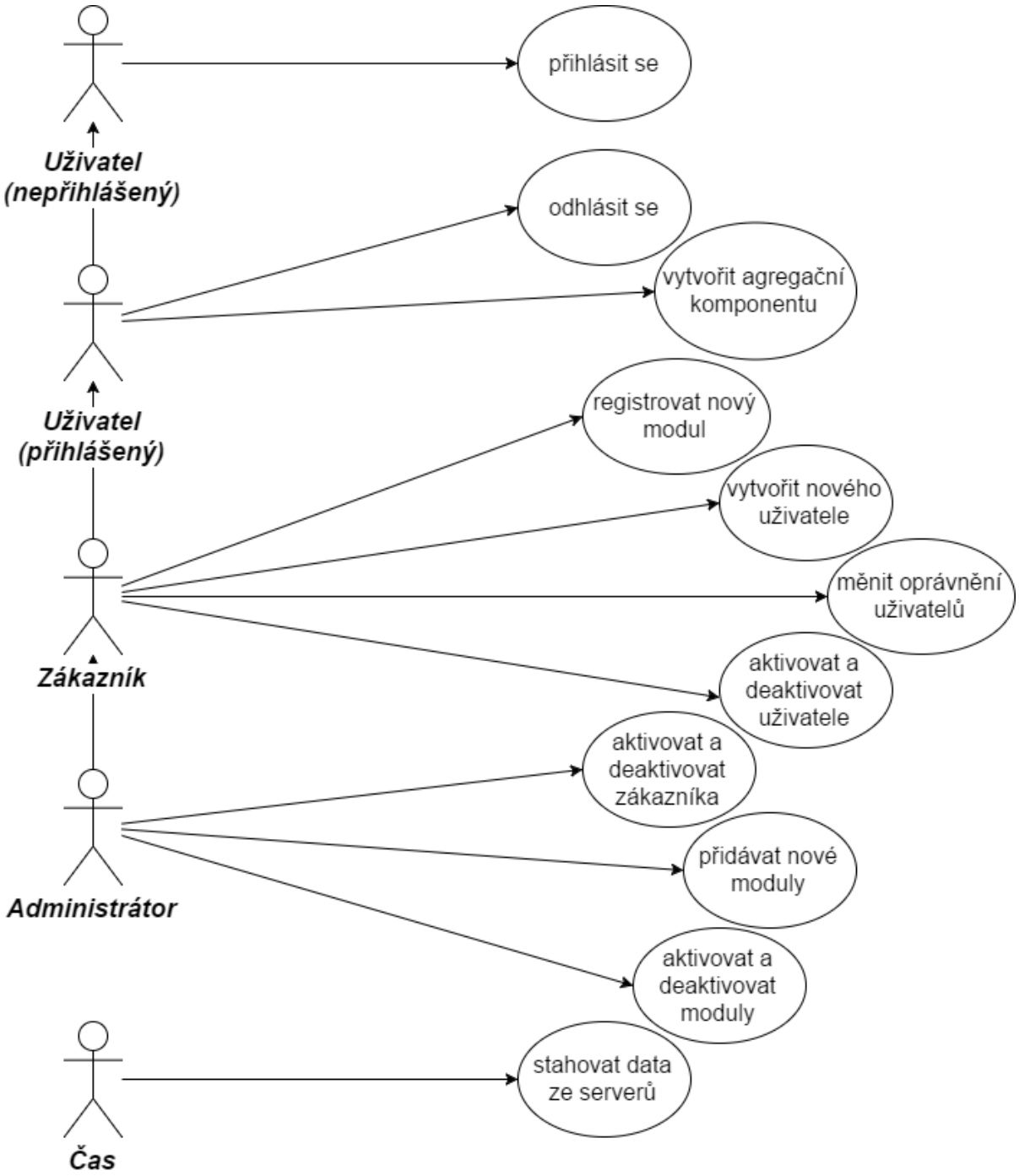
Seznam tabulek

Seznam příloh

Příloha A - Obsah přiloženého CD

- src/ - složka se zdrojovými soubory aplikace
- images/ - složka s přiloženými obrázky
- technicka_zprava.pdf - bakalářská práce ve formátu PDF
- technická_zprava.docx – bakalářská práce ve zdrojovém formátu Word
- README.md – návod na instalaci aplikace

Příloha B - Use case diagram



Příloha C – Navrhnuté šablony