ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název: Analýza, návrh a implementace jednobodové autentizace pro portál Webgarden
Student: Jan Chudomský
Vedoucí: Ing. Lukáš Alexandr
Studijní program: Informatika
Studijní obor: Web a multimédia
Katedra: Katedra softwarového inženýrství
Platnost zadání: Do konce letního semestru 2016/17

Pokyny pro vypracování

Cílem práce je analyzovat požadavky na jednobodovou autentizaci (SingleSignOn - SSO) v potrálu Webgarden, který běží na více doménách. Na základě analýzy vytvořit návrh a implementovat ho.

Požadavky:
1. Nastudujte a krátce popište běžně používané techniky a nástroje (Shibboleth, OAuth2, SAML, apod.).
2. Analyzujte a specifikujte požadavky pro SSO na portálu Webgarden. Uvažujte, že portál běží na různých doménách. Uvažujte řešení centrálního autentikačního serveru i napojení jednotlivých aplikací či modulů portálu.
3. Na základě bodů 1 a 2 proveďte návrh, implementaci, otestování a dokumentaci SSO pro portál Webgarden.

Seznam odborné literatury

Dodá vedoucí práce.

L.S.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 29. října 2015

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra softwarového inženýrství

Bakalářská práce

Analýza, návrh a implementace jednobodové autentizace pro portál Webgarden

Jan Chudomský

Vedoucí práce: Ing. Lukáš Alexandr

13. května 2016

Poděkování

Děkuji Ing. Lukáši Alexandrovi za vedení práce a užitečné rady. Dále děkuji všem, kteří mi pomohli s korekcí gramatických chyb.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, avšak pouze k nevýdělečným účelům. Toto oprávnění je časově, teritoriálně i množstevně neomezené.

V Praze dne 13. května 2016 .....

České vysoké učení technické v Praze
Fakulta informačních technologií
© 2016 Jan Chudomský. Všechna práva vyhrazena.
Tato práce vznikla jako školní dílo na Českém vysokém učení technickém v Praze, Fakultě informačních technologií. Práce je chráněna právními předpisy a mezinárodními úmluvami o právu autorském a právech souvisejících s právem autorským. K jejímu užití, s výjimkou bezúplatných zákonných licencí, je nezbytný souhlas autora.

Odkaz na tuto práci

Chudomský, Jan. *Analýza, návrh a implementace jednobodové autentizace pro portál Webgarden.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2016.

Abstrakt

Tato práce se zabývá návrhem a realizací jednobodové autentizace na portálu Webgarden. Práce představuje řešení problému klasické autentizace, kdy si uživatel musí pamatovat přihlašovací údaje ke každé službě, kterou chce používat. Vytvořené řešení poskytuje možnost jednotně se přihlásit pomocí jedněch přihlašovacích údajů a to hned několika způsoby, včetně přihlášení pomocí Google a Facebooku. Přínosem této práce je usnadnění přihlašovacího procesu, což přispívá k uživatelově pohodlí a spokojenosti.

Klíčová slova autentizace, jednobodová autentizace, modul, SSO, Webgarden, PHP, Single sign-on

Abstract

This thesis describes the design and implementation of a single point authentication for portal Webgarden. Thesis represents solution to the classic problem of authentication, where user has to remember his credentials for each service he uses. Produced solution provides several possibilities to login once, centrally, with one credentials only. Login with Google and Facebook is also included. The contribution of this thesis lies in simplifying login process, which will lead to bigger user's comfort and satisfaction.

Keywords authentication, single point authentication, module, SSO, Webgarden, PHP, Single sign-on

Obsah

Úvod **1**

1 Analýza **3**
1.1 Single sign-on . . . 3
1.2 SAML 2.0 . . . 7
1.3 JSON Web tokens . . . 8
1.4 OAuth 2.0 . . . 11
1.5 OpenID Connect 1.0 . . . 13
1.6 Same-origin policy . . . 15
1.7 CORS . . . 15
1.8 Možnosti realizace SSO . . . 16
1.9 Vybrané řešení . . . 18
1.10 Požadavky na aplikaci . . . 19

2 Realizace **21**
2.1 Návrh architektury aplikace . . . 21
2.2 Modul SSO . . . 24
2.3 JWT . . . 25
2.4 LoginHelper . . . 26
2.5 LoginMethod . . . 26
2.6 Renderer . . . 29
2.7 Přihlašovací metody detailně . . . 31
2.8 Bezpečnost . . . 33
2.9 Návrh architektury databáze . . . 35
2.10 Závěr . . . 36

3 Integrace do systému Webgarden **37**
3.1 Databáze . . . 37
3.2 Iframe metoda . . . 37
3.3 XSLT . . . 39

3.4 CAPTCHA 40
3.5 SP a CAP v jednom 41

4 Testování **43**

Závěr **45**

Literatura **47**

A Seznam použitých zkratek **49**

B Použité nástroje **51**

C Ukázky kódu a obrázky **53**

D Manuál na instalaci demo aplikace **59**
D.1 Nastavení hosts 59
D.2 Nastavení webserveru 59
D.3 Vytvoření databáze 60
D.4 Konfigurace aplikace 60
D.5 Zkouška aplikace 61

E Obsah přiloženého CD **63**

Seznam obrázků

1.1 Diagram průběhu procesu SSO z hlediska UX 5
1.2 Autentizace pomocí JWT 10
1.3 Abstraktní tok OAuth2.0 protokolu 12
1.4 Abstraktní tok OpenID Connect 1.0 protokolu 14
1.5 Obecný průběh přihlášení přes SSO 18

2.1 Příprava Clienta 22
2.2 Příprava EndPointu 23
2.3 Logika metody setOnLoginRequest ve třídě HTTPLogin 27
2.4 Návrh architektury databáze 35

C.1 Části Modulu SSO 53
C.2 Doménový model Clienta v Modulu SSO 54
C.3 Doménový model EndPointu v Modulu SSO 55
C.4 Ukázka XMLHttpRequest CORS požadavku – odpověď ze serveru 56
C.5 Tvorba a ověření digitálního podpisu 57

Úvod

Webgarden je služba pro tvorbu a správu webových stránek, která vznikla v roce 2006. Postupem času služba získala mnoho uživatelů a nyní by jim chtěla umožnit snadnější přihlašování do administračního rozhraní a na všechny stránky vytvořené službou Webgarden.

V současnosti, pokud se uživatel přihlásí do administračního rozhraní Webgarden, je přihlášen pouze tam. V případě, že by chtěl použít mobilní verzi nebo se přihlásit na jiný z webů vytvořený systémem Webgarden, musí svoje přihlašovací údaje zadat znovu.

Cílem této práce je usnadnit přihlašovací proces. Výsledkem bude centralizovaný proces přihlášení, kde uživatel nebude muset opakovaně zadávat svoje přihlašovací údaje a vzroste tím jeho spokojenost. Dále bude uživateli umožněno přihlásit se pomocí webové služby Facebook a Google.

Práce nejprve analyzuje výhody jednobodové autentizace a možnosti jak ji realizovat. V další kapitole popisuje návrh řešení a implementaci. Třetí kapitola popisuje implementaci na portálu Webgarden. Poslední kapitola pojednává o testování.

Kapitola 1

Analýza

1.1 Single sign-on

Single sign-on (SSO) směřuje proces autentizace uživatele u více webových služeb do jednoho centrálního bodu, zdroje důvěry, vůči kterému se uživateli stačí ověřit, aniž by se musel ověřovat jednotlivě v každé službě. Podle [1] je SSO uživatelská zkušenost, kdy se uživatel přihlásí pouze jednou a dále se může pohybovat přes množství aplikací plynule, bez nutnosti zadávat znovu přihlašovací údaje. SSO snižuje množství přihlašovacích údajů, které si musí uživatel pamatovat.

Při zkoumání SSO u různých internetových služeb lze zjistit, že provádějí proces SSO odlišně. Když Google přihlásí uživatele, přihlásí ho automaticky do všech svých služeb, aniž by se ho ptal, jestli to chce. V případě, že uživatel přistoupí např. na Youtube, tak je automaticky přihlášen jako uživatel, který se přihlásil na Google. Pokud je stejný scénář zopakován na Seznam.cz, probíhá jinak. Seznam.cz uživatele přihlásí, ale když uživatel přistoupí na další službu, např. Stream.cz, tak není automaticky přihlášen. Pokud klikne na tlačítko *Přihlásit se*, vyskočí okno, kde je dotázán, zda se chce přihlásit jako aktuálně přihlášený uživatel (ten, který se přihlásil na Seznam.cz) nebo jako někdo jiný.

1.1.1 Důležité pojmy týkající se SSO

- Autentizace – proces ověření identity uživatele neboli ujištění se, že uživatel je opravdu ten, za koho se vydává. Autentizace může probíhat například pomocí jména a hesla, občanského průkazu, otisku prstu, atd. Autentizace říká, kdo je uživatel.

- Autorizace – proces, který ověřuje, zda má uživatel právo přistoupit k chráněnému zdroji. Autorizace říká, na co má uživatel právo.

- Identity provider (IdP) – důvěryhodný zdroj, který spravuje informace o uživatelích a umožňuje ověření uživatelů, např. podle přihlašovacích

údajů. Příkladem může být Facebook, který spravuje obrovské množství uživatelů a umožňuje uživatelům přihlášení skrze jejich Facebook účet do aplikací třetích stran.

- Service provider (SP) – poskytovatel služby. Webová stránka, na které je služba dostupná. Například docs.google.com je poskytovatel služby pro online vytváření a formátování textových dokumentů.

- Central authentication point (CAP) – místo, kam jsou směřovány všechny požadavky na přihlášení případně i odhlášení uživatele.

1.1.2 Rozdílné typy SSO

Podle [1] většina SSO produktů na trhu může být rozdělena do dvou kategorií:

- Webové (Web-based) – také známé jako enterprise SSO nebo ESSO.
  - Internetové – SSO pro internetové aplikace, které interagují se zákazníkem (např. jedna aplikace pro zákaznickou podporu, druhá aplikace pro správu a zaplacení pohledávek)
  - Intranetové – SSO pro intranetové aplikace, kterou pomocí prohlížeče používají zaměstnanci organizace.
  - Vícedoménové – probíhá buď mezi dvěma a více doménami ve stejné organizaci, nebo mezi dvěma a více organizacemi, obvykle partnery.

- Newebové – také známé jako legacy SSO. Uživatel se přihlásí na své pracovní stanici, na které běží software, který si zapamatuje jeho přihlašovací údaje a vyplní za něj všechny další požadavky o přihlášení.

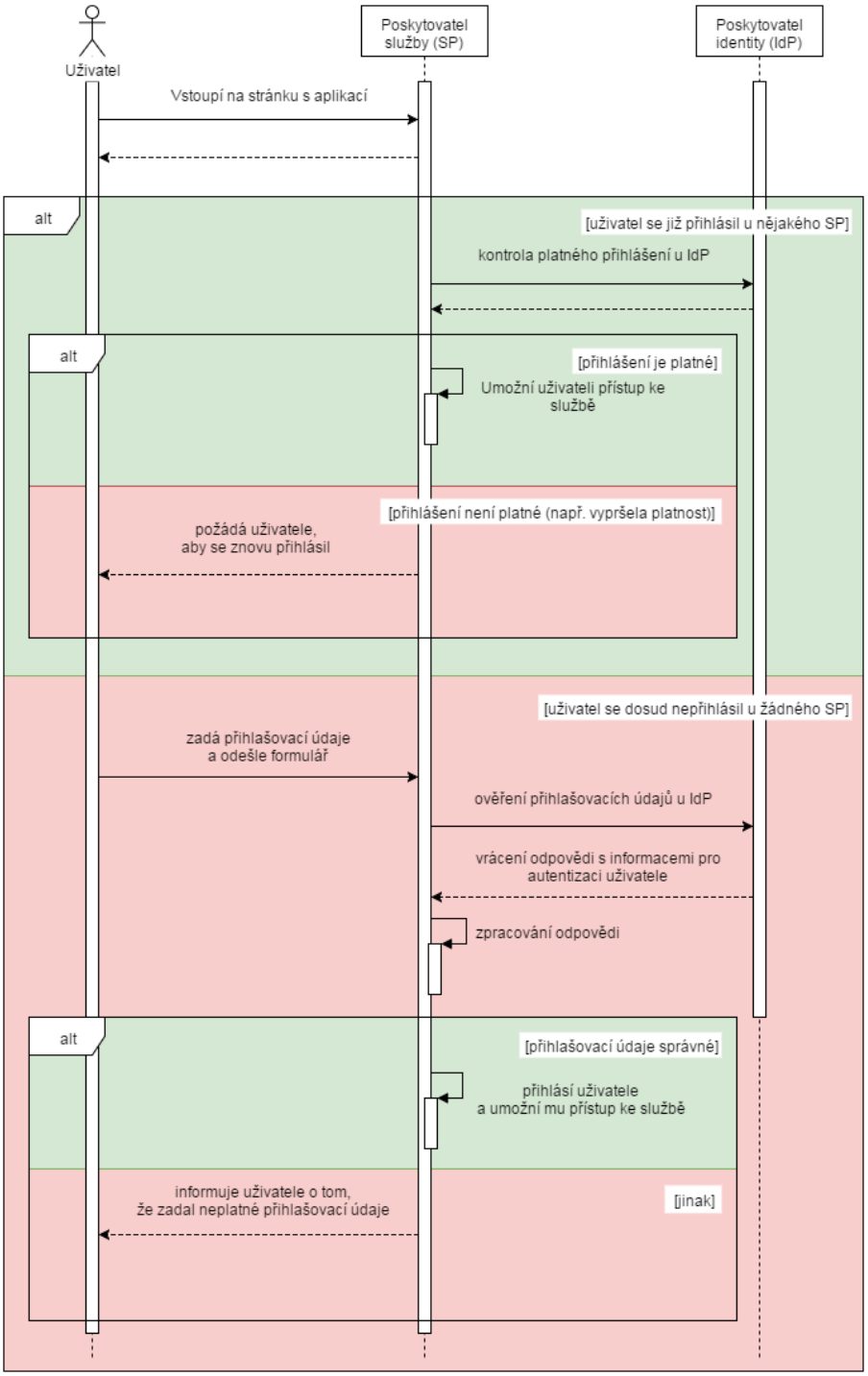
Tato práce se zabývá Web-based internetovým SSO.

1.1.3 Problém s více doménami

V rámci jedné domény lze SSO řešit pomocí *cookies*, které udržují informace o stavu autentizace uživatele, takže není nutné proces autentizace opakovat pokaždé, když uživatel přistoupí na stránku s aplikací.

Problém nastává při vícedoménovém SSO, kdy stav autentizace na jedné doméně není dostupný na doméně druhé, jelikož nelze na jedné doméně nastavovat cookie pro doménu druhou (např. na `mypage.cz` nastavit autentizační cookie pro `webgarden.cz`). Řešením tohoto problému je použití nějakého protokolu, pomocí něhož budou servery na různých doménách komunikovat s CAP, který bude řešit přihlášení uživatele. Jednotlivé stránky vytvořené systémem Webgarden budou hrát roli SP.

1.1.4 Průběh SSO



Obrázek 1.1: Diagram průběhu procesu SSO z hlediska UX

1.1.5 Existující řešení

1.1.5.1 Shibboleth

Shibboleth je open source softwarový balíček pro webové SSO napříč nebo uvnitř organizací. Implementuje široce používané standardy identity, zejména SAML, k poskytnutí Federated SSO a výměny atributů. Uživatel se ověřuje svými údaji nebo údaji jeho organizace. Organizace (nebo IdP) předá minimální informace o totožnosti nezbytné pro SP k tomu, aby rozhodl o autorizaci. Poskytuje také rozšířené funkce ochrany osobních údajů, které uživateli a jejich organizaci umožní kontrolu atributů pro každou aplikaci. Shibboleth server je napsán v jazyce Java [2].

1.1.5.2 CAS

Central Authentication Service (CAS) je standardizovaný mechanismus, kterým by webové aplikace měly autentizovat uživatele. CAS poskytuje službu `ESSO`, která je dobře dokumentovaná, existuje jako komponenta pro Java server, disponuje mnoha knihovnami pro různé programovací jazyky, je integrovaná u Moodle[1], TikiWiki[2] a podobných a má rozsáhlou komunitu. CAS server podporuje i protokoly SAML, OpenID a OAuth [3].

1.1.6 Klady a zápory SSO

1.1.6.1 Klady

- Zvýšení pohodlí uživatele – uživateli stačí jedno heslo, které mu umožní přístup do všech služeb.

- Zvýšení bezpečnosti – uživatel není nucen si pamatovat jednotlivé heslo pro každou službu. Často se stává, že si uživatel volí pro více služeb stejná hesla, což může vyústit v kompromitaci velkého množství údajů, v případě, že nějaký útočník heslo zjistí.

1.1.6.2 Zápory

Pokud by se uživatel přihlašoval do služby standardním způsobem, zadal by svoje přihlašovací údaje, odeslal formulář, stránka by se obnovila a uživatel by byl přihlášen. V případě SSO se může stát, že je uživatel přesměrován k CAP a následně zpět do služby, což může uživatele zmást. Mohl by si myslet, že se děje něco, co nechtěl.

[1] https://moodle.org/
[2] http://tiki.org/

1.2 SAML 2.0

OASIS Security Asertion Markup Language 2.0 definuje framework založený na XML, který slouží k popisu a výměně bezpečnostních informací mezi online byznys partnery. Bezpečnostní informace jsou vyjádřeny ve formě přenositelného SAML tvrzení (assertion), kterému aplikace třetích stran mohou věřit. SAML definuje přesnou syntaxi a pravidla pro vytváření, požadování, komunikaci a používání výše zmíněných SAML tvrzení [4].

1.2.1 SAML účastníci

Každá SAML výměna probíhá minimálně mezi systémovými entitami *SAML asserting party* a *SAML relying party.*

- Asserting party – systémová entita, která vytváří SAML tvrzení. Někdy se jí říká *SAML authority* [4].
- Relying party – systémová entita, která používá tvrzení, která přijala [4].

V každém SAML tvrzení figuruje subjekt (*principal* – entita, která může být autentizována) ke kterému se dané tvrzení vztahuje [4].

1.2.2 SAML role

SAML role jsou vymezeny SAML službami, zprávami protokolu, které použijí a typy tvrzení, které vygenerují. Pro webové SSO SAML definuje roli poskytovatele identity – *identity provider* (IdP) a roli poskytovatele služby – *service provider* (SP) [4].

1.2.3 Případ užití pro webové SSO

Nejdůležitější případ užití SAMLu.

- Vyvolaný IdP (IdP-initiated) – případ, ve kterém je uživatel přihlášen na webu IdP (`idp.example.com`) a přistupuje k prostředkům na těchto stránkách. V určitém okamžiku je přesměrován na web `sp.example.com`, což je partnerský web SP. V tomto případě jsou `idp.example.com` a `sp.example.com` byznys partneři a sdílí mezi sebou uživatelské údaje. IdP zašle tvrzení SP s tím, že uživatele zná, je autentizovaný a má jisté atributy (např. prémiový účet). Jelikož SP věří IdP, věří také tomu, že uživatel je ten správný uživatel, je správně autentizován a tím pádem přihlásí uživatele (vytvoří pro něj session). Po uživateli není vyžadováno opětovné ověření, když přistupuje na web `sp.example.com`.

- Vyvolaný SP (SP-initiated) – obvyklejší, než předchozí případ. Uživatel přistoupí na webové stránky SP (sp.example.com), kde dosud není přihlášen. SP je přesměruje k IdP (idp.example.com) s požadavkem na autentizaci kvůli tomu, aby mohl uživatele přihlásit. Jakmile je uživatel přihlášen, IdP vytvoří tvrzení, které použije SP k ověření uživatelových přístupových práv k chráněnému zdroji.

1.2.4 Komponenty

1.2.4.1 Tvrzení

SAML umožňuje jedné straně prohlašovat o předmětu bezpečnostní informace ve formě tvrzení [4]. SAML tvrzení by mohlo říkat, že uživatel se jmenuje Karel Kaňák, jeho emailová adresa je k.kan@example.com. Spolu s tvrzením lze přenášet další prohlášení o tom, jak proběhlo přihlášení uživatele, jaké má další atributy (např. uživatel Karel Kaňák je člen skupiny „gold members“) a co je oprávněn dělat (např. Karel Kaňák si může koupit cigarety a alkohol).

1.2.4.2 Protokoly

Protokoly jsou využity k přenosu SAML definovaných požadavků a odpovědí. Pomocí protokolů může subjekt iniciovat přihlášení i odhlášení [4].

1.2.4.3 Vazby

Vazby popisují, jak jsou různé SAML protokolové zprávy přenášeny přes základní transportní protokoly. Protokolové zprávy lze přenášet pomocí HTTP GET – jako přesměrování, velikosti omezeno prohlížečem (2–8 KB) i pomocí HTTP POST – jako base64-zakódovaný obsah HTML formuláře (velikost není omezena prohlížečem) [4].

1.2.4.4 Profily

Profily definují způsoby, jak nakombinovat SAML tvrzení, protokoly a vazby k poskytnutí vysoké interoperability pro určitý případ použití [4].

1.3 JSON Web tokens

JSON Web token (JWT) je otevřený standard (RFC 7519[3]), který definuje kompaktní a nezávislý způsob pro bezpečný přenos informací mezi stranami pomocí JSON objektu. Tyto informace lze ověřit a lze jim důvěřovat, protože jsou digitálně podepsány. JWT může být podepsán pomocí tajného hesla (HMAC) nebo pomocí páru veřejný/privátní klíč za použití RSA. Jelikož JWT

[3] https://tools.ietf.org/html/rfc7519

nepoužívá cookies, nemá problém s *Cross-Origin Resource Sharing* (CORS) [5].

- Kompaktnost – JWT má malou velikost, snadno může být poslán přes HTTP GET, HTTP POST nebo uvnitř HTTP hlavičky. Díky malé velikosti je přenos uskutečněn velmi rychle [5].
- Nezávislost – náklad (payload) obsahuje všechny potřebné informace o uživateli, takže není potřeba dalších dotazů na databázi [5].

1.3.1 Struktura JWT

JWT se skládá ze tří částí oddělených tečkami.

- Hlavička (Header) – obvykle obsahuje typ tokenu a hashovací algoritmus jako HMAC SHA256 nebo RSA [5].
- Náklad (Payload) – obsahuje tvrzení o entitě (např. uživatel) a další metadata. Existují tři druhy tvrzení: rezervované (reserved), veřejné (public) a privátní (private) [5].
  - Rezervovaná tvrzení – množina předdefinovaných tvrzení, která nejsou povinná, ale doporučená. Poskytují užitečnou sadu interoperabilních tvrzení. Například: *iss* (issuer), *exp* (expiration), *sub* (subject). Názvy tvrzení jsou pouze tři písmena dlouhá, což přispívá ke kompaktnosti [5].
  - Veřejná tvrzení – definovaná dle vůle toho, kdo používá JWT. Aby se zamezilo kolizím s rezervovanými tvrzeními, měla by být veřejná tvrzení definována v IANA JSON Web token Registry nebo jako URI, které obsahuje kolizím rezistentní jmenný prostor [5].
  - Privátní tvrzení – vlastní tvrzení, vzniklé na sdílení informací mezi stranami, které se dohodly na jejich použití [5].

  Náklad je po vytvoření base64-zakódován a tvoří druhou část JWT.
- Podpis (Signature) – k vytvoření podpisu je zapotřebí vzít zakódovanou hlavičku, zakódovaný náklad, tajný klíč a algoritmus specifikovaný v hlavičce a to vše následně podepsat [5].

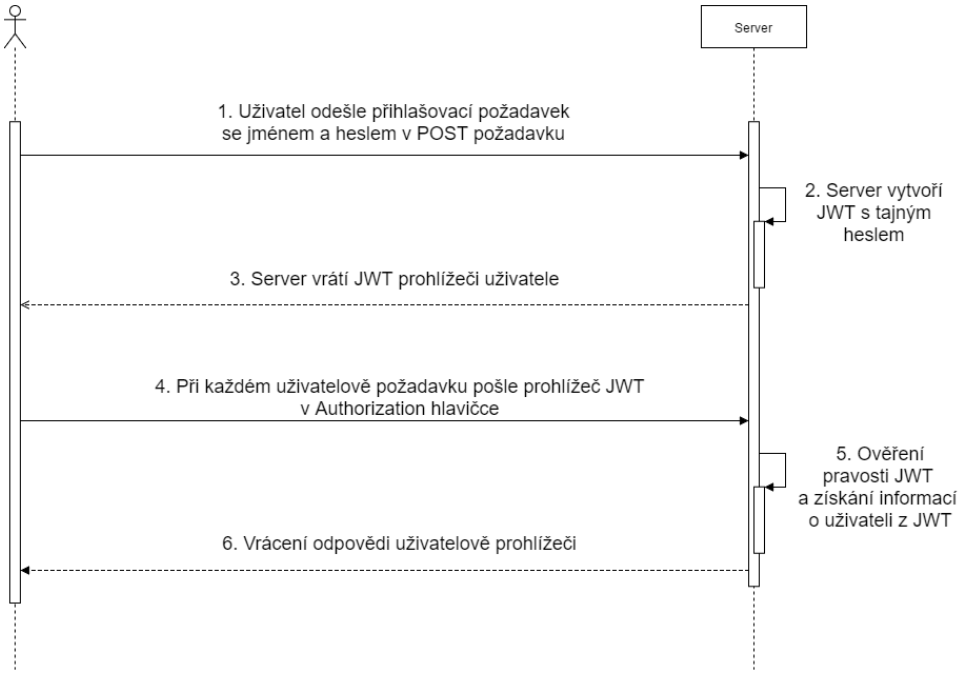
```
HMACSHA256(
  base64UrlEncode(hlavicka) + "." +
  base64UrlEncode(naklad),
  tajne_heslo)
```

Listing 1: JWT tvorba podpisu pomocí HMAC a SHA256

1.3.2 Fungování JWT

Při autentizaci, po úspěšném přihlášení uživatele je namísto vytvoření session a navrácení cookie navrácen JWT, který je lokálně uložen (local storage). Kdykoli chce uživatel přistoupit ke chráněnému zdroji, je JWT posílán spolu s požadavkem typicky jako `Authorization` hlavička s použitím `Bearer` schématu. JWT poskytuje bezestavový autentizační mechanismus, kdy stav uživatele není nikdy uložen v paměti serveru. Chráněné stránky na serveru vždy zkontrolují platnost JWT v autorizační hlavičce a pokud je JWT platný, dovolí uživateli přistoupit na danou stránku [5].

V případě Webgarden SSO bude JWT použit trochu jinak. Nebude posílán s každým požadavkem, bude sloužit pouze k přenosu informací od SSO CAP zpět k poskytovateli služby a bude ukládán do databáze.



Obrázek 1.2: Autentizace pomocí JWT [4]

[4] Převzato z [5]

1.4 OAuth 2.0

OAuth 2.0 je autorizační framework, který umožňuje aplikacím třetích stran získat omezený přístup k HTTP službě. Děje se tak buď jménem *vlastníka služby* (připravením schvalovací interakce mezi vlastníkem služby a HTTP službou) nebo jménem aplikace (*klienta*) třetí strany (povolením získání přístupu jejím jménem). OAuth 2.0 přidává autorizační vrstvu a odděluje roli klienta a vlastníka služby [6].

1.4.1 Tokeny

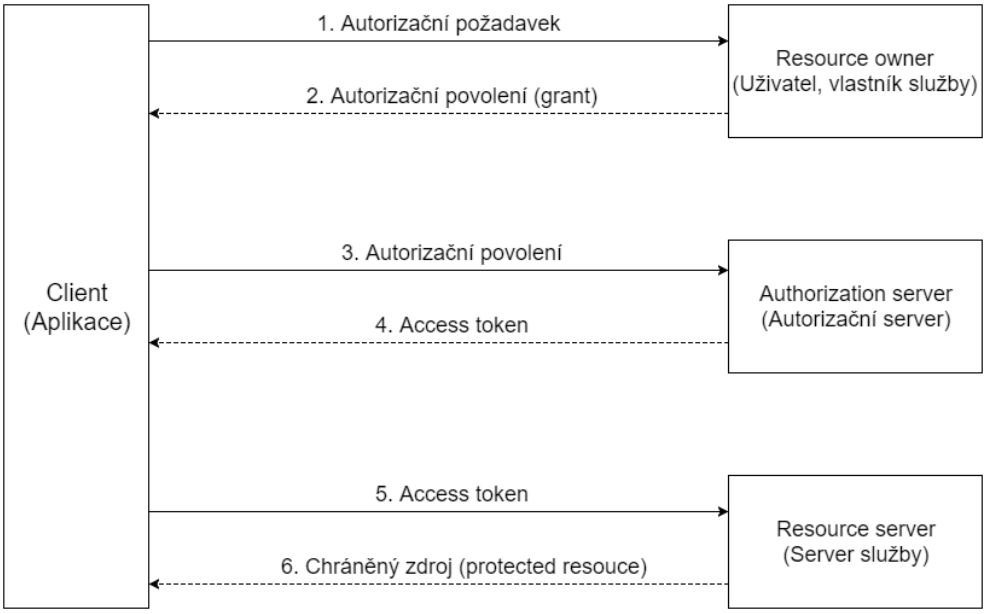
- Access token – používány pro přístup ke chráněnému zdroji. Tvoří abstraktní vrstvu nad přihlašovacími údaji (zabalí přihlašovací údaje jako je jméno a heslo do jednoho řetězce, který *server služby* rozpozná) a obvykle má krátkou životnost. Reprezentuje povolení vydané *klientovi* (specifické rozsahy a doby přístupu vydané vlastníkem služby a vyžadované serverem služby a autorizačním serverem). Může označovat identifikátor použitý k získání autorizačních informací nebo je sám obsahovat [6].

- Refresh token – používán k získání access tokenu. Ve formě řetězce reprezentuje povolení přidělené klientovi vlastníkem služby. Je vytvořen *autorizačním serverem* pro klienta a použit k získání nového access tokenu, když se aktuální access token stane neplatným nebo nebo se zúží jeho rozsah. Určen pouze pro autorizační server, nikdy není posílán na server služby [6].

1.4.2 Typy povolení

Autorizační povolení je pověření (přihlašovací údaj) reprezentující druh autorizace, kterou klient (aplikace) použil vůči vlastníkovi služby. Známým typem povolení je *Authorization code*, který je získán použitím autorizačního serveru jako prostředníka mezi klientem a vlastníkem služby [6].

1.4.3 Abstraktní tok OAuth2.0 protokolu



Obrázek 1.3: Abstraktní tok OAuth2.0 protokolu [5]

1. Klient požaduje autorizaci u vlastníka služby (uživatel). Autorizační požadavek je směrován přímo na vlastníka služby (na obrázku) nebo nepřímo přes autorizační server (plní úlohu prostředníka) [6].

2. Klient získá od vlastníka služby autorizační povolení (pověření reprezentující autorizaci schválenou vlastníkem služby pomocí jednoho ze čtyřech typů povolení) [6].

3. Klient se autentizuje vůči autorizačnímu serveru pomocí autorizačního povolení a vyžádá si access token [6].

4. Autorizační server autentizuje klienta a ověří autorizační povolení. Pokud je povolení platné, vystaví access token [6].

5. Klient si vyžádá chráněný zdroj od serveru služby a autentizuje se access tokenem [6].

6. Server služby ověří access token a pokud je platný, poskytne chráněný zdroj [6].

[5] Převzato z [6]

1.5 OpenID Connect 1.0

OpenID Connect 1.0 je jednoduchá identitní vrstva nad OAuth 2.0 (skloubená identita, autentizace a OAuth 2.0). Umožňuje aplikacím ověřovat identitu koncového uživatele díky autentizaci, kterou obstarává autorizační server. Dále umožňuje aplikacím získat základní informace o koncovém uživateli [7].

1.5.1 Terminologie

- Entity – něco, co má separátní a odlišnou existenci a může být identifikováno v kontextu. Příkladem je koncový uživatel. [7]

- Tvrzení – informace o nějaké entitě [7].

- OpenID Provider (OP) – OAuth 2.0 autorizační server, který je schopný autentizovat koncového uživatele a poskytující tvrzení RP o procesu autentizace a koncovém uživateli [7].

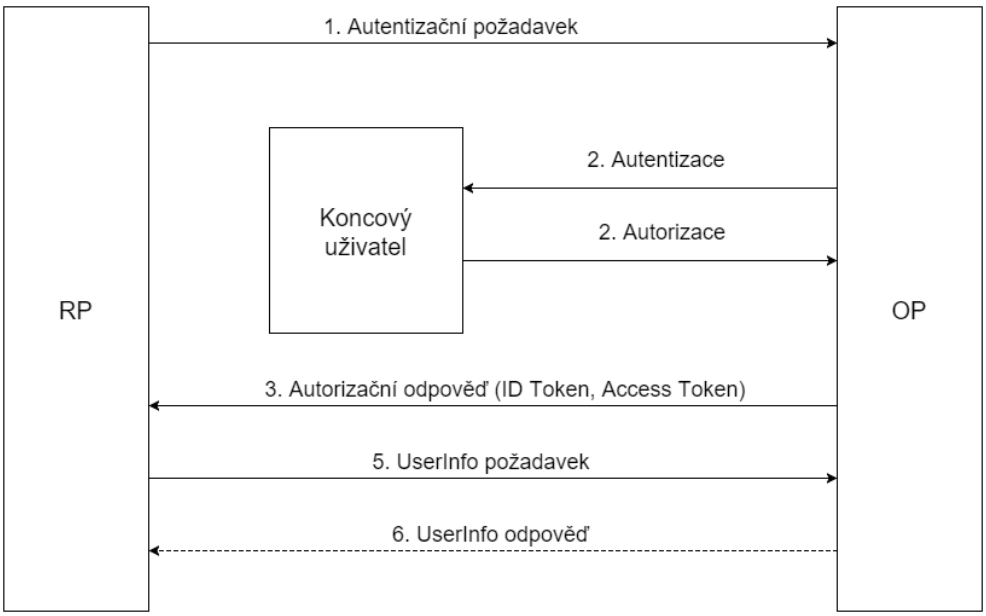
- Relying Party (RP) – OAuth 2.0 klientská aplikace vyžadující autentizaci koncového uživatele a tvrzení od OP [7].

- UserInfo Endpoint – chráněný zdroj, který poskytuje informace o koncovém uživateli, reprezentovanou autorizačním povolením (pokud mu je předložen access token). URL, na které je dostupný musí využívat HTTPS schéma [7].

1.5.2 ID token

ID token tvoří největší rozdíl mezi OpenID Connect 1.0 a OAuth 2.0. ID token je datová struktura ve formě JWT, která umožňuje autentizaci koncového uživatele. Obsahuje tvrzení o autentizaci koncového uživatele [7].

1.5.3 Abstraktní tok OpenID Connect 1.0



Obrázek 1.4: Abstraktní tok OpenID Connect 1.0 protokolu [6]

1. RP pošle autentizační požadavek OP [7].
2. OP autentizuje koncového uživatele a získá autorizaci [7].
3. OP pošle RP odpověď, která obsahuje ID token a obvykle i Access token [7].
4. RP pošle požadavek (včetně Access tokenu) na UserInfo Endpoint [7].
5. UserInfo Endpoint vrací tvrzení o koncovém uživateli [7].

1.5.3.1 Implicit flow

Implicit flow se od obecného abstraktního toku liší v tom, že RP neposílá požadavek na UserInfo EndPoint. Jakmile RP dostane od OP odpověď s ID tokenem, získá z něho informace o subjektu a to mu stačí k tomu, aby subjekt autentizoval [7].

[6] Převzato z [7]

1.6 Same-origin policy

Same-origin policy omezuje interakci dokumentu nebo skriptu načteného z jednoho *originu* (původ, zdroj) s prostředky z jiného originu. Dvě webové stránky mají stejný origin, pokud se shoduje protokol, jméno hosta a port [8].

Například není možné aby dokument získaný z adresy `http://example.com/test.html` mohl číst obsah z dokumentu, do kterého je vložený. K nějakým položkám objektu `window` v rodičovském dokumentu (např. `window.location`, `window.parent`) lze ovšem přistupovat i z vloženého dokumentu.

Prohlížeče obecně aplikují Same-origin policy na síťové požadavky, které zabraňují aplikacím na straně klienta běžících na jednom originu získávat data z aplikace na jiném originu a také omezují nezabezpečené HTTP požadavky, které mohou být automaticky odeslány na destinace, které nemají stejný origin [8].

1.7 CORS

Cross Origin Resourse Sharing (CORS) je mechanismus, který umožňuje provádět cross-origin požadavky na klientské straně [8].

CORS funguje díky přidání nových HTTP hlaviček, které umožňují serverům specifikovat množinu originů, které mají povoleno z nich číst informace skrz prohlížeč. Navíc pro HTTP metodu, která může mít vedlejší efekt na uživatelova data specifikace vyžaduje, aby prohlížeč *preflightoval* (průzkumný požadavek před opravdovým požadavkem) požadavek, získal podporované metody ze serveru pomocí HTTP OPTIONS metody a po schválení ze serveru poslal aktuální požadavek pomocí aktuální HTTP metody [9].

1.7.1 Požadavek s pověřením

Nejzajímavější možností je vytvoření požadavku s pověřením, který pracuje s HTTP Cookies a HTTP Authentication informací. Základně prohlížeč v cross-origin *XMLHttpRequest* (API pro načítání zdrojů [10]) požadavku neposílá pověřovací informace (credentials). Aby prohlížeč poslal pověřovací informace, je nutné nastavit parametr `withCredentials` na `true` před odesláním XMLHttpRequest. Nyní je možné nastavovat na serveru cookies.

```
var invocation = new XMLHttpRequest();
var url = 'http://bar.other/resources/credentialed-content/';

function callOtherDomain(){
  if(invocation) {
    invocation.open('GET', url, true);
    invocation.withCredentials = true;
    invocation.onreadystatechange = handler;
    invocation.send();
  }
}
```

Listing 2: Ukázka XMLHttpRequest `CORS` požadavku s pověřením [7]

Pokud server vrátí hlavičku `Access-Control-Allow-Origin: *` v odpovědi, znamená to, že zdroj může být požadován z jakékoliv domény. V případě, že by si vlastník služby `bar.other` přál omezit přístup ke zdroji pouze pro `foo.example.com`, poslal by v odpovědi hlavičku `Access-Control-Allow-Origin: http://foo.example` a žádná jiná doména než `foo.example.com` by nemohla přistoupit ke zdroji cross-origin způsobem.

`Access-Control-Allow-Origin` hlavička by měla obsahovat hodnotu poslanou v požadavku v `Origin` hlavičce [9].

Příklad komunikace je k nahlédnutí v příloze C.4.

1.7.2 Některé HTTP hlavičky týkající se CORS

- `Access-Control-Allow-Origin: <origin> | *` – parametr origin specifikuje URI, které může přistupovat ke zdroji [9].
- `Access-Control-Allow-Credentials` – určuje, jestli může být odpověď na požadavek vystavena (exposed) [9].
- `Origin: <origin>` – origin je URI zastupující server, ze kterého byl požadavek odeslán. Obsahuje pouze název serveru [9].

1.8 Možnosti realizace SSO

1.8.1 NoScript metoda

V případě, že má uživatel zakázaný *JavaScript* (interpretovaný programovací jazyk s objektově orientovanými možnostmi [11]) nebo jeho prohlížeč nepodporuje JavaScript, lze umožnit SSO pomocí HTTP přesměrování na CAP. Z hlediska uživatelské zkušenosti to není nejlepší metoda. Při přesměrování

[7] Převzato z [9]

na stránku CAP může být uživatel zmaten tím, že je to jiná stránka, než stránka služby, ke které chtěl přistoupit.

1.8.2 Iframe metoda

Iframe je HTML *tag* (značka), který umožní do HTML dokumentu vložit obsah dalšího HTML dokumentu. S obsahem iframe nelze zvenku manipulovat, protože podléhá Same-origin policy. Při autentizaci pomocí iframe směřuje jeho zdroj na CAP. Autentizace se stejně jako u NoScript metody řeší jinde (u CAP), ale uživatel to téměř nemá šanci rozpoznat, jelikož zůstává stále na jedné stránce.

1.8.3 AJAX + CORS metoda

AJAX je soubor technologií, které navzájem efektivně spolupracují. AJAX zahrnuje prezentaci pomocí XHTML a CSS, dynamické zobrazování a interakci pomocí Document Object Model (DOM), výměnu a manipulaci s daty pomocí XML a XSLT, asynchronní získávání dat pomocí XMLHttpRequest a JavaScript, který spojí vše dohromady [12].

Pro AJAX platí Same-origin policy. Lze ho ale parametrizovat tak, že může požadovat zdroj z jiného serveru (zmíněno v 1.7.1).

Oproti Iframe metodě poskytuje AJAX + CORS metoda výhodu snazší manipulace se stránkou podle přijaté odpovědi.

1.8.4 Přihlášení pomocí třetích stran

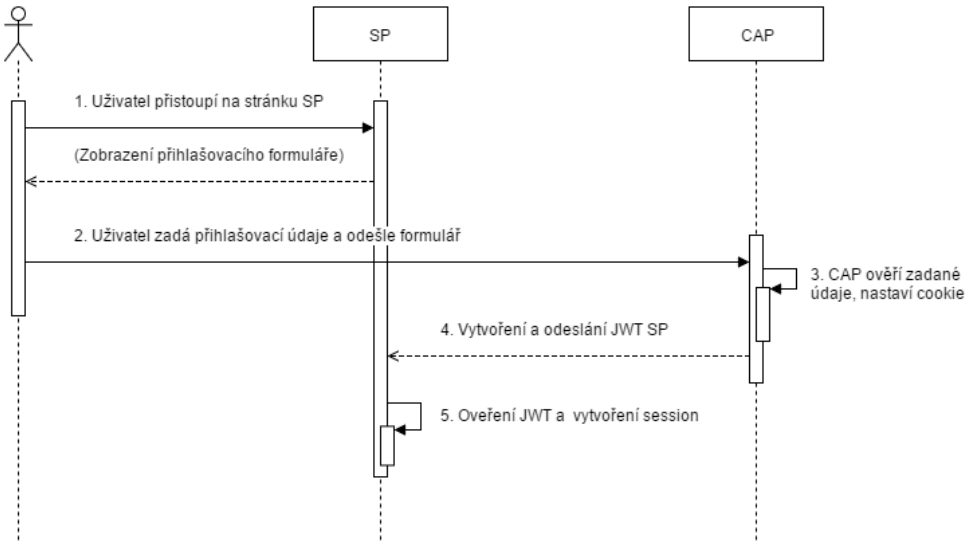
Přihlášení pomocí třetí strany funguje na principu NoScript metody. Podporované služby třetích stran jsou

- Přihlášení pomocí Google
- Přihlášení pomocí Facebooku

Hlavní potřebný údaj je emailová adresa, díky které by si uživatel mohl obnovit přístup do služby Webgarden i po tom, co by smazal svůj Google nebo Facebook účet.

1.9 Vybrané řešení

Nejlepším řešením byl shledán OpenID Connect 1.0 díky své jednoduchosti a využívání JWT. Není třeba využít všechny možnosti, který OpenID Connect 1.0 nabízí a tak pro implementaci SSO na portálu Webgarden vznikne jeho mutace, která bude provádět autentizaci podobně jako Implicit flow. JWT bude použit jako formát pro transport informací.



Obrázek 1.5: Obecný průběh přihlášení přes SSO

1. Uživatel přistoupí na stránku SP, kde mu je zobrazen přihlašovací formulář.
2. Uživatel zadá přihlašovací údaje a odešle formulář.
3. CAP ověří zadané údaje, nastaví cookie reprezentující proběhlou autentizaci uživatele
4. CAP vygeneruje JWT, který podepíše, zabalí do něj informace potřebné k autentizaci uživatele (např. email nebo uživatelovo ID) a pošle ho SP.
5. SP ověří digitální podpis JWT a pokud je v pořádku, vyhledá v databázi uživatele podle informací poskytnutých v JWT a vytvoří pro uživatele session.

1.10 Požadavky na aplikaci

1.10.1 Funkční požadavky

- Přihlášení dosud nepřihlášeného uživatele – uživateli bude umožněno přihlásit se do aplikace pomocí `prvku pro přihlášení`. Uživatel bude přihlášen na CAP a následně poslán zpět do služby, odkud se přihlašoval.
- Přihlášení již přihlášeného uživatele – pokud se uživatel již přihlásil a přistoupí ke službě, kde přihlášen není, bude zobrazen mu `prvek pro pokračování v přihlášení`, který poskytne možnost přihlásit se jako již přihlášený uživatel (přihlášený na CAP) nebo jako někdo jiný.
- Přihlášení pomocí služby třetí strany
- Odhlášení uživatele – aplikace uživateli umožní odhlásit se ze služby, kde se přihlásil a z CAP.
- Přepnutí účtu – aplikace uživateli umožní přepnout účet pokud si bude přát přihlásit se jako jiný uživatel.

1.10.2 Nefunkční požadavky

- PHP5 jako programovací jazyk na straně serveru
- Funkčnost i bez podpory JavaScriptu

Kapitola 2

Realizace

Tato kapitola pojednává o realizaci kostry aplikace. *Kostra aplikace*, dále též označována jako *demo aplikace*, je minimální funkční verze SSO. Při integraci do systému Webgarden, která je popsána v kapitole 3, se vycházelo z kostry aplikace.

2.1 Návrh architektury aplikace

Celá aplikace, dále též nazývána jako *Modul SSO*, je funkčně rozdělena na dvě části. První část, *Client*, připravuje přihlášení a zpracovává JWT. Druhá část, *EndPoint*, zpracovává požadavky na přihlášení a odpovídá na ně.

Client nepřipravuje přihlášení sám, má na to pomocníky, *LoginHelpery*. Každý z pomocníků má svůj *Renderer*, který obstarává předání dat do šablony.

EndPoint obsahuje *LoginMehod*, která za něj vyřídí požadavek na přihlášení. Každá přihlašovací metoda má taktéž svůj Renderer.

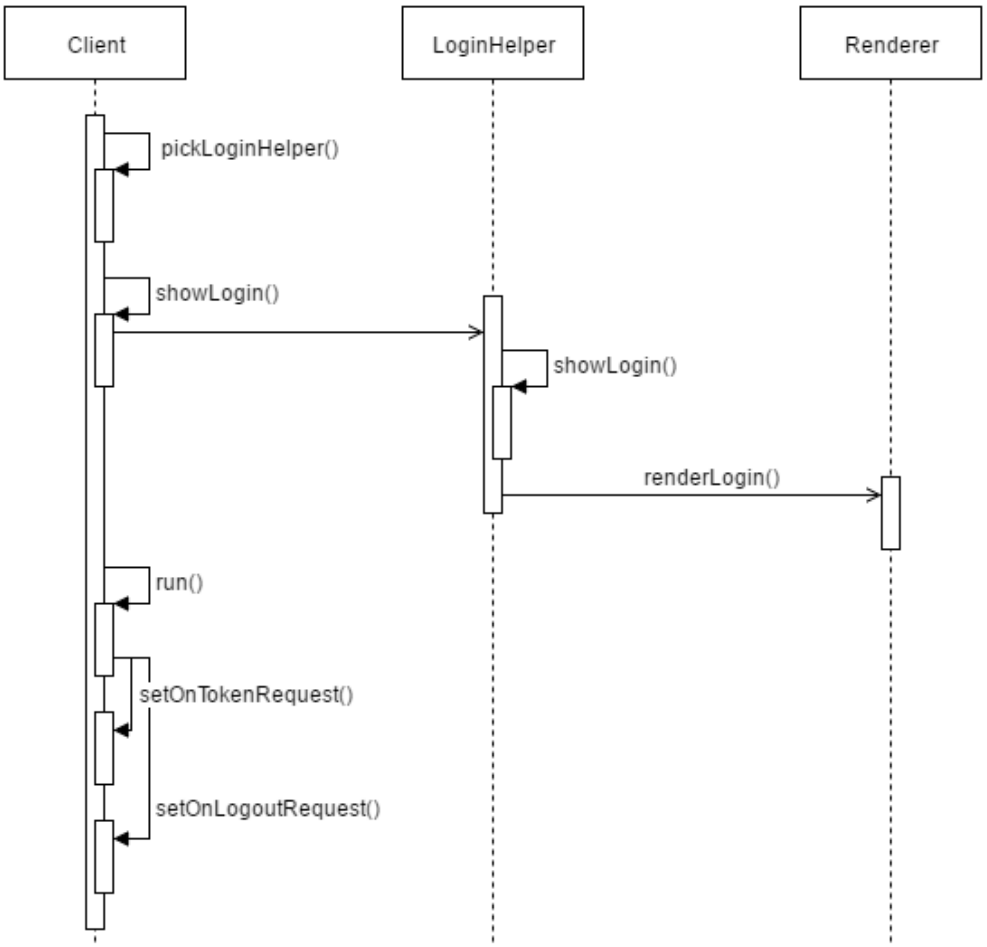
Aplikace je nastavena podle konfiguračního souboru, ve kterém je možné nastavit URL CAP, expiraci SSO cookie, seznam podporovaných prohlížečů a kódy pro přihlášení přes aplikace třetích stran.

Doménový model aplikace lze nalézt v příloze C.1, C.2 a C.3.

2.1.1 Životní cyklus aplikace

Pro správnou funkčnost aplikace je nutné inicializovat třídu `Client` a `EndPoint`. Třídu `Client` využije SP, třídu `EndPoint` CAP.

2.1.2 Client

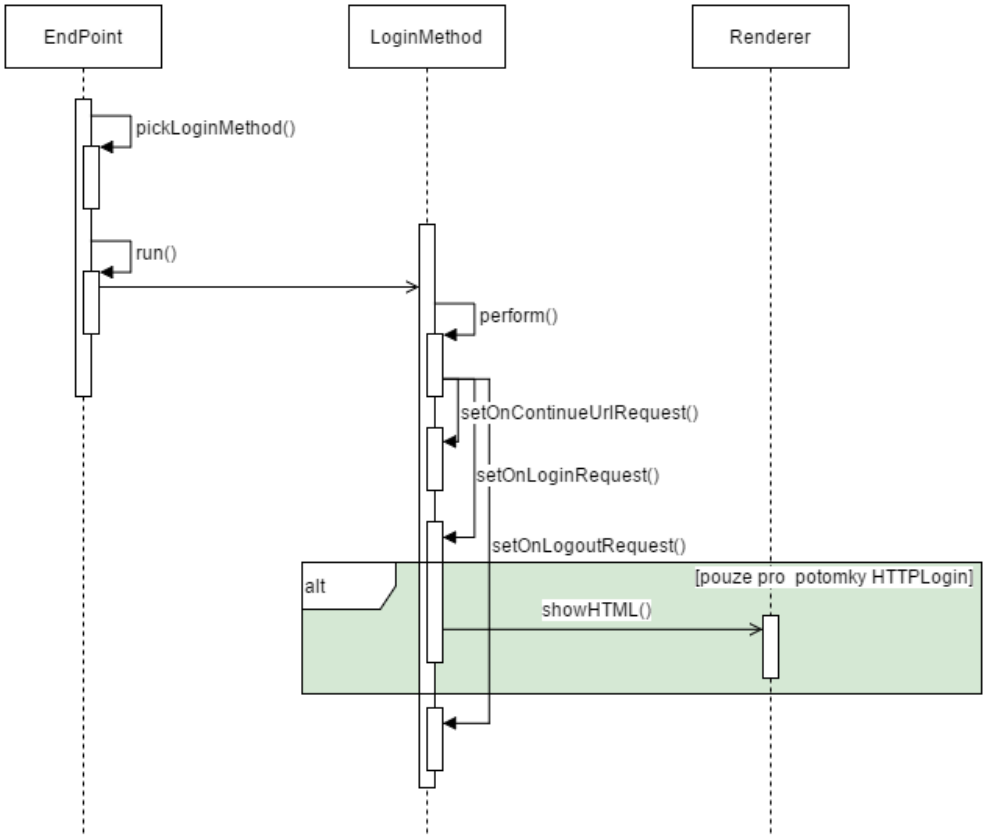


Obrázek 2.1: Příprava Clienta

1. Volba přihlašovacího pomocníka `LoginHelper` podle vlastností prohlížeče.
2. Zavolání metody na zobrazení `prvku pro přihlášení`.
3. `LoginHelper` renderer předá data šablony, která `prvek pro přihlášení` vykreslí.

4. Příprava na zpracování požadavků v metodě `run()`.

2.1.3 EndPoint



Obrázek 2.2: Příprava EndPointu

1. Volba přihlašovací metody `LoginMethod` podle čísla metody obdrženého v HTTP požadavku.

2. Příprava na zpracování požadavků v metodě `run()`.

3. Obsluha požadavků ve třídě `LoginMethod` metodou `perform()`.

   - Pokud je přihlašovací metoda potomkem třídy `HTTPLogin`, tak navíc ještě zobrazí `prvek pro přihlášení` nebo `prvek pokračovat jako přihlášený uživatel`.

```php
<?php
use ModuleSSO\Client;
use Symfony\Component\HttpFoundation\Request;
use ModuleSSO\Client\LoginHelper\Renderer\HTML\HTMLRenderer;

$request = Request::createFromGlobals();
$renderer = new HTMLRendererFactory();

$client = new Client($request, $renderer);
$client->pickLoginHelper();
$client->showLogin();
$client->run();
```

Listing 3: Ukázka kódu pro přípravu třídy Client

```php
<?php
use ModuleSSO\EndPoint;
use Symfony\Component\HttpFoundation\Request;
use ModuleSSO\EndPoint\LoginMethod\Renderer\HTML\HTMLRenderer;

$request = Request::createFromGlobals();
$renderer = new HTMLRendererFactory();

$endPoint = new EndPoint($request, $renderer);
$endPoint->pickLoginMethod();
$endPoint->run();
```

Listing 4: Ukázka kódu pro přípravu třídy EndPoint

2.2 Modul SSO

Funkcionalita Modulu SSO je obalena do abstraktní třídy `ModuleSSO`, která obsahuje konstanty pro názvy parametrů v HTTP požadavcích a abstraktní metodu `run()`.

2.2.1 Client

Hlavním úkolem třídy `Client` je zpracovávat JWT poskytnutý v URL a přihlašovat uživatele a odhlašovat uživatele. `Client` je zodpovědný za správné vybrání `LoginHelperu` pomocí metody `pickLoginHelper()` a implementuje metodu `run()`, ve které se připraví na přijímání požadavku s JWT a požadavku na odhlášení.

```php
<?php
//class Client
public function run()
{
    $this->setOntokenRequest();
    $this->setOnLogoutRequest();
}
```

Listing 5: Metoda run v třídě Client

Metoda `pickLoginHelper()` zvolí vhodnou přihlašovací metodu podle prohlížeče, který uživatel používá. Může se stát, že uživatel nebude disponovat prohlížečem, který umožňuje přihlášení pomocí AJAX + CORS metody. V tom případě bude zvolena Iframe metoda. Pokud nebude v prohlížeči zapnut JavaScript, bude bude zvolena NoScript metoda.

2.2.2 EndPoint

`EndPoint` obstarává správnou volbu `LoginMethod` v metodě `pickLoginMethod()` a v metodě `run()` deleguje přihlášení na metodu `perform()` ve zvolené `LoginMethod`.

```php
<?php
//class EndPoint
public function run()
{
     $this->loginMethod->perform();
}
```

Listing 6: Metoda run v třídě EndPoint

Metoda `pickLoginMethod()` zvolí vhodnou přihlašovací metodu podle přijatého parametru v HTTP požadavku. Pokud je číslo přihlašovací metody předané v požadavku nesprávné nebo chybí, je zvoleno *Přímé přihlášení u CAP* (popsáno v 2.7.5).

2.3 JWT

Generování JWT zajišťuje stejnojmenná třída. JWT je vždy vytvořen pro doménu, ze které přišel požadavek na přihlášení. Obsahuje ID uživatele, který bude přihlášen a čas expirace. Po úspěšném přihlášení je JWT zneplatněn a nejde znovu použít.

2.4 LoginHelper

K odeslání přihlašovacího požadavku je zapotřebí `prvek pro přihlášení`. `LoginHelper` třídy se starají o předání správných dat pro vykreslení `prvku pro přihlášení`. `Prvek pro přihlášení` se prováže s přihlašovací metodou v příslušném rendereru pomocí čísla (`LoginMethod::METHOD_NUMBER`) přihlašovací metody.

2.5 LoginMethod

`LoginMethod` obsluhuje přihlášení a odhlášení. V metodě `perform()` jsou volány metody pro obsluhu HTTP požadavku. Potomci třídy `LoginMethod` si mohou upravit funkčnost obslužných metod podle potřeby.

```php
<?php
//class LoginMethod ...
public function perform()
{
    $this->setOnContinueUrlRequest();
    $this->setOnLoginRequest();
    $this->setOnLogoutRequest();
}
```

Listing 7: LoginMethod peform

- `setOnContinueUrlRequest` – provádí validaci parametru `continue` v HTTP požadavku. Parametr `continue` určuje URL, kam bude uživatel přesměrován po přihlášení. V případě, že parametr `continue` v HTTP požadavku chybí, prozkoumá se session, případně HTTP Referer. Pokud `continue` obsahuje špatně naformátované URL, neodpovídá žádné doméně registrované přes systém Webgarden nebo úplně chybí, je nastaveno na výchozí hodnotu (URL CAP nastavená v konfiguračním souboru aplikace).
- `setOnLoginRequest` – implementována potomky `LoginMethod` třídy. Kontroluje přítomnost `email` a `password` parametru v HTTP požadavku a přihlašuje uživatele.
- `setOnLoginRequest` – čeká na parametr `logout`, ukončí session, odstraní SSO cookie a přesměruje zpět.

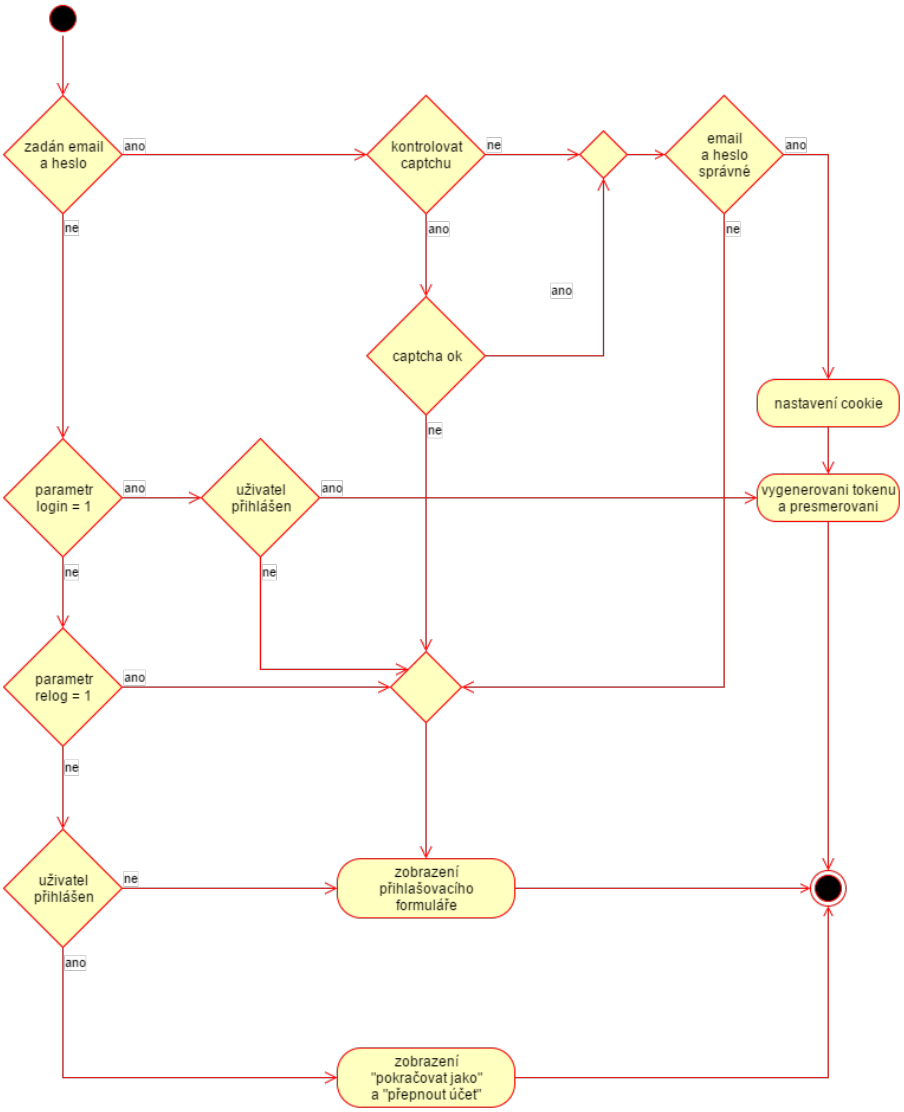
2.5.1 Metoda getUserFromCookie

Pokud se uživatel již přihlásil, bude na CAP nastavena SSO cookie, podle které metoda najde uživatele v databázi a vrátí ho. Pokud je SSO cookie neplatná

nebo se uživatel ještě nepřihlásil, vrátí metoda `null`.

2.5.2 HTTPLogin

Třída `HTTPLogin` implementuje `setOnLoginRequest()` pro `DirectLogin`, `NoScriptLogin` a `IframeLogin`. Chování metody `setOnLoginRequest()` nejlépe vystihuje následující diagram.



Obrázek 2.3: Logika metody setOnLoginRequest ve třídě HTTPLogin

2.5.3 CORSLogin

Hlavičku `Access-Control-Allow-Origin` v odpovědi je nutno nastavit v metodách pracujících s HTTP požadavkem. Hlavičky se nastavují podle `Origin` hlavičky v požadavku. Pokud je doména hlavičky `Origin` stejná jako doména hlavičky `Host`, `Origin` se na server neodešle.

```php
<?php
//class CORSLogin
protected function getOrigin()
{
    $origin = $this->request->server->get('HTTP_ORIGIN');
    $host = $this->request->server->get('HTTP_HOST');
    return $origin ? $origin :
        ($host ? 'http://' . str_replace('www.', '', $host) :
        null);
}

protected function originIsValid($origin)
{
    $parsed = parse_url($origin);
    if (isset($parsed['host'])
        && $this->isInWhiteList($parsed['host'])
    ){
        return true;
    } else {
        return false;
    }
}

public function setOnCheckCookieRequest()
{
    $origin = $this->getOrigin();
    if ($this->originIsValid($origin)) {
        header('Access-Control-Allow-Origin: ' . $origin);
        header('Access-Control-Allow-Credentials: true');
        header('Content-Type: application/json');
        /* some stuff here */
    } else {
        /* another stuff */
    }
}
```

Listing 8: CORSLogin validace Origin hlavičky

- `setOnLoginRequest` – obsluhuje přihlášení uživatele. Vrací JSON, který je dále zpracován pomocí JavaScriptu. Zajímá se pouze o parametry `email` a `password` v HTTP požadavku.
- `setOnCheckCookieRequest` – slouží ke zjištění stavu přihlášení uživatele. Pokud je uživatel přihlášen na CAP, vrátí JSON obsahující JWT k přihlášení uživatele, jinak vrátí JSON obsahující zprávu o tom, co je špatně.
- `perform` – podle parametru v HTTP požadavku rozhodne, jestli zavolá `setOnLoginRequest` nebo `setOnCheckCookieRequest`.

2.5.4 ThirdPartyLogin

Stará se o vytvoření správného URL pro přihlášení pomocí třetí strany, zpracování odpovědi od API třetí strany a přihlášení uživatele.

- `setOnLoginRequest` – vygeneruje přihlašovací URL a přesměruje na něj
- `setOnCodeRequest` – čeká na `code` parametr od aplikace třetí strany. Po přijetí získá uživatelovo ID v aplikaci třetí strany a spojí ho s ID ve službě Webgarden.
- `redirectWithtoken` – páruje uživatele z aplikace třetí strany na uživatele služby Webgarden. Přihlašuje uživatele, nastavuje SSO cookie, generuje JWT a přesměrovává.

2.6 Renderer

Třída, která předává data dál do šablony. O výsledné vykreslení se stará už samotná šablona.

Každý `LoginHelper` i každá `LoginMethod` mají svůj renderer. Renderer pro `LoginHelper` je potomkem `RendererFactory`, což je abstraktní třída, která implementuje rozhraní `IRenderer` pro `LoginHelper`. Obdobně tomu je u `LoginMethod`.

```php
<?php
interface IRenderer
{
    public function renderLogin($params = array());

    public function getRenderer(LoginHelper $loginHelper);
}
```

Listing 9: LoginHelper renderer interface

```php
<?php
interface IRenderer
{
    public function renderLoginForm($params = array());

    public function renderContinueOrRelog($params = array());

    public function getRendererer(LoginMethod $loginMethod);
}
```

Listing 10: LoginMethod renderer interface

2.6.1 LoginHelper renderer

`LoginHelper` renderer je potomkem `RendererFactory`. Obsahuje metodu `renderLogin()`, která předává do šablony data potřebná k vykreslení `přihlašovacího prvku`. `Přihlašovací prvek` je závislý na zvolené přihlašovací metodě. V případě NoScript, Google a Facebook přihlašovací metody je přihlašovacím prvkem odkaz, jinak formulář.

2.6.2 LoginMethod renderer

`LoginMethod` renderer je potomkem `RendererFactory`. V tomto případě zodpovídá za předání dat pro vykreslení přihlašovacího formuláře a `prvku pro pokračování jako přihlášený uživatel`.

2.6.3 Metoda getRenderer

Konkrétní renderer je vybrán v metodě `getRenderer()` podle typu instance objektu předaného v parametru a následně vrácen. Navrácený renderer je navázán na `LoginHeper`, resp. `LoginMethod`, pomocí metody `setRenderer()`.

V kostře aplikace byla použita `HTMLRendererFactory`, která pomocí PHP funkce `echo` rovnou předává data na výstup.

2.7 Přihlašovací metody detailně

Bylo implementováno 6 možných metod přihlášení.

- Přihlášení u SP
  - Přímé přihlášení
    - Přihlášení pomocí AJAX + CORS metody
    - Přihlášení pomocí Iframe metody
    - Přihlášení pomocí NoScript metody
  - Přihlášení pomocí metody třetí strany
    - Facebook
    - Google
- Přímé přihlášení u CAP

Obecně byl průběh SSO popsán v sekci 1.9. Každá metoda modifikuje některé kroky z obecného průběhu SSO.

2.7.1 NoScript metoda

Krok 2. Uživateli se jako přihlašovací prvek zobrazí tlačítko, které ho po kliknutí přesměruje k CAP. Uživatel uvidí přihlašovací formulář, vyplní ho a odešle.

Krok 4. Uživatel je vzápětí přesměrován zpět k SP spolu s JWT.

2.7.2 Iframe metoda

Krok 4. CAP přesměruje uživatele pomocí JavaScriptu zpět k SP spolu s JWT s informacemi o uživateli.

Při implementace Iframe metody vyvstal problém se zobrazením přihlašovacího formuláře tak, aby zapadl do stylu webu, na kterém byl zobrazen. V kostře aplikace je problém řešen pomocí přidání, pro každou doménu specifického, CSS souboru do iframe.

2.7.3 AJAX + CORS metoda

Krok 2. Uživateli se jako přihlašovací prvek zobrazí formulář. Akce odeslání formuláře je odchycena a ověření přihlašovacích údajů je provedeno pomocí AJAXu směřovaného na CAP.

Krok 4. CAP pošle JSON odpověď SP s JWT a informacemi o uživateli.

2.7.4 Metody přihlášení pomocí třetích stran

Metoda přihlášení pomocí Facebooku a Googlu využívají stejnou implementační logiku, liší se pouze získávání informací o uživateli a to kvůli odlišnosti jednotlivých API.

Aby bylo možné přihlásit se pomocí služby třetí strany, musí ve službě třetí strany existovat aplikace, která umožní získat potřebné informace o uživateli. Aplikace byly vytvořeny u služeb třetích stran a požadují email a veřejně dostupné informace.

Přihlášení do Webgarden SSO probíhá pomocí HTTP přesměrování, kterých je větší množství kvůli autorizaci u služby třetí strany.

Krok 2. Uživateli se jako přihlašovací prvek zobrazí tlačítko, které ho po kliknutí přesměruje na přihlašovací URL aplikace třetí strany.

Krok 3.
- I. Pokud uživatel ve službě třetí strany dosud neautorizoval aplikaci Webgarden SSO, bude dotázán, zda-li tak chce učinit a zobrazí se mu seznam práv, která aplikace vyžaduje. Pokud uživatel aplikaci neautorizuje, pokračuje scénář krokem 1.
- II. Po autorizování Webgarden SSO aplikace je u služby třetí strany vygenerován access token.
- III. Uživatel je přesměrován k CAP spolu s access tokenem. Access token umožní CAP získat email uživatele a jeho ID přes API služby třetí strany.

Krok 4. CAP vygeneruje JWT a přesměruje uživatele k SP spolu s JWT s informacemi o uživateli.

2.7.5 Přímé přihlášení u CAP

Přímé přihlášení u CAP realizuje třída `DirectLogin`. Metoda `setOnLoginRequest()` dědí od rodičovské třídy `HTTPLogin`, ale upravuje metodu `generatetokenAndRedirect()`, která nepřesměrovává k SP, ale k CAP a neposílá v HTTP odpovědi JWT.

Krok 4. Negeneruje se JWT

Krok 5. Nepřesměrovává se k SP, jelikož se neví, od kterého SP se přišlo

2.8 Bezpečnost

Bezpečnost přihlašovacích údajů uživatele a bezpečnost přihlášení jako takového je velmi důležitá. Z těchto důvodů je SSO cookie otisk několika parametrů, JWT digitálně podepsaný a přihlašovací požadavek prováděn přes HTTP POST.

2.8.1 Přihlášení přes HTTP POST

HTTP GET požadavek přenáší parametry v URL, což znamená že budou uloženy v logu na serveru, HTTP Referer a v historii prohlížeče. Navíc lze data před odesláním lehce změnit v adresním řádku prohlížeče. Aby se zamezilo tomuto nechtěnému chování, provádí se odesílání přihlašovacího formuláře pomocí HTTP POST požadavku, který přenáší parametry ve svém těle a nevystavuje je do URL.

2.8.2 SSO cookie

SSO cookie je spárovaná s uživatelem, pomocí sloupce `cookie` v tabulce `users` a mění se při každém požadavku na získání uživatele z cookie nebo přihlášení uživatele. Její hodnota je tvořena pomocí uživatelova ID a celého názvu prohlížeče (včetně verze, atd.), který používá. Pokud není SSO cookie shodná s hodnotou uloženou v databázi, musí se uživatel přihlásit znovu, čímž se nastaví opět správná SSO cookie.

SSO cookie je nastavena jako *HTTP-only* na CAP, což znamená že k ní nelze přistupovat přes DOM pomocí JavaScriptu [13].

2.8.3 JWT

JWT vygenerovaný CAP je digitálně podepsaný, má platnost 60 vteřin a SP pokaždé ověřuje integritu RSA podpisu. JWT je dále ukládán do databáze a spárován s doménou, ze které přišel požadavek na přihlášení (pro jinou doménu nelze JWT použít).

RSA je asymetrická šifra, která je vhodná pro podepisování i šifrování. *Asymetrické šifry* mají dva odlišné klíče, soukromý a veřejný. Veřejný klíč je oznámen druhé straně (v případě SSO je to SP) a soukromý si držitel (IdP) pečlivě uschová. Dále platí, že zprávy zašifrované veřejným klíčem lze rozšifrovat pouze příslušným klíčem soukromým a naopak [14]. Ukázku tvorby digitálního podpisu lze nalézt v příloze C.5.

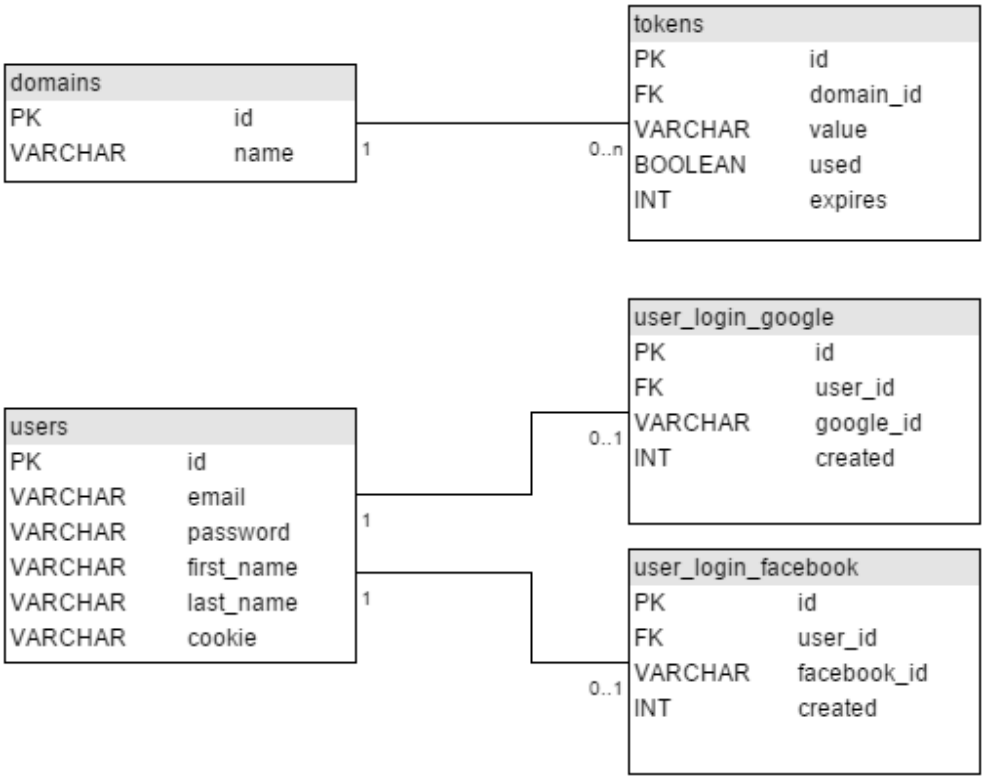
Integrita digitální podpisu je ověřována ve třídě `Client` v metodě `setOnTokenRequest()`.

```php
<?php
if($urlToken = $this->request->query->get(\ModuleSSO::TOKEN_KEY)) {
    try {
        $token = (new Parser())->parse((string) $urlToken);
        $signer = new Sha256();
        $pk = new Key($this->publicKey);

        //check if token is signed and not expired
        if($token->verify($signer, $pk)
            && $token->getClaim('exp') > time()
        ) {
            //verified, login user
        } else {
            //login failed
        }
    } catch (\Exception $e) {
        //login failed
    }
}
```

Listing 11: Ukázka ověření integrity JWT

2.9 Návrh architektury databáze



Obrázek 2.4: Návrh architektury databáze

Schéma 2.4 popisuje databázi nezbytnou pro fungování kostry aplikace Webgarden SSO. Tabulky je možné libovolně rozšířit, případně sloučit podle potřeb implementace.

2.10 Závěr

Pro testovací a ukázkové účely byla vytvořena demo aplikace Webgarden SSO, dostupná na https://github.com/wg-cepi/sso. Aplikace umožňuje:

- Přihlásit se na 2 klientské weby pomocí CAP
- Zvolit si metodu, kterou se chce uživatel přihlásit
  - NoScript
  - Iframe
  - AJAX + CORS
  - Facebook
  - Google
- Přihlásit se přes CAP
- Zobrazit základní informace o uživateli
- Vytvořit uživatele na CAP
- Odhlásit se lokálně z určitého webu
- Odhlásit se z CAP

Manuál ke zprovoznění demo aplikace lze nalézt v příloze D.

Kapitola 3

Integrace do systému Webgarden

3.1 Databáze

Při implementaci do systému Webgarden bylo třeba vytvořit následující tabulky:

- `user_login_facebook`
- `user_login_google`
- `tokens`

Nebylo nutné vytvářet všechny tabulky podle návrhu 2.4, protože některé tabulky systému Webgarden již existovaly a stačilo je poupravit. Tabulka `webgarden.users` byla rozšířena o sloupec `cookie`. Původní tabulka `domains` byla zastoupena tabulkou `webgarden.user_websites`.

Komunikaci s databází v kostře aplikace zajišťovalo PHP rozšíření PDO, které bylo nahrazeno ORM vrstvou Webgarden používanou v celém Webgarden systému.

3.2 Iframe metoda

Pro rozlišení místa, odkud je iframe zobrazován vznik parametr `realm`. Nabývá hodnoty `client` pro weby SP, `zone` pro web CAP a `service` pro web Webgarden helpdesku. V šabloně se pak podle tohoto parametru rozhodne, jak bude obsah iframe vypadat.

3.2.1 Problém s CSS

Správná vizuální podoba přihlašovacího formuláře byla zajištěna nahráním stylů webu do `<style>` tagu v iframe. Pokud by se soubor se styly nahrával pomocí tagu `<link>`, vyvolal by se zbytečně další HTTP požadavek.

3.2.2 Problém se změnou velikosti

Na iframe se vztahuje Same-origin policy, takže s ním není možné manipulovat zvenčí. Problém nastává při proměnlivé velikosti obsahu v iframe. Například `prvek pro přihlášení` zabere více místa než `prvek pro pokračování jako přihlášený uživatel`. Řešením problému se změnou velikostí se staly *postMessage*.

Metoda `window.postMessage` umožňuje bezpečnou cross-origin komunikaci [15]. Na straně odesílatele se stanoví, pro kterou doménu je zpráva určena a co bude posláno. V ukázce je zpráva odesílána z iframe a proměnná `parent` je reference na rodiče.

```
1 //sending side
2 parent.postMessage(
3     "Hello there!",
4     "http://example.webgarden.cz"
5 );
```

Listing 12: Ukázka odeslání postMessage

Na straně příjemce je zapotřebí zkontrolovat správný původ zprávy.

```
//receiving side

// https://davidwalsh.name/window-iframe
// Create IE + others compatible event handler
var eventMethod = window.addEventListener ?
    "addEventListener" : "attachEvent";
var eventer = window[eventMethod];
var messageEvent = eventMethod == "attachEvent" ?
    "onmessage" : "message";

eventer(messageEvent,function(event) {
    if (event.origin === "http://www.webgarden.cz") {
     console.log(event.data);
  }
}
```

Listing 13: Ukázka zpracování přijaté postMessage

Ve výsledku stačí poslat rodiči z iframe zprávu o tom, jakou má aktuálně velikost a rodič podle toho může upravit prvek, ve kterém je iframe vložený.

3.3 XSLT

XSLT je jazyk pro transformaci XML dokumentů do jiných XML dokumentů a je určen pro použití v rámci *XSL* (jazyk na vyjádření stylů pro XML) [16].

Webgarden používá XSLT k transformaci XML dokumentů do HTML, které je zobrazeno uživateli.

Vznikla společná šablona stylů `ViewSSO.xslt`, která obsahuje šablony pro zobrazení `prvku pro přihlášení` a `prvku pokračovat jako přihlášený uživatel`. Šablona `ViewSSO.xslt` je v jiných XSL šablonách importována, čímž je zajištěna konzistence zobrazení SSO prvků ve všech systémech Webgarden.

3.3.1 XMLRenderer

Jelikož Webgarden používá XSL šablony pro transformaci XML dokumentů, bylo proto nutné implementovat renderer, který obdrží XML `DomNode` a přidá do něj vše potřebné pro vykreslení SSO prvků.

```
1  <?php
2  namespace ModuleSSO\EndPoint\LoginMethod\Renderer\XML;
3  class XMLRenderer implements IRenderer
4  {
5      //passed through \DOMNode in constructor
6      protected $xml = null;
7
8      public function renderLoginForm($params = array())
9      {
10         $this->xml->setAttribute('render', 'loginForm');
11         //another stuff...
12         $captchaXML = xml_AddChild($this->xml, 'captcha');
13         //another stuff...
14         return $this->xml;
15     }
16     //another stuff...
17 }
```

Listing 14: Ukázka XMLRenderer

3.4 CAPTCHA

Jako ochrana proti spamu a zneužití byla použita Google reCAPTCHA 2.0, která umožňuje ověřit CAPTCHA (test na rozlišení člověka a robota) i bez podpory JavaScriptu.

Pokud bude uživatel opakovaně zadávat neplatné přihlašovací údaje, bude muset vyřešit CAPTCHA. Po úspěšném vyřešení CAPTCHA a zadání správných přihlašovacích údajů bude uživatel přihlášen.

3.5 SP a CAP v jednom

Na testovacím serveru http://www.manicka2.net je nasazeno SSO trochu neobvykle, jelikož se server chová jako SP i CAP zároveň. Uživatel tak při autentizaci není posílán na jinou doménu, ale pouze na jinou URL. Funkčnost procesu SSO tím není nijak negativně ovlivněna.

Kvůli rozlišení přihlášení do administrace od jiných přihlášení bylo nutné upravit logiku přihlašovací metody a přidat kontrolu původu požadavku.

CAP sídlí na http://www.manicka2.net/sso, přihlášení do administrace na http://www.manicka2.net. Jelikož CAP i přihlášení do administrace jsou na stejné doméně, není nutné generovat JWT. Stačí pouze nastavit SSO cookie a vytvořit session.

Kapitola 4

Testování

Funkčnost demo aplikace byla otestována pomocí unit testů a akceptačních testů.

Unit testy ověřují správnost implementace a fungování systému. Pojem `unit` označuje samostatně testovatelnou část (jednotku), např. třídu nebo funkci.

Akceptační testy slouží k otestování určitého scénáře použití, např. přihlášení uživatele pomocí Iframe metody, který je definován jednotlivými kroky. Díky akceptačním testům je možno částečně simulovat reálného uživatele. Byly otestovány následující scénáře použití:

- Přihlášení úspěšné i neúspěšné
- Lokální i globální odhlášení
- Přihlášení na jednom serveru a následné pokračovaní jako přihlášený uživatel na serveru druhém.
- Přihlášení se jako jiný uživatel

Funkčnost implementace Modulu SSO ve Webgarden bude otestována v budoucnu spolu s testováním na reálných uživatelích.

Závěr

Hlavním cílem práce bylo vytvořit jednobodovou autentizaci pro Webgarden. Cíle bylo dosaženo díky vytvořené aplikaci Modul SSO, která umožňuje centralizovat přihlašovací proces. Podařilo se implementovat několik způsobů jednobodového přihlášení, čímž je zajištěna zpětná kompatibilita pro starší prohlížeče a konfigurovatelnost systému.

Modul SSO řeší i přihlášení pomocí Google a Facebooku, což ocení uživatelé sociálních sítí.

Práce demonstruje, jak pomocí různých technologických postupů vybudovat jednobodovou autentizaci.

V budoucnu by autor chtěl modifikovat aplikaci a použít ji jako rozšíření pro framework Nette[8] nebo Symfony[9].

[8] https://nette.org/cs/
[9] https://symfony.com/

Literatura

[1] Ponnapalli, R.: *Secure implementation of Enterprise single sign-on product in an organization*. SANS Institute, 2014, 3,4 s. Dostupné z: https://www.sans.org/reading-room/whitepapers/authentication/secure-implementation-enterprise-single-sign-on-product-organization-1520

[2] Shibboleth Consortium: What's Shibboleth. [cit. 2015-11-17]. Dostupné z: https://shibboleth.net/about/

[3] Petro, A.: Home - Central Authentication Service - Apereo Wiki. [cit. 2015-11-17]. Dostupné z: https://wiki.jasig.org/display/CAS/Home

[4] Hughes, J.; Maler, E.: Security assertion markup language (SAML) v2.0 technical overview. *OASIS SSTC Working Draft sstc-saml-tech-overview-2.0-draft-08*, 2005, [cit. 2015-11-17]. Dostupné z: https://www.oasis-open.org/committees/download.php/14361/sstc-saml-tech-overview-2.0-draft-08.pdf

[5] Introduction to JSON Web Tokens. [cit. 2015-11-17]. Dostupné z: http://jwt.io/introduction/

[6] Hardt, D.: The OAuth 2.0 authorization framework [online]. 2012, [cit. 2015-11-18]. Dostupné z: http://tools.ietf.org/html/rfc6749

[7] Sakimura, N.; Bradley, J.; Jones, M.; aj.: Openid connect core 1.0. *The OpenID Foundation*, 2014, [cit. 2015-11-18]. Dostupné z: http://openid.net/specs/openid-connect-core-1_0-final.html

[8] W3.org: Cross-Origin Resource Sharing. 2014, [cit. 2015-12-12]. Dostupné z: http://www.w3.org/TR/cors/

[9] Mozilla Developer Network: HTTP access control (CORS). 2015, [cit. 2015-12-12]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Access_control_CORS

[10] Van Kesteren, A.; Jackson, D.: The xmlhttprequest object. *World Wide Web Consortium, XMLHttpRequest*, 2007. Dostupné z: https://www.w3.org/TR/XMLHttpRequest/

[11] Flanagan, D.: *JavaScript: the definitive guide*. O'Reilly Media, Inc., 2006.

[12] Garrett, J. J.; aj.: Ajax: A new approach to web applications. 2005. Dostupné z: https://courses.cs.washington.edu/courses/cse490h/07sp/readings/ajax_adaptive_path.pdf

[13] Zhou, Y.; Evans, D.: Why aren't HTTP-only cookies more widely deployed. *Proceedings of 4th Web*, ročník 2, 2010. Dostupné z: https://www.cs.virginia.edu/~evans/pubs/w2sp2010/http-only.pdf

[14] Truneček, P.; Pelka, T.: Kryptografické protokoly v praxi. *Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií*, 2009. Dostupné z: https://dspace.vutbr.cz/bitstream/handle/11012/11034/DP_el_verze.pdf

[15] Mozilla Developer Network: Window.postMessage(). 2015, [cit. 2016-04-12]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Window/postMessage

[16] Clark, J.; aj.: Xsl transformations (xslt). *World Wide Web Consortium (W3C)*, 1999: str. 103, [cit. 2016-04-12]. Dostupné z: http://www.w3.org/TR/xslt

Příloha **A**

Seznam použitých zkratek

URL Uniform resource locator

URI Uniform resource identifier

API Application Programming Interface

SSO Single sign-on

SP Service provider

IdP Identity provider

CAP Central authentication point

OP OpenID Provider

RP Relying Party

UX User experience

XML Extensible markup language

AJAX Asynchronous JavaScript and XML

CORS Cross Origin Resourse Sharing

DOM Document Object Model

HTTP HyperText transfer protokol

JSON JavaScript object notation

PHP Hypertext Preprocesor

JWT JSON Web tokens

IANA Internet Assigned Numbers Authority

ORM Object-relational mapping

XSL eXtensible stylesheet language

XSLT eXtensible stylesheet language transformations

HTML HyperText Markup Language

CSS Cascading Style Sheets

CAPTCHA Completely Automated Public Turing test to tell Computers and Humans Apart

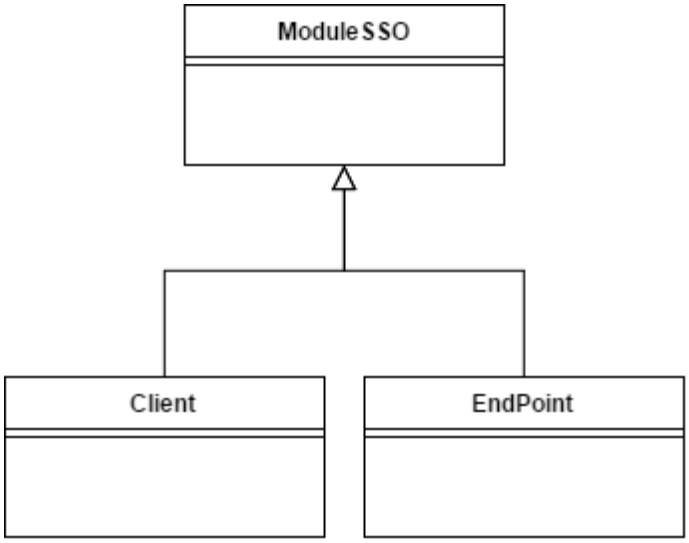
Příloha B

Použité nástroje

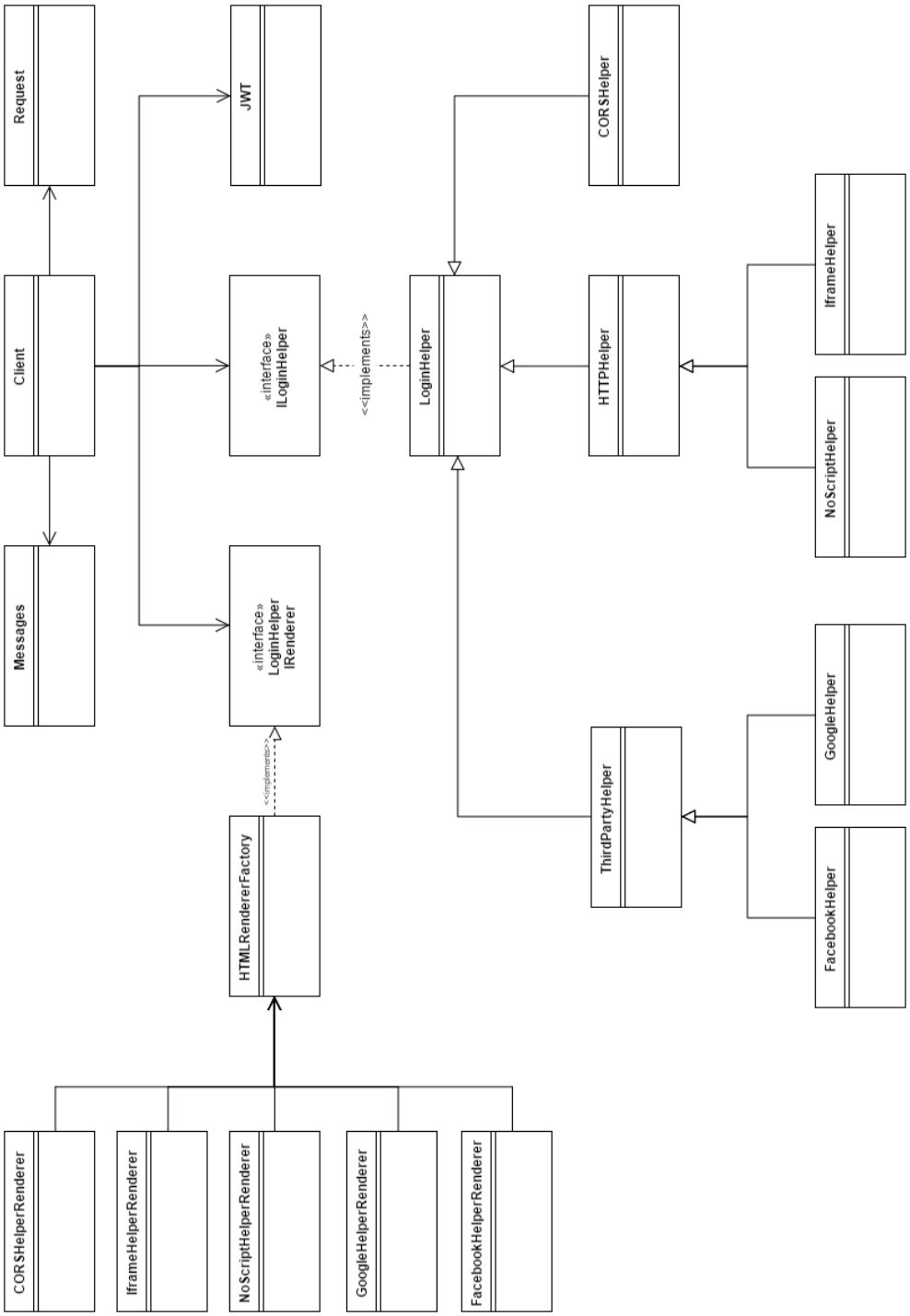
- Browscap - knihovna pro zjištění informací o prohlížeči, dostupná na https://github.com/browscap/browscap-php
- JWT - knihovna usnadňující generování, práci s daty a ověření pravosti JWT. Autorem je Luís Cobucci a knihovna je dostupná na https://github.com/lcobucci/jwt.
- HttpFoundation - knihovna umožňující objektový přístup k položkám HTTP požadavku, dostupná na https://github.com/symfony/http-foundation
- Codeception - framework pro testování aplikací umožňující psát funkcionální, akceptační a unit testy. Dostupný na https://github.com/Codeception/Codeception.
- Facebook SDK for PHP - knihovna poskytující rozhraní pro komunikaci s Facebook Grap API a Facebook Login, dostupná na https://github.com/facebook/facebook-php-sdk-v4
- Google APIs Client Library for PHP - knihovna dovolující práci s Google APIs (Google+, Google Drive, atd.), dostupná na https://github.com/google/google-api-php-client
- Google reCAPTCHA 2.0 - dostupná na https://github.com/google/recaptcha
- Selenium Standalone Server - framework, který umožňuje automatizovat prohlížeč, dostupný na http://www.seleniumhq.org/download/

Příloha C

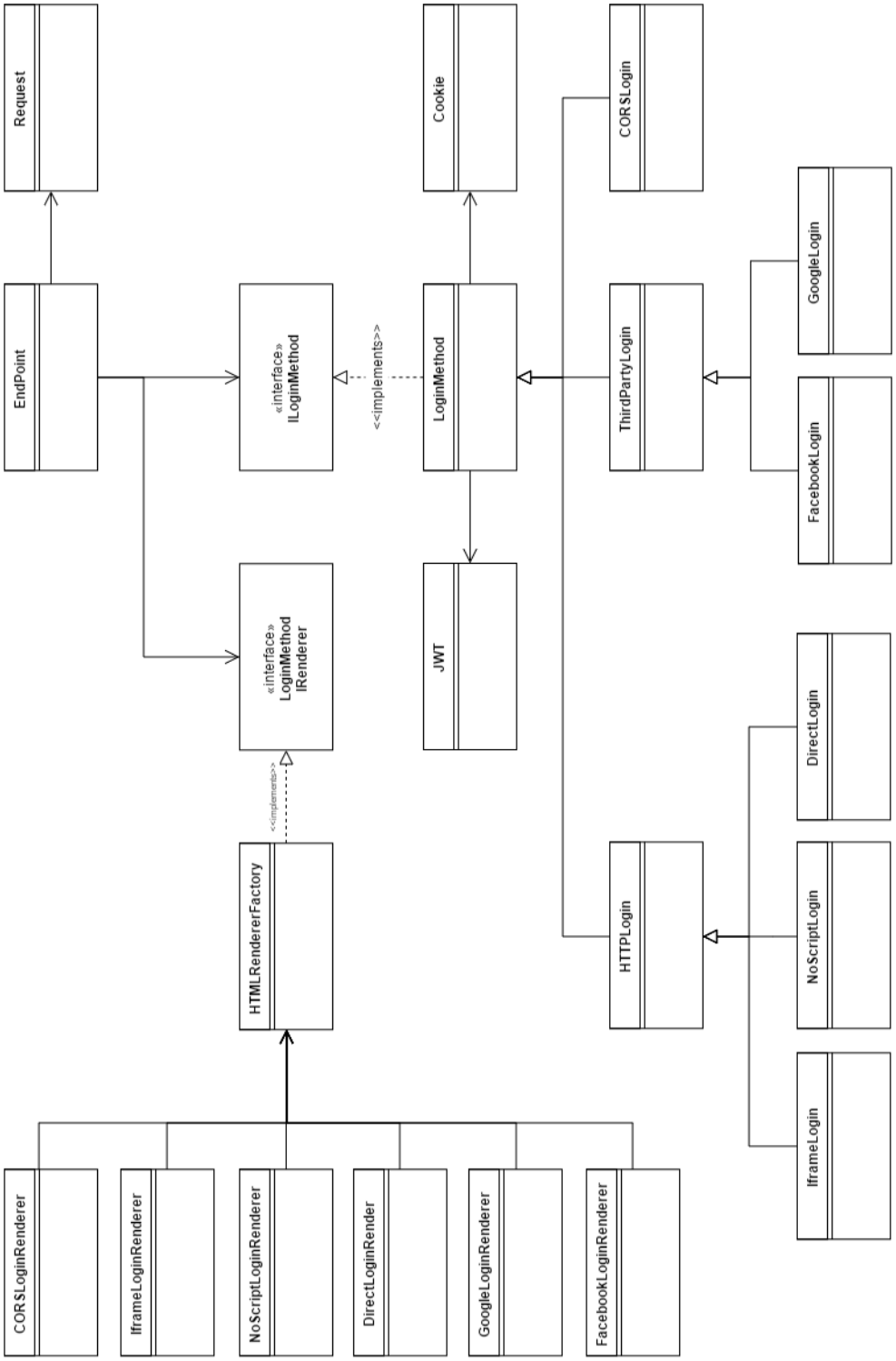
Ukázky kódu a obrázky



Obrázek C.1: Části Modulu SSO



Obrázek C.2: Doménový model Clienta v Modulu SSO



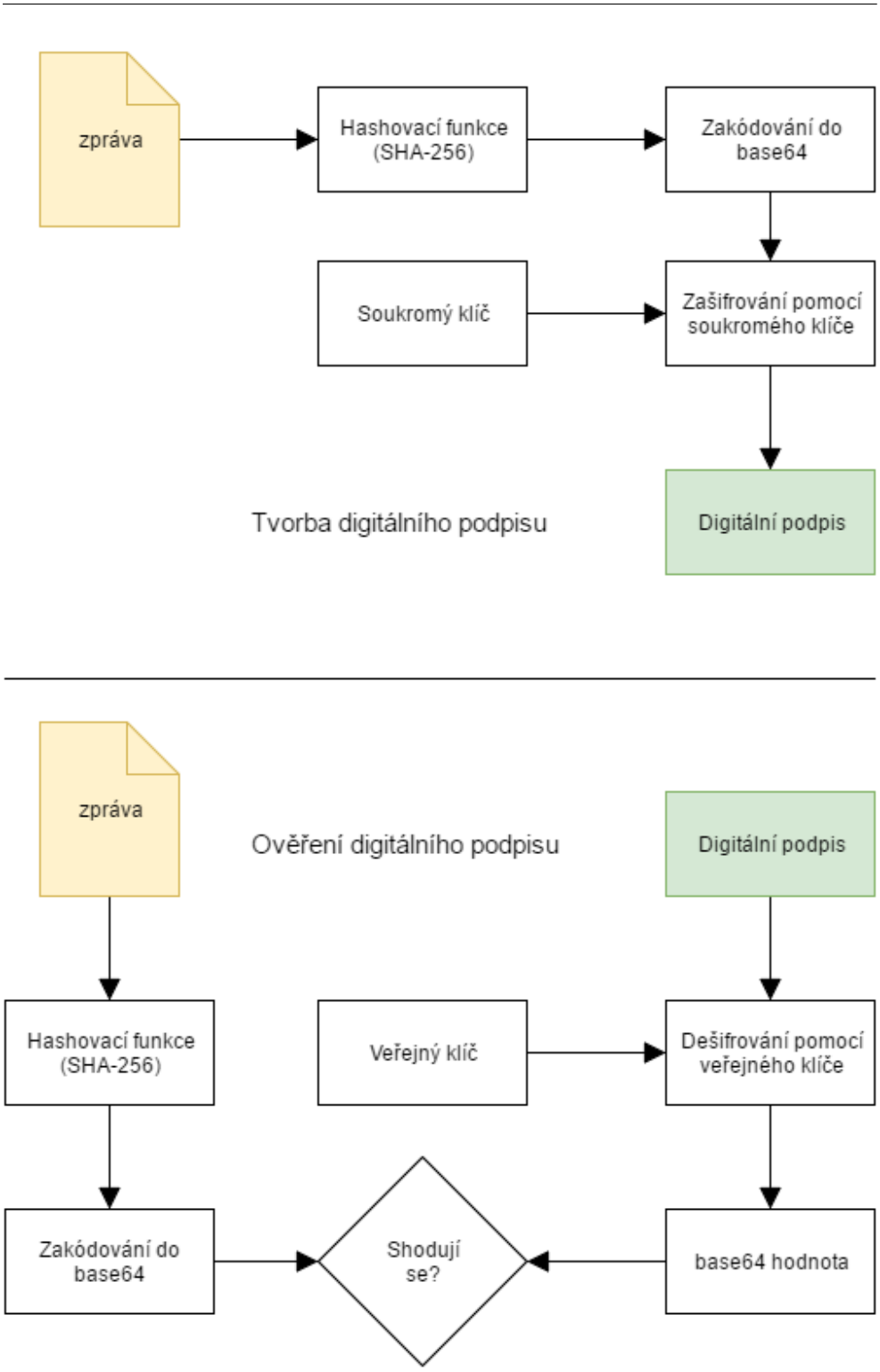
Obrázek C.3: Doménový model EndPointu v Modulu SSO

```
GET /resources/access-control-with-credentials/ HTTP/1.1
Host: bar.other
User-Agent: Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10.5;
↪  en-US; rv:1.9.1b3pre) Gecko/20081130 Minefield/3.1b3pre
Accept:
↪  text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Connection: keep-alive
Referer: http://foo.example/examples/credential.html
Origin: http://foo.example
Cookie: pageAccess=2


HTTP/1.1 200 OK
Date: Mon, 01 Dec 2008 01:34:52 GMT
Server: Apache/2.0.61 (Unix) PHP/4.4.7 mod_ssl/2.0.61
↪  OpenSSL/0.9.7e mod_fastcgi/2.4.2 DAV/2 SVN/1.4.2
X-Powered-By: PHP/5.2.6
Access-Control-Allow-Origin: http://foo.example
Access-Control-Allow-Credentials: true
Cache-Control: no-cache
Pragma: no-cache
Set-Cookie: pageAccess=3; expires=Wed, 31-Dec-2008 01:34:53 GMT
Vary: Accept-Encoding, Origin
Content-Encoding: gzip
Content-Length: 106
Keep-Alive: timeout=2, max=100
Connection: Keep-Alive
Content-Type: text/plain
```

Obrázek C.4: Ukázka XMLHttpRequest CORS požadavku – odpověď ze serveru [10]

[10] Převzato z [9]



Obrázek C.5: Tvorba a ověření digitálního podpisu

Příloha D

Manuál na instalaci demo aplikace

Aplikace funguje na jakékoliv platformě, kde je nainstalováno PHP, MySQL a webserver. Manuál počítá s nainstalovanými a nastavenými výše uvedenými službami.

D.1 Nastavení hosts

Do souboru `C:/Windows/System32/drivers/etc/hosts` přidejte následující řádek:

```
127.0.0.1 domain1.local domain2.local sso.local
```

D.2 Nastavení webserveru

Apache 2.4.17 byl vybrán jako webserver pro demo aplikaci. Konfigurační soubor `httpd.conf` je umístěn v adresáři `conf` v kořenové složce webserveru. Jako `DocumentRoot` uveďte cestu k dané složce v projektu. Pro `domain2.local` to může být např. složka `C:/wamp/www/sso/domain2`

```
<VirtualHost *:80>
  DocumentRoot "/path/to/sso/domain2"
  ServerName domain2.local
  ServerAlias *.domain2.local domain2.local
</VirtualHost>
```

```
<VirtualHost *:80>
  DocumentRoot "/path/to/sso/domain1"
  ServerName domain1.local
  ServerAlias *.domain1.local domain1.local
</VirtualHost>

<VirtualHost *:80>
  DocumentRoot "/path/to/sso/sso"
  ServerName sso.local
</VirtualHost>
```

Kvůli předání odpovědi z Google API na lokální SSO EndPoint je nutné vytvořit soubor `googleLogin.php` v kořenové složce webserveru s následujícím obsahem:

```
<?php
//www folder on your webserver, for example
↪  C:/wamp/www/googleLogin.php
if(isset($_GET['code'])) {
    $code = $_GET['code'];
    header("Location: http://sso.local/googleLogin.php?code=" .
    ↪  $code);
    exit;
}
```

D.3 Vytvoření databáze

Vytvořte MySQL databázi jménem `sso` a importujte do ní soubor `sso.sql`, který se nachází ve složce `dumps/sql` v projektu. Pro akceptační testy je potřeba vytvořit databázi `sso_dummy` a importovat do ní `sso_dummy.sql` soubor.

D.4 Konfigurace aplikace

V souboru `sso/app/config.php` nastavte podle sebe

- `CFG_SQL_HOST` - adresa databázového severu, typicky localhost
- `CFG_SQL_DBNAME` - název databáze
- `CFG_SQL_USERNAME` - přihlašovací jméno do databáze
- `CFG_SQL_PASSWORD` - heslo do databáze

D.5 Zkouška aplikace

Nyní by mělo být vše nastaveno. Pro jistotu ještě restartujte webserver. Importované SQL obsahuje dva dopředu vytvořené uživatele:

- email: joe@example.com, heslo: joe
- email: bob@example.com, heslo: bob

CAP sídlí na `http://sso.local`, první testovací doména na `http://domain1.local` a druhá testovací doména na `http://domain2.local`.

Příloha **E**

Obsah přiloženého CD

```
readme.txt ..................................stručný popis obsahu CD
src
├── impl ....................................zdrojové kódy implementace
└── thesis ..................................zdrojová forma práce ve formátu LaTeX
text ........................................text práce
└── thesis.pdf ..............................text práce ve formátu PDF
```