VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

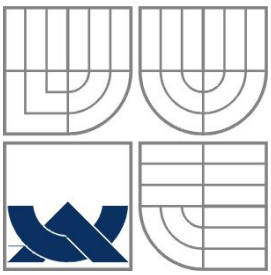
INFORMAČNÍ SYSTÉM S XSLT ŠABLONAMI

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

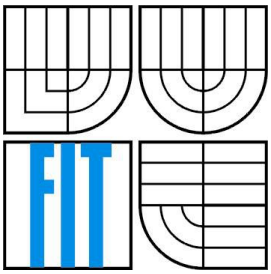
AUTOR PRÁCE
AUTHOR

František Mazura

BRNO 2015



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

INFORMAČNÍ SYSTÉM S XSLT ŠABLONAMI

INFORMATION SYSTEM WITH XSLT TEMPLATES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — František Mazura
AUTHOR

VEDOUCÍ PRÁCE — Ing. Radek Burget, Ph.D.
SUPERVISOR

BRNO 2015

Abstrakt

Cílem této práce je vytvořit univerzální framework pro malé a středně velké informační systémy. Pro řešení zvoleného problému jsem použil jazyk PHP v kombinaci se šablonami XSLT při REST architektuře. Vytvořený framework poskytuje možnost rychlého vytvoření informačního systému s využitím XSLT, AJAX a možností vzdáleného ovládání přes API. Hlavním výsledkem je rychlý framework s jednoduchou modulárností.

Abstract

The goal of this work is to create a universal framework for small and medium information systems. For resolving this problem I use the PHP language together with the XSLT sheets and the REST architecture. The framework provides a fast creation of information systems using XSLT, AJAX and the possibility of a remote access by an API. The main result is a fast framework with a simple modularity.

Klíčová slova

PHP, XSLT, framework, REST, MVC

Keywords

PHP, XSLT, framework, REST, MVC

Informační systém s XSLT šablonami

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením
Ing. Radka Burgeta, Ph.D
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
František Mazura
13. 5. 2015

Poděkování

Chtěl bych poděkovat Ing. Radku Burgetovi, Ph.D. za odborné vedení práce a cenné rady, které mi pomohly při tvorbě této práce.

Obsah

1 Úvod

Obecně framework, když je dobře navržený, je velice důležitá věc při vývoji aplikací. Učí nás dobrým programátorským praktikám a nemusí se řešit rutina a nižší úrovně vývoje. V informačních systémech nám může násobně zvýšit rychlost vývoje a odprostit nás od nižší úrovně programování, jako je obsluha databáze, řízení přístupu do informačního systému, API rozhraní a hlavně takzvané směrování. Díky němu lze zajistit jednoduché přidávání nových modulů a bezpečnost přístupu.

Cílem mé práce je udělat framework, který splňuje tyto vysoké nároky, jak v oblasti výkonu a bezpečnosti, tak v oblasti rychlosti vývoje a náročnosti. Framework, který zajistí směrování informačním systémem, je založen na model-view-controller architektuře, je zaměřen na moduly – stránka v informačním systému se vytvoří zavoláním jednotlivých modulů co se na ni budou nacházet, umožňuje přístup přes webové API, kde je základní podpora XML a JSON výstupu a přístup do API je řešen pomocí OAuth2, má centrálně nastavitelné přístupy k jednotlivým modulům a centrální nastavení posílání dat modulům (POST, GET apod.), přístup do databáze je vyřešen za pomoci PDO (což zajistí i nezávislost na použití databázového serveru), určuje adresářovou strukturu a garantuje možnosti přístupů do jednotlivých složek. S využitím XSLT je další výhoda frameworku, kdy pomocí tohoto přístupu se odděluje programátor PHP a kodér v HTML pomocí XSLT. Je pak samozřejmostí oddělení vývojáře dynamické části webu (AJAX) a vývojáře aplikace, která využívá API rozhraní. Při tomto návrhu je také možná kooperace více PHP programátorů na jednom projektu, kde jednoduchým rozdělením úkolu nenastává problém kolizí a s využitím verzovacího nástroje (například GIT) je plně automatizovaná synchronizace zdrojových kódů. A v neposlední řadě také vyřešit ošetření chyb a to jak přímo v PHP skriptech tak i v modelové části, kde se pracuje s databází a data neprojdou např. regulárním výrazy.

2 Cíle práce

Navrhnout framework pro kooperaci programátorů a kodérů nad vývojem informačního systému. Pro tuto kooperaci využít jazyk XSLT, který vyřeší vzájemné propojení systémové a prezentační části systému.

- Vytvořit PHP framework pro tvorbu informačních systémů
- Použít MVC přístup a zajistit snadnou modulovanost
- Umožnit nezávislý vývoj PHP programátorů, HTML kodérů a vývojářů aplikací propojených se systémem za použití jazyka XSLT
- Zajistit centrální access control list přístupů do systémů
- Umožnit podporu více oprávnění uživatelů informačního systému a zajistit snadné přidávání dalších úrovní oprávnění
- Podporovat nastavitelnost základních výstupu (HTML, JSON a XML) a umožnit přidávání dalších
- Navrhnout modelovou část systému, aby se nové modely přidávaly jednoduše a byla zajištěna automatická kontrola vstupních dat
- Zajistit směrování dle aktuální URL
- Složky se zdrojovými soubory a důležitými daty nemít přístupnou přes URL
- Navrhnout systém tak, aby se výsledný informační systém vytvářel z jednotlivých modulů
- Zajistit ošetření chybových hlášení a jeji logování
- Vytvořit podporu pro API a zabezpečit přístup přes OAuth2

3 Teoretické znalosti a použité technologie

Znalosti a technologie zde uvedené je část používaných. Byly vybrány s ohledem na modularitu, výkon a bezpečnost. Veškeré tyto technologie jsou open source.

3.1 Framework

Je struktura pro převzetí běžných činností při vytváření informačního systému (směrování, oprávnění apod.).

Z hlediska vývoje se jedná o velice důležitou věc, protože nám výrazně zrychluje a zefektivňuje vývoj, díky čemu vytvoříme za méně času více. Pomáhá dobrým programátorským návykům a umožňuje lepší koordinaci v týmu. Řeší nižší vrstvy vývoje (přístup do databáze, směrování ...) a obecně nám garantuje bezpečnost a modularitu.

Tato práce pojímá framework za podpůrný vývojářský základ, na kterém můžete naprogramovat rozsáhlý informační systém. Zajišťuje směrování na konkrétní metody tříd, různé módy používání typu HTML, rozhraní pro API a rozhraní pro AJAX, regulární výrazy a filtry, ošetření chyb, přístup do databáze, kontrolu přístupů. Tento přístup v sobě zahrnuje ošetření chyb a bezpečnost přístupu do webu.

3.2 PHP

PHP: Hypertextový preprocesor je multiplatformní programovací skriptovací jazyk navržený pro vytváření webových stránek [4]. Podporuje objektově orientovaný návrh.

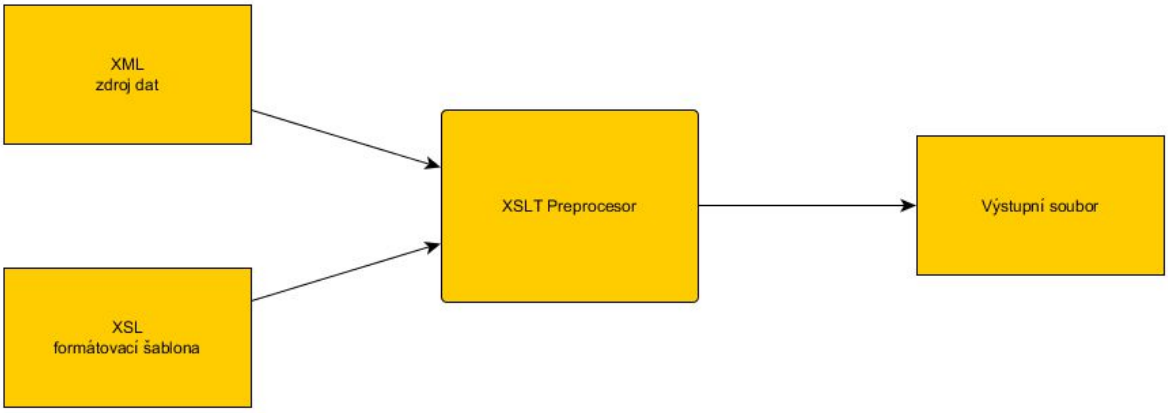
Díky své otevřenosti (Open source licence) a rozšířenosti (přibližně 82 % webů běží na PHP - http://w3techs.com/technologies/details/pl-php/all/all) je pro informační systémy dobrou volbou.

3.3 XSLT

XSLT – eXtensible Stylesheet Language je jazyk pro spojení dat (XML) a šablony (XSLT) do výstupu (např. HTML) [2].

Tato technologie byla zvolena pro oddělení PHP vývojáře a kodéra. Díky tomuto oddělení lze efektivněji (paralelně) pracovat na projektu ve více lidech. Tým se domluví na struktuře XML dokumentu a poté na sobě může nezávisle pracovat.

Technologie XSLT patří pod XML, které patří pod SGML [1]. Stejně jako XML pracuje s Document Object Model (DOM), který se využívá pro odkazování do XML souboru (Xpath) [1].



Ilustrace 1: XSLT - schéma

Z obrázku Ilustrace 1: XSLT - schéma je patrné, jak celý proces funguje. Textový procesor nejprve načte XSLT šablonu. V XSLT šabloně jsou řídící příkazy, kterými se lze odkazovat na zdroj dat – XML dokument, který je před použitím také převeden do DOM. Výstupní soubor může být různorodý. Lze generovat jakékoliv textové výstupy na základě XSLT šablony a XML datech. Ve frameworku se využívá pro generování HTML výstupu.

XSLT jazyk je aplikací jazyka XML [1]. Pro oddělení XSLT příkazů a XML tagů (např.: <klient>Jan Novák</klient>) používá XSLT jmenné prostory (namespace). Jmenný prostor je reprezentován prefixem xsl v elementu stylesheet ve kterém musí být uzavřeno celé XSLT (např: <xsl:stylesheet> xslt přikazy... </xsl:stylesheet>). Základem XSLT jsou šablony [1]. Díky nim lze jednoduše předdefinovat formátování jednotlivých vstupů (např. řádek tabulky uživatele), tyto šablony jsou jednoduše znovupoužitelné a změna formátování se napříč systémem vykonává na jednom místě. XSLT podporuje cykly (cyklus foreach zajistí průchod celým DOM podstromem z XML a jednoduše lze vypsat například tabulku) a podmínky. Lze také využívat rekurzivní volání.

3.4 REST

REST – Representational State Transfer je architektura rozhraní orientované na data [8].

S využitím REST je implicitně řešena podpora pro AJAX a API pro externí přístup do informačního systému. Zajišťuje lepší systémovou modularitu a znovupoužitelnost metod (například metoda getUserDetail se může využívat napříč celým systémem včetně přístupu přes API a bude naimplementována na jednom místě).

Framework používá REST pro AJAX, kde se základní stránka načte spojením XML a XSLT šablonou, z tohoto spojení je vygenerován HTML soubor, který tvoří základní HTML layout. Přes AJAX je tento HTML layout doplněn daty. Ve frameworku se nadefinují funkce (getUserTableAction, createUserAction, editUserAction …), které se po povolení v access control list

mohou volat přes AJAX. Díky těmto funkcím, které vypisují a modifikují data lze informační systém udělat dynamičtější a šetří se server díky možnosti načítání pouze těch dat, která jsou aktuálně potřeba. Obdobným způsobem je řešena podpora API.

3.5 MVC

Model-view-controller je architektura, kde se od sebe oddělují 3 základní složky.

- **Model**

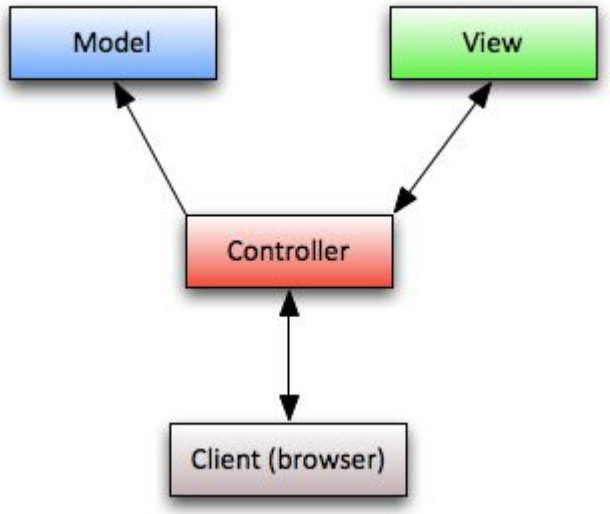
Model je část, kde se stará výhradně o práci s daty, typu vkládání, editace, mazání a čtení.

- **View**

Pohled je část, která nám zajišťuje grafickou reprezentaci, kterou uvidí uživatel informačního systému.

- **Controller**

Řadič, je centrálním prvkem. Volá model v případě potřeby dat, transformuje je a obsluhuje požadavky uživatele informačního systému. Následně zapracovaná data předá k reprezentaci pro uživatele pohledu.



Ilustrace 2: MVC architektura

3.6 AJAX

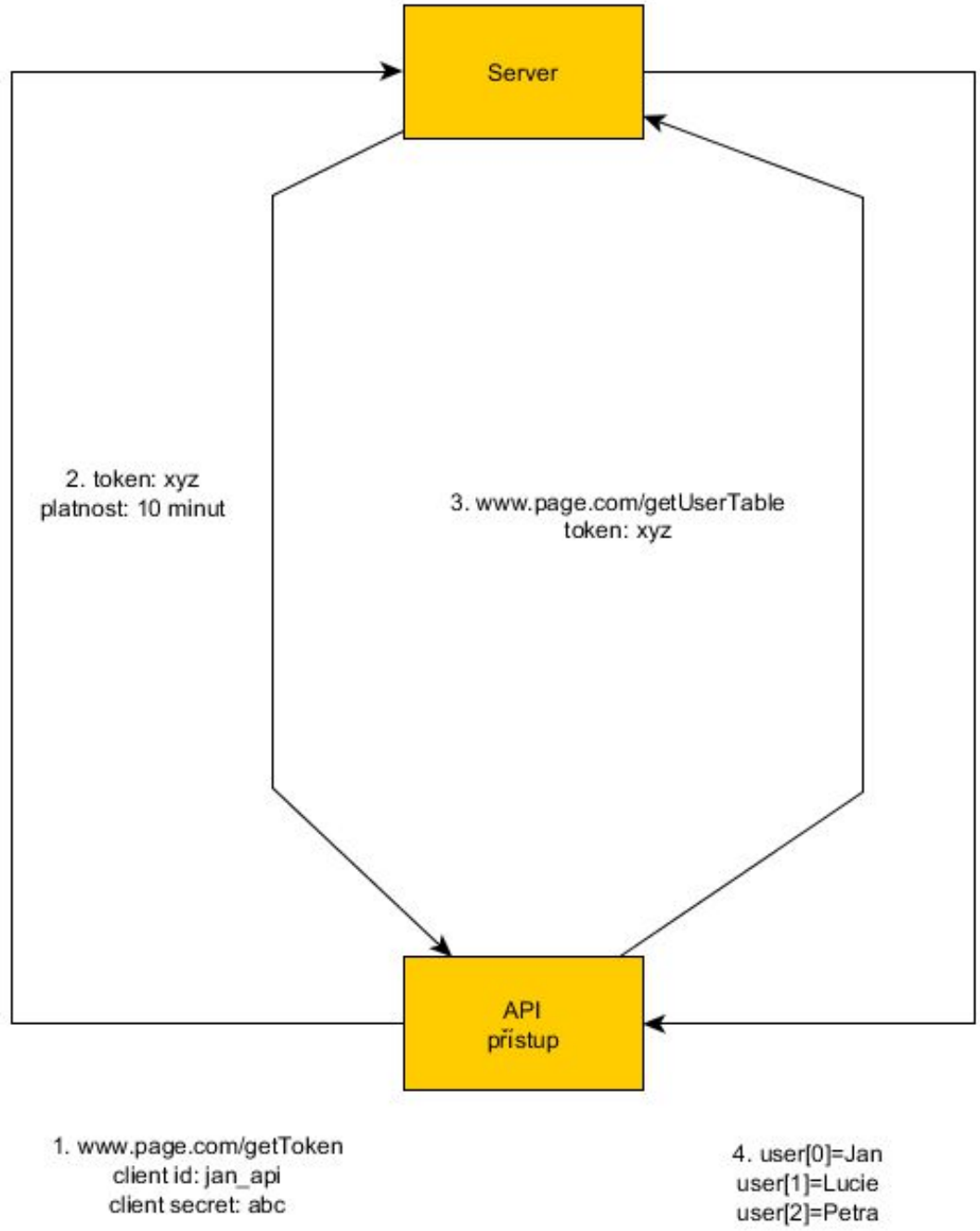
AJAX – Asynchronní JavaScript a XML je způsob architektury komunikace klient-server, kdy klient na pozadí přes JavaScript pošle dotaz na server.

Pomáhá snižovat zátěž serveru, umožňuje realtime aktualizaci stránky bez načtení a lze dělat lépe uživatelsky orientované rozhraní.

3.7 API

API Application Programming Interface – je rozhraní pro strojovou komunikaci s informačním systémem. Umožňuje uživateli používat aplikace třetích stran, popřípadě lze propojit přes toto rozhraní více nezávislých modulů informačního systému.

3.8 OAuth2



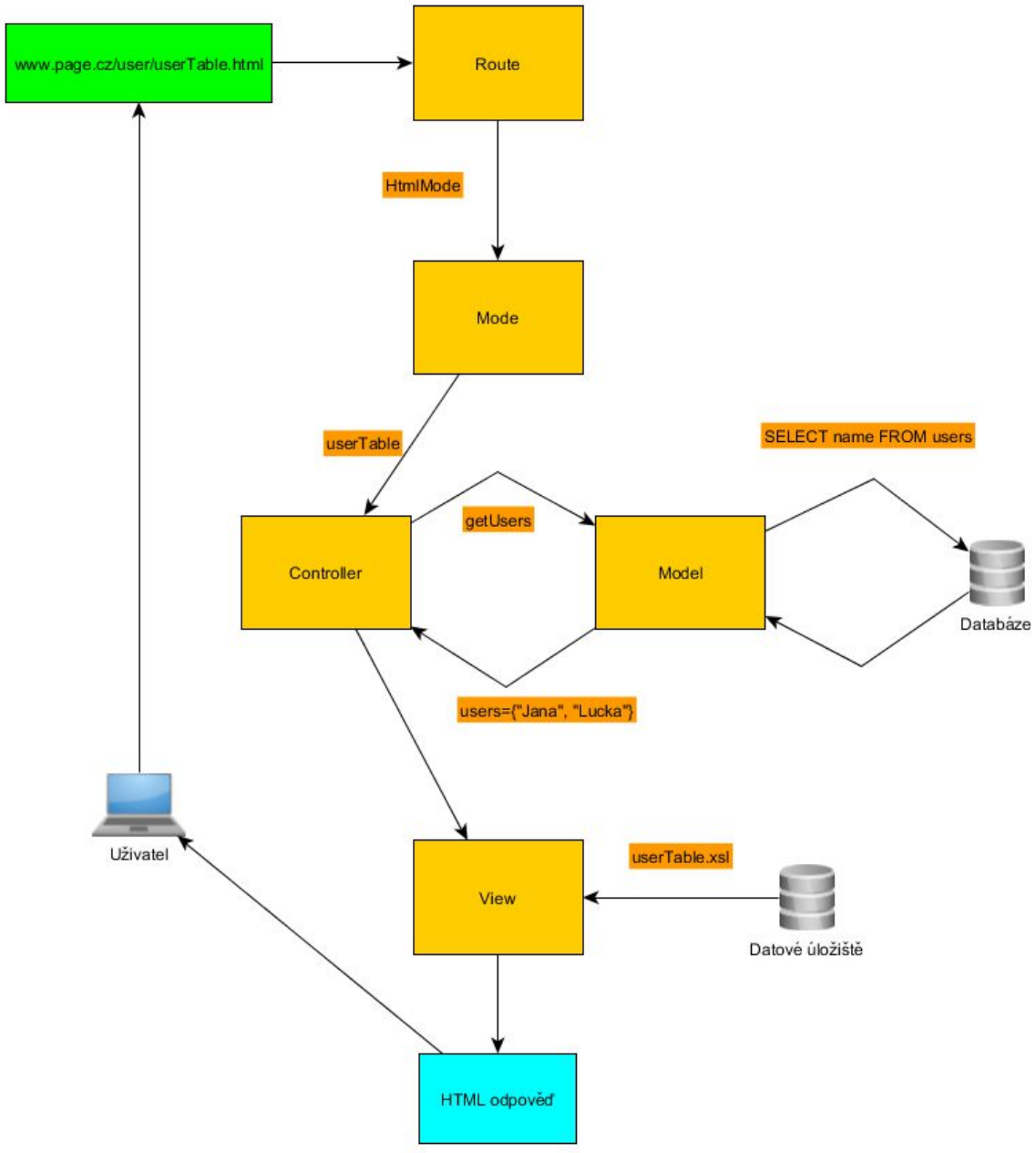
Ilustrace 3: OAuth2 - schéma přihlašování

OAuth2 je otevřený protokol pro zajištění autentizaci a autorizaci uživatele [5]. Tento protokol se používá převážně v API. Pokud chce uživatel informačního systému povolit přístup na svůj účet aplikaci 3. strany, je potřeba zajistit alternativní autentizaci a autorizaci než je klasické uživatelské jméno a heslo (je velké bezpečnostní riziko, aby uživatel informačního systému předal své heslo 3. straně). OAuth2 toto řeší přes Client ID, což je identifikátor uživatele (zajistí autentizaci) a přes

Client secret. Client secret je tajný kód, který zajistí autorizaci. OAuth2 umí i jiné způsoby autentizace a autorizace, které framework v základním sestavení nepodporuje.

4 Návrh systému

Framework je navržen v architektuře model-view-controller, která je zmíněna v teoretické části. Je kladen důraz na jednoduchost přidávání nových věcí a modularitu skládání více věcí do jednoho celku. Každá ze základních řídících tříd má přesně daný účel podle kterého programátor implementuje potomky třídy.



Ilustrace 4: Schéma návrhu frameworku

Na obrázku Ilustrace 4: Schéma návrhu frameworku je zobrazen návrh pro zpracování požadavku na HTML odpověď, kde je využito XSLT. Třída *Route* zastřešuje celý framework a má za úkol z URL rozpoznat kterou funkci má zavolat. Třída *Mód* řeší formát a typ komunikace. Třída *Řadič* v sobě obsahuje logiku informačního systému, kterou aplikuje na data z třídy *Model*. Třída *Model* slouží pro obsluhu práce s daty (převážně databáze). Třída *Pohled* formátuje výstup z *Řadiče* pomocí šablony XSLT.

4.1 Route

Třídou *Route* začíná spuštění celého frameworku. Tato třída *Route* má za úkol nastavit PHP skript. Dále se pak stará o směrování. Z URL které rozdělí na jednotlivé části odvodí, který mód a moduly se mají zavolat.

V základní verzi nastavuje obsluhu chyb. Obsluha chyb je směřována na třídu *ErrorEx*, která dědí PHP třídu Exception. Tato třída *ErrorEx* je navržena pro zakladní ladění a umožňuje přes nastavení proměnné *DEBUG* vypisovat výstup do stránky a implicitně ukládá chybové hlášení PHP skriptu do databáze.

Dále pak nastaví sezení, znakovou sadu frameworkové proměnné a předá obsluhu módu. Do této třídy programátor využívající tento framework při programování informačního systému většinou nezasahuje.

4.2 Mód

Mód je ve frameworku obsažen pro nastavení výstupu. Výstupem je myšleno např.: HTML, JSON, API, CRON adt. Mód má vedle volby výstupu zajistit kontrolu přístupu k řadičům a jejich příslušným metodám. Tato část frameworku je navržena tak, aby bylo možné přidávat nové módy. Základní funkce má na starost abstraktní třída *AbstractMode*, která se v daném módu zdědí. Mód je nezávislý na řadiči a pohledu které volá.

Mód kontroluje přístup k metodám. Pro kontrolu se používá statické pole ACL. Pokud informační systém potřebuje využít dynamického nastavování access control list, tak lze této možnosti využít, kdy se do souboru acl.php naimplementuje třída, která poskytne možnosti dynamické úpravy a čtení ACL a upraví se kontrola přístupu ze statického pole na danou třídu kontroly přístupu.

Tato část frameworku se také stará o obsluhu neoprávněného přístupu (access control list zamítl přístup). Tato obsluha se dá nastavit pro každý mód zvlášť.

4.3 Řadič

Řadič je část frameworku, ve kterém vývojář informačního systému nejvíce pracuje. Každý řadič je jedna úroveň oprávnění. Každá úroveň oprávnění ve frameworku je implementovaná v jedné třídě řadiče. Oprávnění se rozlišují dle názvu třídy. Název třídy vždy končí *Con* (např.: *AdminCon* – třída s metodami pro administrátora).

Ve frameworku není programátor omezován, co se počtu oprávnění týče (stejně jako u módu). Omezení je jen v názvu oprávnění, kde musí být třída nazvána velkým písmenem (jako všude jinde v systému) a může obsahovat pouze znaky z abecedy bez diakritiky (a poté k názvu třídy na konec přidat *Con*). Třída úrovně oprávnění typu řadič, je také povinna dědit třídu *AbstractCon*, která přes svůj konstruktor zajistí kompletní obsluhu třídy úrovně oprávnění řadiče. Programátor se nestará o volání svých funkcí, vše zajišťuje třída *AbstractCon*. Volání funkcí je navrženo přes kořenovou metodu (kterou volá uživatel v url). Např. při url (…)/admin/users, je kořenová metoda user, respektive userAction (přídavek Action označuje metody, které lze volat přes *AbstractCon*). Ta se zavolá. Při volání se kontrolují dvě možnosti přístupu. Implicitní je access control list (nastavení přístupu), kde se nastavuje dané metodě, jaké oprávnění potřebuje. Implicitně je metoda zakázána. Vývojář musí explcitně nastavit přístup k metodě. Druhý přístup je přístup k datům od uživatele. Zde se explicitně nastavuje (implicitně metoda nedostane žádná data) odkud daná metoda může dostat data (GET a POST, popřípadě lze doimplementovat další zdroje získání dat). Pokud je nastaveno, daná metoda do parametru dostane předvolená data. Lze také využít priority dat, pokud např POST není zvolen, lze nastavit aby se použil GET, popřípadě libovolný počet dalších alternativ. Při zavolání kořenové funkce volené uživatelem může programátor volat v návaznosti další metody. Volání obstarává framework. Obstarání volání pomocí frameworku (metodou) je nutné pro uložení kostry volání, která je poté předána pohledu (view) a podle toho se při použití HTML módu automaticky načtou příslušné XSLT soubory pro dané metody. Dále lze v metodě používat model pro přístup k datům. Návratová hodnota metody (pole) je předávána pohledu (view) a slouží jako data v případě módu HTML pro XSLT, u JSON se pole převede do serializačního formátu JSON a pošle se klientovi. Pokud daná metoda nepotřebuje tělo (vrací prázdné pole a nevyžaduje žádnou akci), stačí ji přes metodu z abstraktní třídy řadiče zavolat a nemusí být naimplementovaná. Tohoto se využívá převážně v HTML módu, kdy přes modulární architekturu je například modulu formuláře vložení uživatele, kde není potřeba žádná akce, pouze vypsání daného modulu z XSLT je potřeba. Díky tomu programátor nemusí psát funkci a automaticky se zavolá daný modul a kód zůstává přehlednější.

4.4 Model

Model slouží ve frameworku k práci s databází. Zde vývojář umisťuje své skripty pro účely načítání, vkládání, editování a mazání dat. Pro modelové třídy je připravena třída *AbstractModel*, ze které

všechny třídy dědí. Tato třída *AbstractModel* nám zajišťuje načítání proměnných pro PDO s využitím filtrů (kontroly vstupu). Kontrola vstupu se konfiguruje pro každý model zvlášť. Model je navržen pro používání přes atributy. Každá metoda si z nich vezme potřebná data a zpracuje je.

Pro každý atribut je metoda pro kontrolu vstupu. Tato metoda automaticky zkontroluje vstup a popřípadě nastaví příznaky špatného vstupu. Nastavení kontrolovaných atributů probíhá jednoduše přes pole, kam se vepíší názvy atributů, které se mají kontrolovat a poté metoda pro kontrolu automaticky zavolá všechny potřebné metody. O kontrolu se stará speciální třída *Filter*, ve které jsou metody pro kontrolu, které následně model využívá. Následné špatné vyhodnocení vstupu se získá metodou getError(), která má přístup k chybovému logu a je na programátorovi, aby ji upravil pro své účely. Výstup z této funkce je poté využíván na informování uživatele o špatných hodnotách vstupu. Tento výpis se vypisuje do API nebo AJAXu.

4.5 Pohled

Pohled (view) je další dynamická část frameworku. Počet pohledů odpovídá počtu módů (tzn. že pro každý mód je speciální třída) krát počet řadičů (pro každý řadič – úroveň oprávnění – je potřeba jedna třída pohledu). Toto zajišťuje mnohonásobně větší modulovatelnost a přizpůsobitelnost pro různá použití.

Pohled (view) získá přes mód od řadiče (controller) frameworkem vytvořené pole, které má jasně definovanou stukturu, ve které se ukládají jak návratové hodnoty, tak volané metody a jejich pořadí (od kořenové k poslední volané). Třída *AbstactView* zajistí v konstruktoru automatické volání daných funkcí v pořadí v jakém byli volány v řadiči. Pokud daná metoda k příslušné úrovni oprávnění neexistuje, implicitně se předají data pro další zpracování dále do pohledu (view). Díky tomuto řešení nemusí programátor pracně vytvářet pro každý modul metodu a zároveň si může upravit implicitní metodu pro zpracování. V konkrétních metodách lze poupravit pole, popřípadě spojit více dat z různých modulů. Díky tomu se uspoří výkon a stejná data se nemusí získávat na vícekrát.

Pro HTML pohled, je důležitá kombinace s XSLT šablonamy. Každá metoda musí mít svojí šablonu uloženou na patřičném místě (/template/(*název oprácnění*)/(*název metody*)). Pokud tomu tak není, framework ohlásí chybu a nedovolí další provádění skriptu. Pokud existuje soubor s šablonou, otevře se, načte se a přidá se do skriptem generovaného výsledné XSLT šablony. Tím, že XSLT šablonu poskládá skript z jednotlivých šablon metod se zajišťuje vysoká modulovatelnost a škálovatelnost. Framework obstarává také přístup k datům, kde dynamicky pro každý modul udělá v šabloně proměnnou, která odkazuje na data aktuálního modulu. Tímto řešením kodér si nemusí pamatovat úplnou cestu Xpath, ale stačí mu použít vygenerovaná proměnná. Framework také automaticky zaobalí XSLT zrojový kód do šablony, kterou automaticky pojmenuje na základě názvu zavolané funkce v řadiči. Návrhář (kodér XSLT) například navrhne šablonu stránky, menu a výpis

uživatelů do 3 metod (3 soubory s XSLT šablonami) a uživatel zavolá kořenovou funkci user a v ní programátor zavolá metody šablona stránky, menu a výpis uživatelů. Poté se v pohledu poskládají tyto 3 moduly (XSLT šablony) dohromady do jednoho souboru a stránka se vygeneruje. Touto metodou je také vedle modulovatelnosti zajištěna vysoká znovupoužitelnost (např metody/šablony menu a šablona stránky se mohou vyskytovat na všech stránkách systému a jsou naprogramované a nastylované na jednom místě).

Ostatní módy jsou řešeny obdobným způsobem, kdy se převezme výstup z řadiče a převede se ve většině případu to určitým způsobem naformátovaného textu. Před vypsáním, se u některých pohledů aplikují filtry pro odstranění různých řetězců.

5 Implementace

Framework je naimplementován tak, že se jako první zavolá třída *Route*, která v sobě pak volá třídu *Mode* a takovýmto systémem se volají další části systému. Třída *Route* má dobu životnosti po celou dobu frameworku a všechny další třídy jsou volané v ní. Toto platí rekurzivně na další řídící třídy ve frameworku.

5.1 Route

Třída *Route* spouští celý framework. Při instanci třídy *Route* se postupně spouští jednotlivé metody.

```
<?php
	require_once './../src/route/Route.php';
	$route = new Route();
?>
```

Ilustrace 5: Spuštění frameworku

Obsluha chyb

Jako první se nastaví obsluha chybových hlášení PHP skriptu. Ve frameworku je již implementovaná třída na obsluhu chyb *ErrorEx*, je ale možné si naimplementovat vlastní třídu.

Nastavení sezení

V nastavení sezení se inicializuje PHP globální proměnná *SESSION*. Framework nastaví *SESSION* tak, aby se při uložení na straně klienta používal pouze *coockies* a přístup se omezil na soukromý. Pokud je k dispozici šifrovaný zabezpečený přenos, nastaví se nuceně přenos identifikátoru sezení přes HTTPS. Pro zvýšení bezpečnosti se při každé inicializaci sezení, znovu vygeneruje jednoznačný identifikátor sezení, aby v případě jeho odcizení se zamezilo zneužití.

Nastavení znakové sady

Framework používá standart UTF-8 jak pro databázi, tak pro PHP vstup a výstup. Framework také nastavuje lokaci dle operačního systému, na kterém server běží.

Získání adresy

Framework směruje podle adresy v adresním řádku. Struktura je pevně daná. Pro přístup přes web je to:

řadič/*metoda*./*mód*

5.1.1 Řadič

Je hlavní třída pro obsluhu požadavků. V architektuře model-view-controller zastává pozici controller. Architektura frameworku je navržena tak, aby každý řadič obsluhoval jednu úroveň oprávnění. (např.: třída *IndexCon* bude obsluhovat uživatele, který nemá žádný přístup do systému – návštěvník bez oprávnění. Třída *AdminCon* bude obsluhovat uživatele, který je již přihlášen do informačního systému a má oprávnění administrátora)

5.1.2 Metoda

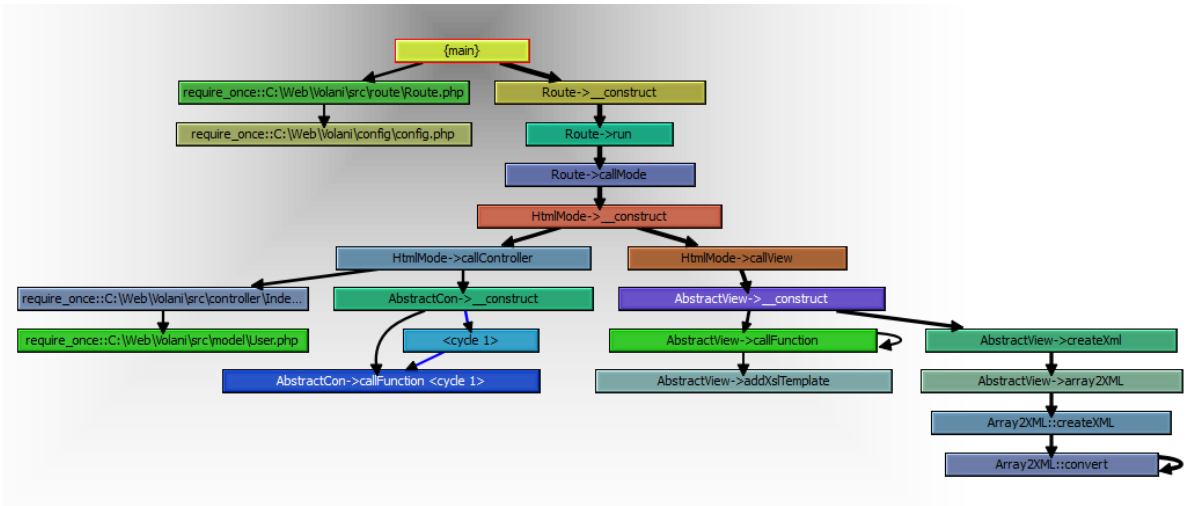
Metoda je metoda řadiče. (např.: v adresní řádce je https://stranka.cz/index/login – zavolá se metoda z řadiče *IndexCon* která obslouží uživatele při přihlášení)

5.1.3 Mód

Pomocí této volby, se volí mód výstupu. V základním frameworku jsou módy HTML, XML a JSON. Vývojář, který framework používá, si může přidat další módy. Implicitní mód je html. (např.: pokud uživatel zadá (…)/admin/users, tak se mu v prohlížeči zobrazí HTML stránka, pokud ale zadá (…)/users.json tak se mu objeví JSON výstup) – toto je zjednodušený popis, dále v dokumentaci je popsáno jak pomocí frameworku nastavovat přístupy k jednotlivým řadičům, metodám a módům.

Zavolání módu

Poslední metoda třídy *Route* , která se volá po po spuštění je *callMode*. Tato metoda předává řízení módu, který zajistí další obsluhu.



Ilustrace 6: Vizualizace průchodu frameworkem

Obrázek Ilustrace 6: Vizualizace průchodu frameworkem znázorňuje průchod frameworkem. Průchod začíná žlutým rámečkem {main}. Každý rámeček symbolizuje volání metody tříd, které jsou popsány níže.

5.2 Mode

Mód je třída, která přebírá kontrolu nad skriptem po třídě *Route*. Tato třída zajišťuje kontrolu přístupu do informačního systému a nastavení formy výstupních dat (HTML, JSON, atd.). Jednotliví potomci a rodičovská třída *AbstractModel* se nacházejí v adresářovém stromu na pozici „src/mode“. Všechny třídy jsou nadstavbou nad Controllerem (řadičem) a View (pohledem). Controllery a View volá každá třída dle potřeby a dle módu (framework má speciální View jak pro HTML tak např. pro JSON).

AbstractMode

Rodičovská třída, kterou dědí všechny třídy typu mode. Tato třída poskytuje svým potomkům kontrolu přístupu a odstranění HTML znaků pro ochranu proti XSS útoku a zabránění nežádoucímu uživatelskému formátování HTML kódu.

HtmlMode

Potomek třídy *AbstractMode*. Zajišťuje obsluhu při zvoleném implicitním HTML módu výstupu.

5.2.1 Access

Nejprve po nastavení atributů se zkontroluje, zda uživatel informačního systému má přístupu k danému modulu. Pro kontrolu je využita metoda rodiče pro kontrolu přístupu, která vyřeší případnou obsluhu neoprávněného přístupu.

5.2.2 CallController

Zde se volá Controller (řadič). Controller je nastaven ze směrovací třídy *Route*. Výstup Controlleru se uloží do atributu instance třídy pro pozdější použití u View (pohledu).

5.2.3 CallView

Zde se volá View (pohled). View se odvodí od nastavení třídy ze směrovací třídy *Route*. Výstup je uložen do atributu pro odeslání uživateli.

5.2.4 SaveSession

Zde se vytvoří instance třídy *Session*, která má na starost ukládání a načítání sezení. Této možnosti lze využít například při přihlášení uživatele do informačního systému a načíst uživatelovo předešlé nastavení před předchozím odhlášení a díky tomu ho přesměrovat na poslední navštívenou stránku. Výhodou volání v módu je, že pro celý systém je uložení sezení globální a programátor se o něj nemusí starat.

JsonMode

Potomek třídy *AbstractMode*. Zajišťuje obsluhu při explicitním módu JSON. Mód se nastavuje přes příponu *.json (např.: (…)/admin/users.json)

5.2.1 Access

Nejprve po nastavení atributů se zkontroluje, zda uživatel informačního systému má přístupu k danému modulu. Pro kontrolu je využita metoda rodiče pro kontrolu přístupu, která vyřeší případnou obsluhu neoprávněného přístupu.

5.2.2 CallController

Zde se volá Controller (řadič). Controller je nastaven ze směrovací třídy *Route*. Výstup Controlleru se uloží do atributu instance třídy pro pozdější použití u View (pohledu).

5.2.3 CallView

Zde se volá View (pohled). View se odvodí od nastavení třídy ze směrovací třídy *Route*. Výstup je uložen do atributu pro odeslání uživateli.

XmlMode

Potomek třídy *AbstractMode*. Zajišťuje obsluhu při explicitním módu XML. Mód se nastavuje přes příponu *.json (např.: (…)/admin/users.xml)

5.2.1 Access

Nejprve po nastavení atributů se zkontroluje, zda uživatel informačního systému má přístupu k danému modulu. Pro kontrolu je využita metoda rodiče pro kontrolu přístupu, která vyřeší případnou obsluhu neoprávněného přístupu.

5.2.2 CallController

Zde se volá Controller (řadič). Controller je nastaven ze směrovací třídy *Route*. Výstup Controlleru se uloží do atributu instance třídy pro pozdější použití u View (pohledu).

5.2.3 CallView

Zde se volá View (pohled). View se odvodí od nastavení třídy ze směrovací třídy *Route*. Výstup je uložen do atributu pro odeslání uživateli.

ApiMode

Potomek třídy *AbstractMode*. Zajišťuje obsluhu při volání API. API se volá např.: (…)/ApiPublic/users.json. Jedná se o odlišný mód od předchozích, protože v sobě kombinuje více módů (JSON, XML). Na tuto třídu je z hlediska bezpečnosti kladen obzvláště vetší důraz.

5.2.1 Konstruktor

Při předání kontroly nad skriptem od třídy *Route* skript pokračuje přes konstruktor. Zde se rozliší 3 možné varianty dalšího postupu:

1. Žádost o přístup k API – pokud se uživate dotazuje na speciálně vyhrazenou funkci pro přístup k API (většinou (…)/ApiPublic/grant), zavolá se funkce *login* a poté se ukončí činnost skriptu.
2. Žádost o přístup k funkci API – zde se kontrolují 2 kritéria pro povolení vstupu do chráněné sekce. První kritérium je, aby byl uživatel přihlášen do systému. Pro kontrolu se využívá metoda *isAllowed*. Druhé kritérium je, aby funkce API byla povolena (kontrolu provádí zděděná metoda *access* od rodiče *AbstactMode*).

5.2.2 Login

Zde se provádí vygenerování přístupového klíče pro uživatele API. Generování klíče a kontrolu přihlašování má na starosti třída *Server*. Třída *Server* využívá pro tyto účely OAuth2.

5.2.3 IsAllowed

Zde se provádí kontrola přístupového klíče uživatele API. Kontrola klíče provádí třída *Server*. Třída *Server* využívá pro tyto účely OAuth2. Implementace OAuth2 je převzatá[1].

1 OAuth2 dostupné na https://github.com/bshaffer/oauth2-server-php pod MIT licencí

5.3 Controller

Řadič je třída, která je volána z módu a vrací výstup do módu. Má za úkol reagovat na uživatele a za pomoci modelu udělat požadované operace. Každá úroveň oprávnění ve frameworku je implementovaná v jedné třídě řadiče. Oprávnění se rozlišují dle názvu třídy. Název třídy vždy končí *Con* (např.: *AdminCon* – třída s metodami pro administrátora). Ve frameworku není programátor omezován, co se počtu oprávnění týče. Omezení je jen v názvu oprávnění, kde musí být třída nazvána velkým písmenem (jako všude jinde v systému) a může obsahovat pouze znaky z abecedy bez diakritiky (a poté k názvu třídy na konec přidat *Con*). Třída úrovně oprávnění typu řadič, je také povinna dědit třídu *AbstractCon*, která přes svůj konstruktor zajistí kompletní obsluhu třídy úrovně oprávnění řadiče.

CallFunction

CallFunction je metoda pro automatické volání funkcí a následném ukládání výstupních dat (pole). Důležitým prvkem této funkce je kontrola náhodně generovaného klíče, který se vygeneruje při příchodu na stránku. Tento klíč se přenáší v hlavičce dotazu a kontroluje se při předávání dat do funkcí. Tento postup slouží k ochraně proti CSRF útoku, kdy skript zabraňuje podvržení cizích příkazů ze strany klienta.

5.4 View

Pohled je třída, která má na starosti prezentaci výstupních dat z řadiče. Je volána pomocí módu. Každý mód má speciální třídu. Výběr je zajištěn dle umístění v adresářové struktuře (view/(mód)/(oprávnění) – např. view/json/user – seznam uživatelů, který se vrací přes JSON).

AbstractView HTML mód

V této abstraktní třídě se generuje HTML kód. Pro vytváření HTML kódu je využívána třída obsažena v PHP – *XSLTProcessor*. Pro zpracování dat z řadiče, je využita převzatá statická třída *Array2XML,*která byla lehce zeditována pro efektivnější využití ve frameworku. Pro vložení do XSLT se data převedou do pomocí třídy *Array2XML* do DOM. Následně se vygeneruje výstupní HTML soubor.

Průchod daty probíhá v pořadí převzatém z řadiče. Využívá se k tomu metoda *callFunction.*

5.4.1 CallFunction

CallFunction je metoda pro automatické volání funkcí modifikující datovou strukturu (pole). V případě že neexistuje jsou použita data z třídy *Controller*. Poté funkce zavolá metody *addXml* a *addXslTemplate*.

5.4.2 AddXml

Tato metoda přidá do výstupního pole dle aktuální funkce pole s daty aktuální funkce.

5.4.3 AddXslTemplate

Tato metoda přídá do výstupní proměnné XSLT template, dle aktuální volané funkce (obsah šablony se načte ze souboru vytvořeného kodérem a samotná šablona – název apod. se dodá automaticky PHP skriptem, šablony pak rušně volá kodér). XSLT šablona je vyžadována, pokud neexistuje, je vyvolána výjimka. Do šablony je pro přidána proměnná odkazující na data příslušící dané šabloně (např: <xsl:variable name="data" select="userTableAction"/>). Díky této proměnné se kodér XSLT nemusí starat o data a používá proměnnou *data.*

6 *Demonstrační aplikace*

Framework byl nasazen do ukázkového informačního systému. Informační systém má jednoduché zadání, které si klade za cíl ukázat většinu funkcí a posloužit jako příklad pro programátory, kteří by tento framework chtěli používat.

Při vypracovávání ukázkového informačního systému jsem spolupracoval s kolegou kodérem, který měl na starosti HTML a stylování vzhledu. Kolega pro tvorbu HTML využíval XSLT a dále pak informační systém nastyloval pomocí CSS a implementoval část JavaScriptového kódu. Při tomto přístupu se v praxi vyzkoušela kooperace vývoje mezi kodérem a programátorem.

V demostrační aplikaci informačního systému jsou v PHP použity převzaté knihovny nacházející se ve složce src/extern. Dále jsou pak použité JavaScriptové knihovny nacházející se ve složce public/extern. Jejich dostupnost a licence jsou uvedeny ve zdrojových kódech popřípadě v souborech LICENSE.md nebo README.md.

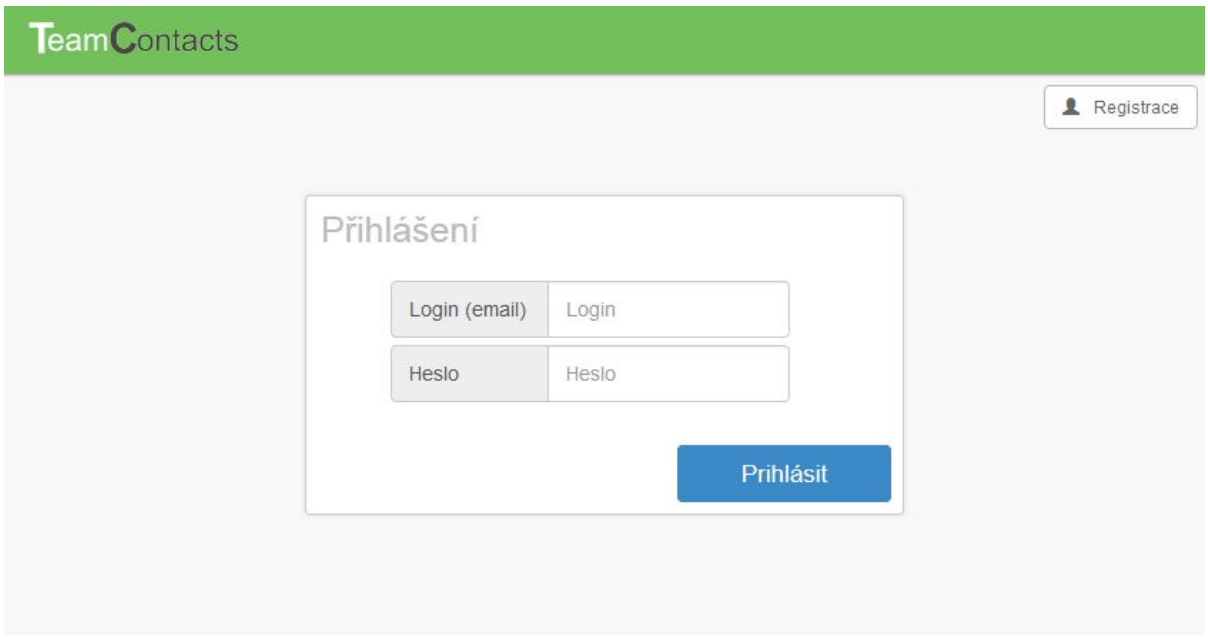
6.1 Zadání

Vytvořte informační systém pro správu kontaktů. Kontakty budou rozděleny do uživatelem definovaných skupin. Skupiny jsou na sobě nezávislé a každá skupina má seznam uživatelů informačního systému, kteří mají do skupiny přístup a mohou ji používat. Systém bude mít cron pro stahování e-mailů z uživatelem nadefinované adresy. Dále bude mít možnost přístup přes API, které se bude moci využívat pro přístup aplikací třetích stran.

6.2 Návrh

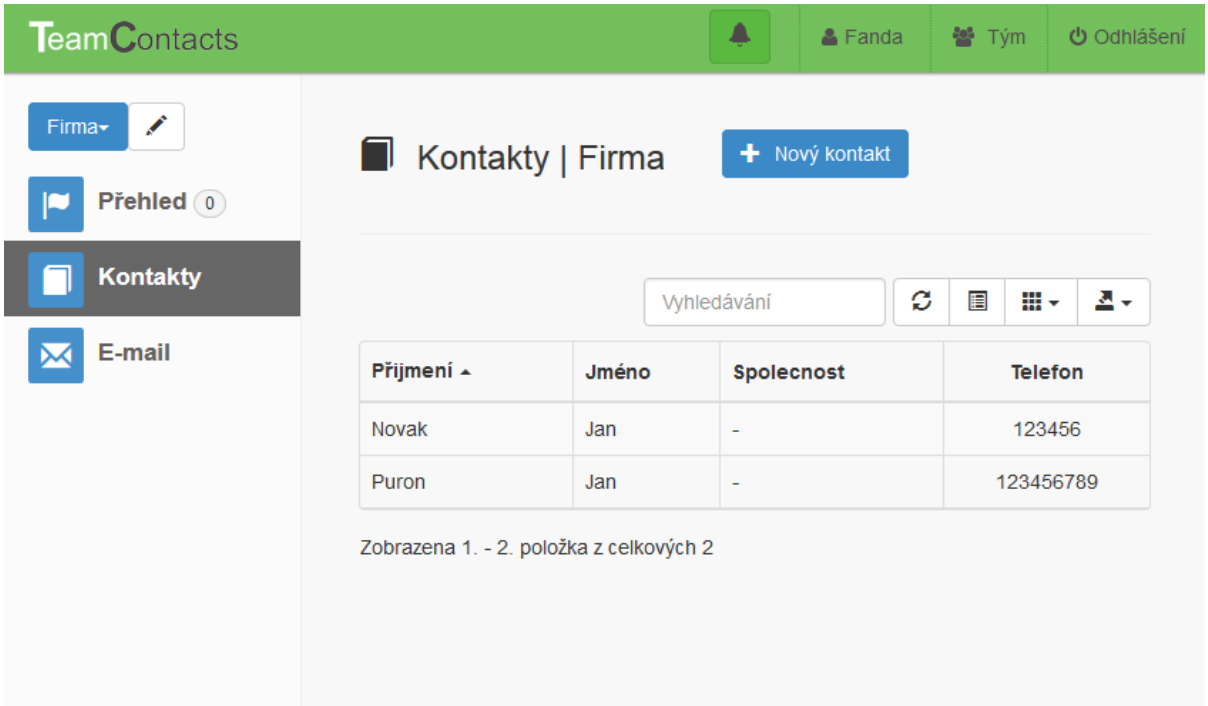
Informační systém je z hlediska frameworku rozčleněn do 4 controllerů. Každý má na starosti jinou část obsluhy informačního systému.

IndexCon – je pro obsluhu nepřihlášeného uživatele, který přišel přes webový prohlížeč. V této části se lze přihlásit nebo registrovat do informačního systému. Zde by byla také hlavní stránka projektu, která by o něm informovala a propagovala ho.



Ilustrace 7: Úvodní stránka systému nepřihlášeného uživatele (http://system.cz/)

UserCon – je pro obsluhu přihlášeného uživatele, který přišel přes webový prohlížeč. Tento řadič obsluhuje uživatele po přihlášení do systému. Je tu použit HTML mód stejně jako u indexu. Pro akce typu vložení kontaktu a podobně, je zde přes JSON mód připravené rozhraní pro používání přes AJAX.



Ilustrace 8: Stránka kontaktů přihlášeného uživatele (http://system.cz/user/index)

CronCon – slouží pro spouštění cronu. Při zavolání stránky http://system.cz/cron/getEmail.json se automaticky stáhnou zprávy ze všech e-mailových schránek uživatelů. Pro použití v praxi by bylo vhodné omezit spouštění crona pouze na IP adresu serveru.

ApiMobile – je připravené API pro použití aplikacemi třetích stran. Zde je využito ověřování přes OAuth2, který je ve frameworku pro tyto účely používán. V dotazu na getSyn je proměnná last_update, která udává poslední čas (unix time), kdy byl požadavek poslán. Tím se posílají pouze změny kontaktů, které od té doby nastali.

```
$ curl http://faktury.cz/apiMobile/grant.json -d 'grant_type=client_credentials
&client_id=fanda@mazura.cz&client_secret=abcd'
{"access_token":"066aed353e52e56b98aac302b749b516e6a6bbb8","expires_in":3600,"to
ken_type":"Bearer","scope":null}
```

Ilustrace 9: Ukázka žádosti o přístupoví token (http://system.cz/apiMobile/grant.json)

Ilustrace 10: Výpis části odpovědi z API rozhraní (http://system.cz/apiMobile/getSyn.json)

7 Testování a závěr

Framework byl testován na testovacím informačním systému, na kterém se vymýšlel koncept a poté se testoval. Po tomto základním testování jsem framework začal používat ve vývoji reálných informačních systémů, které se budou používat v praxi. Díky tomuto používání jsem mohl doladit framework, aby byl více zautomatizován a jednoduchý na použití vývojáři.

Framework splňuje požadavky, které jsou vypsány v kapitole Cíle práce a splnil i mé požadavky, které jsem si dal, aby splňoval a mohl jsem ho začít využívat při vývoji v praxi. Do frameworku jsem se snažil zahrnout mé zkušenosti z předchozího vývoje, při němž jsem vyvíjel informační systémy a webové aplikace jak v čistém PHP, tak i v PHP s frameworkem. Framework je připraven k použití v praxi. Pokud bych framework vyvíjel znova, tak z pozdější zkušenosti používání frameworku bych lépe navrhl předávání parametrů k metodám tříd v části *Controller*. Předávání bych neřešil polem, ale metodou naimplementovanou v abstraktní třídě, která by lépe ošetřovala problémy, které můžou nastat (např.: uživatel nepošle proměnnou, která je po něm vyžadována). Jako další možnosti vývoje, vidím v dynamickém access control list. Po tomto vylepšení by šlo dynamicky přes databázi nastavit uživatelům práva přístupů k jednotlivým modulům. Vývoj tohoto frameworku mi velice pomohl zlepšit své zkušenosti ve vývoji informačních systémů a objektově orientovaného návrhu.

Literatura

[1] XSLT v příkladech. *VŠE O WWW* [online]. 2014 [cit. 2015-05-04]. Dostupné z: http://www.kosek.cz/xml/xslt/

[2] W3C. *XSL Transformations (XSLT)* [online]. 1999 [cit. 2015-05-05]. Dostupné z: http://www.w3.org/TR/xslt

[3] W3SCHOOLS. *XSLT Tutorial* [online]. [cit. 2015-05-05]. Dostupné z: http://www.w3schools.com/xsl/

[4] PHP GROUP. *PHP Manual* [online]. © 2001-2015 [cit. 2015-05-05]. Dostupné z: https://php.net/manual/en/

[5] *OAuth 2.0* [online]. [cit. 2015-05-05]. Dostupné z: http://oauth.net/2/

[6] 2012. The OAuth 2.0 Authorization Framework. *IETF* [online]. [cit. 2015-05-05]. Dostupné z: https://tools.ietf.org/html/rfc6749

[7] 2012. The OAuth 2.0 Authorization Framework: Bearer Token Usage. *IETF* [online]. [cit. 2015-05-05]. Dostupné z: https://tools.ietf.org/html/rfc6750

[8] IBM. 2015. *RESTful Web services* [online]. [cit. 2015-05-06]. Dostupné z: https://www.ibm.com/developerworks/webservices/library/ws-restful/

Seznam příloh

Příloha 1. CD