UNIVERZITA PARDUBICE

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

BAKALÁŘSKÁ PRÁCE

2015 Jakub Stach

Univerzita Pardubice

Fakulta informačních technologií

Autentizační knihovna

Jakub Stach

Bakalářská práce

2015

Zadání BP….

PROHLÁŠENÍ AUTORA

Tuto práci jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

V Pardubicích 7. 3. 2015

Jakub Stach

PODĚKOVÁNÍ

Na úvod bych chtěl poděkovat vedoucímu této práce, Ing. Lukáši Čeganovi, Ph.D. za věcné připomínky a důležité rady při zpracovávání této práce. Dále bych chtěl poděkovat mým přátelům, rodině a spolužákům za pomoc s návrhem této knihovny.

ANOTACE

Tato práce se zabývá autentizačními možnostmi na webových stránkách a jejich rozborem. Postupně budete seznámeni s jednotlivými metodami, ať už zastaralými, tak aktuálními. Budete mít také možnost nahlédnout pod pokličku při návrhu nové autentizační knihovny, zastřešující 3 OAuth rozhraní u nejrozšířenějších sociálních sítí – Twitter, Facebook, Google Plus. V samotném závěru je krok po kroku představena samotná implementace nově vzniklé knihovny do reálného provozu.

KLÍČOVÁ SLOVA

Autentizace, OpenID, OAuth, MultiAuth, Twitter, Google, Facebook

TITLE

Authentication library

ANNOTATION

This work deals with the authentication options on the website and their analysis. Gradually you will be familiar with the different methods, whether obsolete and current. You will also have the opportunity to take a peek at the design of new authentication library, umbrella 3 OAuth interface with the most widely used social networking sites - Twitter, Facebook, Google Plus. In conclusion, it is step by step introduced the actual implementation of the newly established library into real business.

KEYWORDS

Autentization, OpenID, OAuth, MultiAuth, Twitter, Google, Facebook

Obsah

SEZNAM ILUSTRACÍ A TABULEK

SEZNAM ZKRATEK A ZNAČEK

API	Application Programming Interface
CSS	Cascading Style Sheets
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
JS	JavaScript language
OAuth	Open Authorization
PHP	Hypertext Preprocesor
SAML	Security Assertion Markup Language
SSO	Single Sign-On
URL	Uniform Resorce Locator
WWW	World Wide Web
XML	Extensible Markup Lnaguage

Úvod

V této bakalářské práci se budu zabývat tvorbou autorizační knihovny pro obecně rozšířený protokol OAuth, který v současné době využívají především sociální sítě.

První část mé práce bude pojednávat obecně o jazycích HTML, PHP, CSS a JavaScriptu. Zde popíši, kde se s těmito jazyky můžeme setkat a k čemu slouží.

V druhé části se budu zabývat samotným rozborem stávajících možností autorizace, jejich klady a zápory, včetně samotného popisu protokolu OAuth a několika dalších.

Výstupem této bakalářské práce bude autentizační knihovna pro volné použití. Knihovna bude navržena s vysokou mírou abstrakce pro široké uplatnění.

1 Webová stránka

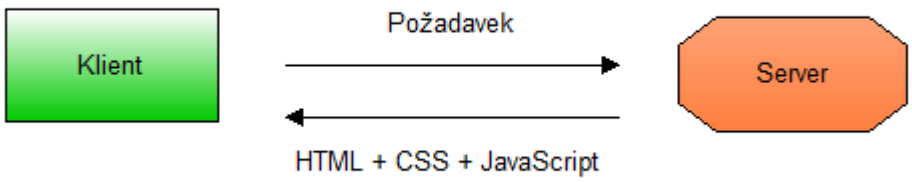
Webová stránka je dokument, stvořený za účelem sdílení informací v síti World Wide Web. Abychom mohli vytvořit webovou stránku, musíme znát a správně použít HTML a XHTML značky.

1.1 Statický web

Statický web je tvořen z webový stránek, jejichž obsah je statický, tedy neměnný. Obsah zůstane konstantní, dokud jej jeho autor či jiná pověřená osoba přímým zásahem do kódu stránky nezmění. Cílovému uživateli je tedy zobrazena přesně tak, jak byla uložena na server jejím autorem. Stránka tohoto typu zobrazuje svůj obsah každému uživateli stejně.

Nesporná výhoda statického webu je rychlost, s jakou můžeme jednoduchý web vytvořit. Vyzdvihnout můžeme také jednoduchost použití a zápisu HTML značek.

Statický web je však nevhodný pro použití na místech, kde často potřebujeme upravovat obsah webu. V tu chvíli se práce stává velmi neefektivní.



Obrázek 1 - Příklad statického webu

1.2 Dynamický web

Jak již název napovídá, dynamický web umožňuje svým provedením dynamicky generovat obsah i strukturu webu. Na rozdíl od statického webu se obsah webu dynamického generuje až ve chvíli, kdy je vydán požadavek na jeho zobrazení.

V dnešní době je většina webových stránek dynamická. Vyplývá to i z potřeby často uveřejňovat nové informace a možnosti zobrazovat personalizovaný obsah (každému uživateli jiný).

Při návrhu dynamického webu si však nevystačíme pouze s HTML, je zde potřeba zapojit libovolný skriptovací jazyk.

1.2.1 Příklad dynamického webu

Na obrázků níže vidíme proces sestavení dynamického webu.

1. Klient vytvoří požadavek na zobrazení stránky.
2. Aplikačnímu serveru je doručen požadavek a ten jej nechá postoupit interpretu PHP.
3. Interpret PHP vykonává program, který vyžaduje data z databáze, vytvoří požadavek.
4. Požadavek na data je doručen databázovému serveru, ten data zpracuje/vyhledá a zašle zpět.
5. Získaná data jsou předána programu, který je zpracuje do výsledné formy.
6. Vygenerovaná stránka či soubor je doručen klientovi.

Obrázek 2 - Příklad generování dynamického webu

2 Technologie

Tvorba webových stránek nespočívá v ovládnutí jedné technologie či programovacího jazyka. Je to promyšlená kombinace několika nástrojů z čehož každý z nich má svou pevně stanovenou roli.

2.1 HTML

Jedná se o značkovací jazyk pro tvorbu webových stránek. Přičemž jednotlivé webové stránky jsou propojeny hypertextovými odkazy. HTML je tvořeno sadou (zpravidla) párových značek, mezi které zapisujeme požadovaný obsah, přičemž jednotlivé značky (tagy) mají svůj význam a sémanticky oddělují bloky obsahu.

2.2 CSS

Kaskádové styly by se daly definovat jako soubor metod a předpisů pro grafickou úpravu webových stránek. Díky CSS jsme schopni hromadně upravovat grafický vzhled stránek, eliminuje se tím dříve používaný zápis vlastností přímo k jednotlivým HTML značkám na stránce. Název „kaskádové“ byl zvolen na základě vrstvení grafických pravidel za sebe.

Mezi výhody CSS patří nejen efektivita, se kterou můžeme jednotlivá pravidla zapisovat a využívat, ale i výsledné snížení přenášeného objemu dat.

V dnešní době je k dispozici CSS verze 3. Před využitím plného rozsahu možností této verze musíme zohlednit fakt, že starší prohlížeče nejsou schopny nová pravidla správně interpretovat. Jako jeden z nejproblémovějších prohlížečů se jeví Internet Explorer 8 a nižší.

2.3 JavaScript

JavaScript je programový jazyk sloužící k tvorbě webových stránek. Nejedná se o serverový programovací jazyk, jelikož běží na straně klienta – typicky v internetovém prohlížeči. Jeho zdrojový kód může být součástí samotného HTML souboru, nebo jej programátor může vložit jako samostatný podpůrný soubor, zpravidla s koncovkou *.js.

Nejčastěji je u webových stránek použit JavaScript k „rozhýbání“ webu. Mění tedy webové stránky na interaktivní.

Okolo roku 2000 byl JavaScript velmi populární především u amatérských tvůrců webu. Dovoloval totiž vložit na samotný web různé animované prvky. Umožňoval (a stále

umožňuje) vyměnit uživateli kurzor za vlastní nebo nechat na uživatele vyskakovat tzv. Pop-up okna. Díky snadnému použití samotného JavaScriptu se tyto animované prvky staly nedílnou součástí samotného webu a to až do takové míry, že uživatele spíše obtěžovaly, než aby mu pomohly či ho potěšily.

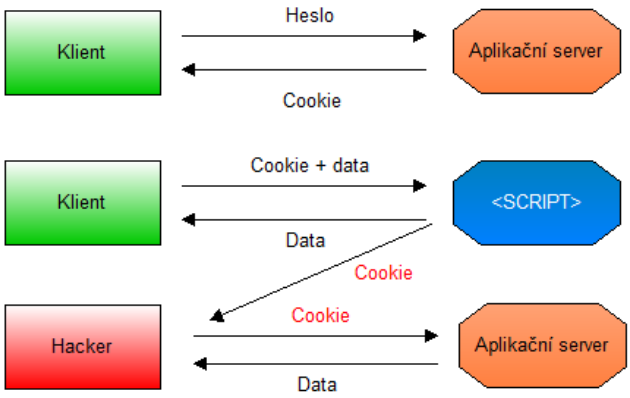
Od této doby programátoři ušli již dlouhou cestu a k používání JavaScriptu se přistupuje značně obezřetněji. Nelze říci, že by JavaScript z webů vymizel, spíše naopak. Rozšířil se ještě více. Ovšem často tam, kde by ho uživatel vůbec nečekal. Stal se z něj výkonný rádce např. při práci s formuláři, zábavný společník pokud jde o animace a nenahraditelný prvek pro interakci na webu. Při použití JavaScriptu je třeba se držet více než jindy pravidla, že všeho moc škodí.

Typickým zástupcem pro využívání JavaScriptu je internetový gigant Google, který nemá snad jedinou webovou aplikaci, kde by hojně nevyužíval možností JavaScriptu.

2.3.1 Cross-Site Scripting

Není tajemstvím, že i JavaScript má své stinné stránky. Lze ho účinně využít k narušení vzhledu webových stránek, funkčnosti nebo dokonce získání citlivých údajů návštěvníků nezabezpečeného webu.

Tato metoda narušení WWW se nazývá Cross-Site Scripting, nebo také XSS. Jedná se o podstrčení a vykonání JavaScriptového kódu do nezabezpečené stránky. Přičemž obrana není příliš složitá, stačí pouze správně ošetřit každou proměnnou, kterou vypisujeme jako část zdrojového kódu stránky. V PHP lze využít funkce `htmlspecialchars()`.



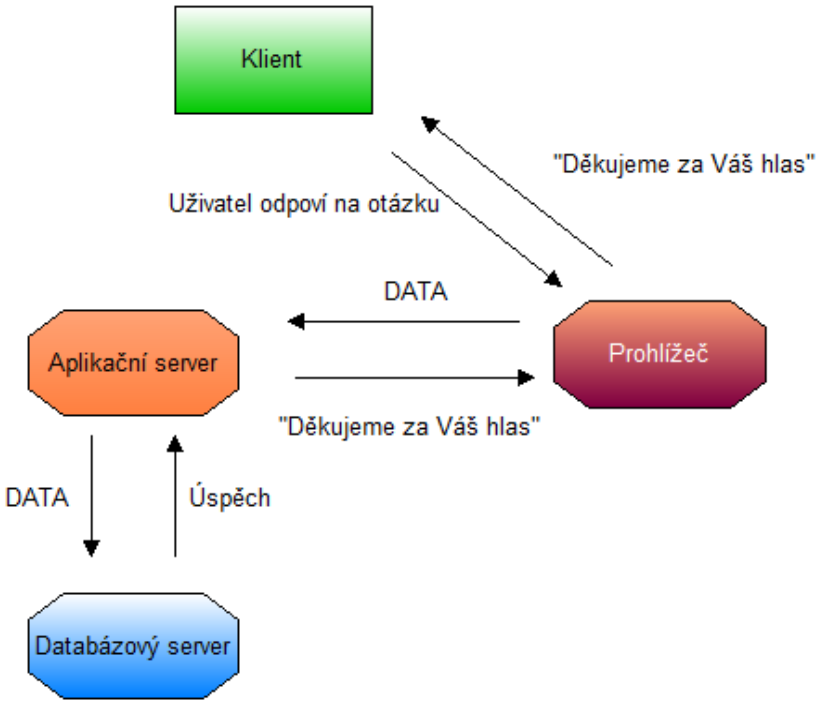
Obrázek 3 - Příklad XSS útoku

2.3.2 AJAX

Často se stává, že na straně klienta potřebuje programátor změnit pouhou drobnost jako výsledek klientské akce. Jako vhodný příklad může posloužit anketa. Chceme docílit, aby poté, co uživatel odpoví na anketní otázku, se místo ankety zobrazilo děkovné hlášení za účast při hlasování. To by samo o sobě nebylo nic zvláštního, ovšem my chceme, aby se vše stalo bez nutnosti znovunačtení celé webové stránky.

K tomu slouží právě AJAX. Jedná se o moderní technologii, která umožní klientské aplikaci komunikovat „v pozadí“, neboli asynchronně. AJAX je kombinací JavaScriptu a XML, nejedná se tedy o novou technologii, nýbrž o nové využití již starých technologií.

AJAX umí odeslat a získat data ze serveru bez nutnosti obnovení stránky, má tak velmi široký rozsah možných použití. V našem modelovém příkladu s anketou by vše mohlo vypadat zhruba následovně:



Obrázek 4 - Příklad využití technologie AJAX

1. Uživatel zvolí odpověď.
2. JavaScript v prohlížeči tuto odpověď zaznamená a vytvoří XMLHttpRequest, ten pošle na server.
3. Server zpracuje odpověď uživatele a výsledek pošle na databázový server.
4. Databázový server data uloží a kladně odpoví.
5. Server pošle prohlížeči odpověď.
6. Prohlížeč odpověď zpracuje např. v podobě vyskakovacího okna s děkovnou zprávou.

2.3.3 jQuery

Potřebujete-li ve svém projektu využívat JavaScript a příliš mu nerozumíte, nabízí se možnost využít tzv. framework[1].

Framework obecně je nadstavba nad klasickým programovacím jazykem, řešící typické problémy dané oblasti. V praxi to znamená značné ulehčení práce, protože většina rutinních kroků je již ve frameworku obsažena. Programátor se tak může v klidu soustředit na důležitější problematiku.

JavaScriptový framework jQuery nám v první řadě velmi usnadňuje zápis jednotlivých příkazů a zároveň díky své jednoduchosti celou práci urychluje.

Ukázka zápisu změny pozadí stránky na červenou:

```
// JavaScript
function changeBackground(color) {
   document.body.style.background = color;
}
onload="changeBackground('red');"

// jQuery
$('body').css('background', '#ccc');
```

Jediná podmínka využití frameworku je zahrnout do našeho projektu zdrojový kód jQuery[2].

[1] http://cs.wikipedia.org/wiki/Framework
[2] https://jquery.com/

2.4 PHP

Pozici nejoblíbenějšího, nebo přinejmenším nejrozšířenějšího skriptovacího jazyka pro webové stránky, zastává PHP. Je to poměrně jednoduchý programovací jazyk určený pro tvorbu dynamických webových aplikací.

PHP není omezeno operačním systémem, ale pro svůj běh potřebuje vlastní webový server. Tento webový server musím mít také nainstalovaný interpret PHP. Zdrojový kód je na serveru uložen v surové formě, zpracovává se až při přijetí požadavku. Interpret PHP vykoná instrukce na serveru a na výstup se pošle až jeho výsledek.

Samotný jazyk lze rozšířit o velké množství knihoven, umožňující podporu např. práci s grafikou, elektronickou poštou či soubory.

Z pohledu programátora má PHP několik zásadních nevýhod. Jednou z nich je velmi slabá typová kontrola, umožňující provést téměř libovolný úkon. Spolu s absencí ladícího nástroje vytvářejí nebezpečnou propast, v níž není těžké ztratit dlouhé hodiny hledáním chyb.

Pro upozornění programátora na případnou chybu nebo nedostatek v kódu, byl vytvořen systém upozornění. Každá nově vzniklá chyba má vlastní úroveň závažnosti. Přiřazení dat nedefinované proměnné vyvolá chybu úrovně `E_NOTICE`, tedy jakási poznámka. Oproti tomu chybějící soubor při vkládání pomocí funkce require_once vyvolá `E_COMPILE_ERROR`, tedy kritickou chybu. V PHP si můžeme nastavit, že chceme být upozorněni pouze na určité typy chyb, zpravidla ty nejzávažnější. Toho docílíme správným nastavením hlášení chyb pomocí funkce `error_reporting()`.

Hlášení chyb „veřejně“, tedy jejich zobrazování přímo v kódu, je vhodné pouze v testovacím prostředí. Výskyt a především zobrazení chyby v produkčním prostředí je nejen nevhodný, ale také nebezpečný, jelikož může v rámci výpisu chyby vyzradit např. přihlašovací údaje k databázi. Abychom se vyhnuli přímému výpisu chyby, nabízí PHP nastavení chybových logů, kam se vzniklé chyby zapíší, namísto přímého výpisu do stránky. Opět je vhodné omezit rozsah zapisovaných chyb, abychom byli informováni pouze o nejzásadnějších chybách. Nutno však podotknout, že správně napsaná aplikace by neměla vyvolávat žádné chyby.

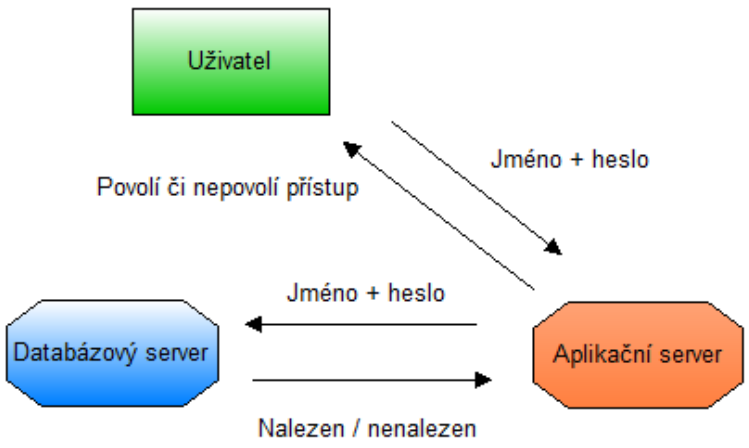
3 Autentizační systémy

Autentizace, neboli ověření identity, je v internetovém prostředí přítomna téměř na každé webové stránce. Musíte prokázat svou identitu, abyste měli přístup např. ke správě webových stránek, k profilu na sociální síti nebo třeba k osobnímu hodnocení na univerzitě.

Každý průměrný uživatel na internetu, dokonce i uživatel-začátečník je dříve nebo později vyzván libovolnou službou k vytvoření uživatelského účtu. Časem však tento uživatel naráží na nové služby a je nucený si tvořit další a další uživatelské účty. Jelikož dbá na své soukromí, dodržuje pravidla pro tvorbu hesel. Platí tedy, že žádné z hesel nepoužívá na více službách, každé heslo má alespoň 9 znaků a je kombinací alfanumerických znaků, různých velikostí písmen a symbolů. Nyní nastává první problém. Jak si má všechny tyto přihlašovací údaje zapamatovat? S rostoucím počtem uživatelských účtů začíná kolotoč přihlašování a hledání těch správných přihlašovacích údajů, který nikdy nekončí. Pomiňme tedy, že každá služba nutí koncového uživatele pamatovat si jedinečné uživatelské jméno a heslo.

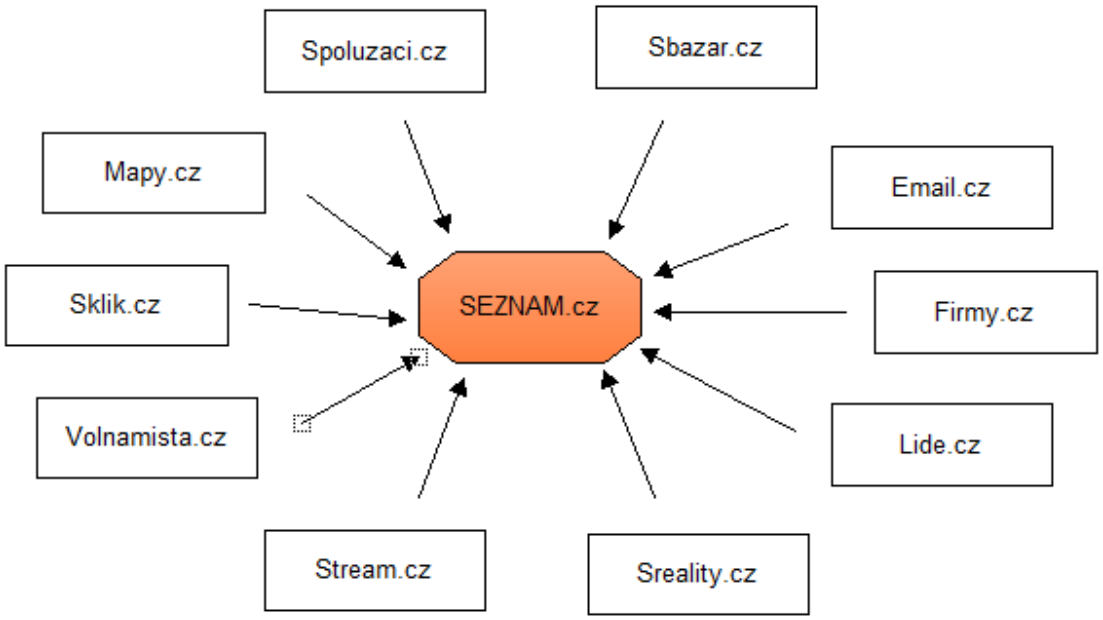
Co bezpečnost? Z pozice uživatele služby lze jen těžko odhadnout, komu svěřujeme své drahocenné přihlašovací údaje. Může to být výborně zabezpečený web, nebo také práce programátora-začátečníka.

Běžná webová stránka s přihlašováním má řešenou autentizaci formou vlastních ověřovacích skriptů a hlavně jedné vlastní databáze. Pokud se chce uživatel přihlásit ke službě X, zadá přihlašovací údaje, služba se podívá do databázového serveru X, jestli účet existuje, případně, jestli jsou všechny údaje správné a na základě toho uživatele vpustí či nikoliv.



Obrázek 5 - Nejčastější způsob přihlašování

Pokročilé weby a především celé webové ekosystémy jsou o krok dále. Byť poskytují několik rozdílných služeb, uživatel má pouze jeden uživatelský účet, skrze který se může dostat na libovolnou službu v daném ekosystému. Vše je tedy řešeno formou jedné centrální databáze uživatelů pro celý jeden ekosystém. Uživateli tedy odpadne nutnost pamatovat si několik různých přihlašovacích údajů.



Obrázek 6 - Ukázka společné databáze uživatelů mezi více službami v rámci ekosystému

Běžného uživatele tento zaběhnutý systém, krom záplavy hesel a přihlašovacích jmen, nijak neomezuje. Problém nastane ve chvíli, kdy se rozhodneme „přenést“ svou identitu na jiný server či službu. Uživatelské jméno, pod kterým vystupujeme, je unikátní pouze v rámci jedné služby, případně v rámci jednoho ekosystému. Co ale v případě, že jsme uznávaným specialistou na jednom webu a chceme, aby nás stejně vnímali i na webu jiném? Stačí si přeci zaregistrovat účet se stejnou přezdívkou či jménem. Tento postup funguje však jen pokud si naše uživatelské jméno již někdo dříve nezaregistroval. V tu chvíli může daný uživatel těžit z našeho dobrého jména či nám naopak škodit, a my mu v tom nemáme jak zamezit.

3.1 Nezávislá autentizační autorita

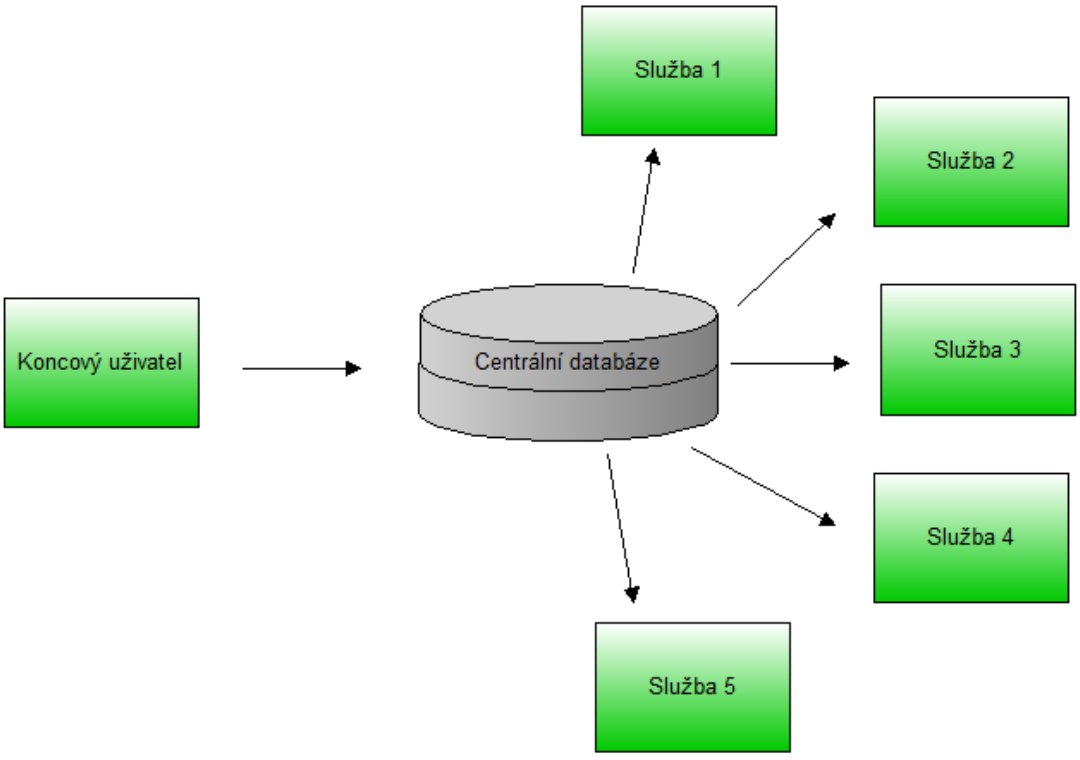
Problémy s odděleným přihlašováním pro každou službu byly již natolik zřejmé, že vznikla myšlenka jedné nezávislé certifikační autority, spravující uživatelské identity.

Vznikla by tedy jedna centrální databáze uživatelských účtů spravovaná třetí stranou, sloužící k přihlašování na libovolný počet služeb.

Výhody plynoucí pro koncového uživatele jsou zřetelné – odpadá nutnost pamatovat si desítky přihlašovacích údajů, všechny tyto údaje by nahradilo přihlašování jediné. Uživatel by se tedy přihlásil do aplikace třetí strany, kde by prokázal svou identitu a s touto identitou by měl otevřený přístup do vybraných služeb. Tento způsob by také vyřešil problém s přenosem identity, jelikož každý uživatel by vystupoval pouze pod jednou identitou u libovolného počtu služeb. Třetí strana by se tedy stala jakýmsi správcem identit.

Z pohledu poskytovatele služeb je toto řešení také pohodlné. Není třeba obstarávat pracné registrování uživatelů, ani spravovat jejich databázi.

I když se toto řešení zdá být téměř ideálním, je zde bezpečnostní problém. V případě prolomení ochrany u třetí strany, našeho správce identit, by útočník rázem dostal přístup k desítkám našich služeb. Také poskytovatelé služeb mohou mít výhrady, přihlašování skrze třetí stranu je může ochudit o přístup k uživatelským datům, která musejí poté získávat dodatečně.



Obrázek 7 - Princip nezávislé autentizační autority

V dnešní době jsou různé druhy autentizačních metod, přičemž jedny řeší jen a pouze autentizaci, tedy prokázání vlastní identity, a druhé jsou připraveny řešit také autorizaci, tedy řízení přístupových práv.

Typickými zástupci první skupiny jsou openID, OAuth, LiveID, a dříve také Facebook Connect nebo BBAuth.

Jak bylo již zmíněno, druhý typ uživatele nejen autentizuje, ale zároveň i autorizuje. Kontroluje tedy jeho oprávnění uživatele, jeho roli a to na úrovni instituce, organizace či služby. Dobrým příkladem je projekt Shibboleth, využívaný např. vysokými školami a státními institucemi.

3.2 OpenID

Jedna z možností, jak řešit autentizaci uživatelů je standard OpenID. Za jeho vývojem stojí společnost Six Apart. Standard OpenID vznikl, když vyvíjela blogovací systém LiveJournal. O správu tohoto standardu se dnes stará společnost JanRain.

Výhodou a zároveň nevýhodou této autentizační metody je její decentralizace. Identity v systému OpenID nespravuje totiž pouze jedna jediná centrální autorita, ale hned několik. Autoritou totiž může být kdokoliv s vlastním serverem. Výhody jsou zřejmé – lze si vybrat důvěryhodnou autoritu dle našich požadavků a té svěřit naši identitu. Nedůvěřivý uživatel si dokonce může založit vlastní OpenID autoritu na vlastním serveru a k ní se připojit.

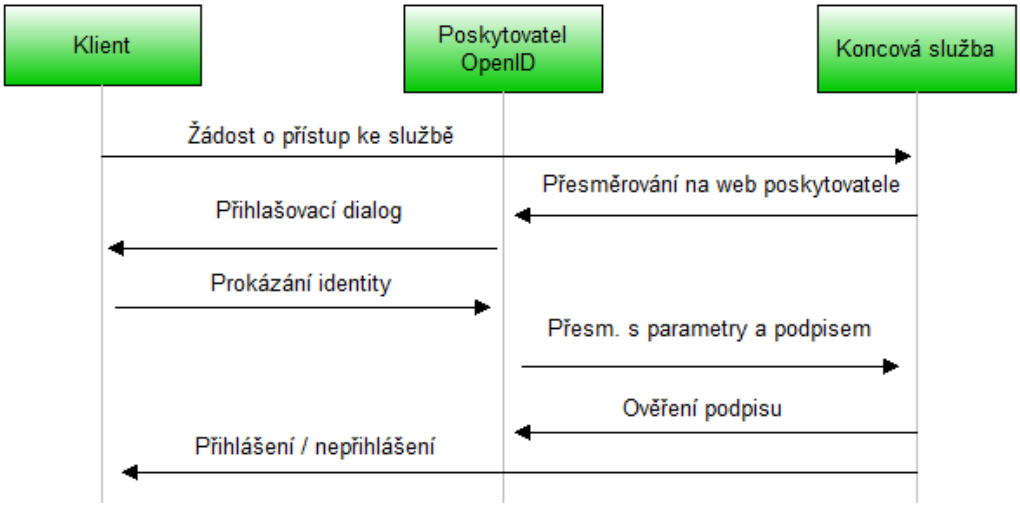
Každému uživateli je přidělen jeden identifikátor ve tvaru běžné URL. Tento identifikátor tedy slouží jako pomyslné uživatelské jméno. OpenID dovoluje uživateli určit přesnou množinu údajů, které chce sdílet s tou kterou službou. V případě potřeby je také možno omezit poskytování sdílených informací pouze na konkrétní časové období.

Zajímavostí samotného standardu je, že nepředepisuje koncové službě, jakým způsobem má být samotné ověření realizováno. Můžeme tedy využít klasické uživatelské jméno a heslo, SMS zprávy, elektronické klíče či certifikáty, to vždy záleží na koncové službě.

Jak již bylo řečeno, mluvíme o decentralizované metodě ověřování. Jak tedy koncová služba zjistí, ke kterému poskytovateli se má připojit, aby si ověřil naši identitu? Stačí otevřít náš identifikátor jako běžnou URL adresu. Jakmile se tak stane, zdrojový kód webové stránky na této adrese obsahuje adresu serverů, kam má služba mířit veškeré své požadavky na ověření. Procesu zjišťování adresy serverů se říká proces Discovery.

Průběh ověření pomocí OpenID:

1. Uživatel sdělí prostřednictvím prohlížeče svůj jedinečný identifikátor.
2. Služba, implementující OpenID (Klient), musí identifikátor normalizovat. Po normalizaci se za pomoci procesu Discovery snaží dosáhnout speciální adresy vytvořené Poskytovatelem.
3. Pro zvýšení bezpečnosti lze veškeré transakce mezi Klientem a Poskytovatelem podepisovat pomocí tzv. Association, kdy si obě strany vymění společný klíč. Ten zabraňuje podvrhnutí údajů během přenosu.
4. Klient přesměruje Uživatele na stránky Poskytovatele s autent. požadavkem v URL.
5. Poskytovatel zkontroluje, jestli je Klient oprávněn prokazovat se daným identifikátorem. Po ověření je Uživatel přesměrován zpět na stránky Klienta s informací o úspěchu či neúspěchu ověření.
6. Klient si z důvodu bezpečnosti ověří informace od Poskytovatele a také podpis transakce. Podpis je možno ověřit buď pomocí vygenerovaného společného klíče, nebo opětovným dotazem na Poskytovatele.



Obrázek 8 - Princip ověření standardu pomocí OpenID

3.2.1 MojeID

Jednou z autorit implementujících OpenID v ČR je MojeID, provozované sdružením CZ.NIC, z.s.p.o. MojeID je nadstavbou samotného standardu OpenID a rozšiřuje jej o vlastní portfolio funkcí.

Přihlásit se skrze MojeID je možné na všech místech, kde je podporováno také OpenID. Služba je poskytována pro koncového uživatele zdarma.

3.2.2 Live ID

Dle názvu bychom řekli, že to je opět varianta OpenID, ale není tomu tak. I když se principiálně v mnoha ohledech podobají, tak nejde o jeden standard. Live ID je služba, kterou poskytuje softwarový gigant Microsoft jako přihlašovací médium ke svým službám. Je to tedy typ přihlášení pro jeden ekosystém.

Live ID identifikátor je na rozdíl od OpenID ve tvaru emailové adresy (OpenID ve tvaru URL). Výhodou je, že k registraci lze využít libovolnou emailovou adresu, není tedy nutné zakládat a pamatovat si další údaj. Live ID nespravuje uživatelská data decentralizovaně, jak tomu je u OpenID. Všechna uživatelská data spravuje jen a pouze spol. Microsoft.

Jaký je tedy důvod k registraci? Pokud nevyužíváte služby jako je Messenger, MSN, OneDrive, Hotmail apod., tak prakticky žádný. V opačném případě budete Live ID bezpodmínečně potřebovat, slouží totiž k přístupu ke všem zmíněným službám.

V roce 2008 Microsoft oznámil, že se stane také zároveň poskytovatelem OpenID, to se však do dnešní doby nestalo.

3.3 OAuth

Konkurentem standardu OpenID je OAuth. Zajímavostí je, že oba protokoly mají jedny autory. Jedná se o autentizační protokol umožňující poskytovat přístup cizím serverům nejen k osobním údajům uživatele, ale i dalšímu jeho chráněnému obsahu jako jsou fotky, videa, příspěvky na osobní zdi (Facebook) apod. Na rozdíl od Open ID není OAuth decentralizovaný a poskytuje přístup k API jeho Autority.

Abychom se neztráceli v pojmech, definujeme si pár pojmů:

- Autorita – služba poskytující své API a údaje uživatele pomocí OAuth.
- Uživatel – reálná osoba vlastnící účet u Autority.
- Služba – koncová služba, která chce využívat chráněné zdroje uživatele u Autority
- Chráněný zdroj – veškerý obsah uživatele umístěný na serverech Autority. Tedy text, video, data, obrázky.

Stroze by šlo říci, že Uživatel může u své Autority autorizovat libovolnou Službu implementující OAuth, aby mohla využívat všech (nebo vybraných) Chráněných zdrojů skrze API autority.

Pokud jsme v pozici Služby, která chce implementovat OAuth a máme zájem o přístup k Chráněným zdrojům Uživatele, musíme si zpravidla u Autority založit tzv. aplikaci, k níž obdržíme také jednoznačné ID a bezpečnostní klíč. Tyto informace slouží jako „přihlašovací údaje“ Služby k prostředkům Autority.

Každá autorita má definovaný rámec informací, jež poskytuje „zdarma“, tedy bez nutnosti speciálního povolení od Uživatele. Typicky se jedná o běžné údaje jako je jméno, příjmení, věk, pohlaví a podobné, veřejně dostupné údaje. Mezi citlivější však patří už emailová adresa, telefonní číslo, adresa bydliště, ale i fotografie, videa a další data. Pro přístup k těmto datům musí mít Služba souhlas Uživatele.

Celý proces výměny dat je poté poměrně jednoduchý a jeho implementace nezabere více jak několik hodin.

Pro lepší představu uvedu příklad služby FOTO, která chce využívat možnosti Facebook API pro přihlášení a také načítání osobního alba uživatele.

Z pohledu služby je vše poměrně jednoduché. Základem tedy je definovat si rámec práv. Vzhledem k požadavku na přihlášení budeme potřebovat nejspíše emailovou adresu, jméno uživatele a také přístup k osobnímu albu. Na základě požadovaných práv vygeneruji přihlašovací odkaz sloužící k autorizaci našich práv na straně Autority. Odkaz obsahuje zašifrovaná požadovaná práva, údaje o naší aplikaci a tzv. token pro identifikaci relace.

Z pohledu uživatele bude vše vypadat následovně:

1. Uživatel chce využít možnosti naší služby, tak klikne na tlačítko „Přihlásit pomocí Facebooku“.
2. Je přesměrován na stránku spol. Facebook, kde je mu zobrazen výčet požadovaných práv. Zároveň je požádán o souhlas s poskytnutím vypsaných práv.
3. Uživatel si požadovaná práva pročte a rozhodne se, jestli je službě poskytne. Neposkytnutí těchto práv však běžně vede k odepření přístupu ke službě.
4. Po odsouhlasení či neodsouhlasení práv je Uživatel přesměrován zpět na stránky služby, kde server získá přístup ke všem odsouhlaseným Chráněným zdrojům.

5. Pokud se v budoucnu Uživatel rozhodne odejmout této službě přístup ke svým zdrojům, stačí na Facebook aplikace odebrat z autorizovaných.

3.4 Hybridní přístup

Každý z výše uvedených způsobů má jistá omezení, proto provozovatelé webových stránek často volí zlatou střední cestu v podobě kombinace otevřeného standardu a zároveň vlastních prostředků.

V praxi to znamená, že provozovatel webových stránek implementuje možnosti přihlášení např. skrze OAuth, ten však používá pouze pro počáteční získání údajů o uživateli. Získané informace slouží především pro vyplnění uživatelského profilu, čímž uživateli našich služeb ušetříme práci.

Standard OAuth nikdy neumožňuje získat společně s ostatními údaji také heslo. V případě využití otevřeného standardu pouze jako zdroje informací, je nutné na straně poskytovatele heslo uživateli vytvořit. Záleží tedy čistě na provozovateli webových stránek, zdali informace získané skrze přihlašování na sociální síť nechá jen jako pomocníka při prvotním vyplňování uživatelského profilu, nebo jako plnohodnotné řešení přihlašování do služeb.

Chytrým řešením se zdá být možnost ponechat přihlašování jak skrze OAuth, tak i skrze vlastní metody. I přes nezanedbatelné množství aktivních profilů na sociálních sítích, nemá svůj profil každý. Ponechat možnost přihlášení pouze skrze sociální síť by bylo tedy krátkozraké. Kombinace obou způsobů umožňuje poskytovateli udržovat uživatelský profil stále aktuální, aniž by musel svá data aktualizovat na více místech a zároveň se na naše služby dostanou i lidé bez profilu na sociální síti.

Nevýhodou tohoto řešení se může zdát nutnost implementace dvou nebo více způsobů ověření.

3.5 Bezpečnost autentizačních metod

Z hlediska bezpečnosti se lze koukat na dostupné autentizační metody dvojím způsobem.

Na jednu stranu výrazně usnadňují manipulaci s uživatelskými údaji. Provozovatelé koncové služby, která implementuje přihlašování pomocí OpenID nebo OAuth nemají přístup k heslům uživatelů. Z toho plyne, že uživatel je chráněn proti možnému zneužití hesla ze strany poskytovatele a to ať už proniknutím útočníka do jeho databáze nebo zneužití hesla samotným poskytovatelem.

Nevýhoda je, že samotná myšlenka těchto metod, mít jedny přihlašovací údaje k více službám, porušuje sama o sobě pravidla bezpečnosti. Pokud se útočník dostane do databáze poskytovatele naší identity, případně zjistí heslo k tomuto jedinému účtu, získá přístup ke všem našim propojeným službám.

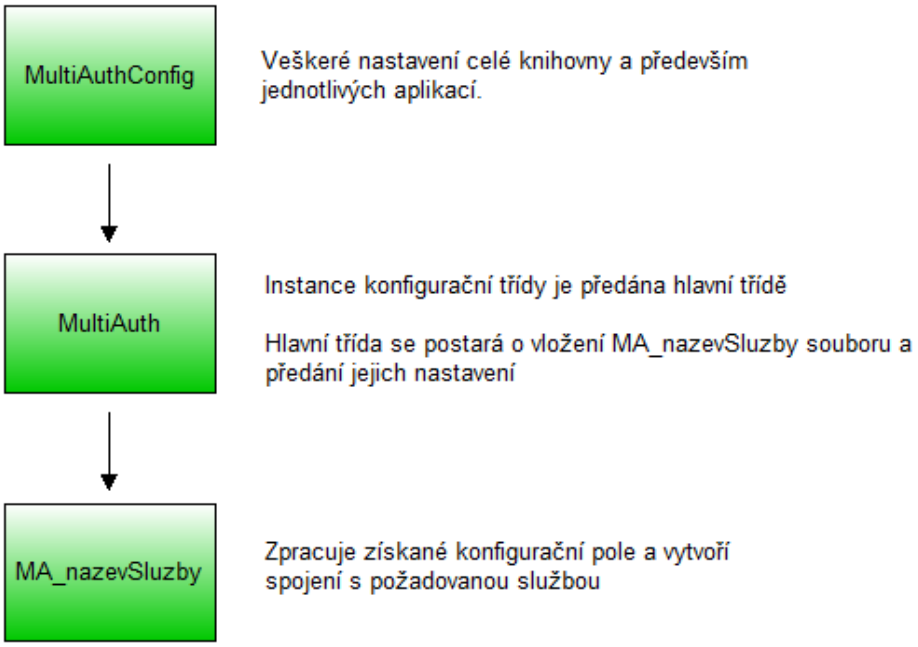
4 MultiAuth

Knihovna `MultiAuth` se dá chápat jako nadstavba pro stávající autentizační mechanizmy sociálních sítí. V základu obsahuje napojení na API sociálních sítí Twitter, Facebook a Google Plus, ale je navržena s ohledem na budoucí rozšiřitelnost.

Hlavní důraz při návrhu této knihovny byl kladen na jednoduchost použití v praxi, její možné rozšíření třetími stranami, ale zároveň zachování plné funkčnosti využitých API.

Knihovna má primárně ulehčit programátorům práci při rutinním získávání dat o uživatelích, které mají později sloužit k jejich přihlašování na vlastních serverech.

Samotná knihovna se skládá z několika drobnějších komponent, které navzájem spolupracují.



Obrázek 9 - Funkce MultiAuth knihovny

4.1 Požadavky

Ke svému běhu vyžaduje knihovna pouze PHP 5.3.0 a vyšší. Další omezení jsou dána pouze použitými API třetích stran. Obecně se však dá předpokládat nutnost cURL knihovny.

4.2 Instalace

Jako prevenci k možným kolizím, využívá knihovna vlastní namespace `MultiAuth`. Na to je třeba brát ohledy při následné implementaci.

4.2.1 Rozhraní IAuthModule

API každé z využívaných služeb je trochu jiné. Je třeba je tedy sjednotit. Potřebnou abstrakci nám poskytuje právě rozhraní `IAuthModule`. Rozhraní obsahuje pouze 5 metod, orientovaných především na práci s přihlašovacími údaji.

```
interface IAuthModule {

    public function init(); // Počáteční nastavení modulu

    public function isLogged(); // Je uživatel přihlášen?

    public function getProfile(); // Vrací získané informace

    public function getLoginUrl(); // Vrací URL pro přihlášení

    public function logout(); // Lokálně odhlásí uživatele
}
```

4.2.2 Třída MultiAuthConfig

Základním stavebním kamenem celé knihovny je právě `MultiAuthConfig`. Už z názvu vyplývá, že se bude jednat o konfigurační třídu celé knihovny. Je to také první a poslední místo, kde se programátor setká s přihlašovacími údaji k jednotlivým API.

Samotná třída je od zbytku aplikace oddělena z ryze praktických důvodů. Bývá zvykem všechny konfigurační údaje spravovat z jednoho místa. Z toho důvodu se také ukládají oddělené od zbytku aplikace v samostatném souboru, zpravidla `config.php`. Díky aplikovanému členění nemusí programátor tuto architekturu porušovat, stačí si nastavit konfigurační třídu a tu pak jako celek předat třídě hlavní.

Nejdůležitějšími metodami této konfigurační třídy jsou metody `allowNazevSluzby()`. Správné vyplnění parametrů jedné či více z těchto metod způsobí zahrnutí vybrané služby mezi požadované. Po nastavení služeb se pro jednotlivé služby automaticky zavedou přístupové klíče a zpřístupní se celé API.

Důležité je také zmínit, že jedna služba může být zavedena v knihovně pouze jednou. V praxi toto omezení pravděpodobně nikoho nepostihne, ovšem v případě, že by se tak stalo, bylo by nutné od sebe nejen oddělit jednotlivé konfigurační třídy, ale mohlo by také docházet ke kolizím názvů jednotlivých sessions.

Třída obsahuje pouze jeden privátní atribut `$allowedServices`, obsahující seznam všech již aktivovaných služeb. Jednotlivé položky seznamu jsou zde reprezentovány jako instance třídy `MultiAuthModule`.

```
  // @var array $allowedServices
  private $allowedServices = array();
```

4.2.2.1 Metoda getAllowedServices()

Pokud tuto metodu použijeme bez parametru, vrátí nám pole všech již aktivovaných služeb, které jsme si v konfigurační třídě nastavili.

V případě, že při volání použijeme také parametr, navrátí metoda jednu konkrétní aktivovanou službu. Není-li námi požadovaná služba aktivní, program vyvolá výjimku.

```
/**
  * @param string $service
  * @return array
  * @return MultiAuthModule
  * @throws Exception
  */
  public function getAllowedServices($service = null) {
        if (is_null($service)) {
            return $this->allowedServices;
        } else {
            if ($this->isAllowed($service)) {
                return $this->allowedServices[$service];
            } else {
                throw new Exception("Tato služba není povolena");
            }
        }
  }
```

4.2.2.2 Metoda registerService()

Další z metod obsluhující zaregistrované služby je `registerService()`. Jedinou úlohou této metody je vložit uživatelem zvolenou službu mezi aktivní, resp. zaregistrované.

Z uvedeného kódu je také patrné, že chceme-li zaregistrovat stejnojmennou službu vícekrát s jinými přihlašovacími klíči, bude platná pouze poslední vložená varianta, všechny předchozí budou přepsány.

```
/**
  * Registruje službu mezi povolené
  * @param MultiAuthModule @Module
  */
  private function registerService(MultiAuthModule $Mod) {
     $this->allowedServices[$Module->getServiceName()] = $Mod;
  }
```

4.2.2.3 Metoda isAllowed()

Pomocná metoda, tzv. helper. Tato metoda vrací `TRUE` v případě, že dotazovaná služba je již mezi registrovanými, v opačném případě vrací `FALSE`.

```
/**
  * Kontrola, jestli je povolená zadaná služba
  * @param string $service
  * @return boolean
  * @throws Exception
  */
  public function isAllowed($service) {
     if (empty($service)) {
         throw new Exception("Název aplikace je prázdný.");
     }
     return array_key_exists($service, $this->allowedServices);
  }
```

4.2.2.4 Metoda debug()

Jednoduchá metoda pro aktivaci výpisu chybových hlášení v PHP.

```
/**
  * Zobrazování PHP chybových hlášení
  */
  public function debug() {
     ini_set("display_errors", 1);
     error_reporting(E_ALL ^ E_NOTICE);
  }
```

4.2.3 Třída MultiAuthServices

Návrh knihovny byl proveden s důrazem na možná budoucí rozšíření o další podporované služby, bylo tedy třeba maximálně zjednodušit cestu pro případné přidání nové služby. Jelikož s názvem požadované služby musí manipulovat jak samotná knihovna, tak i programátor snažící se o její implementaci, mohlo by vznikat velké množství chyb způsobených překlepy a nepřesnostmi.

Z toho důvodu byla vytvořena podpůrná třída `MultiAuthServices`. Z programátorského hlediska se jedná pouze o náhradu nepodporovaného výčtového typu ENUM (jazykem PHP). `MultiAuthServices` tedy obsahuje seznam všech dostupných metod, které knihovna aktuálně dokáže obsloužit.

```
/**
 * Slouží jako výčet podporovaných služeb
 * Náhrada za ENUM, který PHP nepodporuje
 * @author Jakub Stach
 */
abstract class MultiAuthServices {

     // Typ služby Facebook
    const FACEBOOK = "facebook";
     // Typ služby Twitter
    const TWITTER = "twitter";
     // Typ služby Google
    const GOOGLE_PLUS = "googleplus";
}
```

Pro výpis některé z dostupných služeb stačí tedy zavolat:

```
echo MultiAuthServices::FACEBOOK; // Vypíše „facebook“
```

4.2.4 Třída MultiAuthModule

Ve chvíli, kdy programátor vyplní všechny potřebné klíče a další údaje nutné k aktivaci vybrané služby, se vytvoří instance třídy `MultiAuthModule`. Do instance této třídy se uloží veškeré zadané údaje v podobě atributů, aby s nimi bylo možné nadále pohodlně pracovat.

Ke každému z obsažených atributů je vytvořen jak getter, tak setter[3].

Třída je navržena univerzálně a je možné do ní přidávat libovolné množství dalších atributů, přičemž pro zachování správné funkčnosti knihovny je nutné zachovat atributy stávající.

Veškeré atributy této třídy jsou privátní a nepovinné s výjimkou `$serviceName`. Z uvedeného kódu je také patrné, že prázdný atribut `$serviceName` vyvolá výjimku.

```
/**
  * Typ služby (z MultiAuthServices)
  * @var string
  */
  private $serviceName = null;

/**
  * Vrací název služby
  * @return string
  * @throws Exception
  */
   function getServiceName() {
        if (empty($this->serviceName)) {
            throw new Exception("Aplikace nemá žádné jméno");
        }
        return $this->serviceName;
    }
```

Veškeré obsažené atributy třídy jsou výsledkem sjednocení vyžadovaných informací k aktuálně dostupným službám.

[3] http://php.vrana.cz/prace-s-vlastnostmi-pomoci-metod.php

4.2.5 Třída MultiAuthSession

Jako vhodné médium pro uchování dat, se kterými třída `MultiAuth` pracuje, bylo určeno tzv. sezení, neboli session. Pomocí session jsme schopni uchovat data i po znovunačtení webové stránky.

Aby nedocházelo ke kolizím v názvech, ukládá se pro každou službu session ve tvaru „`MA_nazevSluzby`“, kde si vybraná služba uchovává veškeré své parametry, včetně celého načteného profilu ze sociální sítě.

Na následující řádcích je možné vidět jak vypadá session po přihlášení uživatele Facebooku:

```
[MA_facebook]
=> Array(
                [logged] => 1
                [profile] => Array(
                    [id] => 10205008776271518
                    [email] => exquis.pce@gmail.com
                    [first_name] => Jakub
                    [gender] => male
                    [last_name] => Stach
                    [link] =>
            https://www.facebook.com/app_scoped_user_id/1020500877627151
                    [locale] => cs_CZ
                    [name] => Jakub Stach
                    [timezone] => 2
                    [updated_time] => 2015-02-06T23:29:32+0000
                    [verified] => 1
                 )
)
```

Samotná třída má pouze několik obslužných metod a jeden privátní atribut `$service`, obsahující název služby, která využívá danou instanci `MultiAuthSession`.

Konstruktor třídy nastavuje pouze daný atribut `$service` a snaží se zpřístupnit úložiště session, pokud dosud není zpřístupněno.

4.2.5.1 Metoda setSession() a getSession()

Nastavuje konkrétní službě parametr `$variable` na hodnotu `$data`. Opakem je metoda `getSession()`, která vrací hodnotu session s parametrem `$variable`.

```
/**
  *
  * @param string $variable
  * @param mixed $data
  */
  function setSession($variable, $data) {
        if (count($data)) {
            $_SESSION[$this->service][$variable] = $data;
        }
   }
```

4.2.5.2 Metoda deleteSession()

Bezparametrická metoda, která smaže veškerá data vybrané služby.

```
/**
  * Smaže uživatelská data
  *
  */
  function deleteSession() {
      unset($_SESSION[$this->service]);
  }
```

4.2.5.3 Metoda createSessionName()

Vrací jedinečné jméno ve tvaru `MA_nazevSluzby`.

```
/**
  * Vrátí název session pro službu
  *
  * @return string
  */
  private function createSessionName($service) {
        return "MA_" . $service;
  }
```

Celá třída má ještě několik dalších metod, které jsou ovšem pouze nadstavbou již vysvětlených metod, proto je není nutné podrobněji představovat.

4.3 Třída MultiAuth

Srdcem knihovny je třída `MultiAuth`, starající se o distribuci nastavení jednotlivých služeb. Třída má hned několik úkolů:

- převzít instanci konfigurační třídy,
- importovat pro aktivní služby jejich zdrojové soubory,
- předat jednotlivá nastavení koncovým službám,
- v případě vyžádání vytvořit a uchovat instanci vybrané služby.

Programově se skládá z 2 privátních atributů. První atribut je `$config`, ve kterém se uchovává instance celé konfigurační třídy `MultiAuthConfig`, druhým parametrem je pole `$instances`, obsahující všechny již vytvořené instance vybraných služeb.

Konstruktor třídy se stará o uložení instance konfigurační třídy do privátního atributu. Dále v případě neaktivní session zahajuje instanci. V posledním kroku zahrnuje zdrojové soubory koncových tříd služeb.

```
/**
  *
  * @param \MultiAuth\MultiAuthConfig $config
  */
  public function __construct(MultiAuthConfig $config) {
        $this->config = $config;
        $this->startSession();
        $this->importFiles();
  }
```

Metoda `importFiles()` prochází pole vybraných a nastavených služeb a ke každé službě načte soubor s jejími zdrojovými kódy, typicky `MA_nazevSluby.php`.

```
/**
  * Importuje MA_**** soubory k povoleným službám
  * @throws Exception
  */
 private function importFiles() {
     foreach ($this->config->getAllowedServices() as
                                        $key => $service) {
       if (file_exists($this->getModuleClassPath($key))){
            require_once $this->getModuleClassPath($key);
        } else {
           throw new
              \Exception("Zdr. soubory neexistuji:" . $key);
            }
        }
    }
```

Před načtením zdrojových souborů jednotlivých tříd je třeba znát jejich umístění. Očekávanou cestu nám vrací metoda `getModuleClassPath()`.

Doposud však není vytvořena instance žádné ze služeb. To se odehrává až ve chvíli, kdy si programátor požádá o zvolenou instanci metodou `getService()`, jejímž parametrem je název služby, o kterou žádá. Vrací tedy třídu implementující `IAuthModule`.

Metoda zkontroluje, jestli vybraná instance existuje, tedy pokud je uložena v privátním poli `$instances`. Pokud tomu tak není, předá parametr `$service` metodě `createInstance()`.

```
/**
 * Vrátí instanci konkrétní služby
 * @param string $service
 * @return IAuthModule
 */
public function getService($service) {
  if (!($this->instances[$service] instanceof IAuthModule)){
          $this->createInstance($service);
  }
     return $this->instances[$service];
  }
}
```

Poslední z obsažených metod je již zmíněná `createInstance()`. Její úkol spočívá ve vytvoření instance na základě předaného string parametru `$service`. Ještě než třída vytvoří instanci služby, musí zkontrolovat, zdali je tato služba mezi povolenými. Pokud není, je vyvolána výjimka.

```
/**
  * Vytvoří instanci zvolené služby
  *
  * @param string $service
  * @throws Exception
  */
  public function createInstance($service) {
     if ($this->config->isAllowed($service)) {
         $className = "\MultiAuth\Modules\MA_" . $service;
            $this->instances[$service] = new
$className($this->config->getAllowedServices($service));
            $this->instances[$service]->init();
        } else {
            throw new Exception("Služba není povolena");
        }
    }
```

4.4 Třídy MA_nazevSluzby

Jelikož OAuth definuje pouze protokol pro autentizaci a veškeré další práce s údaji probíhají přes vlastní API služby, není možné vytvořit zcela univerzální knihovnu, které by programátor předal pouze klientské ID a klíč SECRET. Z tohoto důvodu je nutné napsat pro každou novou aplikaci vlastní obsluhu jejího API. Ostatně proto je v knihovně přítomno rozhraní `IAuthModule`.

Pro knihovnu `MultiAuth` platí, že každá třída `MA_nazevSluzby` musí být umístěna v kořenovém adresáři knihovny (stejný adresář, ve kterém je umístěn soubor `MultiAuth.php`) a zároveň musí implementovat rozhraní `IAuthModule`.

V souboru „`MA_nazevSluzby.php`“ musí programátor sám řešit vložení všech souborů spojených s API té které služby. Ty bývají k dispozici v adresáři pojmenovaném „`nazevSluzby`“, umístěném na stejném místě, jako soubor „`MA_nazevSluzby.php`“.

K různým službám bývá k dispozici i několik odlišných API, lišící se především v jednoduchosti ovládání. Pro ukázkové implementace jsem využil následující:

- **Twitter** - EpiTwitter[4],
- **Facebook** – originální SDK[5],
- **Google Plus** – originální PHP API[6].

4.5 Rozšíření o vlastní službu

Už bylo zmíněno, že celá knihovna `MultiAuth` je navržena s ohledem na možná budoucí rozšíření.

Jak by tedy vypadalo, když bychom chtěli knihovnu rozšířit např. o podporu služby LinkedIn?

1. Zaregistrujeme službu do `MultiAuthServices`, přidáním následujícího řádku. Po slovíčku `const` zvolíme název služby, pod kterým bude vystupovat v knihovně `MultiAuth`, hodnota může mít libovolný formát. Zavedené služby však napovídají, že by se mělo jednat o název služby psaný malými písmeny, bez diakritiky.

```
const LINKEDIN = "linkedin";
```

2. Ve třídě `MultiAuthConfig` je nutné vytvořit metodu, která zaregistruje službu LinkedIn mezi povolené. Opět dle již stanovených pravidel by mělo jít o veřejnou metodu jména `allowLinkedIn`(). Parametry metody vyčteme z dokumentace služby[7]. Zde jsme zjistili, že to musí být přístupové klíče získané z vývojářské sekce služby LinkedIn[8]. Výsledná metoda bude vypadat zhruba takto:

```
public function allowLinkedIn($APP_ID, $APP_SECRET) {
   $LinkedIn = new Modules\MultiAuthModule();
   $LinkedIn->setServiceName(MultiAuthServices::LINKEDIN);
   $LinkedIn->setClientId($APP_ID);
   $LinkedIn->setClientSecret($APP_SECRET);
   $this->registerService($LinkedIn);
}
```

[4] https://github.com/jmathai/twitter-async/blob/master/EpiTwitter.php
[5] https://developers.facebook.com/docs/reference/php/4.0.0
[6] https://github.com/google/google-api-php-client
[7] https://developer.linkedin.com/docs
[8] https://developer.linkedin.com/

3. Nyní máme uvnitř knihovny dostupné parametry `APP_ID` a `APP_SECRET`. Služba je tedy zaregistrována, ale stále zde není žádný obslužný kód, který by s údaji správně naložil. To napravíme vytvořením souboru „`MA_nazevSluzby.php`“, kde `nazevSluzby` je hodnota parametru const z 1. kroku tohoto návodu. V našem případě bude výsledný soubor „`MA_linkedin.php`“. Soubor musí být umístěn ve stejné složce jako `MultiAuth.php` a musí obsahovat stejnojmennou třídu implementující rozhraní `IAuthModule`.

   Do konstruktoru této třídy předá třída `MultiAuth` konfiguraci služby, skládající se z parametrů, uvedených v bodě 2. Je také potřeba zaregistrovat službu do session. Výsledný konstruktor bude vypadat tedy následovně:

```
/**
  * Konstruktor
  *
  * @param \MultiAuth\Modules\MultiAuthModule $config
  */
 public function __construct(MultiAuthModule $config) {
     $this->config = $config;
     $this->session = new MultiAuthSession($this->config-
>getServiceName());
 }
```

4. Poslední krok se skládá z implementace OAuth přihlašování dle dokumentace vybrané služby[4].

5 Nasazení v praxi

Před využitím knihovny je třeba zdrojové soubory zahrnout do cílového projektu ať už pomocí `require` nebo `__autoload()`.

```
require_once 'MultiAuth.php';
```

Jakmile je knihovna k dispozici, vytvoříme si instanci konfigurační třídy této knihovny.

```
$MaConfig = new MultiAuth\MultiAuthConfig();
```

Konfigurační třída obsahuje metody pro nastavení všech údajů, které pro svůj běh vyžaduje využívané API. Samotná konfigurační třída by nám k ničemu nebyla, potřebujeme vytvořenou konfiguraci předat hlavní třídě a to vložením přímo do jejího konstruktoru.

Hlavní třída se postará o předání konfiguračních údajů koncovým modulům.

```
$Ma = new MultiAuth($MaConfig);
```

Nyní stačí již využít příkazy z rozhraní `IAuthModule`.

Nejprve potřebujeme získat instanci vybrané služby pomocí metody getService().

```
$facebook = $Ma->getService(MultiAuthServices::FACEBOOK);
```

Nyní zkontrolujeme, jestli je uživatel přihlášen. V případě, že je, vypíšeme jeho údaje. Pokud není, zobrazíme mu přihlašovací tlačítko.

```
if ($facebook->isLogged()) {
      $facebook->printData();
} else {
      $facebook->showLoginButton();
}
```

6 Závěr

Snad každá služba v dnešní době vyžaduje prokázání identity. Každý, kdo je na internetu jen trochu aktivní mi dá jistě za pravdu, že z pohledu uživatele je téměř nemožné být vlastníkem desítek či stovek uživatelských účtů. Je na čase se zamyslet, jestli by se nevyplatilo implementovat do všech nově vznikajících služeb libovolnou hromadnou autentizační metodu.

Jistě existuje množina služeb, kde je implementace těchto metod naprosto nežádoucí. Dobrým příkladem může být přihlašování do datové schránky nebo do internetového bankovnictví. Drtivá většina služeb, se kterými běžný uživatel přijde do styku, nemá povahu takto důležité služby a to ani v případě, že by využitý způsob autentizace byl kompromitován.

Standardy jako OpenID či OAuth jistě mají na internetu své místo a svou přítomností ulehčují každodenní práci s mnoha službami.

Cílem této práce bylo vytvořit snadno použitelnou a rozšiřitelnou knihovnu pro práci s více druhy autentizačních mechanizmů. I přesto, že podobné knihovny již existují, u každé jsem narazil na části, ve kterých nesplňovaly mé očekávání. Nejčastěji se jednalo o omezení funkčnosti využitého API, čemuž jsem v mé knihovně zamezil použitím dědičnosti tříd.

Rád bych, kdyby se má knihovna stala často využívaným prostředkem ke komunikaci skrze protokol OAuth. Osobně jsem ji již využil nejen na ukázkovém projektu přiloženém na CD, ale i na osobním projektu pro studenty UPCE FEI.

7 Použitá literatura

[1] Webová stránka. WIKIPEDIA.ORG. *Wikipedia* [online]. 2015 [cit. 2015-04-28]. Dostupné z: http://cs.wikipedia.org/wiki/Webov%C3%A1_str%C3%A1nka

[2] MALÝ, Martin. OAuth – nový protokol pro autentizaci k vašemu API. MALÝ, Martin. *Zdroják.cz* [online]. 2008 [cit. 2015-04-28]. Dostupné z: http://www.zdrojak.cz/clanky/oauth-novy-protokol-pro-autentizaci-k-vasemu-api/

[3] CHAMLING, Saran. Login with Google using PHP API library. CHAMLING, Saran. *Sanwebe* [online]. 2012 [cit. 2015-04-28]. Dostupné z: http://www.sanwebe.com/2012/11/login-with-google-api-php

[4] JANOVSKÝ, Dušan. Javascript - Úvod. JANOVSKÝ, Dušan. *Jakpsatweb.cz* [online]. 2014 [cit. 2015-04-28]. Dostupné z: http://www.jakpsatweb.cz/javascript/javascript-uvod.html

[5] JANOVSKÝ, Dušan. CSS - Úvod. JANOVSKÝ, Dušan. *Jakpsatweb.cz* [online]. 2014 [cit. 2015-04-28]. Dostupné z: http://www.jakpsatweb.cz/css/css-uvod.html

[6] ŠTRÁFELDA, Jan. Co je AJAX. ŠTRÁFELDA, Jan a Dušan JANOVSKÝ. *Adaptic* [online]. 2015 [cit. 2015-04-28]. Dostupné z: http://www.adaptic.cz/znalosti/slovnicek/ajax/

[7] JavaScript and JQuery, What is the difference?. *Purencool.com* [online]. 2015 [cit. 2015-04-28]. Dostupné z: http://purencool.com/javascript-and-jquery-what-is-the-difference

[8] TAMADA, Srinivas. Login with Google Plus OAuth. TAMADA, Srinivas. *9lessons.info* [online]. 2011 [cit. 2015-04-28]. Dostupné z: http://www.9lessons.info/2011/10/login-with-google-plus-oauth.html

[9] MALÝ, Martin. Porovnání moderních autentizačních metod. MALÝ, Martin. *Zdroják.cz* [online]. 2008 [cit. 2015-04-28]. Dostupné z: http://www.zdrojak.cz/clanky/porovnani-modernich-autentizacnich-metod/

[10] VRÁNA, Jakub. 2010. 1001 tipů a triků pro PHP. Vyd. 1. Brno: Computer Press, 456 s. ISBN 978-80-251-2940-1.

[11] LUBBERS, Peter, Brian ALBERS a Frank SALIM. *HTML5: programujeme moderní webové aplikace*. Vyd. 1. Brno: Computer Press, 2011, 304 s. ISBN 978-80-251-3539-6.

[12] MALÝ, Martin. 2009. Implementace přihlašování pomocí Live ID. MALÝ, Martin. Zdroják.cz [online]. [cit. 2015-05-07]. Dostupné z: http://www.zdrojak.cz/clanky/implementace-prihlasovani-pomoci-live-id/

[13] VÁVRŮ, Vlastimil. 2009. OAuth - představení autentizačního protokolu. VÁVRŮ, Vlastimil. ITPlace [online]. [cit. 2015-05-07]. Dostupné z: http://blog.itplace.cz/oauth-predstaveni-autentizacniho-protokolu/

8 Přílohy

Příloha 1 – Stromová struktura zdrojových souborů aplikace

```
MultiAuth
│
│   IAuthModule.php
│   MA_facebook.php
│   MA_googleplus.php
│   MA_twitter.php
│   MultiAuth.php
│   MultiAuthConfig.php
│   MultiAuthModule.php
│   MultiAuthServices.php
│   MultiAuthSession.php
│
├───facebook
├───google
└───twitter
```

Příloha 2 – Diagram tříd knihovny MultiAuth