ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Tarnopolskyi** Jméno: **Dmytro** Osobní číslo: **461279**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Softwarové inženýrství a technologie**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Sémantická autorizační služba

Název bakalářské práce anglicky:

Semantic Authorization Service

Pokyny pro vypracování:

1. Analyzujte současné přístupy k tvorbě autentizačních a autorizačních služeb, a to včetně podpory více různých aplikací (multi-tenancy).
2. Navrhněte autorizační službu, která bude založena na sémantických technologiích a bude umožňovat správu uživatelů a jejich rolí. Aplikace budou na tuto službu delegovat přihlašování.
3. Naimplementujte navrženou autorizační službu, včetně podpory pro různé klientské aplikace (mutli-tenancy).
4. Ověřte správnost implementace jejím použitím v nástroji pro správu sémantických slovníků TermIt.

Seznam doporučené literatury:

[1] D. Allemang, J. Hendler, Semantic Web for the Working Ontologist: Effective Modeling in RDFS and OWL, Morgan Kaufmann, 2011
[2] A. Kukic, The Definitive Guide to Single Sign On, Auth0, 2019
[3] D. Betts, A. Homer, A. Jezierski, M. Narumoto, H. Zhang, Developing Multi-tenant Applications for the Cloud on Windows Azure, Microsoft patterns & practices, 2013

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Martin Ledvinka, skupina znalostních softwarových systémů FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **12.02.2020** Termín odevzdání bakalářské práce: **22.05.2020**

Platnost zadání bakalářské práce: **30.09.2021**

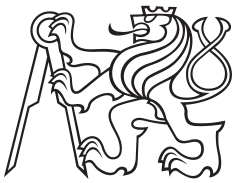
Ing. Martin Ledvinka podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. Datum převzetí zadání

Podpis studenta



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra počítačů

Sémantická autorizační služba

Dmytro Tarnopolskyi

Vedoucí práce: Ing. Martin Ledvinka
Květen 2020

Poděkování

Chtěl bych poděkovat vedoucímu své práce Ing. Martinu Ledvinkovi za jeho cenné rady, čas a trpělivost a zkušenosti, které jsem získal v rámci práce na tomto projektu.

Rád bych také poděkoval své rodině, kamarádům a kolegům za neustálou podporu při studiu.

Prohlášení

I declare that this work is all my own work and I have cited all sources I have used in the bibliography.

Prague, května 21, 2020

Prohlašuji, že jsem předloženou práci vypracoval samostatně, a že jsem uvedl veškerou použitou literaturu v souladu s Metodickým pokynem.

V Praze, 21. May 2020

Abstrakt

Tato bakalářská práce se zabývá úlohou vytvoření autorizační služby, která je založená na sémantických technologiích, umožňuje zprávu uživatelů a jejich rolí a podporuje multi-tenancy.

Byla provedená analýza již existujících řešení a na základě této analýzy bylo navrženo a vytvořeno vlastní řešení.

Při řešení úlohy byl použit programovací jazyk Java verze 8, Spring framework, Intellij IDEA a nástroj na posílání HTTP požadavků Postman.

Klíčová slova: Autorizační služba, sémantické technologie, multi-tenancy, triple store, token

Vedoucí práce: Ing. Martin Ledvinka

Abstract

This Bachelor Thesis deals with the issues of creating a multi-tenant authorization service, based on the semantic technologies, and allows to manage users and their roles.

The analysis of already existing solutions was conducted, which served as a base for application development.

Java 8, Spring Framework, Intellij IDEA, and Postman HTTP-requests sending tool were used to perform the implementation.

Keywords: Authorization service, semantic technologies, multi-tenancy, triple store, token

Title translation: Semantic Authorization Service

Obsah

Obrázky

Tabulky

Kapitola 1
Úvod

1.1 Motivace

V současné době každý člověk používá velké množství různých webových a desktopových aplikací pro různé účely. Většina těchto aplikací poskytuje funkce registrace pro nové a přihlášení pro již existující uživatele, aby došlo k oddělení veřejných dat od soukromých. Pro každou aplikaci si musí každý uživatel vymyslet základní údaje, jako uživatelské jméno a heslo, které potom bude používat i k přihlášení a pamatovat, které údaje k čemu patří, což bude uživatele obtěžovat.

Další důležitým faktem je, že vývojářská komunita se ve stále větší míře přiklání k architektuře orientované na služby či přímo k Microservices. Jedním z hlavních kandidátů na samostatnou službu je autentizace a autorizace uživatelů. Taková služba je základem drtivé většiny systémů, včetně těch vyvíjených na KBSS.

Existuje již hodně různých řešení v oblasti autentizace a autorizace, které se liší technologií, způsobem realizace, složitostí. Má vlastní variace by měla využívat sémantický web a umožňovat správu uživatelů v rámci různých organizací/systémů (tenants) tak, aby více systémů mohlo využívat stejnou autorizační službu.

Cíl práce

Tato práce se zabývá problematikou autentizačních a autorizačních systémů, současnými přístupy k jejich tvorbě včetně podpory více různých aplikací. Jejím cílem je analýza současných přístupů k tvorbě aplikací v oblasti autentizace a autorizace a nalezení řešení, které by splnilo všechny požadavky.

Dalším krokem je vytvoření autorizační služby, založené na sémantických technologiích (RDF, JSON-LD, Triple store), která bude umožňovat správu uživatelů a jejich rolí a kterou by bylo možné využít v doméně informačních systémů využívajících zmíněné technologie sémantického webu s podporou více tenantů.

Posledním krokem je ověření správnosti implementace jejím použitím v nástroji pro správu sémantických slovníků TermIt.

Kapitola 2
Analýza

Úvod

V této kapitole se zabývám analýzou již existujících způsobů autorizace a autentizace uživatele, jejich technologií a realizací, následně pak sémantickými technologiemi, technologií jediného přístupu, multi-tenantní technologií a jejich použitím v autorizačních službách.

Definice

Identifikace je proces porovnání rozmanitých objektů na základě jejich shod nebo rozdílů ve vlastnostech, formách, umístění, složení (struktuře), funkcích, projevech, významu nebo v čase, s cílem zjistit, zda se jedná či nejedná o shodné (identické) objekty.[1]

Autentizace je proces spočívající v ověření identity uživatele služeb.[2] Autentizace říká, kdo je uživatel, patří k bezpečnostním opatřením a zajišťuje ochranu před falšováním identity, kdy se subjekt vydává za někoho, kým není. Autentizace může probíhat například pomocí jména a hesla, občanského průkazu, otisku prstu atd.

Autorizace je proces, při němž dochází k ověření údajů při vstupu do nějakého systému. Při tomto procesu se vyhodnocuje, zda jsou zadané přístupové údaje korektní a zda má uživatel právo ke vstupu.[3]

Sémantický web je Web of Data – dat, titulů, chemických vlastností a jakýchkoliv dalších dat, jež si člověk může představit. Hlavní myšlenkou sémantického webu je podpora distribuovaného webu na úrovni dat spíše než na úrovni prezentace. Na místo toho, aby jedna webová stránka odkazovala na jinou, může jedna datová položka odkazovat na jinou pomocí globálních odkazů s názvem Uniform Resource Identifiers (URI).[4]

RDF (Resource Description Framework) framework popisu zdrojů. Jedná se o základní framework, na kterém je založen zbytek sémantického webu. RDF poskytuje mechanismus, který umožňuje komukoli učinit základní tvrzení o čemkoli a vnoření tohoto tvrzení do jediného modelu. RDF bylo doporučením W3C od roku 2003.[5]

RDFS (The RDF Schema language) - je jazyk, který se používá k

popisu základních pojmů o shodnosti a variabilitě známých objektových jazyků a jiných systémů tříd, právě tříd, podtříd a vlastností.[5]

OWL (Web Ontology Language) je sémantický webový jazyk, který přináší expresivitu logiky do sémantického webu. Umožňuje modelářům vyjádřit podrobná omezení mezi třídami, entitami a vlastnostmi. OWL byl přijat jako doporučení W3C v roce 2003.[4]

Ontologie je pokus komplexně a důkladně formalizovat určitou oblast znalostí pomocí koncepčního schématu. Typicky takové schéma sestává z datové struktury obsahující všechny relevantní třídy objektů, jejich vztahy a pravidla (věty, omezení) přijatá v této oblasti.

Identity provider (IdP) – web, aplikace nebo služba odpovědná za koordinaci identity mezi uživateli a klienty. IdP může poskytnout uživateli identifikační informace a poskytovat tyto informace službám, když uživatel požaduje přístup. [5]

Service Provider (SP) – je poskytovatel služby. Webová stránka, na které je služba dostupná, například YouTube je poskytovatelem služby pro sdílení video souborů.

2.1 Sémantické technologie

2.1.1 Sémantický web

Sémantický web je doplněk ke stávajícímu webu, který byl vynalezen tak, aby informace zveřejněné na internetu byly vhodné pro strojové zpracování. Informace dostupné na webu jsou vhodné pro lidské čtení. Sémantický web je vytvořen proto, aby informace byly vhodné pro automatickou analýzu, syntézu závěrů a transformaci jak samotných dat, tak závěrů, které jsou užitečné v praxi.[4]

Strojové zpracování je možné díky dvěma vlastnostem sémantického webu:

- IRI;
- Použiti sémantických sítí a ontologií.

IRI (Internationalized Resource Identifier) – rozšiřují URI pomocí UCS (Universal Character Set), což je univerzální znaková sada, která obsahuje znaky z různých standardů, včetně množství grafických, matematických a vědeckých symbolů. IRI se používají k pojmenování objektů. Každý objekt globální sémantické sítě má jedinečný IRI, jinými slovy IRI jedinečně pojmenovává objekt.

Samostatné IRI jsou vytvářeny nejen pro stránky, ale také pro objekty skutečného světa (lidé, města, umělecká díla, démoni atd.), a dokonce i pro abstraktní pojmy (například *jméno, pozice, barva*). Vzhledem k jedinečnosti IRI lze stejné objekty nazvat identicky na různých místech sémantického webu. Pomocí IRI lze shromažďovat informace o jedné položce z různých míst. Doporučuje se, aby byl do IRI zahrnut název jednoho z World Wide Web protokolů (HTTP nebo HTTPS). To znamená, že je doporučeno začít adresu

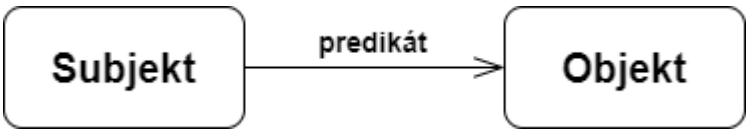
URI pomocí *http://* nebo *https://*. Taková adresa může být současně použita jako IRI a jako adresa webové stránky (URL). Na webových stránkách, jejichž adresy URL odpovídají IRI, W3C doporučuje zveřejnit popis předmětu. Popis je žádoucí poskytnout ve dvou formátech:

- Ve formátu vhodném pro lidské čtení;
- Ve formátu vhodném pro strojové čtení.

2.1.2 RDF

Jako strojově čitelný formát W3C navrhuje používat jazyk RDF. Jazyk RDF umožňuje popsat strukturu sémantické sítě ve formě grafu. Každému uzlu a každému oblouku grafu lze přiřadit samostatný IRI.

RDF umožňuje zápis informace v podobě třech propojených datových kusů, které se nazývají *triple* (Obrázek 2.1) a které se pák ukládají do triplestoru. Triply jsou také označovány jako *tvrzení* nebo *tvrzení RDF*.



Obrázek 2.1: RDF triple

„Subjekt → predikát → objekt“ je schopen vzít jakýkoli předmět a připojit jej k jakémukoli jinému objektu pomocí predikátu k zobrazení typu vztahu existujícího mezi subjektem a objektem.[6]

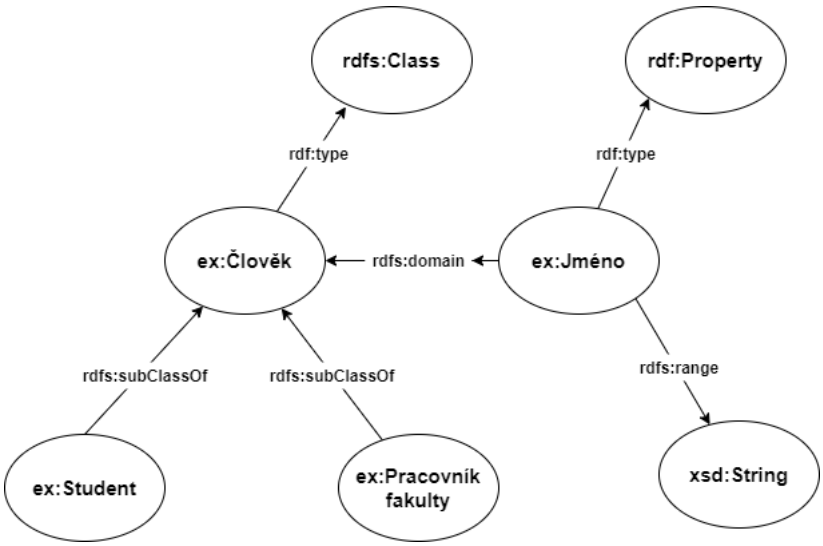
Například věta „Jakub prodává knihy“ může být uložena jako tvrzení RDF v triplestoru a popisuje vztah mezi subjektem věty *Jakub* a objektem *knihy*. Predikát *prodává* ukazuje, jak jsou subjekt a objekt propojeny.

2.1.3 Využití sémantických sítí a ontologií

Data na World Wide Web jsou obvykle prezentována jako text zaznamenaný v přirozených jazycích. Tyto texty jsou určeny pro lidské vnímání, ale stroj dokáže porozumět jejich významu pomocí jedné z metod zpracování přirozeného jazyka. Metody provádějí frekvenční analýzu a/nebo lexikální analýzu textu.

Tvrzení, psaná v RDF, lze interpretovat pomocí ontologií. Pro vytváření ontologií se doporučuje používat jazyky RDF Schéma a OWL. Ontologie jsou vytvářeny k získání logických závěrů z dat a jsou založené na matematických formalismech nazývaných popisná logika. Ontologie umožňují formální popis dat. Určují jak třídy objektů a vlastnosti vztahů, tak jejich hierarchické pořadí.

Toto schéma (Obrázek 2.2) zobrazuje, že prvek *Člověk* je typu *Class* a je doménou vlastnosti *Jméno*, které má *String* jako obor hodnot. *Student* a *Pracovník fakulty* jsou podtřídy třídy *Člověk*.



Obrázek 2.2: Příklad RDF Schématu

2.1.4 Triplestore

Triplestory jsou grafickou databází a ukládají sémantická data jako síť objektů s materializovanými vazbami mezi nimi. Díky tomu je RDF triplestorem preferovanou volbou pro správu vysoce propojených dat. Triplestory jsou například flexibilnější a méně nákladné než relační databáze.[6]

Databáze RDF se často nazývá databáze sémantických grafů, je také schopna zvládnout výkonné sémantické dotazy a využívat inference pro odkrývání nových informací ze stávajících vztahů.

2.2 Autorizační služby

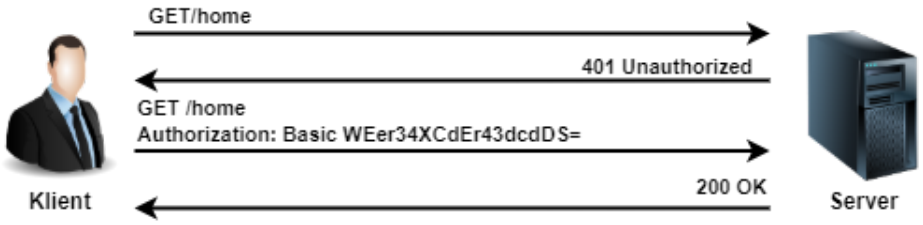
V dnešní době se při návrhu aplikace, která podporuje procesy autorizace a autentizace uživatelů, používají autorizační služby, postavené na různých technologiích. V této sekci jsou uvedeny informace a srovnání různých způsobů realizace těchto služeb.

2.2.1 Přihlášení pomocí hesla

Autentizace pomocí uživatelského jména a hesla je dnes samozřejmostí, protože poskytuje jednoduchý proces, se kterým je většina lidí seznámena. Tato metoda je postavená na tom, že uživatel musí pro úspěšnou identifikaci a autentizaci zadat uživatelské jméno a heslo, které si vytváří při registraci v systému. Kvůli tomu, že se přihlašovací údaje ukládají do databáze, hesla musí být hashovaná a solená. Ukládání hesel prostého textu je velmi nebezpečné, protože pokud je databáze napadena, útočník bude mít pravděpodobně přístup k mnoha účtům, protože uživatelé obvykle znovu používají stejnou sadu přihlašovacích údajů pro více aplikací. Existuje několik různých protokolů pro autentizace pomocí hesla.[5]

HTTP Basic Authentication

Nejjednodušším způsobem autorizace pomocí hesla je Basic access authentication. Tento protokol už se málokdy používá na internetu, ale používá se ve vnitřních korporačních systémech, protože většina z nich byla vyvíjena dávno.



Obrázek 2.3: HTTP Basic Authentication

Obrázek 2.3 ukazuje, že první věc, kterou neautorizovaný uživatel uvidí, je chyba *401 Unauthorized*. Uživatel zadává uživatelské jméno a heslo, které se zabalí do kódování Base64, jež převádí data z binární podoby do posloupnosti tisknutelných znaků a ta se posílají na server k ověření. Když ověření proběhne úspěšně, server posílá uživateli zprávu *200 OK*, což znamená, že už je autorizován a odpovídá požadovanými daty. Pro každý další request musí klient posílat kombinaci jména a hesla. V daném případě existuje hodně rizik a jedním z nich je «man-in-the-middle attack». Tato hrozba znamená, že pokud se používá nespolehlivé připojení k internetu, data mohou být ukradena během procesu jejich posílaní od klienta na server nebo v opačném směru.

HTTP Digest Authentication

Dalším krokem ve vývoji technologie autorizace je systém HTTP digest authentication, který používá hashovací algoritmus MD-5 v různých variacích. Hashovaní pomáhá bránit uhádnutí uživatelského jména a hesla, ale stále existuje mnoho různých rizik při použití tohoto systému, protože dnes už MD-5 není rozhodně bezpečným algoritmem.

NTLM

Autentizace, která je postavená na metodě challenge-respone, kdy se heslo nepohybuje v otevřené podobě. Není standardem HTTP, ale je podporovaná většinou prohlížečů webových servisů. Obvykle se používá pro autentifikace uživatelů Windows Active Directory ve webových aplikacích.

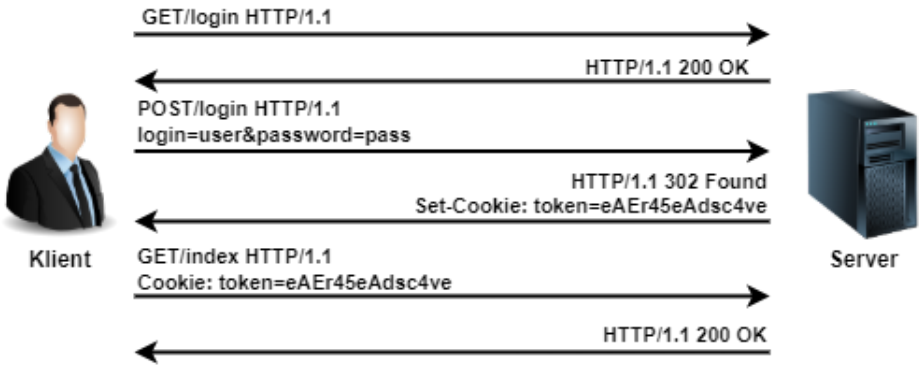
Negotiate

Ještě jeden příklad autentizace z rodiny Windows Authentication, který dovolí klientovi vybírat mezi NTML a Kerberos autentizací. Kerberos je bezpečný protokol, který je založen na principu SSO (Single Sign-On), ale klient a

server se musejí nacházet v oblasti intranetu a musí být součástí domény Windows.[7]

2.2.2 Forms Authentication

Pro tento protokol neexistuje určitý standard, a proto všechny jeho realizace jsou různé a specifické pro určité části autentizace frameworků. Autentizace v tomto případě probíhá na vyšší úrovni abstraktního modelu.



Obrázek 2.4: Forms Authentication

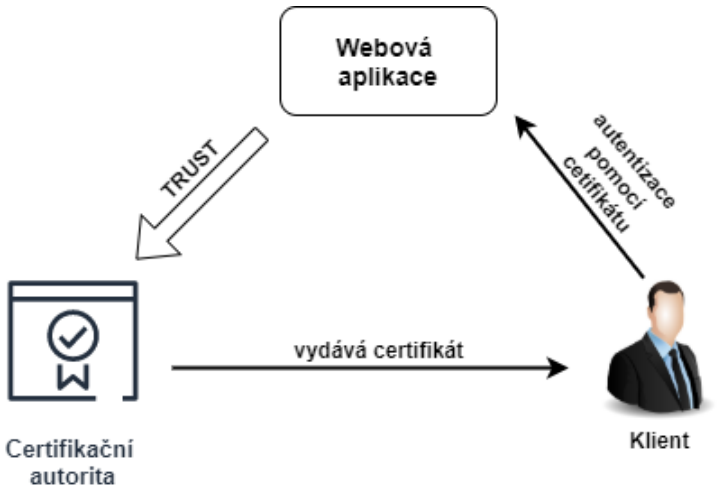
HTTP server neukazuje chybu přístupu, ale přesměruje neautorizovaného uživatele na další stránku, která obsahuje přihlašovací formu, což je pole pro uživatelské jméno a heslo. Jakmile uživatel vyplní své přihlašovací údaje, vytvoří se POST požadavek a data se posílají na server buď přes spolehlivý kanál, HTTP nebo plaintext. Server si vytváří a posílá uživateli token nebo session identifikátor, který se ukládá do Cookies a používá se pro další přístup ke stránce.(Obrázek 2.4)

2.2.3 Autentizace pomocí certifikátu

Certifikát je sada atributů, které identifikují jejich vlastníka. Je schválený CA (certificate authority). Certifikát je kryptograficky spojený s privátním klíčem, který se nachází u vlastníka certifikátu a dává možnost jednoznačně ověřit vlastnictví certifikátu.

Na straně klienta může být klíč uložen v operačním systému, v browseru, v souboru, na zvláštním zařízení (smart card, USB token). Obvykle se pro přístup a užití používá ke klíči heslo nebo PIN.

Ve webových aplikacích se používají certifikáty standardu X.509. Autentizace pomocí takového certifikátu probíhá v okamžiku spojení se serverem a je součástí protokolu SSL/TLS. Tento způsob je podporovaný prohlížeči, které dávají možnost použít tento certifikát pro přihlášení, pokud to webová stránka podporuje.



Obrázek 2.5: Autentizace pomocí certifikátu

Během autentizace server ověřuje certifikát podle následujících pravidel:

- Jestli je certifikát schválený CA;
- Jestli certifikát nevyexpiroval.

Po úspěšném ověření může webová aplikace provést autorizaci požadavků na základě dat certifikátu, jako jsou *subjekt* (jméno majitele), *issuer*, *seriál number* (sériové číslo certifikátu) nebo *thumbprint* (data, vytvořená na základě privátního klíče certifikátu).

Použití autorizace pomocí certifikátu, které je zobrazeno na obrázku 2.5 je mnohem spolehlivější a bezpečnější způsob, než autorizace pomocí hesla. Je to z toho důvodu, že se během autorizace vytváří podpis, který ukazuje, že byl použit klíč. Kromě spolehlivosti má tato metoda autorizace problém s rozšiřováním a podporou certifikátů, což dělá tento způsob autentizace málo použitelným.

2.2.4 Autorizace pomocí jednorázového hesla

Autorizace pomocí jednorázového hesla se obvykle používá spolu s autorizací pomocí hesla, což je two-factor-authentication (2FA). Uživatel pro autorizaci musí znát přihlašovací údaje (heslo nebo uživatelské jméno) a zároveň musí mít přístup k zařízení, které jednorázové heslo generuje (obvykle mobilní telefon). Přítomnost dvou faktorů pomáhá zvýšit bezpečnost, což je u webových aplikací potřeba.[8]

Existuje jiné použití této metody, například doplňková autentizace uživatele pro dokončení operace, jako peněžní poukázka, změna nastavení atd.

Možnosti pro doručení jednorázového hesla:

- SMS – náhodně se generuje kód, který se posílá na telefonní číslo, které bylo uvedené při autorizaci. Uživatel musí mít přístup k mobilnímu telefonu, aby heslo dostal a použil;

- Osobní tokeny – tokeny, které generují jednorázové heslo na základě tajného klíče, který se nachází na serverové straně, což dává možnost zkontrolovat jednorázové heslo.

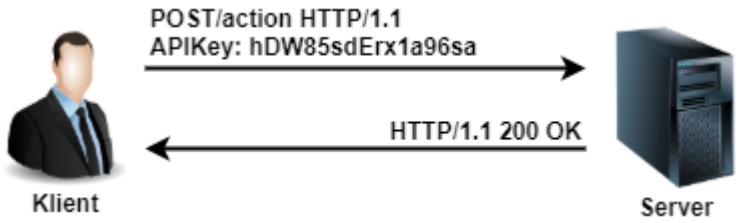
Ve webových aplikacích se tato metoda autorizace realizuje pomocí Forms Authentication. Po první autorizaci heslem se vytváří session uživatele, ale stále nemá přístup k funkcionalitě stránky, dokud neproběhne doplňková autorizace pomocí jednorázového hesla.

2.2.5 Autorizace pomoci Access Key

Tato metoda se nejčastěji používá pro autentizaci zařízení, servisů nebo aplikací pro přístup k webovým aplikacím. Obsahuje přístupový klíč nebo API klíč. Jsou to unikátní řetězce, které mají v sobě náhodnou posloupnost znaku, která je náhradou za kombinaci uživatelského jména a hesla.

Ve většině případů generace přístupového klíče probíhá na serveru jako odpověď na požadavek uživatele, jenž si pak uloží tento klíč v klientské aplikaci. Před vytvořením přístupového klíče existuje možnost nastavení doby platnosti klíče a přístupové úrovně, které budou klientům přiřazeny.

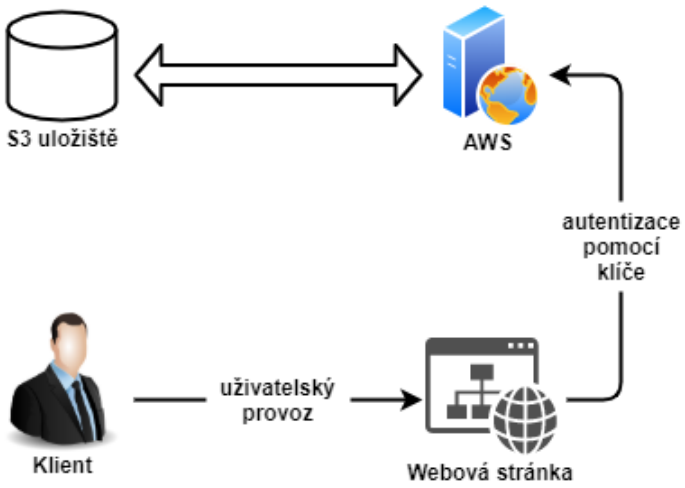
Příkladem použití autorizace pomocí Access Key je cloud Amazon Web Services. Předpokládáme, že uživatel má webovou aplikaci, která splňuje funkce nahrávání a přehledu fotografií, a on chce používat Amazon S3 jako úložiště souboru. V tomto případě si musí přes konzoli AWS vytvořit klíč, který má omezený přístup ke cloudu, právě čtení/zápis souboru na Amazon S3. Tento klíč pak je možné použít pro autentizaci aplikace v cloudu AWS.



Obrázek 2.6: Příklad autentizace pomocí Access Key

Použitím přístupového klíče (Obrázek 2.6) se lze vyhnout předání přihlašovacích údajů aplikacím na straně. Mají mnohem složitější strukturu na rozdíl od hesla, a proto není možno odhadnout přístupový klíč, což je velkou výhodou v bezpečnosti autorizace. Pokud dojde ke ztrátě klíče, tak stačí ho vynulovat a vytvořit si nový klíč.

Z technického hlediska neexistuje jediný protokol pro přenášení klíčů, může se umístit v HTTP požadavku, URL query, request body nebo HTTP headeru (Obrázek 2.7).



Obrázek 2.7: Autentizace pomoci Access Key, který se přenáší v hlavičce HTTP

Existuje složitější variace autorizace pomocí přístupového klíče, která se skládá ze dvou částí: veřejné a privátní. Veřejná část slouží pro identifikaci uživatele, privátní – pro vytvoření podpisu.

2.2.6 Autorizace pomocí tokenů

Autorizace pomocí tokenů se nejčastěji používá pro implementaci systému SSO (Single Sign-On). Realizace takové metody autentizace spočívá v tom, že identity provider (IdP) poskytuje data o uživateli v podobě tokenu, který service provider používá pro identifikaci, autentizaci a autorizaci uživatele.

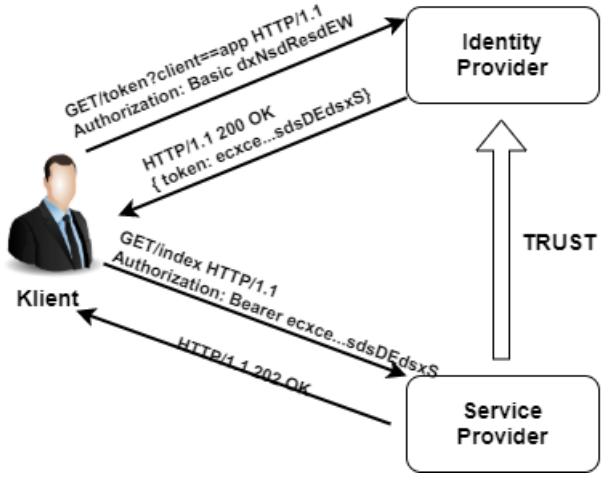
V hlavních rysech takový proces vypadá následovně:

1. Klient se autentizuje v identity provideru příslušným způsobem (heslo, access key, certifikát, Kerberos atd.);

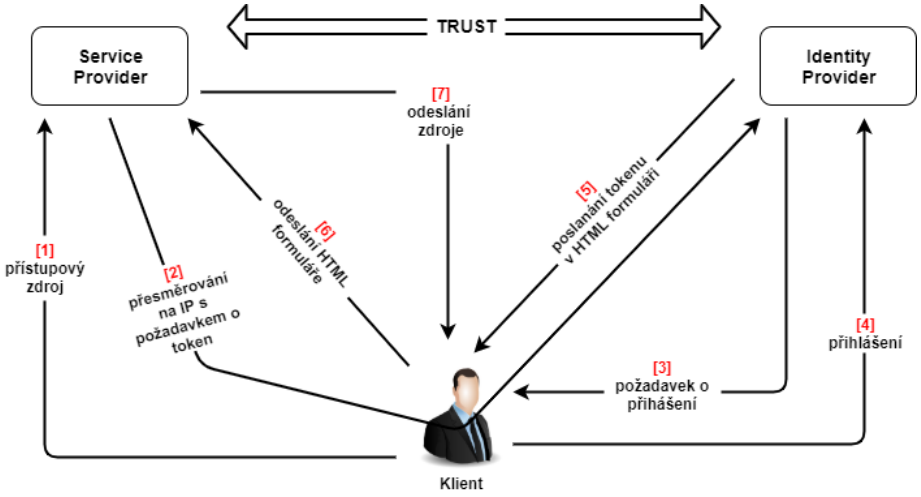
2. Klient posílá identity provideru požadavek o token pro přístup ke konkrétní service provider aplikaci. Identity provider generuje token a posílá ho klientovi;

3. Klient se autentizuje v service provider aplikaci pomocí tohoto tokenu.

Z obrázku 2.8 je vidět, jak se autentizuje aktivní klient, který může vykonávat posloupnosti nějakých příkazů. Co se týče pasivního klienta, jako je browser, který může jenom zobrazovat stránky požadované uživatelem, tak autentizace probíhá automatickým přesměrováním mezi webovými stránkami, identity providerem a service providerem (Obrázek 2.9).



Obrázek 2.8: Autorizace „aktivního“ klienta pomocí Bearer tokenu



Obrázek 2.9: Autorizace „pasivního“ klienta metodou přesměrování

Existuje několik různých standardů, které definují protokoly interakce mezi klienty (aktivní a pasivní) a IS/SP aplikacemi a formát tokenů. Nejznámější jsou OAuth, OpenID Connect, SAML a WS-Federation.

Token je datová struktura, obsahující informace o tom, kdo daný token vygeneroval, kdo token může dostat, životní dobu, informace o uživateli. Kromě toho token ještě obsahuje podpis, aby nedošlo k neschváleným změnám.

Pro autentizaci pomocí tokenu musí SP-aplikace vyplnit následující kontroly:

1. Token byl vygenerován identity providerem (pole issuer);
2. Token je určen dané SP-aplikaci (pole audience);
3. Token je platný (pole expiration date);

4. Kontrola podpisu (podpis).

Jestli kontrola proběhla úspěšně, SP-aplikace vyplní autentizaci uživatele na základě dat z tokenu.

2.2.7 Formáty tokenů

- Simple Web Token (SWT).

Simple Web Token (SWT) – nejjednodušší formát, reprezentující sadu dat jméno-hodnota s kódováním HTML form. Má několik rezervovaných jmen: Issuer, Audience, ExpiresOn, HMAC SHA256 (Obrázek 2.10). Token se podepisuje pomocí symetrického klíče tak, že IP a SP aplikace musejí mít klíč, aby token byl vytvořen nebo ověřen.

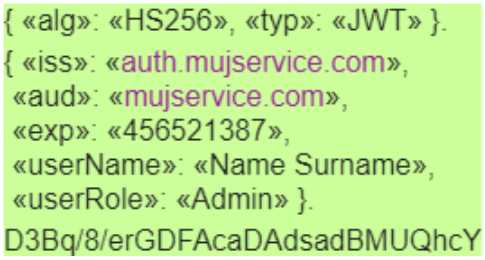
```
Issuer=http://auth.mujservice.com&
Audience=http://mujservice&
ExpiresOn=456521387&
UserName=Name Surname&
UserRole=Admin&
HMACSHA256=ADSERwewx4sdaSSDAWsadwDAS
```

Obrázek 2.10: Příklad Simple Web Tokenu

- JSON Web Token (JWT). JSON Web Token (JWT) – obsahuje tři bloky dat, která jsou rozdělena tečkami: hlavička, náklad a podpis.[9]
  - Hlavička (Header) – se obvykle skládá ze dvou částí: typ tokenu, což je JWT a hashovacího algoritmu, jako HMAC SHA256 nebo RSA.
  - Náklad (Playload) – obsahuje tvrzení o entitě v podobě jméno-hodnota a po vytvoření je zakódovaný ve formátu base64. (Obrázek 2.11) Existují tři druhy tvrzení: reservované (reserved), veřejné (public) a privátní (private).
    - Rezervované tvrzení – jedná se o soubor předdefinovaných tvrzení, která nejsou povinná, ale doporučená, aby poskytovala sadu užitečných interoperabilních tvrzení. Některé z nich jsou: iss (emitent), exp (expirační doba), sub (předmět), aud (publikum) a další;[10]
    - Veřejná tvrzení – mohou být libovolně definovaná tím, kdo JWT používá. Aby se zabránilo kolizím, měla by být definovaná v registru Token IANA JSON nebo by měla být definovaná jako URI, které obsahuje jmenný prostor odolný proti kolizi;
    - Privátní tvrzení – jsou obvykle tvrzení vytvořená za účelem sdílení informací mezi stranami, které se dohodly na jejich použití a nejsou ani registrovanými ani veřejnými nároky.

- Podpis (Signature) – chcete-li vytvořit podpis, je třeba vzít zakódovanou hlavičku, zakódovaný náklad, tajný klíč, algoritmus specifikovaný v hlavičce a vše podepsat.

Cílem JWT je možnost ověření autenticity dat – skutečnosti, že data nebyla cestou změněna. Nikoli však skrýt obsah dat. Účelem zakódování dat je převod struktury dat do lépe přenositelné podoby.[9]



Obrázek 2.11: Příklad JSON Web Tokenu

SAML

SAML (Security Assertion Markup Language) je standard, který usnadňuje výměnu bezpečnostních informací. SAML, vyvinutý organizací OASIS (rganization for the Advancement of Structured Information Standards), je frameworkem založeným na XML. SAML 2.0. umožňuje různým organizacím (s různými bezpečnostními doménami: poskytovatelem identity (IDP) a poskytovatelem služeb (SP)) bezpečně si vyměňovat informace o autentizaci a autorizaci.[11]

SAML definuje přesnou syntaxi a pravidla pro vyžádání, vytvoření, komunikaci a použití těchto SAML tvrzení.

Základní části jsou:

- Tvrzení (Assertions) – je zpráva, která říká poskytovateli služeb (SP), že je uživatel přihlášen. Výroky SAML obsahují všechny informace nezbytné pro poskytovatele služeb k potvrzení totožnosti uživatele, včetně zdroje tvrzení, času, kdy bylo vydáno, a podmínek které činí tvrzení platným;
- Protokoly (Protocols) – sada zprav, které se přenáší mezi účastníky (požadavek na vytvoření nového tokenu, seznam existujících tokenů, logout atd.);
- Vazby (Bindings) – popisují, jak jsou různé SAML protokolové zprávy přenášené přes základní transportní protokoly (HTTP Redirect, HTTP POST, HTTP artifact, SAML SOAP, SAML URI atd.);
- Profily (Profiles) – scénáře použití standardu, určující assertions, protocols a bindings pro jejich realizace. Příkladem je Web Browser SSO.

SAML definuje formát přenesení metadat mezi účastníky, které obsahují seznam rolí, protokolů, atributů, šifrovací klíče atd.

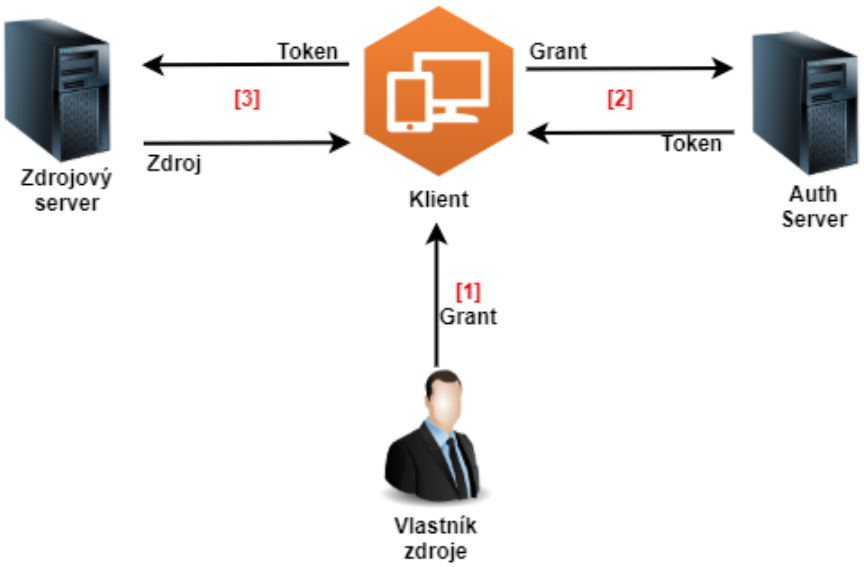
OAuth a OpenID Connect

OAuth 2 je autorizační protokol, jímž uživatel, vlastnící zdroje na resource serveru, zplnomocňuje cizí aplikaci, aby jeho jménem se zdroji zacházela.[5] Protokol se pomáhá vyhnout předání uživatelského jména a hesla každé aplikaci a omezit práva, které aplikace má.

OpenID Connect je ověřovací protokol založený na OAuth 2.0 specifikaci. Zpracovává autentizaci prostřednictvím JSON Web Tokens (JWT) dodaných pomocí protokolu OAuth 2.0.[5]

Proces delegování funkce přístupu (Obrázek 2.12) se skládá z:

- Uživatel (resource owner) povoluje aplikaci (client) přístup k nějakému zdroji v podobě grantu;
- Aplikace komunikuje s autorizačním serverem a vyžádá si access token pomocí grantu. Autorizační server autentizuje klienta a ověří autorizační povolení. Pokud je povolení platné, vystaví access token;
- Aplikace používá tento token pro vyžádání dat od zdrojového serveru, pokud token je platný, data od serveru dostane.



Obrázek 2.12: Delegování funkce přístupu

- Access Token – aplikace je používají k vytváření požadavků API jménem uživatele. Token přístupu představuje oprávnění konkrétní aplikace pro přístup ke konkrétním částem uživatelských dat.

  Přístupové tokeny musí být během přenosu a skladování důvěrné. Jediné strany, které by měly přístupový token vidět, jsou samotná aplikace, autorizační server a server prostředků. Aplikace by měla zajistit, aby úložiště přístupového tokenu nebylo přístupné jiným aplikacím na stejném

zařízení. Přístupový token lze použít pouze přes připojení https, protože jeho předání nekódovaným kanálem by pro třetí strany znamenalo triviální zachycování. [12]

- Refresh token – nese informace potřebné k získání nového přístupového tokenu. Jinými slovy, kdykoli je pro přístup ke konkrétnímu prostředku vyžadován přístupový token, klient může použít obnovovací token k získání nového přístupového tokenu vydaného ověřovacím serverem. Mezi běžné případy použití patří získání nových přístupových tokenů po vypršení platnosti starých nebo získání přístupu k novému zdroji poprvé. Obnovovací tokeny mohou také vypršet, ale mají poměrně dlouhou životnost. Obnovovací tokeny obvykle podléhají přísným požadavkům na úložiště, aby se zajistilo, že nedojde k úniku. Autorizační server je také může zakázat.[13]

Standardy WS-Trust a WS-Federation

- WS-Trust

WS-Trust je specifikace WS- * a standard OASIS, který poskytuje rozšíření WS-Security, konkrétně se zabývá vydáváním, obnovováním a ověřováním bezpečnostních tokenů, jakož i způsoby, jak zjistit, posoudit přítomnost a důvěru zprostředkovatele, vztahy mezi účastníky zabezpečené výměny zpráv. Pomocí rozšíření definovaných ve WS-Trust se aplikace mohou zapojit do zabezpečené komunikace navržené pro práci v rámci webových služeb.

- WS-Federation

WS-Federation (což je zkratka pro Web Services Federation) je protokol, který lze použít k vyjednávání o vydání tokenu. Tento protokol lze použít pro vaše aplikace (například pro aplikaci založenou na systému Windows Identity Foundation) a pro poskytovatele identity (jako jsou služby Active Directory Federation Services nebo Azure AppFabric Access Control Service).

- Standard WS-Federation definuje:
  - Formát a způsoby přenášení metadat o servisech;
  - Funkce jediného přístupu (SSO);
  - Servis atributů obsahující další informace o uživateli;
  - Servis pseudonymů, které dovolují vytváření alternativních uživatelských jmen;
  - Podporu pasivních klientů metodou přesměrování

Můžeme říct, že WS-Federation a WS-Trust řeší stejné problémy, ale s odlišnou realizací a přístupy.

2.3 SSO (Single sign-on)

Ověřování pomocí jednotného přihlášení (SSO) je nyní vyžadováno více než kdy jindy. V dnešní době téměř každý web vyžaduje určitou formu autentizace pro přístup k jeho funkcím a obsahu. S rostoucím počtem webových stránek a služeb se stal nezbytným centralizovaný přihlašovací systém.

Single Sign-On (SSO) je technologie, která umožňuje uživatelům přihlásit se pouze jednou sadou přihlašovacích údajů – například jedním uživatelským jménem a heslem – a dále se plynule pohybovat přes množství aplikací bez nutnosti zadávat znovu přihlašovací údaje. SSO může pracovat na jakémkoli prohlížeči nebo zařízení a usnadňuje uživatelům používání několika různých programů nebo webů, aniž by museli zadávat několik různých uživatelských jmen a hesel, které si musí pamatovat.

Příkladem je účet Google, který přihlášenému uživateli poskytuje přístup ke všem svým službám, aniž by se ho ptal, jestli to chce. V případě, že uživatel přistoupí například na YouTube, tak je automaticky přihlášen jako uživatel, který je přihlášen na účet Google.

2.3.1 Existující řešení

Okta

Okta, podle [14], je populární poskytovatel SSO a služba správy identit. Okta se integruje do podnikových adresářů, stejně jako do on-premise, cloudových a mobilních aplikací. S Okta lze zadat své přihlašovací údaje jednou a přistupovat ke všem svým pracovním aplikacím a nástrojům, aniž byste byli vyzváni k přihlášení v každé z nich.

Okta naváže zabezpečené spojení s prohlížečem uživatele a poté autentizuje uživatele do aplikací spravovaných Okta pomocí jedné ze dvou metod integrace SSO:

- Okta's Secure Web Authentication (SWA);
- Federated (podporující SAML nebo jiný proprietární federovaný ověřovací protokol).

Shibboleth

Shibboleth je softwarový balíček s otevřeným zdrojovým kódem, založený na standardech pro jednotné přihlášení na web přes hranice organizace nebo v rámci nich. Umožňuje webům činit informovaná rozhodnutí o autorizaci pro individuální přístup k chráněným online zdrojům způsobem ochrany soukromí.[15]

Software Shibboleth implementuje široce používané standardy federované identity, hlavně OASIS Security Assertion Markup Language (SAML), aby poskytoval federovaný rámec pro jednotné přihlášení a výměnu atributů. Uživatel se autentizuje pomocí svých organizačních údajů a organizace (nebo

poskytovatel identity) předává poskytovateli služeb minimální informace o totožnosti, nezbytné k povolení rozhodnutí o autorizaci. Shibboleth také poskytuje rozšířené funkce ochrany osobních údajů, které umožňují uživateli a jejich domovskému webu kontrolu atributů pro každou aplikaci.

Keeper

Keeper SSO Connect je softwarová aplikace, implementující standard federované identity SAML, která je nainstalována na podnikové firemní, soukromé nebo veřejné cloudové infrastruktuře. Všechny šifrovací klíče uživatelů jsou spravovány službou Keeper SSO Connect, což zákazníkovi poskytuje plnou kontrolu nad klíči, které se používají k šifrování úschoven koncových uživatelů. Aplikační služba Keeper SSO Connect může být nainstalována na soukromém serveru nebo na cloudovém serveru. Podporovány jsou operační systémy Windows a Linux.[16]

V prostředí Microsoft Windows běží aplikace Keeper SSO Connect jako standardní služba Windows. To zajistí, že služba nebude ukončena, když se někdo odhlásí z počítače, a automaticky se spustí po restartu. Na všech platformách lze SSO Connect nakonfigurovat pro vysokou dostupnost. Aby byla služba vždy aktivní, lze připojení Keeper SSO nainstalovat na více serverů, které jsou umístěny za vyrovnávačem zátěže.

2.3.2 Výhody a nevýhody SSO

Výhody

- Usnadňuje přístup uživatelů k jejich aplikacím tím, že si uživatelé nemusí pamatovat všechna hesla do všech aplikací;
- Snižuje zatížení zapamatování několika hesel;
- Zjednodušuje procesy přidělování a změn hesel;
- Snadná implementace a připojení k novým zdrojům dat.

Nevýhody

- Uživatel má díky SSO jediné heslo do mnoha aplikací. Únik takového hesla nebo celé identity otevírá útočníkovi více dveří.
- Při selhání SSO dojde ke ztrátě přístupu ke všem souvisejícím systémům;
- Nastavení a správa SSO vyžaduje určité úsilí a investice;
- Podvádění identity v externích přístupech uživatele.

2.4 Multi-tenancy

Multi-tenancy je termín v softwarové architektuře odkazující na obsloužení mnoha zákazníků z jedné instance aplikace. Každý zákazník se nazývá tenant. Tenanty mohou mít možnost přizpůsobit některé části aplikace, například barvu uživatelského rozhraní (UI) nebo obchodní pravidla, ale nemohou upravit kód aplikace.[5]

V multi-tenant architektuře fungují ve sdíleném prostředí více instancí aplikace. Tato architektura je schopna fungovat, protože každý tenant je fyzicky integrován, ale logicky oddělen, což znamená, že jedna instance softwaru bude spuštěna na jednom serveru a poté bude sloužit více tenantům. Tímto způsobem může softwarová aplikace v multi-tenant architektuře sdílet sdílenou instanci konfigurací, dat, správy uživatelů a dalších vlastností.

Multi-tenant aplikace mohou sdílet stejné uživatele, displeje, pravidla – ačkoli uživatelé je mohou přizpůsobit do určité míry – a databázová schémata, která si mohou tenanty také přizpůsobit.

2.4.1 Význam multi-tenancy

Multi-tenantní architektury se nacházejí ve veřejných cloudových i soukromých cloudových prostředích, což umožňuje oddělit data každého nájemce od sebe. Například ve veřejném multi-tenant cloudu budou stejné servery použity v hostovaném prostředí k hostování více uživatelů. Každému uživateli je na těchto serverech přidělen samostatný a ideálně bezpečný prostor pro ukládání dat.

Multi-tenancy je důležité pro škálovatelnost veřejných a soukromých cloudů a pomohlo učinit multi-tenancy standardem. Multi-tenant architektura může také pomoci zajistit lepší návratnost investic organizacím a také zrychlit tempo údržby a aktualizací pro tenanty.

2.4.2 Typy multi-tenantní architektury

Existují tři hlavní typy multi-tenantích modelů, všechny s různou úrovní složitosti a nákladů.

- Jediné schéma sdílené databáze je multi-tenant model s multi-tenant databází. Toto je nejjednodušší forma ze tří a má relativně nízkou cenu pro nájemce kvůli použití sdílených zdrojů. Formulář používá jednu instanci aplikace a databázi k hostování nájemců a ukládání dat. Použití jediného schématu sdílené databáze umožňuje snadnější škálování; provozní náklady však mohou být vyšší;
- Další multi-tenancy architektura zahrnuje použití jedné databáze s více schématy. Systém používá jednu instanci aplikace s jednotlivými schématy pro každého tenanta. Tato architektura má navíc vyšší náklady a vyšší režii s každou databází. Je to cenná architektura, protože s daty různých tenantů je třeba zacházet odlišně – jako kdyby museli procházet různými geografickými předpisy;

- Třetí typ multi-tenancy architektury hostí data ve více databázích. Tento model je z hlediska správy a údržby poměrně složitý, ale nájemci mohou být odděleni podle vybraného kritéria.

2.4.3 Výhody a nevýhody multi-tenancy

S multi-tenancy přichází řada výhod a nevýhod. Mezi výhody patří:

- Nižší náklady díky úsporám z rozsahu;
- Tenanty se nemusejí starat o aktualizace, protože jsou vytlačeny poskytovatelem hostitele;
- Tenanty si nemusí starat o hardwaru, na kterém jsou jejich data hostována;
- Poskytovatelé musí sledovat a spravovat pouze jeden systém;
- Architektura je snadno škálovatelná. Nájemci se mohou přizpůsobit podle potřeby – noví uživatelé získají přístup ke stejné instanci v softwaru, obvykle pro zvýšení přírůstkové sazby.

Některé nevýhody, které přicházejí s architekturou multi-tenancy, však zahrnují:

- Multi-tenant aplikace bývají méně flexibilní než aplikace v jiných architekturách, jako je například single-tenant;
- Multi-tenancy je obecně složitější než single-tenancy;
- Multi-tenancy aplikace vyžadují pro zabezpečení přísnější ověření a řízení přístupu.

Kapitola 3

Implementace

3.1 Úvod

V této kapitole se zaměřuji na popis použitých technologií, popis architektury systému a komunikaci mezi jejími jednotlivými vrstvami, ale také na způsob řešení zabezpečení aplikace.

3.2 Požadavky

3.2.1 Funkční požadavky

- Komunikace mezi uživatelem a serverem bude zajištěna pomoci REST API;
- Uživatel bude mít možnost registrace;
- Uživatel bude mít možnost přihlášení, pokud už má účet;
- Uživatel bude mít možnost změny následujících uživatelských údajů: jméno, příjmení, email;
- Rozdělení uživatelů a jejich možnosti podle role;
- Podpora více tenantů;
- Každý uživatel musí patřit právě jednomu tenantu;
- Administrátor bude mít možnost vytvoření nových tenantů;
- Administrátor bude mít možnost blokování a odblokovaní uživatele;
- Administrátor bude mít možnost měnit název tenantu.

3.2.2 Nefunkční požadavky

- Java jako programovací jazyk;
- Integrace s nástrojem pro správu sémantických slovníků TermIt

3.3 Použité technologie

Již od začátku bylo jasné, že budu aplikaci vyvíjet pomocí programovacího jazyka Java, protože mám s ním největší zkušenosti a je jedním z nejpoužívanějších programovacích jazyků pro webové aplikace. Jako framework jsem si zvolil Spring, protože usnadňuje práci a eliminuje potřebu psát velké množství se opakujícího kódu. Co se týče databáze, tak používám kombinace PostgreSQL jako relační databázi a Ontotext GraphDB, což je RDF uložiště pro sémantickou variantu běhu aplikace.

3.3.1 Java

Java je jedním z nejdůležitějších a nejpoužívanějších programovacích jazyků na světě a tento status si drží mnoho let. Mezi důležité kocepty Javy patří:

- JVM (Java Virtual Machine), který poskytuje základ pro nezávislost na platformě;
- Automatizované techniky správy úložiště, například *garbage collection*;
- Syntaxe jazyka, která je podobná syntaxi jazyka C.

Výsledkem je jazyk, který je objektově orientovaný a efektivní pro programování aplikací.[17] Java navíc podporuje primitivní datové typy jako int, char atd.

Kódy Java jsou kompilovány do bajtového kódu nebo strojově nezávislého kódu. Tento bajtový kód je spuštěn na JVM (Java Virtual Machine).

Java je jedním z nejpopulárnějších používaných programovacích jazyků, zejména pro webové aplikace typu klient-server.

3.3.2 REST

REST (Representational State Transfer) je architektura softwarového rozhraní pro distribuované prostředí, jako WWW (World Wide Web), které se používá pro vývoj webových služeb.[18]

REST je, na rozdíl od XML-RPC či SOAP, orientován datově, nikoli procedurálně. Webové služby definují vzdálené procedury a protokol pro jejich volání, REST určuje, jak se přistupuje k datům.[19]

Rozhraní REST je použitelné pro jednotný a snadný přístup ke zdrojům. Zdrojem mohou být data, stejně jako stavy aplikace (pokud je lze popsat konkrétními daty). Všechny zdroje mají vlastní identifikátor URI a REST definuje čtyři základní metody pro přístup k nim, které jsou známé pod označením CRUD, tedy vytvoření dat (Create), získání požadovaných dat (Retrieve), změnu (Update) a smazání (Delete). Tyto metody jsou implementovány pomocí odpovídajících metod HTTP protokolu.

- GET (Retrieve)

Základní metodou pro přístup ke zdrojům je získání zdroje – metoda GET. Setkává se s ní každý uživatel webu dnes a denně – není to nic jiného než požadavek na stránku.

- POST (Create)

  Pro vytvoření dat slouží metoda POST, známá (minimálně) z HTML formulářů.

- DELETE

  Zdroj lze smazat pomocí volání URI HTTP metodou DELETE. Volání je obdobné volání metody GET.

- PUT

  Operace změny je podobná operaci vytvoření (create, metoda POST), s tím rozdílem, že voláme konkrétní URI konkrétního zdroje, který chceme změnit, a v těle předáme novou hodnotu (jako u metody POST). Na rozdíl od POST je u úprav zdroje jeho URI už známá, takže ji lze zadat.

3.3.3 Spring Framework

Spring Framework je platforma Java, která poskytuje komplexní podporu infrastruktury pro vývoj aplikací Java. Jaro zpracovává infrastrukturu, takže se lze soustředit na vaši aplikaci.[20]

Základní funkcionalitou frameworku je Dependency Injection, což jednoduše řečeno znamená, že objekty, které potřebují další objekty, si tyto objekty nevytváří pomocí klíčového slova new samy, ale přenechávají tuto činnost Springu (Spring kontejneru). Ten je pak zodpovědný za vytváření objektů (za celý životní cyklus objektu od vzniku po zánik). Důvod pro tuto činnost je ten, že díky Springu nejsou komponenty těsně se sebou svázány.

Spring Security je součásti Spring Frameworku, je výkonný a vysoce přizpůsobitelný framework pro ověřování a kontrolu přístup. Je to de-facto standard pro zabezpečení Spring aplikací. Spring Security je rámec, který se zaměřuje na poskytování autentizace a autorizace Java aplikacím. [21]

3.3.4 JOPA

Java OWL Persistence API, tedy JOPA, je knihovna, která implementuje podobné řešení jako úprava JPA. Její hlavním cílem je poskytnout možnost pracovat s daty jako s Java objekty. Entitám a jejich vlastnostem se přidávají anotace, které obsahují identifikační IRI namísto běžných JPA anotaci @Table a @Column. Ukázku kódu s použitím JOPA je vidět na obrazku 3.1

```
@ParticipationConstraints(nonEmpty = true)
@OWLDataProperty(iri = Vocabulary.user_username)
@Column(name = "username")
private String username;
```

Obrázek 3.1: Příklad použiti JOPA anotace

3.3.5 Databáze

V rámci implementace autorizačního systému byly použity dvě úložiště dat kvůli tomu, že jsou naimplementovány dva režimy běhu aplikace. První režim je založen na JPA a používá relační databázi, druhý – sémantický režim, který ukládá data do triple storu.

Řešeno je to tak, že je třeba aktivovat jeden ze dvou *profilů*: *jpa* (relační databáze) (Obrázek 3.4) a *sem* (sémantický režim) (Obrázek 3.3), který se nastavuje před spuštěním aplikace v souboru *application.properties* jako *spring.profiles.active* (Obrázek 3.2).

```
# Application profile: sem/jpa
spring.profiles.active=jpa
```

Obrázek 3.2: Příklad nastavení režimu aplikace

Na tom, jaká varianta bude zvolena, záleží, které DAO (Data Access Objekt) se budou používat, buď jpa, nebo sémantické. Každá DAO třída má nastavený profil pomocí anotace @Profile, jak je vidět na obrázku 3.3.

```
@Repository
@Profile("sem")
public class UserSemanticDao implements UserDao {
```

Obrázek 3.3: DAO v sémantickém profilu

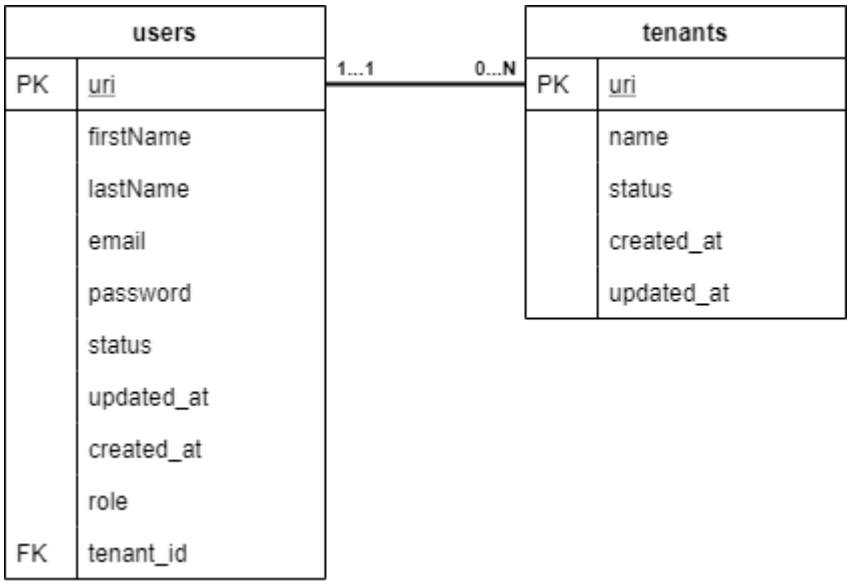
V tomto případě aplikace bude běžet nad RDF úložištěm a ukládat a vybírat data právě z něho.

```
@Repository
@Profile("jpa")
public interface UserRepository extends JpaRepository<User, Long>, UserDao {
```

Obrázek 3.4: DAO s režimu JPA

Aplikace má dvě hlavní entity, které jsou propojené mezi sebou, jako je zobrazeno na obrázku 3.5. První je users (uživatele), která v sobě bude chránit

data uživatelů (uri, uživatelské jméno, jméno, příjmení, email, heslo, status, datum vytvoření, datum poslední opravy, role a identifikátor tenantu, kterému daný uživatel patří). Druhá entita – tetants, slouží ke chránění dat o tenantu (uri, jméno, status, datum vytvoření, datum poslední opravy).



Obrázek 3.5: Návrh databáze

- Relační databáze

  Používám PostgreSQL, což je systém pro správu relačních databází s otevřeným zdrojovým kódem. Pro komunikace s relační databází je použito JPA, což je standard programovacího jazyka Java, který umožnuje objektově-relační mapování, které usnadňuje práci s ukládáním objektů do databáze a naopak.

- Tripple store

  V sémantickém režimu je používáno Ontotext GraphDB, což je RDF uložiště, které je optimalizováno pro správu metadat, analýzu grafů a publikování dat. Pro práce s daty, které se ukládají v tripple storu je používaný dotazovací jazyk SPARQL a OWL (Web Ontology Language) pro popis dat a jejich propojení mezi sebou.

3.3.6 JSON-LD

JSON-LD (JavaScript Objekt Notation for Linked Data) je jednou z metod přenosu souvisejících dat pomocí textového formátu. Je založen na již úspěšném formátu JSON a poskytuje způsob, jak pomoci datům JSON spolupracovat na webu. JSON-LD je ideální datový formát pro programovací prostředí, REST webových služeb.[22]

JSON-LD používá koncept kontextu pro podporu datového modelu RDF. Kontext spojuje vlastnosti objektů v dokumentu JSON s prvky ontologie. Pro

vytvoření shody mezi syntaxi JSON-LD a RDF musí být hodnoty JSON-LD převedeny na konkrétní typ nebo označeny značkou jazyka. Kontext může být přímo v dokumentu JSON-LD nebo může být umístěn na adrese URL určené pro kontext. Například pro běžné dokumenty JSON lze kontext určit v záhlaví Propojení protokolu HTTP.

Jakákoli data ve formátu JSON-LD a ve formátu JSON jsou sadou párů klíč-hodnota. Na rozdíl od formátu JSON poskytuje formát JSON-LD vyhrazené klíče, které lze použít k definování kontextu popisu nebo k vazbě objektů různými způsoby. Například „@context“ definuje slovník objektů a „@id“ slouží unikátním identifikátorem IRI.

3.3.7 Maven

Maven je nástroj pro správu a porozumění projektu na základě popisu jejich struktury v souborech POM (Project Object Model), což je schéma v jazyce XML, který vývojářům poskytuje kompletní rámec sestavení životního cyklu.

V případě prostředí více vývojových týmů může Maven nastavit způsob práce podle standardů ve velmi krátké době. Protože většina nastavení projektu je jednoduchá a znovu použitelná, Maven usnadňuje život vývojářům při vytváření kontrol, sestavování a testování nastavení automatizace.

Maven je velmi vhodný takové účely, jako build a použití externích závislostí software.[23]

3.3.8 GIT

Git je bezplatný open source systém pro správu verzí, který původně vytvořil Linus Torvalds v roce 2005. Na rozdíl od starších centralizovaných systémů pro správu verzí, jako jsou SVN a CVS, je Git distribuován: každý vývojář má lokálně celou historii svého úložiště kódů. Díky tomu je počáteční klon úložiště pomalejší, ale následné operace, jako je commit, blame, dif, merge a log, jsou výrazně rychlejší.

Git má také vynikající podporu pro větvení, slučování a přepisování historie úložiště, což vedlo k mnoha inovativním a výkonným pracovním postupům a nástrojům. Pull žádosti jsou jedním z takových populárních nástrojů, které umožňují týmům spolupracovat na větvích Git a efektivně kontrolovat každý další kód. Git je dnes nejrozšířenějším systémem pro správu verzí na světě a je považován za moderní standard pro vývoj softwaru.[24] Gitlab je příkladem specializované stránky pro umístění vzdáleného repositáře.

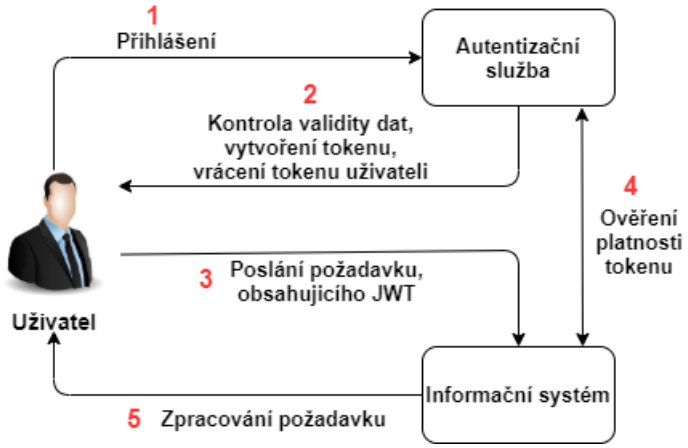
3.4 Use-cases

Aplikace má dva druhy uživatelů podle jejich role – běžný uživatel a admin. V této části jde o hlavních případech užiti, které poskytuje autorizační služba, jako je registrace, přihlášení, vytvoření tenantu a změna dat.

3.4.1 Použiti v rámci informačního systému

Obrázek 3.6 ukazuje, jak scénář, který popisuje použití v rámci informačního systému.

1. Uživatel se přihlásí s použitím uživatelského jména a hesla;
2. Proběhne kontrola dat. Jestliže bude úspěšná, vytvoří se token, který bude vrácen uživateli;
3. Uživatel při přístupu na REST API aplikace posílá token;
4. Aplikace vezme jeho token a ověří platnost pomoci autorizační služby;
5. Pokud bude token platný, zpracuje požadavek uživatele.



Obrázek 3.6: Použití v rámci informačního systému

3.4.2 Registrace běžného uživatele

Jestli se člověk chce zaregistrovat jako nový uživatel, tak musí použit požadavek typu POST s URL */api/v1/users/*. Tělo požadavku by mělo obsahovat sadu informací o uživateli, jako uživatelské jméno, jméno, příjemní, email a heslo ve formátu JSON (Obrázek 3.7), v headeru musí být uveden název tenantu, do kterého se uživatel chce zaregistrovat.

```
{
    "username":"test_user",
    "firstName":"user",
    "lastName":"test",
    "email":"test_user@fel.cvut.cz",
    "password":"test"
}
```

Obrázek 3.7: Příklad těla požadavku na registrace

Jestli uživatel neuvede nějakou položku buď v těle požadavku nebo název tenantu v headeru, tak server jeho požadavek odmítne a vypíše zprávu, obsahující informace o tom, co je špatně. V případě, že všechna data jsou uvedena ve správném formátu a požadavek je v pořádku, dostane uživatel odpověď s kódem *201 Created*, což bude znamenat, že nový uživatel byl vytvořen a uložen v databázi.

3.4.3 Přihlášení

Klient, buď webový prohlížeč nebo mobilní aplikace, který se chce přihlásit, musí mít účet v systému a poslat POST požadavek */api/v1/users/login*, v těle kterého jsou uvedené uživatelské jméno a heslo (Obrázek 3.8) a v headeru tohoto požadavku, stejně jako v požadavku na registraci, musí uvádět název tenantu, do kterého se chce přihlásit. Důležitým je že se uživatel může přihlásit jenom do toho tenantu, ve kterém má účet.

```
{
    "username":"test_user",
    "password":"test"
}
```

Obrázek 3.8: Příklad těla požadavku na přihlášení

Při zpracování požadavku server kontroluje, jestli takový uživatel existuje v databázi, uvedl-li správné heslo a jestli má aktivní status. Pokud kontrola byla úspěšná, generuje se JWT (JSON Web Token), který obsahuje zakódovaná data uživatele a je zabezpečen symetrickým algoritmem HS256 s tajným klíčem. Tento token je odpovědí na požadavek klienta. Na základě tohoto tokenu se dělají další požadavky.

3.4.4 Update uživatele

Jestli chce uživatel svá data upravit, buď změnit jméno, příjmení nebo email, tak k tomu potřebujete požadavek typu PUT */api/v1/users/current*, který ve svém těle bude mít nová data, které si uživatel chce nastavit (Obrázek 3.9). Důležitým je skutečnost, že tělo musí obsahovat kompletní sadu dat (jméno, příjmení, email) i když je požadována změna pouze jednoho parametru.

```
{
    "firstName":"new_firstName",
    "lastName":"new_lastName",
    "email":"new_email@fel.cvut.cz"
}
```

Obrázek 3.9: Příklad těla požadavku na změnu dat uživatele

V případě, že všechno bylo uděláno správně, obdrží zadavatel odpověď se statusem *200 OK*, která bude obsahovat kompletní uživatelská data s obnovenými daty.

3.4.5 Zobrazení uživatelských dat

Pokud se klient potřebuje dozvědět nějakou informaci o uživateli, pak může použít GET požadavek */api/v1/users/current*. Jestli takový uživatel existuje a token klienta je platný, dostane odpověď, která bude obsahovat data o uživateli (Obrázek 3.10).


```
{
    "@id": "http://example.org/users/test_usr",
    "@type": [
        "http://example.cz/api/v1/user"
    ],
    "http://example.cz/api/v1/firstName": "user",
    "http://example.cz/api/v1/lastName": "test",
    "http://example.cz/api/v1/user_role": "ROLE_USER",
    "http://example.cz/api/v1/username": "test_usr",
    "http://example.cz/api/v1/status": "ACTIVE",
    "http://example.cz/api/v1/email": "test_user@fel.cvut.cz",
    "http://example.cz/api/v1/user_tenant": {
        "@id": "http://example.org/tenants/admin_tenant",
        "@type": [
            "http://example.cz/api/v1/tenant"
        ],
        "http://example.cz/api/v1/created": 1589291811256,
        "http://example.cz/api/v1/lastModified": 1590152826439,
        "http://example.cz/api/v1/status": "ACTIVE",
        "http://example.cz/api/v1/tenant_name": "admin_tenant"
    }
}
```


Obrázek 3.10: Příklad zobrazení uživatelských dat

3.4.6 Změna hesla

V případě, že uživatel potřebuje změnit heslo, musí použít HTTP požadavek typu PUT s URL */api/v1/users/current/password*. Tělo požadavku musí obsahovat nové heslo.

3.4.7 Vytvoření tenantu

Vytvoření nového tenantu je administrátorská funkce, což znamená že ji může splnit jenom ten uživatel, který má role *ROLE_ADMIN*. Provede se to pomocí POST požadavku s URL */api/v1/admin/tenants/registration*, který ve svém těle musí obsahovat název budoucího tenantu. Po přijetí takového požadavku server zkontroluje pomocí tokenu, jestli uživatel má roli administrátora a jestli je všechno v pořádku, pak pošle odpověď se statusem *201 Created*.

3.4.8 Požadavek o zobrazení dat tenantu

Pokud klient chce, aby se zobrazila data tenantu, tak musí mít roli *ROLE_ADMIN* stejně jako pro vytvoření nového tenantu. Používá se na to GET požadavek s

URL */api/v1/tenants/test_tenant*, který obsahuje ve svém URL název tenantu. Jako odpověď dostane data o tenantu ve formátu JSON-LD (Obrázek 3.11).


```
{
    "@id": "http://example.org/tenants/test_tenant",
    "@type": [
        "http://example.cz/api/v1/tenant"
    ],
    "http://example.cz/api/v1/created": 1589120870206,
    "http://example.cz/api/v1/status": "ACTIVE",
    "http://example.cz/api/v1/tenant_name": "test_tenant"
}
```


Obrázek 3.11: Příklad zobrazení dat tenantu

3.4.9 Deaktivace a aktivace uživatele

Administrátor, jinými slovy uživatel, který má roli *ROLE_ADMIN* může deaktivovat (příp. aktivovat) uživatele pomocí požadavku typu *DELETE* (příp. POST) s URL */api/v1/admin/test_user/status*, který ve svém URL má uživatelské jméno tohoto klienta, na kterého je funkce směrovaná. Deaktivace uživatele bude pro něj znamenat, že nebude mít možnost se přihlásit, tím pádem nebude mít přístup autorizační službě.

Kapitola 4
Testování

Úvod

Po implementaci aplikace bylo potřeba ověřit, jestli splňuje požadavky na svou funkčnost a funguje-li vše tak, jak by mělo, jinými slovy, bylo tedy třeba aplikaci otestovat. Testování bylo provedeno třemi způsoby, první byl pomocí Unit testů, druhý – pomocí programu zvaného Postman a naposled ověřila se správnost implementace propojením s TermItem.

4.1 Unit testy

Testování pomocí unit testů je fáze testování, která se zaměřuje na nejmenší testovatelné části aplikace. Jde o testování jednotlivých tříd a metod.

K testování byl použit open source framework JUnit, který poskytuje anotace pro identifikací testovacích metod a asserty pro vyhodnocování očekávaných výsledků. Dalším frameworkem, který byl použitý pro testování byl Mockito, který pomáhá izolovaně otestovat funkčnost tříd.

Pokrytí Unit testy se u důležitých balíčků aplikace, právě model, repository, servis, a controller bliží k 100%, avšak celkové pokrytí projektů skladá 53%. Byly na to použité následující testovací strategie: repository nebo DAO bylo otestováno jenom v sémantické variantě nad in-memory triple storem, protože ve variantě s použitím JPA je DAO generované Springem a testování nepotřebuje. Servisy byly otestované nad in-memory úložištěm. Co se týče controllerů, tak ty byly testované nad mocky business logiky aplikací.

Pro jistotu byly provedeny i negativní testovací scénáře z důvodu ujištění, že aplikace funguje správně i v případě chybných vstupních parametrů. Celkový počet testů je 50.

4.2 Testování s Postmanem

Testování API zahrnuje kontrolu, zda splňuje očekávání funkčnosti, spolehlivosti, výkonu a zabezpečení a vrací správnou odpověď.

Testování API se používá k určení, zda je výstup dobře strukturovaný a užitečný pro jinou aplikaci, či nikoli, kontroluje odpověď na základě vstupního

(request) parametru a kontroluje, kolik času API trvá k načtení a autorizaci dat.

Postman je aplikace pro testování API, zasláním požadavku na webový server a získáním zpětné odpovědi. Níže jsou uvedeny některé funkce Postmanu, pomocí kterých lze zvýšit produktivitu testování.

- Umožňuje uživatelům nastavit všechny hlavičky (headers) a soubory cookies, které API očekává, a zkontroluje odpověď;
- Postman umožňuje uživatelům vytvářet kolekce pro postupné volání API. Každá kolekce může vytvářet podsložky a více požadavků. To pomáhá při organizaci testovacích souprav;
- Kolekce a prostředí lze importovat nebo exportovat, což usnadňuje sdílení souborů. Přímý odkaz lze také použít ke sdílení kolekcí;
- Ke každému volání API lze přidat kontrolní body, jako je ověření úspěšného stavu odpovědi HTTP, což pomáhá zajistit pokrytí testem.

Testování pomocí Postmanu se provádělo i během vývoje aplikace pro testování endpointů, právě pro kontrolu správnosti jejich fungování a časovou náročnost. Požadavky se vytvářely pro každý existující v REST kontrolérech endpoint. Pro některé endpointy bylo vytvořeno několik požadavků, obsahujících stejný URL, ale různá těla a headery pro kontrolu chování serveru v případě chybných argumentů nebo parametrů.

Kromě jednotlivých požadavku byly vytvořeny i kolekce požadavků, takzvané testovací scénáře. Jejich výsledky jsou exportované ve formátu JSON a přidané do projektu v zvláštní složce „postman“. Seznám scénářů je uvedeny níže a některé z nich s dentálním popisem.

- **Název kolekce**: ActionsWithoutLogin
  Nepřihlášeny uživatel se snaží dostat informace o uživateli, změnit jeho data a změnit heslo.
  - **Název požadavku**: TestGetUserInfo
    Požadavek: GET /users/current
    Očekáváno: *403 Forbidden*
    Výsledek: *403 Forbidden*
  - **Název požadavku**: TestChangePassword
    Požadavek: PUT /users/current/password
    Tělo: JSON objekt s atributem „password“
    Očekáváno: *403 Forbidden*
    Výsledek: *403 Forbidden*
  - **Název požadavku**: TestUpdateUser
    Požadavek: PUT /users/current
    Tělo: JSON objekt s atributy „firstName“, „lastName“, „email“
    Očekáváno: *403 Forbidden*
    Výsledek: *403 Forbidden*

- **Název kolekce**: ChangePasswordTest
  Nový uživatel se registruje, pak se přihlásí a změní si heslo.
  - **Název požadavku**: TestCreateUserForPasswordChange
    Požadavek: POST /users/
    Hlavička: tenant - admin_tenant
    Tělo: JSON objekty s atributy „username“, „firstName“, „lastName“, „email“, „password“
    Očekáváno: *201 Created*
    Výsledek: *201 Created*
  - **Název požadavku**: TestLoginUser
    Požadavek: POST /users/login
    Hlavička: tenant - admin_tenant
    Tělo: JSON objekt s atributy „username“ a „password“
    Očekáváno: *200 OK*
    Výsledek: *200 OK*
  - **Název požadavku**: GetUserInfoBeforeChangePassword
    Požadavek: GET /users/current
    Očekáváno: *200 OK* a data uživatele
    Výsledek: *200 OK* a data uživatele
  - **Název požadavku**: ChangePassword
    Požadavek: PUT /users/current/password
    Tělo: JSON objekt s atributem „password“
    Očekáváno: *204 No Content*
    Výsledek: *204 No Content*
  - **Název požadavku**: TestLoginWithNewPassword
    Požadavek: POST /users/login
    Hlavička: tenant - admin_tenant
    Tělo: JSON objekt s atributy „username“ a „password“
    Očekáváno: *200 OK*
    Výsledek: *200 OK*

- **Název kolekce**: DisableAndEnableUserTest
  Nový uživatel se registruje,pak se přihlásí a administrátor jeho účet zablokuje, ověří status a pak odblokuje a zase ověří status.

- **Název kolekce**: UpdateUserTest
  Nový uživatel se registruje,pak se přihlásí a změní uživatelské jméno, příjmení a email.

- **Název kolekce**: LoginWithWrongPasswordTest
  Nový uživatel se registruje,pak se zkusí přihlásit se špatným heslem.

- **Název kolekce**: UserTryAdminFunctions
  Nový uživatel se registruje,pak se přihlásí a zkusí vytvořit nový tenant, zablokovat účet a odblokovat ho.

4.3 Integrace s TermItem

Dalo se propojit mojí autorizační službu s nástrojem pro správu sémantických slovníků, který se nazývá TermIt. Po integraci proces byl následující: objevuje se login obrazovka TermItu, kde uživatel zadává uživatelské jméno a heslo, jež se v rámci autentizačního procesu pošlou na autorizační službu a ta vrátí uživateli token. Potom si TermIt stáhne aktuální metadata uživatele a synchronizuje je s lokální kopií (dělá se kvůli konzistenci dat). Nakonec je token vrácen uživateli v autorizační hlavičce. Při dalších requestech na TermIt dojde vždy k ověření platnosti tokenu oproti autorizační službě.

Kapitola 5
Závěr

5.1 Shrnutí

Cílem této bakalářské práce bylo vytvoření sémantické autorizační služby, která by umožňovala správu uživatelů a jejich rolí a kterou by bylo možné využít v doméně informačních systémů využívajících zmíněné technologie sémantického webu s podporou více tenantů. Pro splnění tohoto zadání bylo nutno se nejprve seznámit se sémantickými technologiemi a jejích použitím v rámci autorizačních služeb.

Dalším krokem byla analýza již existujících řešení v oblasti autorizačních systémů. V rámci této analýzy byly identifikovány výhody a nevýhody každé metody a porovnány jsem je mezi sebou. Výsledky porovnání jsou uvedeny v tabulce 5.1.

Na základě těchto výsledků jsem dal přednost technologii autorizace pomocí hesla s použitím JWT pro tvorbu vlastní aplikace.

Následně byla provedena analýza technologie jediného přístupu, takzvaná SSO (Single Sign On), která umožňuje uživatelům přihlásit se pouze jednou sadou přihlašovacích údajů – například jedním uživatelským jménem a heslem – a dále se plynule pohybovat přes množství aplikací bez nutnosti zadávat znovu přihlašovací údaje.

Posledním krokem analýzy bylo seznámení s architekturou multi-tenancy, ve které jediná instance softwarové aplikace slouží více zákazníkům, což bylo součástí zadání bakalářské práce.

Dále následovala implementace navržené přihlašovací služby, která by splňovala všechny podmínky a byla založena na provedené analýze. Implementace byla provedená v jazyce Java verzi 8 s použitím knihovny JOPA, která poskytuje možnost práce se sémantickými daty jako s Java objekty. Bohužel jsem špatně odhadl objem práce, kterou by při vytváření takto robustního řešení bylo potřeba vykonat kvůli tomu, že seznámení s novými pro mě technologiemi zabralo více času, proto jsem v průběhu nestihl implementovat některé z mnoha svých nápadů.

Výsledek vývoje jsem otestoval pomoci Unit testů a aplikace pro testování API, nazývané Postman. Testování umožnilo ověřit, že identifikované funkční požadavky byly splněny, a že aplikace funguje tak, jak má.

Díky práci na tomto projektu jsem si rozšířil znalosti v oblasti autorizačních

Způsob	Základní použití	Protokoly	Bezpečnost
Heslo	Autorizace uživatelů	HTTP, Forms	Průměrná
Certifikáty	Autentizace uživatelů v bezpečných aplikacích, autentifikace uživatelů	SSL/TLS	Vysoká
Jednorázové heslo	Doplňková autentizace uživatelů (two-factor authentication)	Forms	Vysoká
Přístupový klíč	Autentifikace služeb a aplikace	-	Vysoká
Tokeny	Delegování autentizace uživatelů, delegována autorizace aplikaci	SAML, WA-Trust, WA-Federation, OAuth, OpenID Connect	Vysoká

Tabulka 5.1: Porovnání autorizačních metod

systémů a získal povědomí o sémantických technologiích a architektuře multi-tenantu a jejich využitích v rámci vývoje autorizačních služeb. Získal jsem také zkušenosti s typografickým systémem LaTeX a psaním technické zprávy. Výsledkem této práce byla vlastní zkušenost s tvorbou většího projektu a zlepšil jsem si znalost programování webových aplikací a frameworku Spring, který slouží pro tyto účely.

5.2 Výhled do budoucna

Implementovaná aplikace funguje a splňuje funkční a nefunkční požadavky, ale je jen základem a dá se zlepšit. Přidal bych funkce přihlášení pomocí jiných metod, například Access Key. Dá se zlepšit i zabezpečení aplikace použitím jiného algoritmu hashování hesla a vytvoření tokenu.

Dala by se rozvíjet i uživatelská funkcionalita o možnost mít jeden účet pro několik tenantů a možnost se mezi nimi pohybovat. Přidal bych i omezení na počet chybných pokusů přihlášení tak, aby se účet blokoval.

Dal by se udělat měřič síly hesla a rady pro uživatele při registraci tak, aby ukazovaly, jestli je heslo lehce zlomitelné nebo ne.

Příloha A

Literatura

[1] Říha Zdeněk a kolektiv Rak Roman, Matyáš Václav. *Biometrie a identita člověka*. Grada Publishing, a.s., 2008. ISBN: 9788024723655.

[2] České Vysoké Účetní Technické. Informační systémy a technologie. pojmy. Dostupné z: `https://ist.cvut.cz/pojmy/autentizace/`, navštíveno 5.1.2020.

[3] ITBIZ. Slovník. informační technologie it., 2011. Dostupné z: `https://www.itbiz.cz/slovnik/informacni-technologie-it/autorizace`, navštíveno 5.1.2020.

[4] Jim Hendler Dean Allemang. *Semantic Web for the Working Ontologist: Effective Modeling in RDFS and OWL*. Morgan Kaufmann, 2011. ISBN: 9780080558387.

[5] Ado Kukic. *The Definitive Guide to Single Sign On*. OAuth, 2019.

[6] Ontotext. What is rdf triplestore? `https://www.ontotext.com/knowledgehub/fundamentals/what-is-rdf-triplestore/`, navštíveno 18.4.2020.

[7] Chris Adams Henrik Walther, Mark Horninger. *The Best Damn Exchange, SQL and IIS Book Period*. Syngress, 2011. ISBN: 9780080556888.

[8] Mark Stanislav. *Two-Factor Authentication*. IT Governance Ltd, 2015. ISBN: 9781849287340.

[9] Zooom CZ; Zápisník vývojáře webových stránek. JSON Web Tokens (JWT), 2019. Dostupné z: `https://zooom.cz/json-web-tokens-jwt/`, navštíveno 10.3.2020.

[10] Matthias Biehl. *OpenID Connect – End-user Identity for Apps and APIs*. API-University Press, 2019. ISBN: 9781979718479.

[11] SecureAuth. An introduction to saml (security assertion markup language). Dostupné z: `https://www.secureauth.com/blog/introduction-to-saml`, navštíveno 8.4.2020.

[12] OAuth. Access tokens. Dostupné z: https://www.oauth.com/oauth2-servers/access-tokens/, navštíveno 16.5.2020.

[13] Auth0. Refresh tokens. Dostupné z: https://auth0.com/blog/refresh-tokens-what-are-they-and-when-to-use-them/, navštíveno 18.4.2020.

[14] Okta. Overview of managing apps and sso. Dostupné z: https://help.okta.com/en/prod/Content/Topics/Apps/Apps_Overview_of_Managing_Apps_and_SSO.htm, navštíveno 18.4.2020.

[15] Slibboleth. What 's slibboleth? Dostupné z: https://www.shibboleth.net/index/, navštíveno 18.4.2020.

[16] Keeper. Overview. Dostupné z: https://docs.keeper.io/sso-connect-guide/overview, navštíveno 18.4.2020.

[17] Oracle. Java developer's guide. overview of java. Dostupné z: https://docs.oracle.com/en/database/oracle/oracle-database/12.2/jjdev/Java-overview.html#GUID-17B81887-C338-4489-924D-FDDF2468DEA7, navštíveno 17.5.2020.

[18] Roy Thomas Fielding. Representational state transfer., 2000. Dostupné z: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm, navštíveno 18.4.2020.

[19] Sanjay Patni. *Pro RESTful APIs: Design, Build and Integrate with REST, JSON, XML and JAX-RS*. Apress, 2017. ISBN: 9781484226650.

[20] Spring. Overview of spring framework. Dostupné z: https://docs.oracle.com/en/database/oracle/oracle-database/12.2/jjdev/Java-overview.html#GUID-17B81887-C338-4489-924D-FDDF2468DEA7, navštíveno 17.5.2020.

[21] Spring. Spring security. Dostupné z: https://spring.io/projects/spring-security, navštíveno 15.4.2020.

[22] JSON-LD. Json for linking data. Dostupné z: https://json-ld.org/, navštíveno 8.1.2020.

[23] Tutorials Point. Simply Easy Learning. Maven - overview. Dostupné z: https://www.tutorialspoint.com/maven/maven_overview.htm, navštíveno 18.4.2020.

[24] Atlassian Bitbucket. Getting git right. Dostupné z: https://www.atlassian.com/git, navštíveno 18.4.2020.

Příloha B

Obsah přiloženého CD

```
/
├── login-demo
│   ├── .idea
│   ├── .mvn
│   ├── postman ..... kolekce postman testů
│   ├── scr
│   │   ├── main
│   │   │   ├── java ..... adresář s java soubory
│   │   │   └── resources
│   │   │       └── application.properties ..... konfigurační soubor
│   │   └── test
│   │       ├── java ..... adresář s testovacími soubory
│   │       └── resouces
│   └── pom.xml
└── BP2020-tarnodmy.pdf ..... bakalářská práce ve formátu PDF
```

Příloha C

Návod ke spuštění

Pro spuštěni dané aplikace je třeba mít nainstalované následující technologie:

- Java (ve verzi 8 a vyšší);
- Apache Maven;
- Vývojové prostředí (Intellij IDEA, NetBeans atd.);
- PostgreSQL.

C.1 Spuštění aplikace

1. Otevřete projekt ve vývojovém prostředí;
2. Pro nastavení režimu otevřete soubor **application.properties** a nastavte proměnu **spring.profiles.active** na **sem** nebo **jpa**;
3. Pro nastavení databáze buď relační nebo sémantické otevřete soubor **application.properties** a upravte si adresu databáze, uživatelské jméno a heslo;
4. V záložce **Maven** rozklikněte **login-demo**, pak **Lifecycle** a **compile**;
5. Spusťte třídu **LoginDemoApplication**.

Aplikace se spustí na portu 8080 a bude možné posílat HTTP požadavky pomocí Postmanu.