VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

SPRÁVA IDENTIT V INFORMAČNÍCH SYSTÉMECH

IDENTITY MANAGEMENT IN INFORMATION SYSTEMS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **THANH QUANG TRAN**
AUTHOR

VEDOUCÍ PRÁCE — **doc. Ing. RADEK BURGET, Ph.D.**
SUPERVISOR

BRNO 2024

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

155014

Ústav:	Ústav informačních systémů (UIFS)
Student:	**Tran Thanh Quang**
Program:	Informační technologie
Název:	**Správa identit v informačních systémech**
Kategorie:	Informační systémy
Akademický rok:	2023/24

Zadání:

1. Seznamte se se současnými technologiemi pro implementaci webových klient-server informačních systémů se zaměřením na řešení autentizace a autorizace uživatelů.
2. Prostudujte existující systémy a protokoly pro správu identit a centralizovanou autentizaci v informačních systémech.
3. Na základě konzultací s vedoucím navrhněte architekturu informačního systému založenou na mikroslužbách s využitím centrální autentizace.
4. Implementujte ukázkový informační systém vhodně demonstrující použitelnost navržené architektury.
5. Proveďte testování navrženého systému.
6. Zhodnoťte dosažené výsledky.

Literatura:

- Bertocci, V.: OAuth2 and OpenID Connect: The Professional Guide, Okta, Inc., 2022
- Dále dle pokynů vedoucího.

Při obhajobě semestrální části projektu je požadováno:
Body 1 až 3.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce:	**Burget Radek, doc. Ing., Ph.D.**
Vedoucí ústavu:	Kolář Dušan, doc. Dr. Ing.
Datum zadání:	1.11.2023
Termín pro odevzdání:	9.5.2024
Datum schválení:	30.10.2023

Abstrakt

Tato práce se zabývá problematikou správy identit na webu a implementací informačního systému, který tuto problematiku řeší. Navržený informační systém využívá systému Keycloak pro správu identit a delegaci procesu autentizace centrálně na třetí stranu. Díky Keycloaku lze integrovat externí zdroje identit pomocí široce používaných protokolů řešící jednotné přihlášení jako SAML 2.0 a OpenID Connect. Samotný informační systém je postaven na architektuře mikroslužeb, kde jednotlivé mikroslužby jsou implementovány v programovacím jazyce Python. Webový klient informačního systému je implementován ve webovém frameworku Vue.js s rozšířením Vuetify, které slouží pro snadné vytváření webového uživatelského rozhraní.

Abstract

This thesis deals with the issue of identity management on the web and the implementation of an information system that solves this issue. The proposed information system utilizes Keycloak for identity management and delegating the authentication process centrally to a third party. Thanks to Keycloak, external identity sources can be integrated using widely used protocols such as SAML 2.0 and OpenID Connect. The information system itself is built on a microservices architecture, where individual microservices are implemented in the Python programming language. The web client of the information system is implemented in the Vue.js web framework with the Vuetify extension, which is used for easy creation of the web user interface.

Klíčová slova

správa identit, autentizace, autorizace, jednotné přihlášení, Json Web Token, SAML 2.0, OAuth 2.0, OpenID Connect, REST API, mikroslužby, kontejnery, Docker, Kubernetes, Python, Flask, JavaScript, Vue.js, Vuetify

Keywords

Identity Management, Authentication, Authorization, Single sign-on, Json Web Token, SAML 2.0, OAuth 2.0, OpenID Connect, REST API, Microservices, Containers, Docker, Kubernetes, Python, Flask, JavaScript, Vue.js, Vuetify

Citace

TRAN, Thanh Quang. *Správa identit v informačních systémech.* Brno, 2024. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce doc. Ing. Radek Burget, Ph.D.

Správa identit v informačních systémech

Prohlášení

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

.....
Thanh Quang Tran
9. května 2024

Poděkování

Rád bych poděkoval panu doc. Ing. Radku Burgetovi, Ph.D za rady, pravidelné konzultace a trpělivost při tvorbě této bakalářské práce. Dále bych chtěl vzdát velké díky mé rodině, která mě při studiu i přes překážky a problémy vždy podporovala.

Obsah

Seznam obrázků

Kapitola 1

Úvod

Tato bakalářská práce se zabývá výzkumem protokolů jednotného přihlášení na webu, systémům řešící správu identit a tvorbou demonstračního informačního systému, který tyto protokoly a systémy integruje. Správa identit a správné řešení autentizace jsou klíčovým prvkem moderních webových aplikací a jejich bezpečného provozu.

Jednotné přihlášení s centrální autentizací umožňuje uživatelům pohodlně využívat služeb informačního systému s použitím již existující identity, kterou uživatel vlastní u jiné organizace či služby. Uživatel tak není nucen opakovaně vytvářet nové účty a používat stejná hesla u různých služeb, která mohou být při nesprávné implementaci systému ohrožena. Opakované zadávání hesel také zpravidla odrazuje uživatele a negativně ovlivňuje jejich uživatelskou zkušenost s informačním systémem, který neposkytuje možnost jednotného přihlášení pomocí externích identit [29].

Moderním trendem při vývoji webových aplikací je architektura mikroslužeb, která umožňuje rozdělení funkcionality aplikace na menší a nezávislé komponenty, nazývané mikroslužby. Při použití této architektury se může webová aplikace skládat z mnoha různých mikroslužeb, které mohou být provozovány v různých serverech a technologiích. V takové architektuře, kde by uživatel jinak musel opakovaně zadávat své přihlašovací údaje pro každou jednotlivou službu nebo mikroslužbu, je jednotné přihlášení s centrální autentizací vhodným nástrojem pro zjednodušení autentizace a zabezpečení identit uživatelů.

Práce se tedy zaměřuje na implementaci informačního systému, který je založen na architektuře mikroslužeb a bude napojen na centrální systém správy identit Keycloak. Autentizace uživatelů bude delegována na tento systém a umožní autentizaci za pomocí externích identit využívajících protokolů jednotného přihlášení.

Práce má celkově 8 kapitol. Kapitola 2 se zabývá lehkým úvodem do problematiky správě identit v informačních systémech. Kapitola 3 do základu popíše běžně používané mechanismy autentizace a autorizace na webu. Kapitola 4 prozkoumá populární specifikace a protokoly jednotného přihlášení s centrální autentizací a autorizací, konkrétně SAML 2.0, OAuth 2.0 a OpenID Connect. Kapitola 5 se věnuje implementací webového systému, což zahrnuje přípravu Keycloaku jako centrální systém pro správu identit a implementaci informačního systému, využívající služeb Keycloaku. Kapitola 6 se zabývá testování výsledného informačního systému. Práce končí kapitolou 7, která obsahuje shrnutí dosažených výsledků a potenciálních vylepšení v rámci bezpečnosti či rozšíření systému.

Kapitola 2

O správě identit

Správa identit, ang. Identity Management, nebo pod jiným názvem *správa identit a přístupu*, ang. Identity and Access Management, má za úkol spravovat uživatele či jiné entity v určité organizaci a zajistit, aby správní uživatelé měli přístup ke správným zdrojům nebo aby správné zdroje byly přiřazeny správnému uživateli. Hlavní motivací ke správě identit bylo omezení přístupu k chráněným zdrojům jako například k souborům, aplikacím nebo sítím. Pro zajištění správného přístupu je zapotřebí provést ověření (autentizace) uživatele a následné udělení oprávnění (autorizace) uživateli. *Systémy správy identit* umožňují organizacím zajišťovat procesy autentizace jednotlivých uživatelů ke spojení jejich identit či autorizace pro správu přístupů [26].

2.1 Systémy správy identit

Systém správy identit, také pod jiným názvem *systém správy identit a přístupu*, je systém vytvořený specificky pro správu identit v rámci organizace. Mezi hlavními prvky systému typicky zahrnují [26]:

- Autentizace: Je proces, kterým entita dokazuje, kým je. Proces může být definován například kombinací doložení uživatelského jména a hesla či jednorázových kódů.
- Autorizace: Je proces řízení přístupu dle identity. Určuje, jako činnosti má uživatel dle identity oprávnění provádět. To může být založeno podle rolí identity nebo skupinách, do kterých identita patří.
- Správa jednotlivých identit: Obsahuje procesy pro správu identit jako vytváření, editace či mazání jednotlivých identit.
- Jednotné přihlášení (Single Sign-On): Je autentizační mechanismus, který umožňuje autentizaci k jiným aplikacím či systémům pomocí jednotné sady přihlašovacích údajů, která se nachází pouze v daném systému správy identit.

2.1.1 Federativní identita

Termín *Federativní identita*, ang. Federated Identity, umožňuje uživatelům centrálně sdílet a využívat svoji identitu a přístupová práva napříč různými organizacemi či systémy. To zahrnuje například sdílení služeb související s autentizací a autorizací. Mezi technologie využívané pro federovanou identitu může patřit například SAML 2.0, OAuth 2.0, OpenID

Connect a JSON Web Token, které budou popsány v kapitole 3. Federativní identita je úzce spjatá s autentizačním mechanismem jednotného přihlášení [17][25][20].

2.1.2 Federace identit

Federace identit, někdy zkráceně *federace*, je synonymní a v určitých zdrojích i významově stejná s federativní identitou. Běžně se federace identit definuje jako kolekce organizací, které mají vzájemnou dohodu a sadu pravidel o sdílení a správě identit, jejich právních rámců či technických specifikací. Některé federace rozlišují jednotlivé organizace ve federaci na *členy federace*, což jsou organizace poskytující identity (obvykle univerzity či výzkumné instituce), a na *provozovatele služeb*, což jsou organizace poskytující pouze služby a používají identit od členů federace. [25].

2.1.3 Národní federace identit

Mnoho zemí provozuje vlastní federaci identit pro univerzitní a výzkumné instituce, které se nachází v daném zemi. Těmto federacím se říká *národní federace identit* [8].

eduID.cz

Jedna z národních federací identit v České republice je Česká akademická federace identit eduID.cz, kterou provozuje sdružení CESNET[1]. Cílem federace eduID.cz je poskytnout organizacím, které jsou součástí federace, sdílení identit uživatelů z jiných organizací. Mezi členy federace patří primárně univerzity nebo výzkumné instituce, které obvykle spadají pod státní správou. Federace eduID.cz je součástí služby eduGAIN [7].

eduGAIN

eduGain je globální služba, která propojuje národní i nenárodní federace identit po celém světě, což umožňuje organizacím využívat identity a služeb jiného státu přes jejich národní federaci. Aktuálně je v eduGain zapojeno přes 79 federací, 5637 organizací poskytující identity a 3676 organizací poskytující služby. eduGain je úzce spjatá s REFEDS [8][24].

REFEDS

REFEDS (Research and Education FEDerations) je komunita zabývající se sjednocení procesů souvisejicích s federací identit ve výzkumném a univerzitním sektoru. Poskytuje fórum pro spolupráci souvisejících s federací identit a pracuje na vývoji společných postupů a standardů, aby umožnila interoperabilitu v rámci komunity [21].

[1] https://www.cesnet.cz/o-nas

Kapitola 3

Autentizační a autorizační metody na webu

Tato kapitola popisuje autentizační a autorizační metody v prostředí webových aplikací s architekturou klient-server v HTTP. Kapitola zmíní a popíše ty nejčastěji používané.

3.1 HTTP Basic Authentication

HTTP Basic Access Authentication je jednoduchý autentizační mechanismus definován ve specifikaci RFC 7617[22]. Při každém HTTP požadavku se zasílá dvojice jméno a heslo. Dvojice je oddělená dvojtečkou a je před zasláním zakódována v Base64. Zakódovaná hodnota se při požadavku nachází v HTTP hlavičce `Authorization`. Před zakódovanou hodnotou se musí nacházet řetězec `Basic`, který specifikuje autentizační mechanismus a je oddělená od zakódované hodnoty mezerou.

Příklad: Pokud je dvojice údajů Honza:HokejFotbal123, pak HTTP hlavička má po zakódování tuto podobu:

```
Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==
```

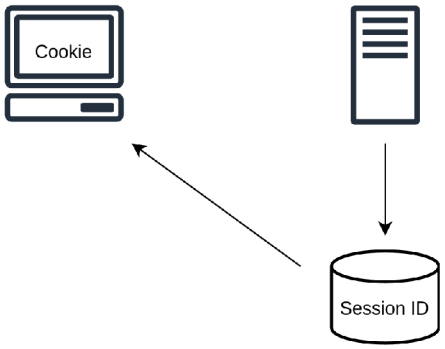
Server přijímající požadavek poté ověří údaje se svoji databází a podle toho odpoví. Pokud údaje nesedí, server odpoví uživateli s HTTP kódem 401 s hlavičkou `WWW-Authenticate` s hodnotou, která obsahuje název autentizační metody a relevantní uživatelské domény.

HTTP basic je jednoduchá metoda na použití, ale hodnota v hlavičce `Authorization` není nijak šifrovaná a údaje jsou tedy vidět při každém požadavku během komunikace. Heslo může být tedy lehce zachyceno, pokud není použito HTTPS.

3.2 Autentizace založená na relaci a cookie

V této variantě si server udržuje *relaci* (jiným termínem *sezení*) o koncovém uživateli. Uživatel se přihlašuje obvykle za pomocí webového přihlašovacího formuláře zpravidla pomocí svého uživatelského jména a hesla. Po autentizaci uživatele je serverem vytvořena relace, která se udržuje mezi klientem a serverem. Informace o přihlášeném uživateli jsou uloženy na serveru a klient obvykle obdrží identifikátor sezení (Session ID), který si klient udržuje v rámci svých následných požadavků na server, aby prokázal, zda je již autentizován. Na

serveru se informace na straně serveru udržuje obvykle v paměti serveru. Na klientovi je identifikátor relace zase obvykle uložen pomocí HTTP Cookie.



Obrázek 3.1: Jednoduchý model fungování relace

Mezi hlavními výhodami patří:

- Jednoduchá implementace a funkčnost.
- Pohodlné užití ze strany uživatele.

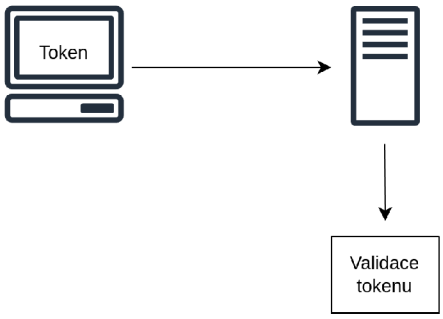
Mezi hlavní nevýhodami patří:

- Uchovávání stavu na straně serveru. Server si musí udržovat stav každého přihlášeného uživatele, což znamená více užití zdrojů. Udržování stavu je také proti REST principu o bezstavovosti. Zároveň to více komplikuje možnost horizontálního škálování aplikace, kdy aplikace je rozdělena na více instancí na různých serverech a zátěž distribuována na tyto servery.
- Složitá integrace Session ID v mobilních aplikacích.
- Metoda je více náchylná na útok CSRF (Cross-Site Request Forgery), kdy útočník může oklamat uživatele, odcizit identifikátor sezení a vykonat nežádoucí požadavky s jeho identifikátorem.

3.3 Autentizace založená na tokenu

Principem této metody autentizace je delegování identity a oprávnění skrze unikátní webový token, který klient obdrží od serveru. Uživatel se přihlašuje obvykle stejným způsobem jako v předchozí metodě za pomocí webového formuláře se jménem a heslem. Po autentizaci je vygenerován klientovi token. Tento token již není identifikátorem pro server a obsahuje informace o samotném uživateli. Může být generován různými způsoby, často je závislý na kombinaci uživatelských přístupových údajů a dalších faktorů. Po obdržení tokenu může klient provádět další požadavky na server, přičemž server validuje platnost tokenu a na základě informací o uživateli v tokenu server určuje, zda má klient oprávnění provést danou akci.

Je však nezbytné zajistit pravost tokenu a jeho bezpečné předávání a uchovávání na straně klienta, aby nedošlo k jeho zneužití nebo odhalení citlivých údajů uživatele. Zneužití tokenu lze zamezit například krátkou platností tokenu, podepisováním tokenu nebo šifrováním tokenu. V dnešní době se jako populární řešení tokenu používá JSON Web Token (sekce 3.3.1).



Obrázek 3.2: Jednoduchý model fungování tokenu

Mezi hlavními výhodami tokenu patří:

- Server si neudržuje stav o uživateli, tudíž není využito více zdrojů a splňuje princip bezstavovosti REST API. To také umožňuje lehčí možnost horizontálního škálování aplikace.
- Jednodušší pro implementaci jednotného přihlášení, kdy různé nezávislé aplikace mohou přijímat stejné tokeny.
- Podpora mobilních aplikací.

Mezi hlavními nevýhodami patří:

- Útoky kvůli při špatné implementaci nebo užití tokenu jako CSRF nebo XSS.

3.3.1 Json Web Token

Json Web Token, zkráceně JWT, je otevřený standard definován v RFC 7519[13], který definuje bezpečný způsob přenosu informací pomocí serializačního formátu JSON. Token se obvykle dělí na tři části: hlavičku, data a volitelného podpisu (signatura), kde jednotlivé části jsou oddělené tečkou. Hlavička a data jsou ve formátu JSON. Podpis je generován pomocí vybraného podpisového algoritmu a slouží pro ověření pravosti tokenu. Popis užití podpisových algoritmů je popsán ve specifikaci JSON Web Signatures (Sekce 3.3.4). Token je zakódován upravenou variantou Base64 pro URL řetězce [20]. Příklad zakódovaného tokenu vypadá následovně:

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.
SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c
```

Po dekódování hlavičky a obsahu vypadají informace takto:

Hlavička:

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

Obsah:

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022
}
```

Existuje i specifikace JSON Web Encryption pro šifrování Json Web Tokenu, který je definován v RFC 7516.

3.3.2 Hlavička JWT

Hlavička obsahuje informace o užitých podpisových či šifrovacích algoritmech a popřípadě jak zpracovat zbytek tokenu. Obsahuje následující klíče [20]:

- `alg`: Jediný povinný klíč. Obsahuje informaci o užitém podpisovém či šifrovacím algoritmu.
- `typ`: Nepovinný klíč. Obsahuje informaci o typu tokenu. V praxi je hodnota většinou `JWT`.
- `cty`: Nepovinný klíč. Používá se pouze v případě, pokud datová část tokenu obsahuje další Json Web Token. Pokud datová část neobsahuje další token, tak se tento klíč nesmí použít. Pokud datová část obsahuje token, tak hodnota klíče je vždy `JWT`. Vnořené tokeny v tokenu se vyskytují zřídka, tudíž tento klíč se příliš nepoužívá.

3.3.3 Datová část JWT

V druhé části tokenu lze nalézt data spojené s uživatelem. Také zde najdeme informace o tokenu samotném, například kým a kdy byla vydána nebo jaké rozsahy práv token má. Datová část obsahuje tyto *registrované* klíče [20]:

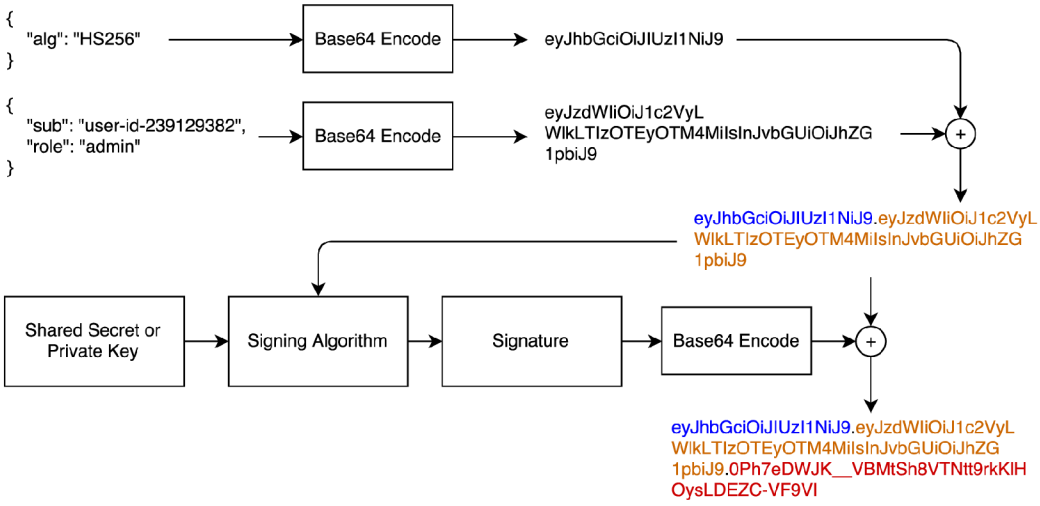
- `iss`: Obsahuje hodnotu, která definuje vydavatele tokenu. Z anglického slova *issuer*.
- `sub`: Hodnota klíče definuje subjekt, o kterém token přenáší informace. Z anglického slova *subject*.
- `aud`: Obsahuje hodnotu, která definuje vydavatele tokenu. Z anglického slova *audience*.
- `exp`: Hodnota definuje expiraci tokenu.
- `nbf`: Zkráceně pro *not before*. Hodnota definuje čas, kdy začíná platnost tokenu.
- `iat`: Zkráceně pro *issued at*. Hodnota definuje čas, kdy byl token vydán.
- `jti`: Zkráceně pro *JWT ID*. Definuje identifikátor tokenu.

3.3.4 JSON Web Signatures

Popis specifikace JSON Web Signatures lze nalézt v RFC 7515[12]. JWS popisuje formát pro digitální podpis a využívá se primárně pro zajištění integrity a validity JSON Web Tokenu. Existuje mnoho typů podpisových algoritmů, dole jsou vypsány nejdůležitější [20]:

- HMAC s použitím SHA-256 (HS256 v JWA specifikaci).
- RSASSA PKCS1 v1.5 s použitím SHA-256 (RSA256 v JWA specifikaci).
- CDSA s použitím P-256 a SHA-256 (ES256 v JWA specifikacie).
- a další jako HS384, HS512, RS384, RS512, ES384, ES512, PS256, PS384

Výsledek podpisového algoritmu je třetí část JSON Web Tokenu v Base64. Výsledek algoritmu se vytváří z již zakódované hlavičky a datové části, které jsou oddělené tečkou.



Obrázek 3.3: Proces podepisování Json Web Tokenu [20]

3.3.5 Útoky spojené s JWT tokeny

Tato sekce popisuje možné útoky s Json Web tokeny, které jsou obvykle způsobeny jeho špatnou implementací a udržováním [17][20].

Cross-Site Request Forgery

CSRF (Cross-Site Request Forgery) útoky operují na principu zasílání neautorizovaných požadavků na webové stránky, na kterých je uživatel aktuálně přihlášen, přičemž tyto požadavky jsou odesílány z jiného zdroje než samotná stránka.

JSON Web Token, stejně jako jakákoliv jiná data uložená na straně klienta, mohou být uchovávána v souborech cookie a zneužita z jiného zdroje. Pokud tedy cílová stránka nedisponuje žádnými bezpečnostními mechanismy k ochraně před CSRF útoky, jsou požadavky od útočníka brány jako pravé.

Pro zamezení těchto útoků lze použít tokeny s krátkou dobou platnosti. Pokud je token uložen v cookie, tak je pro mitigaci CSRF útoku žadoucí použít atribut `SameSite`. Pokud JWT nejsou uchovávány jako soubory cookie, ale jsou uchovávána například v uložišti prolížeče, tak je mnohem obtížnější tyto útoky provést [17][20].

Cross-Site Scripting

XSS (Cross-Site Scripting) je druh útoku, při kterém útočník vloží škodlivý kód do stránky či webové aplikace. Tento škodlivý kód je často napsán v JavaScriptu, ale může se i jednat například o HTML nebo CSS. Útočník tedy má tedy možnost s tímto kódem odcizit Json Web Token a provádět nežádoucí úkony bez vědomí uživatele. Pokud je token uložen v cookie, tak je pro mitigaci XSS útoku žadoucí použít atribut `HttpOnly`, který zabrání čtení tokenu JavaScriptem [17][20].

Kapitola 4

Protokoly jednotného přihlášení

V této kapitole jsou popsány specifikace a protokoly umožňující funkcionalitu jednotného přihlášení. Prozkoumají se do základu široce používané protokoly řešící jednotné přihlášení a správu identit, do kterých patří SAML 2.0 a OpenID Connect. OpenID Connect vychází z OAuth 2.0 specifikace, jehož fungování bude také rozvedeno.

4.1 SAML 2.0

Security Assertion Markup Language 2.0 je protokol pro výměnu informací o identitě mezi různými doménami. K výměně těchto informací v síti využívá *SAML tvrzení* 4.1.2. Protokol není omezen na specifický síťový aplikační protokol, nicméně největší uplatnění má v HTTP [27][3].

4.1.1 Aktéři v SAML 2.0

Při výměně SAML tvrzení zmiňuje protokol tyto hlavní aktéry [3]:

- *Poskytovatel identit*, ang. Identity Provider, je entita, která vlastní a sdílí identity poskytovatelům služeb.
- *Poskytovatel služeb*, ang. Service Provider, je entita poskytující služby uživatelům. Závisí na poskytovateli identit pro autentizaci a poskytování identit. V praxi je poskytovatel služeb software, který chce delegovat autentizaci a identity u poskytovatele identit.
- *Principal* - V protokolu definován jako subjekt, většinou ve formě uživatele, který má identitu u poskytovatele identit. Ve zbytku této práce bude Principal bude zmiňován jako **identita** nebo jednoduše jako **uživatel**.

4.1.2 SAML tvrzení

SAML tvrzení obsahuje informace o identitě a je definován pomocí značkovacího jazyka XML s kořenovým tagem `<saml:Assertion>`, kde prefix `saml` je jmenný prostor XML, který je definován ve formě URN[1] s hodnotou `urn:oasis:names:tc:SAML:2.0:assertion`. SAML Tvrzení se může skládat z mnoha částí, která jsou důležitá pro poskytovatele služeb. Důležité je zmínit tyto tři hlavní části [3][27]:

[1] Uniform Resource Name

- **Autentizační část**: Obsahuje informace o autentizačním procesu. V SAML tvrzení je tato část v XML tagu `<saml:AuthnStatement>`.
- **Atributová část**: Obsahuje informace o identitě ve formě atributů. Část se nachází v XML tagu `<saml:AttributeStatement>`.
- **Autorizační část**: Popisuje povolená práva a přístupy v rámci Poskytovatele identit.

Žádost o SAML tvrzení obvykle iniciuje poskytovatel služeb. Výpis 4.1 ukazuje příklad SAML tvrzení.

```
<saml:Assertion xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    ID="_21bcd72413a8da157f30cf3373d50436b129e68d80"
    Version="2.0"
    IssueInstant="2024-01-21T21:04:49Z">
    <saml:Issuer>https://www.vutbr.cz/SSO/saml2/idp</saml:Issuer>
    ...
    <saml:Subject>
        <saml:NameID SPNameQualifier="https://meta.cesnet.cz/sp/shibboleth"
                     Format="urn:oasis:names:tc:SAML:2.0:nameid-format:transient"
                     >_27cd5ef1ce0901c2c3481cb4d6430784bcf0f32617</saml:NameID>
        <saml:SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:bearer">
            <saml:SubjectConfirmationData
                NotOnOrAfter="2024-01-21T21:09:49Z"
                Recipient="https://login.cesnet.cz/proxy/module.php/saml/sp/saml2-acs.php/default-sp"
                InResponseTo="_77c2f1c12e4f88c1fd7a97076f4ae3b33697800b54"/>
        </saml:SubjectConfirmation>
    </saml:Subject>
    ...
    <saml:AuthnStatement AuthnInstant="2024-01-21T21:04:49Z"
                         SessionNotOnOrAfter="2024-01-22T05:04:49Z"
                         SessionIndex="_5bfe7468c0b2ed358f1bc70599f2b2c1c2e50e8b92">
        <saml:AuthnContext>
            <saml:AuthnContextClassRef>
            urn:oasis:names:tc:SAML:2.0:ac:classes:PasswordProtectedTransport
            </saml:AuthnContextClassRef>
        </saml:AuthnContext>
    </saml:AuthnStatement>
    <saml:AttributeStatement>
        <saml:Attribute Name="urn:oid:2.16.840.1.113730.3.1.241"
                        NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri">
            <saml:AttributeValue xsi:type="xs:string">Thanh Quang Tran</saml:AttributeValue>
        </saml:Attribute>
        <saml:Attribute Name="urn:oid:1.3.6.1.4.1.25178.1.2.9"
                        NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri">
            <saml:AttributeValue xsi:type="xs:string">vutbr.cz</saml:AttributeValue>
        </saml:Attribute>
        <saml:Attribute Name="urn:oid:1.3.6.1.4.1.5923.1.1.1.6"
                        NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri">
            <saml:AttributeValue xsi:type="xs:string">221999@vutbr.cz</saml:AttributeValue>
        </saml:Attribute>
        ...
    </saml:AttributeStatement>
</saml:Assertion>
```

Výpis 4.1: Příklad SAML tvrzení

SAML tvrzení 4.1 reprezentuje identitu studenta VUT v Brně. Zároveň reprezentuje koncový výsledek autentizačního procesu, který iniciovala aplikace pro přenos objemných souborů Cesnet Filesender[2] v roli poskytovatele služeb, který je definován v tagu `<saml:Subject>`. Poskytovatel služeb požaduje identitu u univerzity VUT v Brně (tag `<saml:issuer>`), která má roli poskytovatele identit a ve kterém probíhá autentizace uživatele.

[2] https://filesender.cesnet.cz/

4.1.3 Autentizace v SAML 2.0

Poskytovatel identit a poskytovatel služeb navzájem komunikují pomocí *SAML zpráv*, které jsou také ve formátu XML. Žádost o autentizaci u poskytovatele identit většinou iniciuje poskytovatel služeb pomocí SAML zprávy s kořenovým tagem `<samlp:AuthnRequest>` [3][27].

```
<samlp:AuthnRequest xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
                    xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
                    AssertionConsumerServiceURL="https://sp.example.com/demo1/metadata.php"
                    Destination="https://idp.example.com/SSOService.php"
                    ForceAuthn="false"
                    ID="ID_8684c800-a03b-41f4-9d6b-05c7929ddf8e"
                    IssueInstant="2023-10-16T14:06:33.810Z"
                    ProtocolBinding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST"
                    Version="2.0"
                    >
    <saml:Issuer>https://sp.example.com/demo1/metadata.php</saml:Issuer>
    <samlp:NameIDPolicy AllowCreate="true"
                        Format="urn:oasis:names:tc:SAML:2.0:nameid-format:transient"
                        />
</samlp:AuthnRequest>
```

Výpis 4.2: Příklad SAML požadavku o autentizaci

Tato zpráva se pošle poskytovateli identit, který poté začne proces autentizace. Po autentizaci zašle poskytovatel identit SAML tvrzení poskytovateli služeb a obě strany si vytvoří relaci (sekce 3.3).

4.1.4 Komunikační metody v SAML 2.0

SAML 2.0 specifikuje v HTTP protokolu tři hlavní metody výměny SAML zpráv mezi poskytovatelem identit a poskytovatelem služeb [4][27]:

- *HTTP Redirect Binding*
- *HTTP POST Binding*
- *HTTP Artifact Binding*

HTTP redirect binding

HTTP redirect binding je společně s HTTP POST Bindingem jedním z častěji využívaných metod, jakým může být prováděna výměna SAML zpráv mezi poskytovatelem služeb a poskytovatelem identit. Při použití HTTP redirect bindingu je SAML zpráva přenášena v URL parametru při HTTP GET požadavku.

Tato metoda je vhodná pro krátké SAML zprávy, například zpráva s kořenovým tagem `<samlp:AuthnRequest>` požadující autentizaci. SAML zpráva projde DEFLATE kompresí a poté se zakóduje v Base64. Pokud je SAML zpráva typu požadavek, tak se již zakódovaná zpráva pošle v URL parametru s klíčem `SAMLRequest`. Pokud jde o SAML zprávu typu odpověď, tak se zpráva pošle v URL parametru s klíčem `SAMLResponse` [27][4].

HTTP POST Binding

Při HTTP POST Bindingu je zakódovaná SAML zpráva zasílána jako data formuláře v těle HTTP POST metody s hlavičkou `Content-Type: application/x-www-form-urlencoded`.

Tato metoda je vhodná pro dlouhé SAML zprávy jako například SAML tvrzení, kde informace o identitě můžou být rozsáhlé [27][4].

HTTP Artifact Binding

V metodě HTTP Artifact Binding je SAML zpráva nahrazena krátkým identifikátorem, nazývaným *artifact*. Poskytovatel služeb požádá poskytovatele identit o vytvoření a předání hodnoty artifactu, který je typicky kratší a snáze přenositelný než samotná SAML zpráva. Poskytovatel služeb následně využije hodnotu artifactu k dotazu na poskytovatele, který odpoví poskytnutím samotné SAML zprávy, která je vnořená v XML tagu `<samlp:ArtifactResponse>`. Tato metoda není příliš v praxi využívána [27][4].

4.1.5 Komunikační tok SAML 2.0

Tok komunikace k získání SAML tvrzení se v protokolu nazývá *SAML Profiles* [6]. Nejběžněji užívaný komunikačním tok v protokolu je *Web browser SSO profile*, který se soustřeďuje na komunikaci za pomocí webové prohlížeče využívající HTTP protokolu. Existuje mnoho variant komunikačního toku dle výběru komunikačních metod ze sekce 4.1.4. Nejtypičtěji se používá kombinace HTTP Redirect Bindingu a HTTP POST Bindingu, kdy poskytovatel služeb využívá k požadavku HTTP Redirect Bindingu a poskytoval identit k odpovědi HTTP POST Bindingu [6][27].

1. Uživatel za pomocí webového prohlížeče požádá poskytovatele služeb o autentizaci u poskytovatele identit.
2. Poskytovatel služeb vytvoří SAML požadavek s tagem `<samlp:AuthnRequest>` a vytvoří HTTP odpověď s kódem 302 k přesměrování k poskytovateli identit. Zakódovaný SAML požadavek se poté uloží v URL parametru `SAMLRequest` přesměrovacího URL.

```
HTTP/1.1 302 Found
Location: https://idp.example.org/SAML2/SSO/Redirect?SAMLRequest=<encoded-request>
```

3. Prohlížeč provede HTTP GET požadavek nad přesměrovacím URL poskytovatele identit. Poskytovatel identit extrahuje a zpracuje SAML požadavek z parametru `SAMLRequest` a provede případný proces autentizace, pokud nemá uživatel u poskytovatele identit relaci.
4. Po případné autentizaci si vytvoří poskytovatel identit relaci a odpoví prohlížeči s HTML formulářem obsahující data se SAML tvrzením.
5. Prohlížeč poté předloží poskytovateli služeb získaný HTML formulář se SAML tvrzením v těle HTTP POST metody s hlavičkou:
   `Content-Type: application/x-www-form-urlencoded`.
6. Poskytovatel služeb extrahuje SAML tvrzení z těla HTTP POST požadavku. Po ověření SAML tvrzení vytvoří poskytovatel služeb relaci a umožní uživateli využívat jeho služeb.

4.1.6 SAML metadata

Aby mohla proběhnout důvěra mezi poskytovatelem služeb a poskytovatelem identit, musí si poskytovatelé navzájem vyměnit svá metadata, která jsou konformní dle SAML protokolu o metadatech [5]. SAML metadata jsou také ve formátu XML a pro poskytovatele identit a poskytovatele služeb je obsah SAML metadat definován zvlášť [27].

Metadata poskytovatele identit

Obsah metadat poskytovatele identit je popsán v kořenovém tagu `<EntityDescriptor>`. Výpis 4.3 ukazuje výňatek SAML metadat VUT[3] jako poskytovatele identit.

```
<md:EntityDescriptor xmlns:md="urn:oasis:names:tc:SAML:2.0:metadata"
    xmlns:shibmd="urn:mace:shibboleth:metadata:1.0" xmlns:mdui="urn:oasis:names:tc:SAML:metadata:ui"
    xmlns:ds="http://www.w3.org/2000/09/xmldsig#" entityID="https://www.vutbr.cz/SSO/saml2/idp">
    <md:IDPSSODescriptor WantAuthnRequestsSigned="true"
        protocolSupportEnumeration="urn:oasis:names:tc:SAML:2.0:protocol">
        ...
    </md:IDPSSODescriptor>
    <md:Organization>
        <md:OrganizationName xml:lang="en">Brno University of Technology</md:OrganizationName>
        <md:OrganizationName xml:lang="cs">Vysoké učení technické v Brně</md:OrganizationName>
        <md:OrganizationDisplayName xml:lang="en">
            Brno University of Technology
        </md:OrganizationDisplayName>
        <md:OrganizationDisplayName xml:lang="cs">
            Vysoké učení technické v Brně
        </md:OrganizationDisplayName>
        <md:OrganizationURL xml:lang="en">https://www.vutbr.cz/en</md:OrganizationURL>
        <md:OrganizationURL xml:lang="cs">https://www.vutbr.cz</md:OrganizationURL>
    </md:Organization>
    <md:ContactPerson contactType="technical">
        <md:Company>VUT</md:Company>
        <md:GivenName>CVIS</md:GivenName>
        <md:SurName>Oddělení webu VUT V Brně</md:SurName>
        <md:EmailAddress>mailto:portal@vutbr.cz</md:EmailAddress>
    </md:ContactPerson>
</md:EntityDescriptor>
```

Výpis 4.3: Výňatek metadat poskytovatele identit VUT

Uvnitř výpisu 4.3 se nachází tag `<md:IDPSSODescriptor>` s informacemi umožňující jednotné přihlášení. Mezi důležitými tagy patří `<md:SingleSignOnService>`, který definuje koncový bod služby poskytovatele identit pro jednotné přihlašení a tag pro jednotné odhlašení `<md:SingleLogoutService>`.

```
<md:IDPSSODescriptor WantAuthnRequestsSigned="true"
    protocolSupportEnumeration="urn:oasis:names:tc:SAML:2.0:protocol">
    <md:Extensions>
    ...
    </md:Extensions>
    <md:KeyDescriptor use="signing">
        ...
    </md:KeyDescriptor>
    <md:KeyDescriptor use="encryption">
        ...
    </md:KeyDescriptor>
    <md:SingleLogoutService
        Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect"
        Location="https://idp.vutbr.cz/saml2/idp/BUTSingleLogoutService.php"/>
    <md:NameIDFormat>urn:oasis:names:tc:SAML:2.0:nameid-format:transient</md:NameIDFormat>
    <md:NameIDFormat>urn:oasis:names:tc:SAML:2.0:nameid-format:persistent</md:NameIDFormat>
    <md:SingleSignOnService Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect"
    Location="https://idp.vutbr.cz/saml2/idp/SSOService.php"/>
</md:IDPSSODescriptor>
```

Výpis 4.4: Obsah tagu `<md:IDPSSODescriptor>` poskytovatele identit

[3] https://metaman.eduid.cz/entities/509/showmetadata

Metadata poskytovatele služeb

Metadata poskytovatele služeb také obsahuje kořenový tag `<md:EntityDescriptor>`. Uvnitř se nachází tag `<md:SPSSODescriptor>` s informacemi o poskytovateli služeb.

Výpis 4.5 ukazuje výňatek SAML metadat služby evidence členů Studentské unie FIT VUT v Brně[4] jako poskytovatele služeb.

```
<md:EntityDescriptor xmlns="urn:oasis:names:tc:SAML:2.0:metadata"
    xmlns:md="urn:oasis:names:tc:SAML:2.0:metadata" xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
    xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
    ...
    entityID="https://su-int.fit.vutbr.cz/kis">
    <md:SPSSODescriptor WantAssertionsSigned="true" AuthnRequestsSigned="true"
        protocolSupportEnumeration="urn:oasis:names:tc:SAML:2.0:protocol">
        ...
    </md:SPSSODescriptor>
    <md:Extensions xmlns:alg="urn:oasis:names:tc:SAML:metadata:algsupport">
        ...
    </md:Extensions>
    <md:Organization>
        <md:OrganizationName xml:lang="cs">Studentská unie FIT VUT v Brně</md:OrganizationName>
        <md:OrganizationName xml:lang="en">FIT BUT Students' Union</md:OrganizationName>
        <md:OrganizationDisplayName xml:lang="cs">SU FIT VUT</md:OrganizationDisplayName>
        <md:OrganizationDisplayName xml:lang="en">SU FIT BUT</md:OrganizationDisplayName>
        <md:OrganizationURL xml:lang="cs">https://su.fit.vutbr.cz</md:OrganizationURL>
        <md:OrganizationURL xml:lang="en">https://su.fit.vutbr.cz</md:OrganizationURL>
    </md:Organization>
    <md:ContactPerson contactType="technical">
        <md:GivenName>Jakub</md:GivenName>
        <md:SurName>Budiský</md:SurName>
        <md:EmailAddress>mailto:xbudis02@stud.fit.vutbr.cz</md:EmailAddress>
    </md:ContactPerson>
    ...
</md:EntityDescriptor>
```

Výpis 4.5: Výňatek SAML metadat poskytovatele služeb

Uvnitř výpisu 4.5 se nachází tag `<md:SPSSODescriptor>` s informacemi o poskytovateli služeb.

```
<md:SPSSODescriptor WantAssertionsSigned="true" AuthnRequestsSigned="true"
    protocolSupportEnumeration="urn:oasis:names:tc:SAML:2.0:protocol">
    <md:Extensions>...</md:Extensions>
    <md:KeyDescriptor>...</md:KeyDescriptor>
    <md:AssertionConsumerService Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST"
        Location="https://su-int.fit.vutbr.cz/kis/api/auth/eduid/assertion" index="1" />
    <md:AttributeConsumingService index="0">
        ...
        <md:ServiceDescription xml:lang="cs">
            Evidence členů Studentské unie FIT VUT v Brně
        </md:ServiceDescription>
        <md:RequestedAttribute FriendlyName="eduPersonUniqueId"
            Name="urn:oid:1.3.6.1.4.1.5923.1.1.1.13"
            NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri" isRequired="true" />
        <md:RequestedAttribute FriendlyName="cn" Name="urn:oid:2.5.4.3"
            NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri" isRequired="true" />
        <md:RequestedAttribute FriendlyName="mail" Name="urn:oid:0.9.2342.19200300.100.1.3"
            NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri" isRequired="true" />
    </md:AttributeConsumingService>
</md:SPSSODescriptor>
```

Výpis 4.6: Obsah tagu `<md:SPSSODescriptor>` poskytovatele služeb

Mezi důležité tagy patří `<md:AssertionConsumerService>`, který obsahuje koncový bod poskytovatele služeb zpracovavající SAML tvrzení od poskytovatele identit.

[4] https://metaman.eduid.cz/entities/445/showmetadata

Dále se běžně nachází v metadatech tag `<md:AttributeConsumingService>`, který obsahuje seznam atributů uživatele, který požaduje poskytovatel služeb.

Jelikož VUT v Brně je česká univerzita, tak patří do federace eduID.cz (viz. sekce 2.1.3), která má sdílená pravidla definic atributů jednotlivých uživatelů v rámci svých členů.

Výpis 4.6 konkrétně požaduje tyto atributy:

- `eduPersonUniqueId`: Je jedinečný identifikátor uživatele v rámci celé federace eduID.cz[5].
- `cn`: Celé jméno včetně diakritiky (bez akademických titulů!)[6]
- `email`: Adresa elektronické pošty uživatele[7]

Všechny atributy a jejich význam lze nalézt na stránkách eduID.cz[8].

[5] https://www.eduid.cz/cs/tech/attributes/edupersonuniqueid
[6] https://www.eduid.cz/cs/tech/attributes/cn
[7] https://www.eduid.cz/cs/tech/attributes/mail
[8] https://www.eduid.cz/cs/tech/attributes

4.2 OAuth 2.0

OAuth 2.0 je autorizační framework definován v RFC 6749[9]. Jelikož OAuth 2.0 je framework, tak se nejedná o protokol. OAuth 2.0 slouží primárně k delegování přístupů a práv k chráněným zdrojům se zastoupením vlastníka těchto zdrojů [23][2]. OAuth 2.0 neřeší správu identit a jednotné přihlášení. K tomu byl vytvořen OpenID Connect, který z OAuth 2.0 vychází a bude popsán v sekci 4.3.

4.2.1 Aktéři v OAuth 2.0

Ve specifikaci OAuth najdeme následující aktéry a pojmy[9]:

- *Chráněné zdroje*, ang. Protected resources: OAuth přesně nespecikuje význam chráněného zdroje, nicméně v praxi to může být jakákoliv informace s omezeným přístupem, která je nějakým způsobem chráněna.
- *Vlastník zdrojů*, ang. Resource Owner: Je entita, typicky koncový uživatel, který vlastní chráněný zdroj.
- *Klient*, ang. Client: Je software, který chce přistoupit k chráněným zdrojům ve jménu vlastníka zdroje. Přístupy k chráněným zdrojům získává pomocí *přístupového tokenu*, ang. Access Token. Klientem v OAuth 2.0 se nerozumí klientská aplikace v modelu Klient-server. Klientem tedy může být jakýkoliv software nebo program, který chce přistoupit k chráněným zdrojům.
- *Autorizační server*, ang. Authorization server. Je server, který se typicky stará o autentizační a autorizační úkony. Má důvěru chráněného zdroje a vydává přístupové tokeny klientům.

Autorizační server

Účelem autorizační serveru je autorizovat klienty a předávat jim tokeny. Autorizační server musí mít implementován *Authorization endpoint*, což je koncový bod pro autorizaci klientů, který má běžně podobu `/auth` nebo `/authorize`, a koncový bod *Token Endpoint*, který má běžně podobu `/token` a slouží pro vydání přístupového tokenu [9][23].

Klient

Klient je software, který zastupuje vlastníka s omezenými právy. Aby mohl zastupovat vlastníka a být autorizován, musí být klient registrován na autorizačním serveru. Registrace klienta na autorizačním serveru je realizováno pomocí identifikátoru s názvem *Client ID* (`client_id`). Klientovi může být dále přidělen *Client Secret* (`client_secret`), což je údaj, který slouží pro autentizaci klienta. Někteří klienti nicméně Client Secret mít nemusí [9][23].

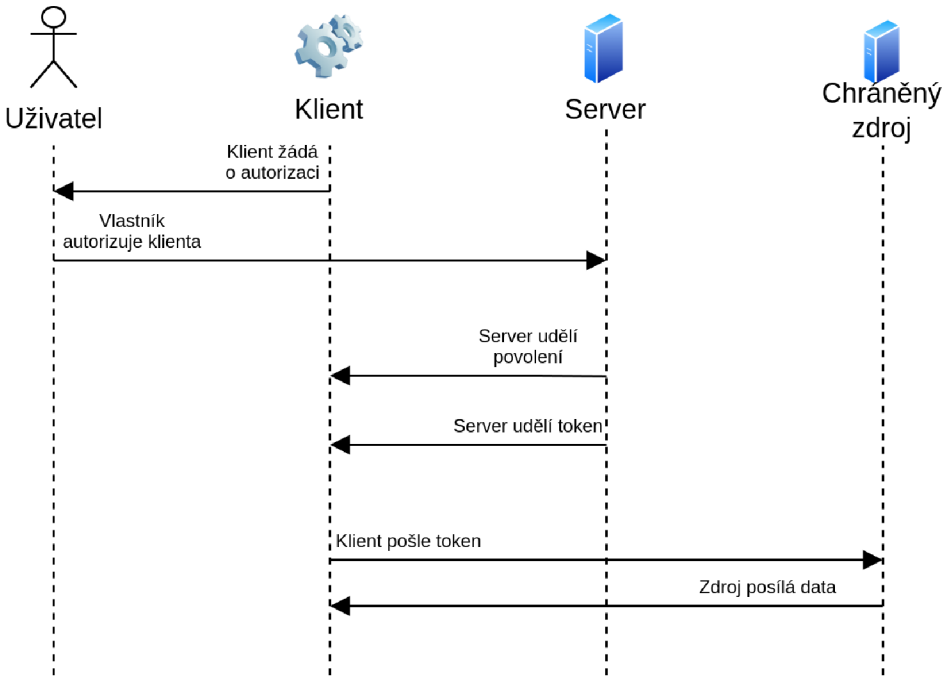
Klient je rovněž charakterizován pomocí metadat. Metadata klienta můžou byt například již zmíněné Client ID nebo seznam validních URI pro zpětné přesměrování ke klientovi (`redirect_uri`) [9].

Klienti se mohou na autorizačním serveru také registrovat dynamicky. Popis, jak klienty dynamicky registrovat lze najít v RFC 7591[9].

[9] https://www.rfc-editor.org/rfc/rfc7591.txt

4.2.2 Komunikační tok OAuth 2.0

Obrázek 4.1 ukazuje vysokoúrovňovou komunikaci jednotlivých aktérů v OAuth 2.0 [9][23].



Obrázek 4.1: Vysokoúrovňové zobrazení autorizace klienta v OAuth 2.0 [23]

Aby klient získal oprávnění se zastoupením vlastníka, musí klient přesměrovat vlastníka k autorizačnímu serveru. Vlastník se případně autentizuje na autorizačním serveru a je autorizačním serverem dotázán, jestli může klient přistoupit s určitými právy k chráněným zdrojům. Práva k chráněným zdrojům mohou být omezena a později upravena dle přání vlastníka zdroje. Klient oprávnění získá ve formě přístupového tokenu. Obecný tok k získání tokenu se dle specifikace dělí na tyto kroky [9][23]:

- **Autorizační požadavek**, ang. Authorization Request, kdy klient žádá o autorizaci.
- **Autorizační odpověď**, ang. Authorization Response, kdy klient dostane povolení k získání přístupového tokenu.
- **Žádost o token**, ang. Token Request, kdy klient žádá o přístupový token.
- **Odpověď s tokenem**, ang. Token Response, kdy klient získá přístupový token.

4.2.3 Přístupový a obnovovací token

Klientům je autorizačním serverem přidělován přístupový token. S jeho pomocí mohou klienti přistupovat k chráněným zdrojům vlastníka s omezenými právy. Je tedy žádoucí, aby přístupový token měl pouze rozsah práv, který klient nezbytně potřebuje [23].

Obnovovací token je další typ tokenu, který může být přidělován autorizačním serverem. Obnovovací token umožňuje klientům obnovit přístupový token po jeho vypršení, aniž by museli opakovaně žádat vlastníka o autorizaci. [9][23].

OAuth 2.0 nespecifikuje jak by přístupový nebo obnovovací token měl vypadat. Na webu jsou přístupové tokeny v dnešní době ve formě Json Web Tokenů (viz. sekce 3.3.1).

4.2.4 OAuth scopes

OAuth scopes je mechanismus, který umožňuje klientům získat určité přístupy k informacím nebo provádět určité operace s těmito informacemi, jako například informace o uživateli [9]. Tyto operace jsou obvykle v rámci určité služby nebo API, kde služba nebo API definuje své vlastní scopes ke svým zdrojům. OAuth 2.0 nespecifikuje jaké scopes by se měly poskytovat, nicméně některé běžné příklady scopes mohou zahrnovat:

- `read`: Oprávnění ke čtení informací. To může zahrnovat čtení profilu uživatele, jejich e-mailových adres, seznamu kontaktů apod.
- `write`: Oprávnění k zápisu a změn informací.
- `delete`: Oprávnění k mazání informací.
- `profile`: Oprávnění ke čtení informací o uživateli.
- `email`: Oprávnění ke čtení k e-mailové adrese uživatele.

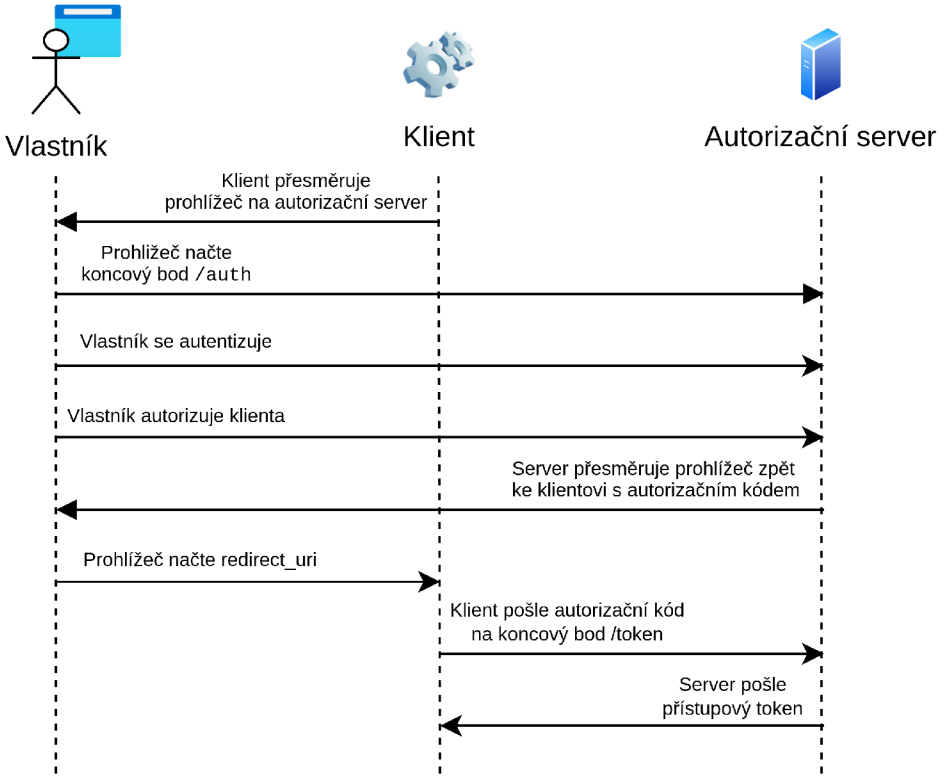
4.2.5 Metody získání tokenu

V této sekci je popis metod jak získat přístupový token v protokolu HTTP. OAuth 2.0 popisuje několik metod jak získat token. V angličtině se těmto metodám říká *Grant Types*, zjednodušeně *Grant*. Jednotlivé metody jsou vhodné podle typu klienta. OAuth 2.0 definuje tyto hlavní metody [9]:

- *Authorization Code*
- *Implicit*
- *Client Credentials*
- *Resource Owner credentials grant type*

Authorization Code Grant

Authorization Code Grant je nejběžněji používaná metoda pro získání tokenu. Obrázek 4.2 ukazuje získání přístupového tokenu za pomocí webového prohlížeče [9][23]



Obrázek 4.2: Authorization code grant [9][23]

1. Klient připraví autorizační požadavek a přesměruje prohlížeč k autorizačnímu serveru. V HTTP má přesměrování podobu HTTP odpovědi s kódem 302.

```
HTTP/1.1 302 Found
Location: http://authserver.com/authorize?response_type=code&client
_id=oauth-client&redirect_uri=http%3A%2F%2Fclient-example.com%2Fcallback&scope=foo&
state=Lwt50DDQKUB8U7jtfLQCVGDL9cnmwHH1
```

   Prohlížeč poté provede HTTP GET metodu s URL autorizačního požadavku, které má tyto parametry:

   - `response_type`: Povinný parametr specifický pro autorizační požadavek, specifikuje typ metody. V tomto případě jde o hodnotu `code`, kdy autorizační server vrátí klientovi autorizační kód.
   - `client_id`: Povinný parametr. Identifikuje klienta iniciující autorizační požadavek a přístupový token.
   - `redirect_uri`: Nepovinný parametr. Specifikuje URL kam se má autentizační server vrátit. Dané URL poté získá autorizační kód.
   - `scope`: Nepovinný parametr, který specifikuje rozsah práv přístupového tokenu.
   - `state`: Nepovinný parametr s jakoukoliv hodnotou. Tato hodnota se očekává po autorizační odpovědi k mitigaci CSRF útoku.

Autorizační server ověří hodnoty `client_id` a `redirect_uri`, pokud jsou parametry validní v rámci autorizačního serveru, tak umožní uživateli se případně autentizovat.

2. Po případné autentizaci přesměruje autorizační server prohlížeč zpátky ke klientovi dle hodnoty `redirect_uri`. K `redirect_uri` je přidám parametr `code` s autorizačním kódem. Pokud byl definován state parametr, je taky přidán jako parametr.

```
HTTP/1.1 302 Found
Location: http://client-example.com/callback?code=cavvwku-ld0n8-uilnunn&
state=Lwt50DDQKUB8U7jtfLQCVGDL9cnmwHH1
```

   Prohlížeč poté provede HTTP GET metodu s URL klienta, které obsahuje parametr `code`, což je jednorázový autorizační kód pro výměnu tokenu. Zde končí autorizační odpověď.

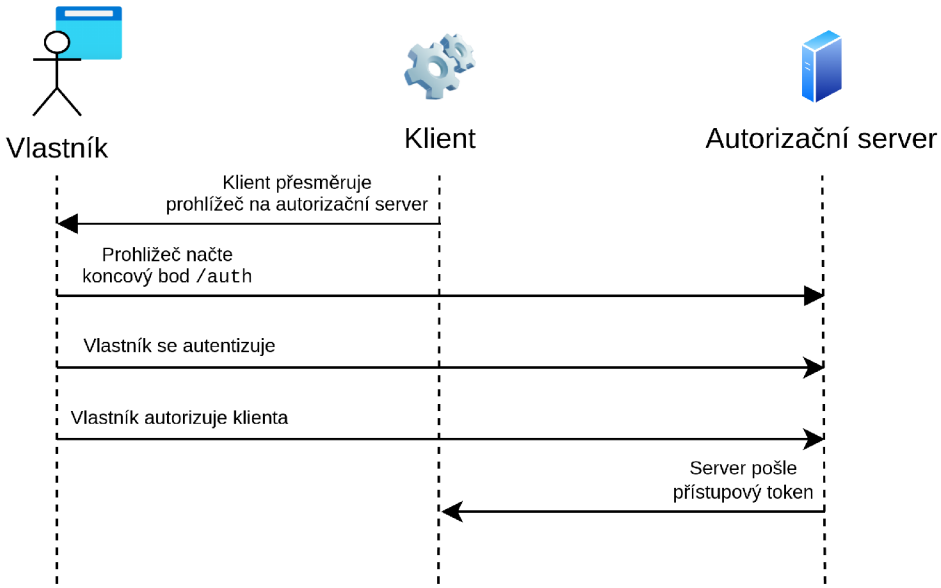
3. Klient připraví žádost o token. Klient zpracuje přijatý autorizační kód a výmění kód za přístupový token na koncovém bodě `/token` autorizačního serveru pomocí HTTP POST metody. Klient musí při požadavku dodat v HTTP hlavičce `Authorization` dodat svoje údaje Client ID (`client_id`) a Client Secret (`client_secret`) ve formě HTTP Basic.

4. Autorizační server validuje požadavek a pošle přístupový a případně obnovovací token klientovi.

Tato metoda je vhodná pro backendové či mobilní aplikací a měla by být vždy prvním volbou při integraci OAuth do aplikace [23].

Implicit Grant

Tato metoda se podobá metodě Authorization Code Grant, ale nemá kroky související s autorizačním kódem, tudíž se přístupový token přiděluje hned v autorizační odpovědi. Obrázek 4.3 ukazuje získání přístupového tokenu s webovým prohlížečem [23].



Obrázek 4.3: Implicit grant [9][23]

1. Klient připraví autorizační požadavek a přesměruje prohlížeč k autorizačnímu serveru. V HTTP má přesměrování podobu HTTP redirectu s kódem 302.

```
HTTP/1.1 302 Found
Location: http://authserver.com/authorize?response_type=token&client
_id=oauth-client&redirect_uri=http%3A%2F%2Fclient-example.com%2Fcallback&scope=foo&
state=Lwt50DDQKUB8U7jtfLQCVGDL9cnmwHH1
```

   Prohlížeč poté provede HTTP GET se získaným URL, které má stejné parametry jako v Authorization Code Grant, pouze `response_type` parametr má v této metodě hodnotu `token`, která naznačuje, aby autorizační server poskytl token ihned v autorizační odpovědi. Autorizační server ověří hodnoty `client_id` a `redirect_uri` a pokud jsou parametry validní, tak umožní uživateli se případně autentizovat.

2. Po případné autentizaci autorizační server přesměruje prohlížeč zpátky ke klientovi dle hodnoty parametru `redirect_uri` s přístupovým tokenem, který je uložen v URI fragmentu.

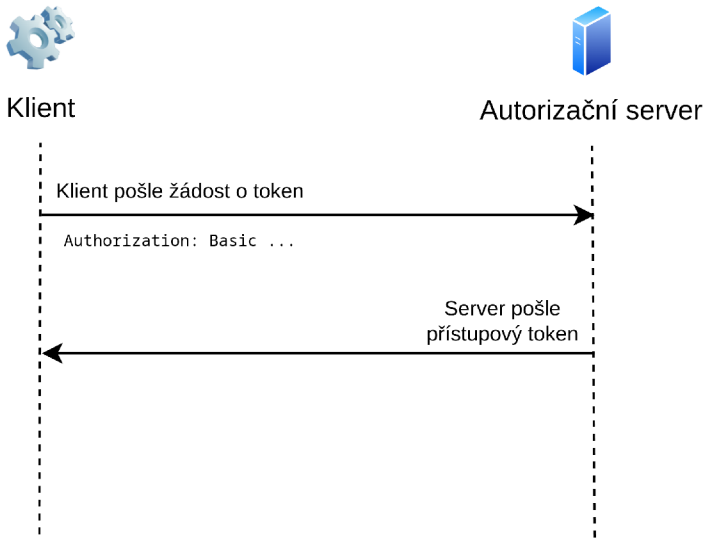
```
HTTP/1.1 302 Found
Location: http://client-example.com/callback#access_token
```

Implicit metoda je určena pro aplikace bežící čistě jen na prohlížeči například za pomocí jazyka JavaScript, které nedokážou uchovávat bezpečně citlivá data jako Client Secret, proto se v této metodě neobjevuje ověřování hodnoty Client Secret. Všimněme si, že autorizační server neposílá obnovovací token.

Novější OAuth 2.1 odstranil z bezpečnostních důvodů Implicit Grant ze své specifikace, takže se již tato metoda nedoporučuje používat.

Client Credentials Grant

V této metodě k získání tokenu probíhá komunikace pouze mezi klientem a autorizačním serverem, vlastník s prohlížečem se tudíž do této metody nijak nezapojuje a vyskytující se zde pouze kroky s tokenem. K získání tokenu využívá klient HTTP Basic autentizace [23].



Obrázek 4.4: Client Credentials Grant [9][23]

1. Klient připraví HTTP POST požadavek k autorizačnímu serveru na koncovém bodě `/token`. Client ID a Client Secret jsou použity jako přístupové údaje a zakódovány dle HTTP Basic specifikace v hlavičce `Authorization`. Tělo požadavku pouze obsahuje `grant_type=client_credentials` pro identifikaci metody.

```
POST /token HTTP/1.1
Host: http://authserver.com
Content-Type: application/x-www-form-urlencoded
Authorization: Basic BASE64_ENCODED_CLIENT_CREDENTIALS

grant_type=client_credentials
```

2. Autorizační server ověří požadavek a pokud je požadavek validní, pošle klientovi přístupový token.

Některé implementace autorizačních serverů jako Auth0 přijímají i požadavek bez HTTP Basic autentizace a přikládají Client ID a Client Secret v těle požadavku.

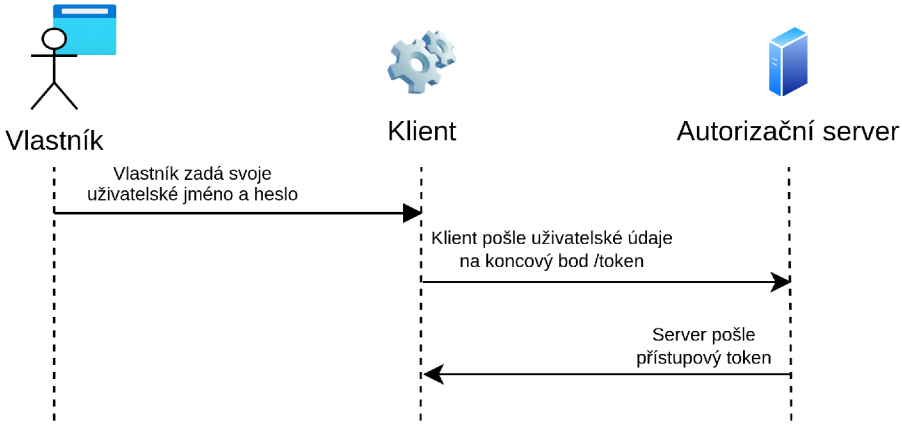
```
POST /token HTTP/1.1
Host: http://authserver.com

grant_type=client_credentials&client_id=<client_id>&client_secret=<client_secret>
```

Client credentials metoda je vhodná, pokud potřebuje backendová aplikace přímo komunikovat s autorizačním serverem a získat data bez účasti vlastníka.

Resource Owner Credentials

Tato metoda umožňuje získat přístupový token přímo pomocí přihlašovacích údajů uživatele, jako je uživatelské jméno a heslo.



Obrázek 4.5: Resource Owner Credentials Grant [9][23]

1. Uživatel zadá klientovi své přihlašovací údaje.
2. Klient poté vytvoří HTTP POST požadavek o token k autorizačnímu serveru. Tělo požadavku obsahuje formulářová data s identifikací metody (`grant_type` s hodnotou `password`), uživatelské jméno (`username`) a heslo (`password`). [frame=single]

```
POST /token HTTP/1.1
Host: http://authserver.com
Content-Type: application/x-www-form-urlencoded

grant_type=password&username=USERNAME&password=PASSWORD
&client_id=CLIENT_ID&client_secret=CLIENT_SECRET
```

3. Autorizační server po validaci údajů pošle klientovi přístupový token.

Použití této metody není doporučováno, protože klient vidí přístupové údaje uživatele, které mohou být při komunikaci ohroženy různými útoky. Je vhodné tuto metodu používat bez zapojení klienta, kdy přístupový token od autorizačního serveru vyžaduje přímo vlastník. Framework OAuth 2.1 tuto metodu ze své specifikace z bezpečnostních důvodů odstranil.

Jiné Grant Types

- **Refresh Token**: Přístupový token lze získat po předložení platného obnovovacího tokenu. Běžně se tentýž platný obnovací token dá použít opakovaně, ale lze i obnovovacího token zneplatnit po získání přístupového tokenu, pokud to má autorizační server takto nastavené.
- **SAML Profile for OAuth 2.0 Client Authentication and Authorization Grants**: definována v RFC 7522, je Grant metoda pro získání tokenu pomocí SAML tvrzení.

4.3 OpenID Connect

OpenID Connect je protokol, který byl zveřejněn v roce 2014 a vydán nadací OpenID Foundation. Řeší funkcionalitu jednotného přihlášení a podporuje webové i mobilní aplikace.

OpenID Connect není vylepšením původního protokolu OpenID. Jedná se o samostatnou specifikaci postavenou na autorizačním frameworku OAuth 2.0, která přináší nové funkcionality a lepší zabezpečení. Protokol je třetí generací technologie OpenID a přidává dodatečnou autentizační vrstvu pro OAuth 2.0. Umožňuje klientům ověřit totožnost koncového uživatele na základě autentizace provedené autorizačním serverem a získat základní informace o koncovém uživateli [14][19][2].

4.3.1 Aktéři v OpenID Connect

Jelikož je protokol založen na frameworku OAuth 2.0, tak má stejné aktéry (sekce 4.2.1) s těmito rozdíly [14]:

- *Poskytovatel OpenID*, ang. OpenID Provider, je ekvivalentní k autorizačnímu serveru v OAuth 2.0, má stejné role jako autorizační server a navíc spravuje a udržuje identity. Tato práce bude zmiňovat poskytovatele OpenID také jako ***OP***.
- OAuth klient se v OpenID Connect protokolu v angličtině označuje jako *Relying Party*. Tato kapitola bude nadále zmiňovat Relying Party jako ***RP*** nebo jednoduše **klient**.

4.3.2 Tok v OpenID Connect

Protokol dělí tok získání tokenu podobně jako v OAuth 2.0 (sekce 4.2.2) následovně [14]:

- **Autentizační požadavek**, ang. Authentication Request, kdy klient žádá o autentizaci.
- **Autentizační odpověď**, ang. Authentication Response, kdy klient dostane povolení k získání ID tokenu a přístupového tokenu.
- **Žádost o token**, ang. Token Request, kdy klient žádá o ID token a přístupový token.
- **Odpověď s tokenem**, ang. Token Response, kdy klient získá ID token a přístupový token.

4.3.3 ID Token

OpenID Connect zavadí nový typ tokenu s názvem ID token. Tento token obsahuje informace o uživateli, který se autentizoval v OP a vydává se obvykle společně s přístupovým tokenem po žádosti o token. ID Token je společně s přístupovým tokenem ve formě Json Web Tokenu. ID Token obsahuje jako v JWT následující povinné hodnoty [14]:

- `iss`: Definuje vydatele tokenu
- `sub`: Unikátní hodnota, která definuje lokálně identitu a je primárně určen pro klienta
- `aud`: Publikum, pro které je token určen. Součástí hodnoty musí obsahovat `client_id` klienta. Hodnota může být ve formě pole nebo jednohodnového řetězce.
- `exp`: Konec platnosti tokenu. Obvykle ve formě unixového času.

4.3.4 Autentizace a metody získání tokenu

OpenID Connect provádí autentizaci pro účely přihlášení koncového uživatele nebo pro určení, zda je koncový uživatel již přihlášen. Server implementující OpenID Connect vrátí klientovi výsledek autentizace bezpečným způsobem, aby klient mohl klient výsledku autentizace důvěřovat [14].

Po autentizačním procesu může klient žádat o ID token a přístupový token. Protokol používá stejných metod získání tokenu obsahující kroky spojené s autentizačním požadavkem a autentizační odpovědí. Konkrétně jde o metody **Authorization Code Flow** a **Implicit Flow** (OpenID Connect používá termínu *Flow* místo *Grant*), které byly již zmíněny v sekci 4.2.2. Tato sekce zmíní pouze metodu Authorization Code Flow. Implicit Flow je odstraněna v novější specifikaci OAuth 2.1 a neměla by být v budoucnu používána [9][14].

Autentizace s Authorization Code Flow

Kroky a definice parametrů jsou stejné jako v sekci 4.2.2 o Authorization Code Grant [14]:

1. Klient připraví autentizační požadavek s parametry `response_type` s hodnotou `code`, `redirect_uri` s validní hodnotou k navrátu zpět ke klientovi a `client_id` s Client ID klienta. Pro protokol je dále povinné přidat parametr `scope` s hodnotou `openid`, která naznačuje, že klient požaduje autentizaci a informace o identitě uživatele v OpenID Connect protokolu.

   ```
   HTTP/1.1 302 Found
   Location: http://oidc-provider.com/authorize?response_type=code&client
   _id=oidc-client&redirect_uri=http%3A%2F%2Fclient-example.com%2Fcallback&scope=openid&
   state=Lwt50DDQKUB8U7jtfLQCVGDL9cnmwHH1
   ```

   Prohlížeč přesměruje uživatele k **OP**, který poté ověří hodnotu Client ID. Pokud jsou parametry validní, tak umožní uživateli se případně autentizovat.

2. Po případné autentizaci přesměruje **OP** prohlížeč zpátky ke klientovi dle hodnoty `redirect_uri`. K `redirect_uri` je přidám parametr `code` s autorizačním kódem.

   ```
   HTTP/1.1 302 Found
   Location: http://client-example.com/callback?code=cavvwku-ld0n8-uilnunn&
   state=Lwt50DDQKUB8U7jtfLQCVGDL9cnmwHH1
   ```

   Prohlížeč poté provede HTTP GET k URL klienta, které obsahuje parametr `code`, což je jednorázový autorizační kód pro výměnu tokenu. Zde končí autentizační odpověď.

3. Klient připraví žádost o token. Díky získanému autorizačnímu kódu může vyměnit kód za přístupový token a ID token na koncovém bodě `/token` **OP** pomocí HTTP POST metody. Klient musí při požadavku dodat v HTTP hlavičce `Authorization` dodat svoje údaje Client ID (`client_id`) a Client Secret (`client_secret`) ve formě HTTP Basic autentizace.

4. **OP** validuje požadavek a pošle přístupový token, ID token a případně obnovovací token.

4.3.5 OpenID Connect Discovery 1.0

OpenID Connect Discovery 1.0 je specifikace, která umožňuje dynamické získání informací o poskytovateli OpenID (OP) pomocí specifických koncových bodů. To umožňuje klientům (RP) jednoduše nalézt některé nezbytné informace o poskytovateli OpenID. Specifikace

stanovuje metadata, která by měl poskytovatel OpenID obsahovat, například metadata koncových bodů pro autorizaci a pro token nebo podporovaných podpisových algoritmů.

Jedna z hlavních koncových bodů je `/.well-known/openid-configuration`, který obsahuje všechna metadata o OP dle specifikace [15].

4.3.6 Odhlašování v OpenID Connect

Protokol definuje tři mechanismy odhlašování:

- RP-Initiated
- Front-Channel
- Back-Channel

RP-Initiated odhlašování

Odhlašování inicuje klient (RP) na žádost uživatele (například po kliknutí tlačítka), který pošle požadavek poskytovateli OpenID (OP), že se uživatel má odhlásit. Poskytovatel obvykle požadavek přijme a ukončí relaci uživatele. Komunikace toku procesu odhlašování probíhá primárně mezi klientem a poskytovatelem OpenID [16].

Front-Channel odhlašování

Komunikace při odhlášování probíhá přímo mezi prohlížečem uživatele a poskytovatelem OpenID (OP), bez jakékoliv komunikace mezi servery. To je obvykle dosaženo pomocí přesměrování nebo jiných mechanismů na straně prohlížeče [11].

Back-Channel odhlašování

Odhlašování v tomto mechanismu probíhá obvykle pouze mezi klientem a poskytovatelem OpenID (OP) bez účasti uživatele, tudíž při odhlašování komunikují navzájem pouze servery [10].

4.3.7 OpenID Connect v bankovnictví

V Evropě je OpenID Connect adoptován ve velké škále v oblasti bankovnictví. Tradičně měly banky a jiné finanční instituce monopol s identitami svých bankovních zákazníků, kdy dříve zákaznící měli vlastní klientské číslo, pomocí kterého se přihlašovali do online bankovnictvní, což omezovalo inovaci v oblasti digitalizace. Aby poskytly zákazníkům co nejlepší uživatelskou zkušenost, investovaly banky do vývoje modernějších online služeb, kde zákazník může používat svoji bankovní identitu i v jiných službách [18].

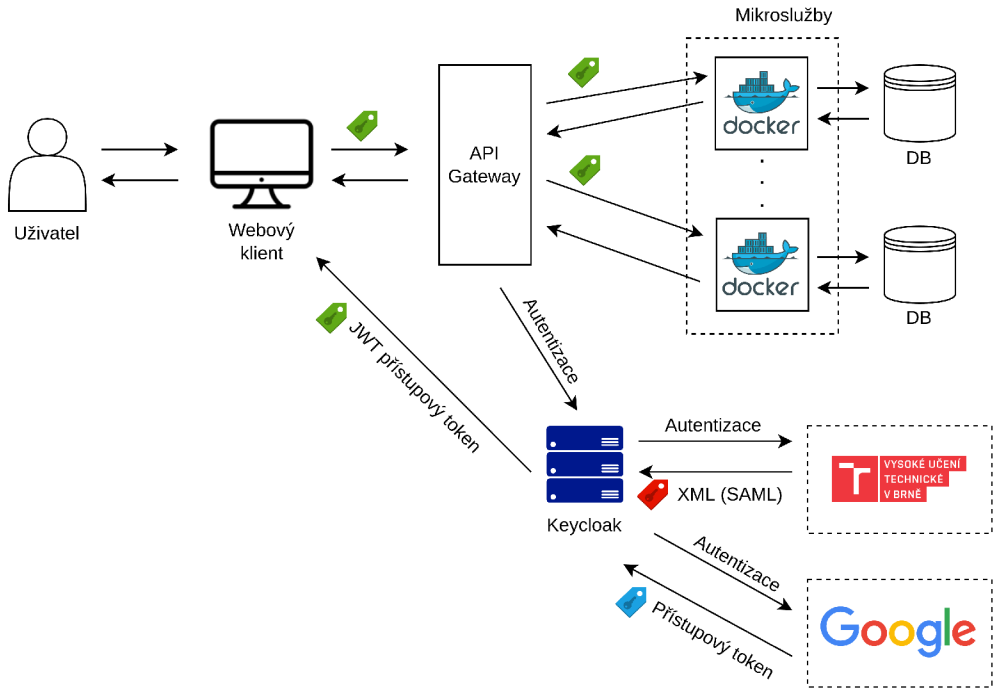
V České republice v rámci digitalizace existuje služba Bankovní identity (Bank iD) [1], která implementuje otevřené bankovnictví a umožňuje digitální ověření uživatelů poskytované bankami. Otevřené bankovnictví umožňuje sdílení finančních dat mezi bankami a poskytovateli služeb pomocí API. V Evropské unií je otevřené bankovnictví ovlivněná směrnicí o platebních službách PSD2 (Payment Services Directive), která se zaměřuje na podporu rozvoje a využívání inovativních online a mobilních plateb prostřednictvím otevřeného bankovnictví [28]. Služby jako Bank iD implementují svá API rozhraní pomocí protokolu OpenID Connect.

Česká bankovní asociace poskytuje i standard pro implementaci autentizace bankovních identit v OpenID Connect, která je podporována 16 významnými českými bankami [31].

Kapitola 5

Demonstrační webový systém

Tato kapitola se zaměřuje na integraci různých zdrojů identit k informačnímu systému založeného na architektuře mikroslužeb. Webový systém se bude skládat ze dvou aplikací, jmenovitě **Keycloak**, který bude popsán v sekci 5.1 a samotný informační systém `VUTBids`, který bude popsán v sekci 5.3. Informační systém bude využívat k přihlašování uživatelů systému Keycloak, který zároveň také slouží pro správu identit uživatelů. Keycloak bude mít ve webovém systému roli centrálního systému pro autentizaci a autorizaci uživatelů. Řízení přístupu bude primárně řízen přes Json Web Tokeny. Bude dále brán ohled na práva a role podle typu uživatele a správnou práci se zpracováváním Json Web Tokenu. Obrázek 5.1 ukazuje jednoduché schéma architektury systému.



Obrázek 5.1: Jednoduché schéma webového systému

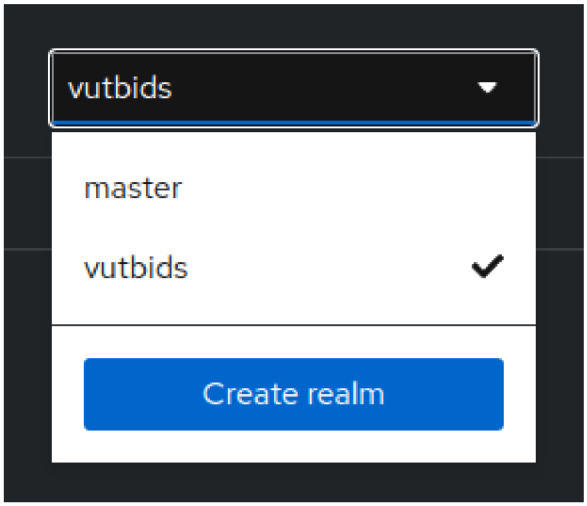
5.1 Autentizační systém Keycloak

Keycloak je populární open-source systém pro správu identit implementovaný v jazyce Java, který vyvíjí primárně společnost RedHat[1]. Systém slouží primárně pro zabezpečení aplikací a jiných systémů za pomocí protokolů řešící autentizaci a autorizaci. Podporuje protokoly SAML 2.0 a OpenID Connect (potažmo i OAuth 2.0), ale i integraci identit síťových protokolů jako LDAP nebo Kerberos. Keycloak je robůstní a flexibilní, což ho činí ideálním řešením pro organizace hledající bezpečné a spolehlivé řešení pro správu identit. V této práci bude použit Keycloak verze 23.

5.2 Příprava a konfigurace Keycloaku

Keycloak nabízí široké možnosti nastavení a správy svých jednotlivých komponent, které jsou pro nové uživatele relativně složité. V této sekci jsou popsány postupy, jak připravit Keycloak, aby fungoval s naší aplikací a aby poskytoval požadované identity od Googlu a VUT v Brně.

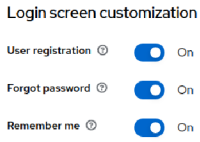
Keycloak zavadí vlastní komponentu s názvem *realm*, která slouží jednoduše jako jmenný prostor pro oddělení uživatelů, rolí, skupin a dalších dílčích komponentů Keycloaku. Keycloak má předem definovaný výchozí realm s názvem *master*, který obsahuje konfigurace, role a uživatele, které se používají pro správu samotného Keycloaku. Realm master by měl sloužit pouze pro samotný Keycloak a nic jiného. Je tedy nutné, abychom vytvořili separátní realm s názvem *vutbids*, který bude pouze pro informační systém VUTBids.



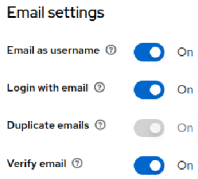
Obrázek 5.2: Seznam dostupných realmů v Keycloaku a možnost realm vytvořit

Keycloak Realm bude nastaven tak, aby email byl nastaven jako uživatelské jméno, tudíž nesmí být v databázi Keycloaku dva uživatelé se stejným emailem. Dále bude možnost registrace nového uživatele přes formulář. Dále bude možnost obnovit zapomenuté heslo a funkcionalita `Zapamatuj si mě`. Tato nastavení lze nastavit v `Realm settings` → `Login`.

[1] https://www.redhat.com/



Obrázek 5.3: Nastavení přihlašovací stránky



Obrázek 5.4: Nastavení emailu

5.2.1 Nastavení Keycloak klienta

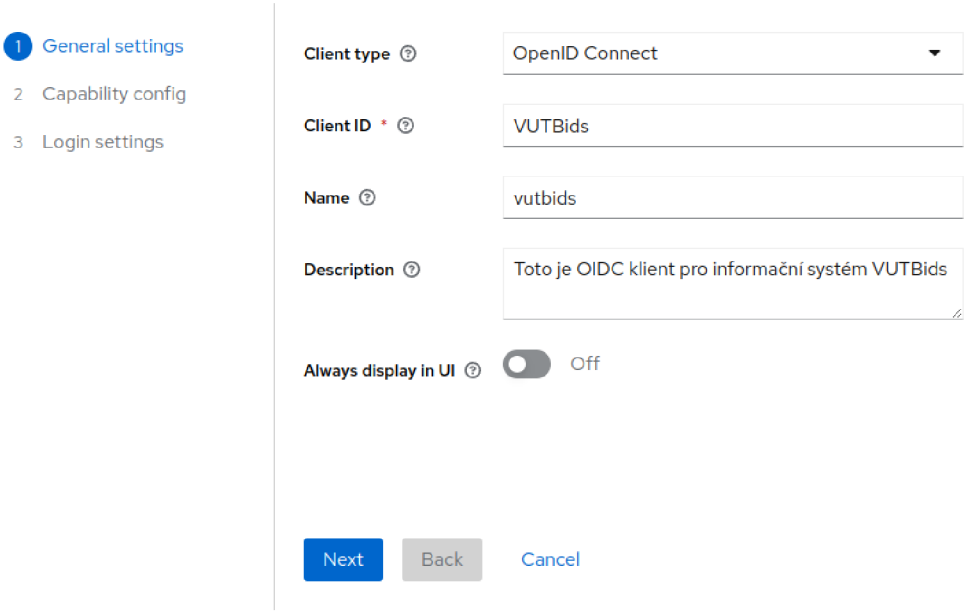
Keycloak podporuje registraci klientů manuálně přes jeho administrátorské rozhraní, přes jeho REST API nebo přes dynamickou registraci definovanou v RFC 7591. Klienti v Keycloaku představují aplikace nebo služby, které mají důvěru Keycloaku a mohou využívat jeho služeb. To může zahrnovat například přihlašování uživatelů, správu uživatelů, správu rolí nebo skupin, získání přístupového tokenu, definice tokenu a dalších operací související s autentizací a autorizací v rámci systému.

Pro informační systém VUTBids stačí jedna definice klienta, který bude komunikovat s Keycloakem. Klient byl registrován manuálně jako OpenID Connect klient. Klienti se registrují v administrátorském rozhraní na levé liště v sekci `Clients` → `Create client`. Jelikož o služby Keycloaku žádá backendová aplikace, tak je klientovi přiděleno Client Secret, pomocí kterého se může u Keycloaku autentizovat. Client Secret lze získat povolením `Client Authentication` v nastavení klienta. Tabulka 5.1 ukazuje minimální nezbytné hodnoty nastavení klienta, který je již připraven na produkci.

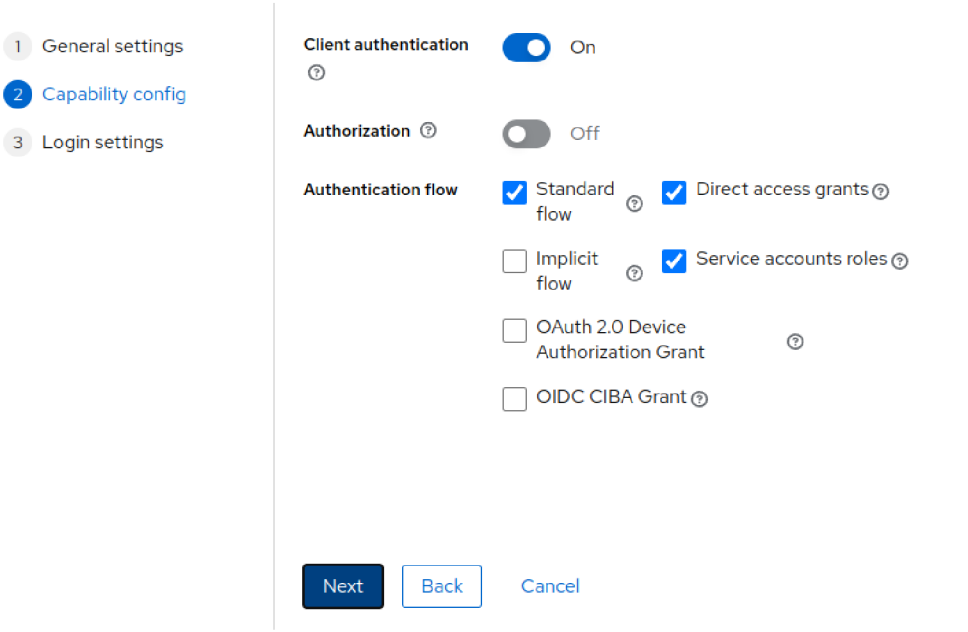
Client ID	vutbids	Identifikátor klienta, jde o stejný Client ID jako v OAuth 2.0 specifikaci
Valid redirect URIs	https://vutbids.cz/*	Validní URI pro přesměrování zpátky ke klientovi
Client Authentication	Povolen	Klient bude považován za tajného a bude mu přidělen `Client Secret`
Authentication flow	Povolen: `Standard flow`, `Direct Access grants`, `Service Account Roles`	Povolené toky k získání tokenu

Tabulka 5.1: Pole a hodnoty formuláře při vytváření klienta.

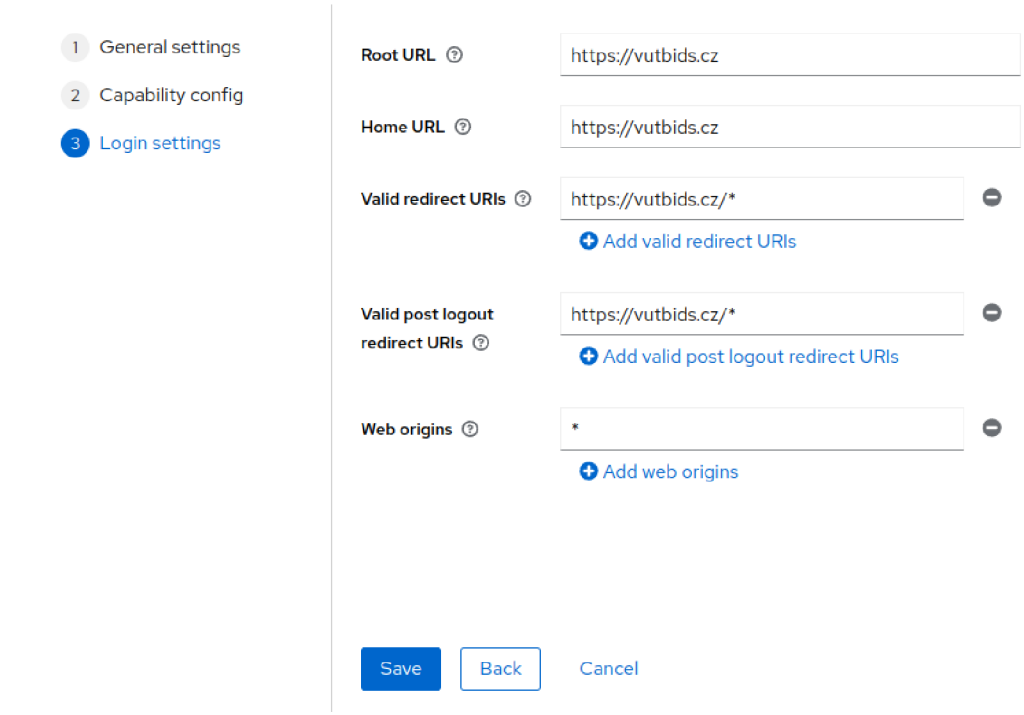
Keycloak nazývá toky k získání tokenu pod jinými názvy. `Standard flow` je ekvivalentní s Authorization Code Flow v OAuth 2.0, `Direct Access grants` je ekvivalení s Resource Owner Password Grant a `Service Account Roles` s Client Credentials Grant (sekce 4.2.5). Direct Access Grant by se neměl používat v produkci, nicméně může být povolen za účelem testování Keycloaku samotného. Obrázky 5.5, 5.6 a 5.7 ukazují registraci tohoto klienta v administrátorském rozhraní Keycloaku.



Obrázek 5.5: Registrace Keycloak klienta v administrátorském rozhraní, 1. část



Obrázek 5.6: Registrace Keycloak klienta v administrátorském rozhraní, 2. část



Obrázek 5.7: Registrace Keycloak klienta v administrátorském rozhraní, 3. část

Pro větší bezpečnost by se neměl vyskytovat žolíkový znak `*` v seznamu validních URI a měly by se definovat pouze URI s koncovými body, které s Keycloakem přímo pracují.

5.2.2 Role a skupiny

Keycloak podporuje definici rolí a skupin pro uživatele. Uživatelé tedy mohou mít přidělené role a také mohou patřit do skupin. Role a skupiny v rámci správy zdrojů nejsou nijak technicky rozdílné, oboje plní oprávnění k určitým omezeným zdrojům dle atributu uživatele. Uživatel s rolí `Admin` by měl tedy z pohledu systému stejné přístupy jako uživatel ve skupině `Admins`. Nicméně v sémantickém pojetí jsou role určeny pro správu přístupů a skupiny pro správu uživatelů.

V systému Keycloak lze nalézt již nemalé množství předdefinovaných rolí (servisní role), které slouží pro přístup a správě zdrojů Keycloaku. Tyto role mohou být přiděleny uživatelům nebo klientům. Klient vutbids bude mít tyto servisní role:

- `view-users`: Role umožňující čtení informací o uživateli.
- `query-users`: Role umožňující vyhledávání uživatelů dle parametrů.
- `manage-users`: Role umožňující správu uživatelů.

Jelikož je již mnoho předdefinovaných rolí, tak pro jednoduchost budou v systému VUTBids uživatelé diferencováni podle skupin. Budou definovány dvě skupiny:

- `Admins`, kde budou přiřazeni uživatelé s oprávněním administrátora.
- `Auctioneers`, kde budou přiřazeni uživatelé s oprávněním licitátora.

5.2.3 Integrace zdrojů identit

Uživatel bude mít k dispozici možnost se přihlašovat v Keycloaku přes tři zdroje identit:

- Google, který bude využívat OAuth 2.0
- VUT, který bude vnímán Keycloakem jako poskytovatel identit v protokolu SAML 2.0
- Lokálním Keycloak formulářem

Integrace Google identit

Integrace identit populárních organizací jako Google, Facebook nebo Github je v Keycloaku velmi jednoduchá, jelikož existuje návod pro každou z nich. V případě Googlu stačí v Keycloaku v realmu *vutbids* přejít na `Identity providers` → `Add provider`. Ve formuláři je potřeba zadat hodnoty Client ID a Client Secret, které musí poskytnout Google. Google má dostupnou dokumentaci pro vývojáře jak Client ID a Client Secret získat[2].

Integrace VUT identit

VUT aktuálně poskytuje identity přes SAML 2.0 protokol v roli poskytovatele identit, tudíž Keycloak má v tomto kontextu roli poskytovatele služeb. Pro rozjetí přihlašování přes VUT je potřeba mít k dispozici VUT metadata poskytovatele identit. Definice metadat poskytovatele identit VUT byla již ukázána ve výpisu **??**.

Pro vytvoření nového zdroje identit je potřeba v Keycloaku v administrátorském prostředí přejít na *Identity Providers* → *Add Provider* → *SAML v2.0*, kde bude představen formulář pro definici poskytovatele identit využivajícího SAML 2.0 protokolu. Tabulka 5.2.3 ukazuje použité hodnoty formuláře pro VUT:

Alias	BUT	Identifikátor, který musí být jedinečný v seznamu poskytovatele identit
Display name	Vysoké učení technické v Brně	Název poskytovatele, který je viditelný na přihlašovací stránce
Service provider entity ID	Výchozí hodnota	Entity ID poskytovatele služeb, ponechání ve výchozím stavu
Use entity descriptor	Vypnuto	Pokud povolen, tak import metadat přes URL, jinak přes XML soubor

Hodnota políčka `Alias` bude poté použita při vytváření vlastního přihlašovacího motivu.

Pro import VUT metadat použijeme XML soubor místo importu přes vzdáleného URL. Je tedy potřeba mít ve formuláři vypnuté políčko `Use entity descriptor`. Po importu metadat je dále ve formuláři potřeba definovat další pole dle tabulky 5.2.3.

[2] https://developers.google.com/identity/protocols/oauth2

Name ID policy format	Transient
Principal type	Attribute [Name]
Principal attribute	urn:oid:1.3.6.1.4.1.5923.1.1.1.13[3]

A formulář potvrdíme. Pro minimální funkčnost stačí nechat ostatní pole formuláře ve výchozím stavu. Zbytek polí lze později nastavit dle potřeby. Význam a použití dalších polí ve formuláři lze nalézt na ikonce otazníku.

Dále je potřeba mapovat jednotlivé atributy uživatele u poskytovatele identit VUT na atributy uživatele v Keycloaku. Keycloak umožňuje mapování atributů pomocí mapperů. Pro potřeby informačního systému VUTBids stačí požadujeme pouze atribut `mail`, což je atribut adresy elektronické pošty uživatele. Pro funkčnost je dále potřeba, aby byla SAML metadata Keycloaku registrována u poskytovatele identit VUT. K tomu je potřeba zaslat metadata odpovědné osobě, která má metadata ze strany VUT na starost. Jakmile budou Keycloak metadata registrována u VUT, tak

5.2.4 Nastavení tokenů

Keycloak nabízí velké množství nastavení jak přístupového, obnovovacího a ID tokenu, které jsou ve formě JWT. K podepisování tokenů v rámci realmu lze využít z široké škály dostupných podpisových algoritmů, včetně symetrických (např. HS256) nebo asymetrických (např. RS256 nebo ES256) za pomocí veřejných klíčů.

Realm vutbids bude podepisovat tokeny pomocí asymetrického podpisového algoritmu RS256, který je dnes nejvíce využívaný a bezpečnější. Klienti by si tedy ukládali pouze veřejný klíč, který by byl použit pouze pro validaci tokenu.

Pokud by se očekávalo, že by webová aplikace měla enormní návštěvnost, kde by docházelo k velkému množství procesů dekódování tokenu najednou, tak lze uvažovat i o použití ES256 algoritmu, pomocí kterého lze dekódovat tokeny rychleji, jelikož používá veřejný klíč s menší velikostí.

Životnost přístupového tokenu

Přístupový token v Keycloaku se nastavuje v `Realm settings` → `Tokens`. Životnost přístupové tokenu by měla být krátká, aby se zmenšilo riziko zneužití tokenu v případě odcizení. Realm *vutbids* má nastavenou životnost přístupového tokenu na 5 minut.

Životnost obnovovacího tokenu

Jelikož nechceme, aby se uživatel po 5 minutách musel pokaždé přihlašovat, tak využijeme obnovacího tokenu. Obnovovací token nelze v Keycloaku 23 nastavit napřímo a jeho životnost se nastavuje relativně složitě.

Keycloak si po přihlášení uživatele drží jeho relaci. Nastavení relace definuje maximální životnost obnovacího tokenu. Nastavení relace se nachází v `Realm settings` → `Sessions` a má dvě hlavní pole:

- `SSO Session Idle`: Doba, po kterou může být relace nečinná, než vyprší.
- `SSO Session Max`: Maximální doba, kdy je relace platná.

Pomocí těchto dvou polí lze nastavit životnost obnovacího tokenu následovně:

[3] https://www.eduid.cz/cs/tech/attributes/edupersonuniqueid

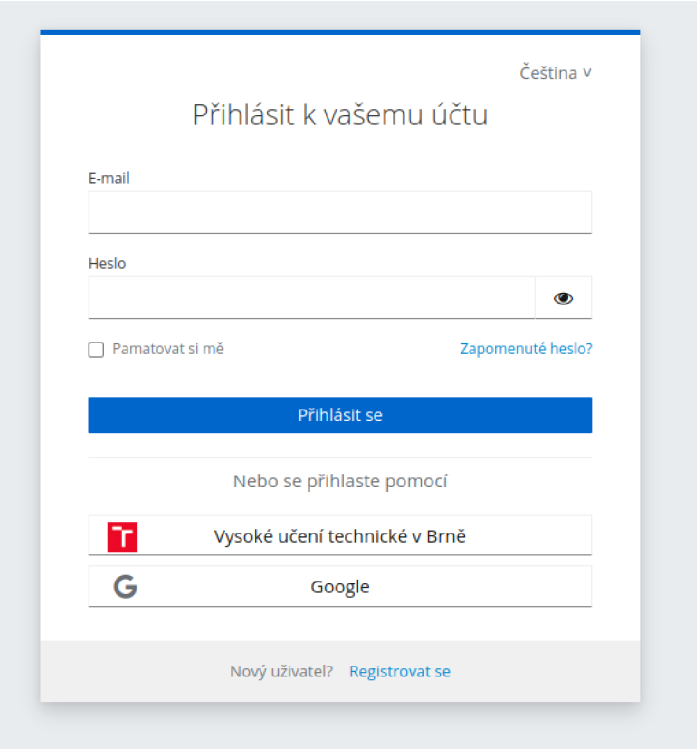
- Pokud je `SSO Session Max > SSO Session Idle`, tak životnost obnovacího tokenu je doba `SSO Session Idle`.
- Pokud je `SSO Session Max <= SSO Session Idle`, tak životnost obnovovacího tokenu je doba `SSO Session Max`.

5.2.5 Integrace vlastního motivu vzhledu

Keycloak poskytuje uživatelům možnost vytvářet vlastní motivy vzhledu svých individuálních webových stránek, například vzhled přihlašovací stránky. To přispívá k jednotnému vizuálnímu stylu napříč různými aplikacemi a zlepšení uživatelské zkušenosti.

Keycloak 23 umožňuje úpravy vzhledu pomocí šablonovacího systému **FreeMarker**[4], který slouží pro generování statických webových stránek navržen podle vzoru MVC.

Pro vytvoření vlastního motivu, který bude připraven na nasazení, je potřeba vytvořit Java Service Provider Interface (SPI), který v adresáři `resources` obsahuje soubory šablon FreeMarker, JavaScript či soubory kaskádových stylů. Keycloak poskytuje nemalé množství předdefinových stránek, které lze rozšířit či upravit dle potřeby za pomocí šablon, JavaScriptu a CSS souborů. Více informací jak rozšířit a upravit předdefinované stránky a připravit motiv vzhledu, lze najít v dokumentaci Keycloaku[5]. Pro účely infomačního systému VUTBids stačí rozšířit a upravit předdefinovanou stránku `login.ftl`, což je šablona pro přihlašování uživatelů.



Obrázek 5.8: Přihlašovací stránka pro vutbids klienta

[4] freemarker.apache.org
[5] https://www.keycloak.org/docs/latest/server_development/#_themes

5.2.6 Nasazení Keycloaku

Keycloak je nasazen za pomocí technologie kontejnerizace a připraven jak na nasazení čistě v Dockeru nebo v cloudu v Kubernetes clusteru. Jelikož Keycloak je samostatný software, musí běžet Keycloak na jiné adrese než klienti, kterým poskytuje služby. Samozřejmostí je použití HTTPS pro šifrování komunikace mezi Keycloakem a klienty.

5.3 Informační systém VUTBids

Účelem informačního systému VUTBids je pořádání internetových aukcí určených primárně pro studenty VUT, ale i externích subjektů. Každá aukce má iniciálně nějaké atributy: typ aukce (nabídková, poptávková), autor aukce, popis zboží/majetku a vyvolávací cenu, apod.

Uživatelé budou moci dále informační systém použít konkrétně v zjednodušeném pojetí následujícím způsobem:

Administrátor

- Spravuje uživatele v rámci informačního systému. Uživatele spravuje v Keycloaku.
- Má rovněž práva všech následujících rolí.

Licitátor

- Schvaluje navrhované aukce registrovaných uživatelů (stává se organizátorem aukce).
- Potvrzuje registrace účastníků.
- Plánuje, spouští, ukončuje, vyhodnocuje aukce a zveřejňuje jejich výsledky.
- Má rovněž práva registrovaného uživatele (nicméně nesmí být licitátorem v rámci aukcí, ve kterých je autorem nebo účastníkem).

Registrovaný uživatel

- Zakládá a navrhuje aukce – po schválení se stane autorem aukce (stává se autorem aukce).
- Nastavuje a upravuje parametry aukce, kde je autorem.
- Registruje se do aukcí, kde není autorem – po schválení se stane účastníkem aukce.
- Sleduje stav jeho registrace.
- Sleduje stav aukcí, provádí příhozy k zůčastněným aukcím.
- Vidí výsledky aukce po zveřejnění vítěze licitátorem.

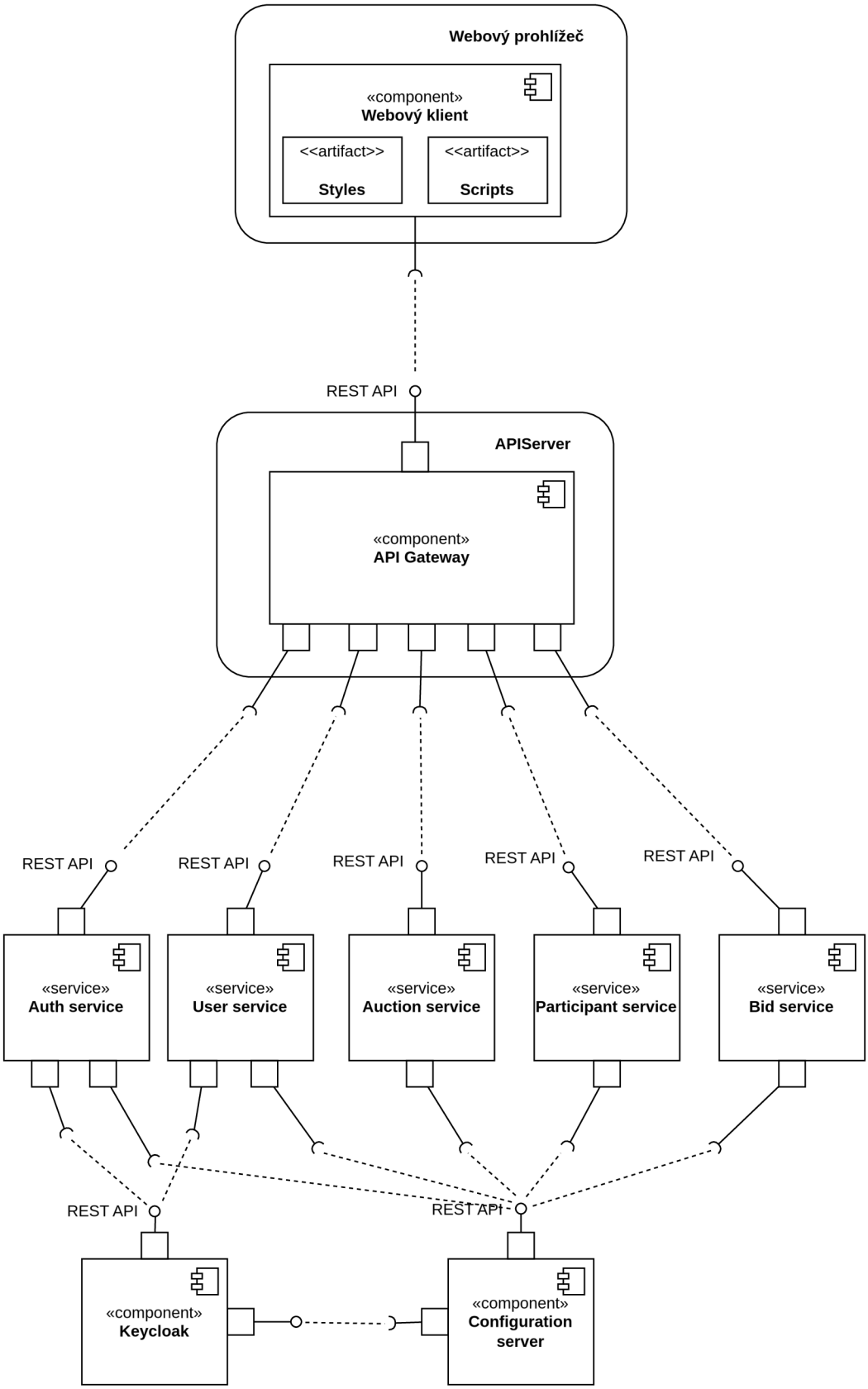
5.3.1 Model informačního systému

Systém se skládá z těchto mikroslužeb:

- **Auth service**: Mikroslužba zabývající se přihlašování, odhlašování a správou tokenů uživatelů. Bude využívat služeb Keycloaku pro autentizaci a ověřování tokenů.
- **User service**: Mikroslužba, která pracuje s informacemi o uživatelích, kteří jsou uloženi v Keycloaku.
- **Auction service**: Mikroslužba, která udržuje a pracuje s aukcemi uživatelů. Služba bude mít vlastní databázi s jednou tabulkou, která udržuje aukce.
- **Bid service**: Mikroslužba, která udržuje a pracuje s příhozy aukce. Podobně jak Auction service bude mít tato služba také vlastní databázi s jednou tabulkou, která obsahuje záznamy příhozů jednotlivých aukcí.
- **Participant service**: Mikroslužba, která eviduje a pracuje se zůčastněnými uživateli v určité aukci.

Jednotlivé mikroslužby v rámci aplikace musí typicky sdílet konfigurace jako veřejný klíč pro dekódování tokenu. K tomu budou využívat služeb **konfiguračního serveru**, kde jednotná konfigurace bude dostupná přes jeho REST API.

Jednotlivé mikroslužby budou maskované za **API gateway**, která bude směrovat HTTP požadavky na odpovídající mikroslužby. Obrázek 5.9 na další straně ukazuje UML diagram komponent informačního systému.



Obrázek 5.9: UML diagram komponent informačního systému VUTBids

5.3.2 Použité technologie

Technologie, které byly využity při vývoji této webové aplikace, byly vybrány s ohledem na aktuální trendy vývoje aplikací založeného na architektuře mikroslužeb. Důraz byl kladen na nástroje a technologie, které souvisí s technologií *kontejnerů* a *kontejnerizace*[6], což je typ virtualizace. Kontejnery poskytují flexibilní a izolované prostředí pro běh aplikací, což umožňuje snadné nasazení, škálování a správu jednotlivých mikroslužeb.

Docker

Docker je open-source software, který umožňuje vytvářet a spouštět aplikace v kontejnerech. Kontejnery mají samostatné běhové prostředí, které obsahuje všechno potřebné pro běh aplikace, včetně knihoven, závislostí a konfigurace. Jednotlivé kontejnery jsou tedy od sebe izolované a nijak si nepřekáží. K vytvoření kontejneru v Dockeru je potřeba definovat kontejner *image*, což je šablona, která obsahuje definice kontejneru jako například prostředí, na kterém bude kontejner běžet, knihovny, které aplikace vyžaduje a aplikaci samotnou.[7]

Kubernetes

Kubernetes, zkráceně často mezi vývojáři jako K8s, je open-source software pro orchestraci kontejnerů. Kubernetes sámostatně nedokáže kontejnery přímo rozjíždět a ani vytvářet kontejner image, tudíž musí bežet společně s kontejner softwarem jako Docker. Kubernetes využívá již existujicích kontejner image pouze pro automatizaci procesů s kontejnery jako nasazení, správu či škálování. Podporuje jak vertikální škálování, kdy kontejneru je dodáno více hardwarových zdrojů, tak i horizontální škálování, kdy aplikace je dostupná z více instancí stejného kontejneru a zátěž je rozdělena mezi nimi.

Kontejnery běží v takzvaném *Kubernetes clusteru*, kdy cluster může být rozdělen do několika *Kubernetes node*. Nody mohou být fyzické nebo virtuální servery, které jsou propojeny pomocí síťového rozhraní, aby mohly komunikovat a sdílet stav kontejnerů mezi sebou. Nody komunikují přes HTTP nebo HTTPS protokol a Kubernetes orchestrátor zajistí správné nasazení kontejneru a škálování podle definovaných pravidel a požadavků.

Kubernetes definuje objekty, kde každý objekt reprezentuje určitý zdroj nebo pravidlo v rámci kontejnerů a virtualizace. Budou zmíněny pouze ty nejdůležitější:

- *Pod*: Je objekt a nejmenší nasaditelná jednotka pro nasazení a běh kontejnerů. V podu může být jeden nebo více kontejnerů. Kontejnery v podu sdílejí disková pole, kde jedno diskové pole může být připojeno k více kontejnerům, a síťové rozhraní, kde dva různé kontejnery ve stejném podu nemohou být vázáni na stejný síťový port. V podu mohou být různé definice *volumes*, síťových portů, minimálních hardwarový požadavků či limitů atd. Pody jsou součástí jiných objektů, jako *ReplicaSet*, *Deployment* a *StatefulSet*, což jsou objekty pro běh více podů a jejich kontejnerů. V běžném provozu by měl mít pod pouze jeden kontejner s případnými menšími pomocnými kontejnery.
- *ReplicaSet*: Je objekt, který definuje Pod a počet jeho instancí (replik), které by měly běžet. ReplicaSet je součástí objektu *Deployment*, který se v praxi využívá častěji.
- *Deployment*: Podobný objektu ReplicaSet, ale vysokoúrovnější a s většími možnostmi pravidel. Obsahuje definice podu a jeho repliky. Tento objekt by měl být pro bezstavové aplikace, které si neudržují stav.

[6] Více o kontejnerech: https://en.wikipedia.org/wiki/Containerization_(computing)
[7] Dokumentace Dockeru: https://docs.docker.com/

- *StatefulSet*: Tento objekt je podobný objektu Deployment, ale je pro aplikace, které si udržují stav, jako například databáze.
- *Service*: Je objekt a abstrakce, která definuje způsob síťového přístupu k určitým replikám podů a jak se na ně mohou ostatní pody připojit. Tento objekt umožňuje komunikaci mezi různými mikroslužbami v rámci clusteru v rámci interního DNS. To umožňuje objekt Service typu *ClusterIP*.

Minikube

Minikube je jedna z implementací Kubernetes pro běh na jednom lokálním počítači. Poskytuje jednoduchý Kubernetes cluster pro vývoj a testování aplikací v Kubernetes. Je vhodný pro vývojáře, kteří chtějí rychle vytvořit a spustit Kubernetes prostředí bez nutnosti složitého nasazení na vzdálené datacentrum serverů.

Kong Gateway

Kong Gateway je open-source API gateway, která umožňuje správu, monitorování a zabezpečení mikroslužeb a jejich API. Poskytuje funkce jako směrování, autentizace, autorizace a rate limiting. V rámci této práce bude Kong Gateway využit jen pro směrování koncových bodů ke správným mikroslužbám.

Flask

Flask je open-source webový framework napsaný v jazyce Python pro vytváření webových aplikací a API. Flask je známý svou jednoduchostí a flexibilitou, což z něj činí oblíbenou volbu pro začátečníky i pokročilé vývojáře. Všechny mikroslužby budou implementovány v tomto frameworku.

Flask-SQLAlchemy

Flask-SQLAlchemy je rozšíření pro framework Flask, které poskytuje integraci mezi Flaskem a SQLAlchemy, což je populární knihovna v jazyce Python umožňující ORM (Object Relational Mapping) s databázemi. Toto rozšíření usnadňuje práci s relačními databázemi a umožňuje vývojářům efektivněji manipulovat s daty pomocí objektů.

Vue.js

Pro implementaci webového klienta byl zvolen javascriptový framework Vue.js, který slouží pro vytváření uživatelských rozhraní a je oblíbenou volbou mezi vývojáři kvůli jednoduché syntaxi a počáteční účící křivce. Vue.js umožňuje deklarativní způsob definice UI komponent, což usnadňuje organizaci kódu a jeho znovupoužitelnost. S Vue.js byl využity jeho rozšíření Pinia a Vuetify.

Pinia

Pinia je knihovna pro udržování a řízení stavů pro Vue.js framework. Pinia nabízí reaktivní uložiště stavů, což znamená, že všechny UI komponenty využívající tohoto uložiště se aktualizují při změně dat v uložišti. Hlavní motivace existence této knihovny je zlepšení organizace a zjednodušení zdrojového kódu při aktualizací dat v komponentech, což vede k lepší čitelnosti a udržovatelnosti kódu.

Vuetify

Vuetify je populární knihovna UI komponent založená na Vue.js, která nabízí rozsáhlou sadu předdefinových komponent pro rychlé vytváření moderních uživatelských rozhraní. Knihovna je postavena na základech Material Designu od Googlu, což zajišťuje moderní a konzistentní vzhled aplikace.

5.4 Příprava a nasazení konfiguračního serveru

Jak už bylo řečeno, existují určité konfigurace, které musí být sdíleny mezi všemi mikroslužbami v rámci celé aplikace, jako je například veřejný klíč pro dekódování tokenu. Pokud by každá mikroslužba uchovávala vlastní kopií téže konfigurace, zvyšovalo by to riziko chyb při údržbě a vývoji aplikace, například pokud by se zapomněla aktualizovat konfigurace jedné mikroslužby.

Konfigurační server eliminuje tyto potenciální chyby tím, že centralizuje řízení konfigurace z jednoho zdroje. Každá mikroslužba poté může extrahovat aktuální konfigurace, které budou potřebovat v rámci své funkcionality a běhu. Aktuálně konfigurační server poskytuje konfigurace na koncovém bodě `/config` ve formě JSON, kde jsou všechny sdílené definice konfigurace.

```
{
  "KEYCLOAK_URL": "https://login.vutbids.cz",
  "HOST_URL": "http://vutbids.cz",
  "CLIENT_ID": "vutbids",
  "CLIENT_SECRET": "K32feyoig299Ke...",
  "PUBKEY": "MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AM..."
}
```

Konfigurační server by obvykle měl být nasazen a běžet na separátní adrese, která by měla být přístupná pouze mikroslužbám.

5.5 Implementace mikroslužeb

Jednotlivé mikroslužby jsou samostatně jednoduché na implementaci a plní svoji specifickou roli v rámci celé aplikace. Proto je v budoucnu možné je snadno přenést do jiného programovacího jazyka, který by byl pro danou mikroslužbu vhodnější. Mikroslužby navzájem komunikují přes REST API.

Každá mikroslužba prošla vývojem jako samostatná aplikace ve Flask frameworku s rozšířením Flask-RESTful. Díky Flask-RESTful je možné jednotlivé koncové body aplikace definovat jako třídy, kde jednotlivé HTTP metody jsou implementovány jako metody těchto tříd.

```
class HelloWorld(Resource):
    def get(self):
        return {'message': 'Toto je GET požadavek!'}

    def post(self):
        return {'message': 'Toto je POST požadavek!'}

api.add_resource(HelloWorld, '/helloworld')
```

Všechny mikroslužby sdílí ve zdrojovém kódu dekorátor `@token_required` pro validaci tokenu. Tento dekorátor se přidává před definicí metody, která zpracovává specifickou HTTP metodu konkrétního koncového bodu. Pokud je tento dekorátor přidán, tak token bude ověřen před zpracováním požadavku.

```
class HelloWorld(Resource):
    def get(self):
        return {'message': 'Toto je GET požadavek!'}

    @token_required
    def post(self):
        return {'message': 'Token úspěšně ověřen!'}
```

Samotný dekorátor využívá pro dekódování a validaci tokenu Python knihovny `PyJWT`[8] za pomocí metody `decode()`.

```
try:
    decoded = jwt.decode(token,
                         key_binary,
                         audience=["account"],
                         algorithms=["RS256"])
    app.logger.debug("Token is valid")
    g.decoded_token = decoded
except jwt.ExpiredSignatureError as e:
    app.logger.error(e)
    app.logger.error("Token expired")
    abort(401)
```

Po ověření tokenu pomocí dekorátoru je token uložen v objektu `g`, což je objekt dostupný po celou dobu trvání požadavku a je specifická pro Flask framework.

[8] pyjwt.readthedocs.io

5.5.1 Auth service

Tato mikroslužba vykonává úkony související s autentizací a odhlašování uživatelů v Keycloaku a manipulací s tokeny získané z Keycloaku.

Jelikož mikroslužba pracuje s Keycloakem v roli klienta v protokolu OpenID Connect (viz. sekce 5.2.1), tak je nezbytné, aby mikroslužba pracovala s knihovnami nebo rozšířeními, které řeší OpenID Connect nebo OAuth 2.0 funkcionality v roli klienta.

Tato mikroslužba využívá konkrétně Python knihovny *Authlib*[9], která usnadňuje práci s OAuth 2.0 a OpenID Connect ve webových aplikacích založené na Pythonu. Knihovna podporuje integraci populárních Python webových frameworků jako Django a Flask.

Dle dokumentace knihovny je nutné v Python aplikaci registrovat OAuth 2.0/OpenID Connect klienta pomocí objektu `OAuth(app)`, kde argument `app` je instance Flask aplikace.

Pro registraci klienta v aplikaci je potřeba definovat Client ID a Client Secret. Dále je potřeba definovat koncové body Keycloaku *Authorization Endpoint* (`authorize_url`) a *Token Endpoint* (`access_token_url`) ze sekce 4.2.1 a případně volitelný koncový bod OIDC Discovery, který obsahuje metadata Keycloaku jako poskytovatele OpenID. OAuth scopes má povinnou hodnotu `openid` a další hodnoty `email` a `profile`, pro čtení emailu a profilu uživatele.

```
KEYCLOAK_URL = https://login.vutbids.cz
oauth = OAuth(app)
oauth.register('keycloak',
              client_id=CLIENT_ID,
              client_secret=CLIENT_SECRET,
              access_token_url=KEYCLOAK_URL +
              "/realms/vutbids/protocol/openid-connect/token",
              authorize_url=KEYCLOAK_URL +
              "/realms/vutbids/protocol/openid-connect/auth",
              server_metadata_url=KEYCLOAK_URL+
              "/realms/vutbids/.well-known/openid-configuration",
              client_kwargs={
                  'scope': 'openid email profile'
               },
          )
```

[9] authlib.org

Koncové body

Tabulka 5.2 ukazuje hlavní koncové body, které tato služba implementuje:

Koncový bod	Popis
`/login`	Koncový bod pro iniciaci procesu přihlášení uživatele
`/authorize`	Koncový bod pro získání tokenu, který je nezbytný při komunikačním toku OpenID Connect
`/logout`	Koncový bod pro odhlášení uživatele
`/api/account/linked`	Koncový bod vrací připojené externí účty aktuálně přihlášeného uživatele
`/api/token/refresh`	Koncový bod dodá nový přístupový token od Keycloaku při dodání obnovovacího tokenu
`/api/token/introspect`	Koncový bod, který ověřuje platnost přístupového tokenu v Keycloaku

Tabulka 5.2: Tabulka příkladů hlavních koncových bodů mikroslužby Auth service

/login

Koncový bod `/login` žádá o autentizaci Keycloaku s metodou **Authorization Code Flow**, která byla popsána v sekci 4.2.2.

```
# Login endpoint
class Login(Resource):
    def get(self):
        redirect_uri = url_for('authorize', _external=True)
        resp = oauth.keycloak.authorize_redirect(redirect_uri)
        return resp
```

Metoda `oauth.keycloak.authorize_redirect` vytvoří HTTP odpověď s přesměrováním prohlížeče ke Keycloaku. Přesměrovací URL Keycloaku obsahuje parametr `redirect_uri`, který má návratové URI informačního systému s koncovým bodem `/authorize`. To odpovídá prvnímu kroku metody Authorization Code ze sekce 4.2.5 a sekce 4.3.4. Pokud uživatel ještě nemá v Keycloaku relaci, tak se přihlásí pomocí formuláře (obrázek 5.8).

Po případném přihlášení uživatele přesměruje Keycloak prohlížeč zpátky k informačnímu systému ke koncovému bodu `/authorize` s autorizačním kódem v URL parametru.

/authorize

Koncový bod `/authorize` slouží pro získání přístupového tokenu a měl by se provést pouze po přesměrování z Keycloaku, kdy Keycloak při přesměrování na tento koncový bod poskytl po autentizačním požadavku autorizační kód.

```
# Authorize endpoint
class Authorize(Resource):
    def get(self):
        token = oauth.keycloak.authorize_access_token()
        resp = make_response(redirect(HOST_URL))
        resp.set_cookie('access_token',
                        str(token['access_token']),
                        httponly=True,
                        samesite=True)
        resp.set_cookie('id_token',
                        str(token['id_token']),
                        httponly=True,
                        samesite=True)
        resp.set_cookie('refresh_token',
                        str(token['refresh_token']),
                        httponly=True, samesite=True)
        return resp
```

Metoda `oauth.keycloak.authorize_access_token` vymění získaný autorizační kód za přístupový token, obnovovací token a ID token. Jednotlivé tokeny jsou poté uloženy v HTTP Cookie s atributy `SameSite` a `HttpOnly`.

Ukládání tokenů

Ukládání tokenů v HTTP cookie s těmito atributy se považuje za jednu z více bezpečných, ale má určité nevýhody. Při každém požadavku se přenáší obsah cookie, což znamená větší objem dat při HTTP komunikaci.

Existuje více způsobů jak uložit tokeny v prohlížeči mimo HTTP cookie, a to například pomocí `localStorage`. Mezi výhodami pro ukládání tokenu v `localStorage` patří [30]:

- Jednoduchý přístup k tokenům a použití tokenů v JavaScriptu.
- Při každém požadavku se nepřenáší žádná data cookie. Token je možné volitelně přenášet přes HTTP hlavičku `Authorization`, což znamená menší objem dat při HTTP komunikaci.

Mezi nevýhodami patří:

- Zranitelnější vůči útokům typu XSS (Cross-Site Scripting), kdy útočník může spustit škodlivý JavaScript a jednoduše token extrahovat z `localStorage`. Tato zranitelnost je obzvláště nebezpečná v případě obnovovacího tokenu, který má dlouhou životnost.

Bylo tedy zvoleno ukládání tokenů v cookie s atributem `HttpOnly`, díky kterému tokeny nelze zpřístupnit JavaSriptem. Nicméně `localStorage` je taky dostupná varianta, pokud je správně implementována a rizika jsou v úvahu.

`/logout`

Koncový bod odstraňuje relaci uživatele v Keycloaku a odstraní všechny tokeny z cookie. Volá koncový bod Keycloaku `/realms/vutbids/protocol/openid-connect/logout`, který odstraní relaci uživatele. Tento koncový bod Keycloaku k odhlašování podporuje jak GET požadavek tak i POST požadavek. K odhlášení bylo využito GET metody, který potřebuje následující parametry k odhlášení uživatele

- `client_id`: Client ID klienta, v našem případě *vutbids*.
- `refresh_token`: Obnovovací token uživatele, který je spojen s jeho relací v Keycloaku.
- `post_logout_redirect_uri`: Návratové URI po odhlášení.
- `id_token_hint`: ID token, se kterým bude Keycloak pracovat.

5.5.2 User service

Mikroslužba komunikuje s Keycloakem pro extrakci informací o uživateli. Dále zobrazuje aukce uživatele, které vlastní nebo vyhrál. Tato mikroslužba nijak nevytváří a ani nespravuje uživatele. Ke správě identit a uživatelů slouží právě samotný Keycloak. Identifikátory uživatelů (`id`) v Keycloaku jsou definovány jako UUID[10]. Tabulka 5.3 ukazuje příklady koncových bodů REST API, které tato mikroslužba implementuje.

Koncový bod	**Popis**
`/api/users/<id>`	Najde uživatele v Keycloaku dle ID
`/api/users/<id>/auctions/owned`	Vrací vlastněné aukce uživatele dle jeho ID.
`/api/users/<id>/auctions/won`	Vrací vyhrané aukce uživatele dle jeho ID.

Tabulka 5.3: Tabulka příkladů hlavních koncových bodů mikroslužby Auth service

Koncové body, které vrací aukce uživatele, komunikují s mikroslužbou Auction service (sekce 5.5.3) pro extrakci a filtraci aukcí dle identifikátoru uživatele.

[10] https://en.wikipedia.org/wiki/Universally_unique_identifier

5.5.3 Auction service

Tato mikroslužba vykonává úkony související s aukcemi, které vlastní registrovaní uživatelé. Mikroslužba pracuje s databází, která obsahuje jednu SQL tabulku 5.4 reprezentující aukce.

Název sloupce	Datový typ	Popis
id	INT	Jedinečný indentifikátor aukce
name	VARCHAR	Název aukce
desc	VARCHAR	Popis aukce
state	ENUM	Aktuální stav aukce
initial_price	FLOAT	Počáteční cena aukce
created	DATETIME	Čas, kdy byla aukce vytvořena
date_start	DATE	Datum, kdy aukce začíná
date_end	DATE	Datum, kdy aukce končí
author_id	VARCHAR	Identifikátor autora aukce
auctioneer_id	VARCHAR	Identifikátor licitátora aukce
winner_id	VARCHAR	Identifikátor vítěze aukce

Tabulka 5.4: Struktura SQL tabulky pro aukce

Aukce se vytváří a spravují pouze přes REST API na následujících koncových bodech:

- `/api/auctions`: Přijímá GET požadavek na čtení aukcí a POST požadavek k vytvoření nové aukce.
- `/api/auctions/<id>`: Přijímá GET, PUT a DELETE požadavek pro čtení, editace a mazání jediné aukce dle identifikátoru.

Při GET požadavku lze aukce filtrovat pomocí URL parametrů na základě jednotlivých hodnot sloupců SQL tabulky. HTTP požadavky, které nějakým způsobem mění stav databáze jsou chráněny za přístupovým tokenem za pomocí dekorátoru `@token_required`. Po validaci a dekódování přístupového tokenu se provedou operace, který je uživatel autorizován vykonat dle informací z tokenu.

Založit aukci může jakýkoliv uživatel, který má platný přístupový token z Keycloaku. Uživatel vyplní název, popis, předběžný termín začátku aukce a předběžný termín konce aukce. Po založení se aukce dostane do stavu `PLANNED`, což znamená, že aukce je definována, ale nemá licitátora. Pouze uživatel ve skupině `Auctioneers` (viz. sekce 5.2.2) může být licitátorem aukce a změnit aukci z `PLANNED` na `READY`, což je stav, kdy aukce již má licitátora a je připravená. Licitátor případně ještě může změnit datum začátku aukce a konce aukce. Jakmile aukce začne, změní se na stav `ONGOING`.

Účastníci a příhozy účastníků se mohou přidávat pouze v průběhu začátku a konce aukce (stav `ONGOING`). Účastníci se vytvářejí a spravují v mikroslužbě Participant service (sekce 5.5.4) a příhozy v Bid service (sekce 5.5.5)

Po skončení se aukce změní ze stavu `ONGOING` na `ENDED` a licitátor zvolí vítěze podle příhozů účastníků.

5.5.4 Participant service

V této mikroslužbě se vytvářejí a spravují účastníci aukce. Mikroslužba také pracuje s nezávislou databází, která také obsahuje pouze jednu SQL tabulku 5.5 reprezentující účastníky aukce.

Název sloupce	Datový typ	Popis
id	INT	Jedinečný identifikátor účasti
participant_id	VARCHAR	Identifikátor účastníka
auction_id	VARCHAR	Identifikátor aukce
joined	DATETIME	Čas, kdy byla účast vytvořena
approved	BOOLEAN	Sloupec říká, jestli je účast potvrzena

Tabulka 5.5: Struktura SQL tabulky pro účastníky

Účastníci se také vytváří a spravují pouze přes REST API na následujících koncových bodech:

- `/api/participants`: Přijímá GET požadavek na čtení účastníků a POST požadavek k vytvoření nového účastníka.
- `/api/participants/<id>`: Přijímá GET, PUT a DELETE požadavek pro čtení, editace a mazání účastníka dle identifikátoru.

Všechny koncové body pro vytváření či úpravu účastníků jsou chráněné přístupovým tokenem. Účastníci se mohou vytvářet pouze v aukcích, které jsou ve stavu `ONGOING`. Po vytvoření nejsou účastníci potvrzeni, potvrdit je může pouze licitátor dané aukce, které se chtějí účastnit.

5.5.5 Bid service

V této mikroslužbě se vytvářejí a spravují příhozy účastníků aukce. Mikroslužba také pracuje s nezávislou databází, která také obsahuje pouze jednu SQL tabulku 5.6 reprezentující příhozy aukce.

Název sloupce	Datový typ	Popis
id	INT	Jedinečný identifikátor příhozu
participant_id	VARCHAR	Identifikátor účastníka
auction_id	VARCHAR	Identifikátor aukce
created	DATETIME	Čas, kdy byl příhoz vytvořen
amount	FLOAT	Suma příhozu

Tabulka 5.6: Struktura SQL tabulky pro příhozy

Účastníci se také vytváří a spravují pouze přes REST API na následujících koncových bodech:

- `/api/bids`: Přijímá GET požadavek na čtení příhozů a POST požadavek k vytvoření nového příhozu.

- `/api/bids/<id>`: Přijímá pouze DELETE požadavek pro mazání příhozu dle identifikátoru.

Stejně jako v Participant service jsou všechny koncové body pro vytváření či úpravu příhozů jsou chráněné přístupovým tokenem. Příhozy lze pouze vytvářet a upravovat pokud je uživatel účastníkem dané aukce.

5.5.6 Gateway service

Gateway služba, provozovaná pomocí open–source programu Kong Gateway, umožňuje přístup ke koncovým bodům mikroslužbeb pod jednu adresu. Služba směruje koncové body ke správným mikroslužbám.

- Koncové body `/login`, `/authorize`, `/logout` apod. → **Auth Service**
- Koncové body `/api/users/*` → **User Service**
- Koncové body `/api/auctions/*` → **Auction Service**
- Koncové body `/api/participants/*` → **Participant Service**
- Koncové body `/api/bids/*` → **Bid Service**

Konfigurace směrování je v Kong Gateway definována za pomocí konfiguračního souboru `kong.yml`.

5.6 Webové rozhraní

Webové rozhraní bylo implementováno v javascriptovém frameworku Vue.js s rozšířením Vuetify pro jednoduché vytváření UI komponentů a rozšířením Pinia pro pohodlné udržování stavů dat mezi komponenty.

Webové rozhraní využívá *Vue router* pro řízení navigace v aplikaci na straně klienta. To umožňuje dynamické zobrazení různých komponent a návštěv jiných koncových bodů URL adres bez znovunačtení stránky. Tabulka 5.7 ukazuje příklady koncových bodů, které servírují komponenty a pohledy webového rozhraní Vue.js.

Koncový bod	**Popis**
`/`	Domovská stránka, zobrazuje všechny aukce z `/api/auctions`
`/account`	Zobrazuje informace o přihlášeném uživateli
`/account/owned`	Zobrazuje vlastněné aukce aktuálně přihlášeného uživatele
`/users/<id>`	Zobrazuje informace o uživateli z `/api/users/<id>`
`/auctions/<id>`	Zobrazuje aukci z `/api/auctions/<id>`

Tabulka 5.7: Tabulka příkladů hlavních koncových bodů webového rozhraní

Pro komunikaci s REST API mikroslužeb je použita knihovna *Axios*. K instanci objektu Axios je vázána logika k obnovení přístupového tokenu za pomocí další knihovny *axios-auth-refresh*, která detekuje HTTP odpovědi s kódem `401 (Unauthorized)`. Po detekci HTTP

odpovědi s tímto kódem provede knihovna volání nad koncovým bodem `/api/token/refresh` mikroslužby Auth service (sekce 5.5.1), který pošle nový přístupový token po předložení obnovacího tokenu, který se nachází v HTTP cookie. Pokud Keycloak nezašle obnovovací token, zavolá se koncový bod pro odhlášení a odstranění tokenů.

Zdrojové kódy webového rozhraní zachovávají běžnou adresářovou strukturu Vue.js aplikací. Adresář *components* obsahuje UI komponenty webového rozhraní. Adresář *router* obsahuje definice koncových bodů a UI komponentů, které se k těmto bodům vážou a jsou definovány v adresáři *views*. Adresář *store* obsahuje definice objektů, které extrahují a manipulují s daty z REST API a reaktivně aktualizují tato data v UI komponentech, které tyto objekty využívají.

5.7 Nasazení

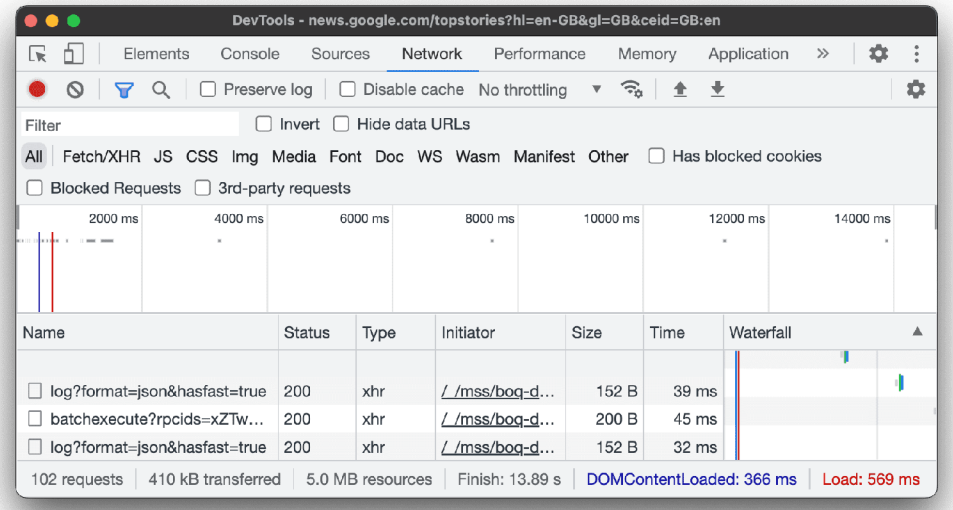
Každá mikroslužba je připravená na rozjetí v Docker kontejneru a v Kubernetes clusteru jak na lokálním počítači, tak v nějakém produkčním clusteru. V každém zdrojovém kódu mikroslužby se nachází definice příslušných *Dockerfile* souborů a Kubernetes manifestů. Tudíž je potřeba pouze upravit tyto `yml` soubory a konfigurace pro specifické Kubernetes prostředí. Nasazení není pro nového vývojáře přímočaré a jednoduchý návod nestačí, jelikož každý Kubernetes cluster není stejný a pro celkové rozjetí aplikace je potřeba vývojáře, který má znalosti o Kubernetes a všech mikroslužbách, například na jakých portech běží jednotlivé mikroslužby, aby mohl všechny mikroslužby komunikovat jako celek. Především je důležité se ujistit, jestli API Gateway směruje na správné mikroslužby a jestli všechny mikroslužby odpovídají.

Kapitola 6

Testování

Jednotlivé mikroslužby byly při vývoji testovány manuálně jako samostatná aplikace, kde každá mikroslužba běžela při vývoji a testování lokálně na svém separátním dostupném portu. Bylo nezbytné nejdříve implementovat a otestovat mikroslužbu Auth service, která se stará o autentizaci a předávání tokenů. Testování této mikroslužby probíhalo za pomocí aplikace Postman, která umožňuje pohodlné testování koncových bodů REST API na danou mikroslužbu přes jeho rozhraní.

Po implementaci a testování funkcionality mikroslužby Auth service bylo již možné získávat přístupové tokeny pro testování přístupu a práv dalších mikroslužeb. Testování následujících mikroslužeb probíhalo ručně za pomocí webového prohlížeče a jeho vývojové konzole. Díky vývojové konzoli bylo možné pozorovat celý tok komunikace s podrobnostmi mezi webovým rozhraním a mikroslužbami.



Obrázek 6.1: Vývojová konzole Google Chrome pro vývojáře

Přístupový token pro účely pohodlného testování měl dlouhou životnost. Při testování byly samozřejmě v některých mikroslužbách nalezeny chyby v business logice, kde se některé stihly opravit a některé ne vzhledem k množství času.

Kapitola 7

Závěr

Cílem této bakalářské práce je informační systém, který deleguje autentizaci a správu identit na separátní centrální systém Keycloak, který, pokud nakonfigurován správně, je již samostatně velmi bezpečný. Jedním z hlavních cílů této práce bylo navrhnou a vytvořit informační systém s Keycloakem tak, aby informační systém bezpečně uchovával a zacházel s identitami uživatelů a správně komunikoval při výměně identit s Keycloakem. K tomu bylo potřeba poznat jednotlivé autentizační mechanismy a porozumět protokolům řešící jednotné přihlašení. Výsledkem je celkově bezpečná aplikace, která nijak neuchovává trvale identitu a pracuje s ní jen, pokud je to zapotřebí. Nicméně žádný systém není plně bezpečný.

Dalším hlavním požadavkem informačního systému, který byl ve výsledku realizován, bylo integrovat externí zdroje identit vysoce populárních organizací, ale i menších organizací, které využívají staršího protokolu SAML 2.0 nebo novějšího OpenID Connect. Tato integrace externích zdrojů identit poskytuje uživatelům pohodlnou a bezpečnou možnost přístupu k informačnímu systému pomocí svých existujících účtů u poskytovatelů identity, jako jsou Google nebo VUT. To znamená, že uživatelé nemusí vytvářet nové přihlašovací údaje a hesla, což zvyšuje pohodlí a snižuje riziko zapomenutí hesel a zároveň se snižuje implementační či administrativní zátěž spojená s vytvářením a správou uživatelských účtů v rámci samotného informačního systému.

Informační systém byl postaven na architektuře mikroslužeb, která je dnes populárním řešení při vývoji nových webových aplikací. Jelikož je tato architektura relativně nová, tak problematika autentizace a bezpečnosti v této architektuře se stále rozvíjí a je vzhledem k vlastnosti architektury někdy i velmi komplexní. S rozdělením aplikace na samostatné mikroslužby přicházejí nové výzvy ohledně správy identity, autentizace a autorizace.

Celkově se práce zabývala zajímavým tématem. Prohloubil jsem si znalosti o problematice správy identit a proč je to důležité v každém informačním systému důkladně řešit.

Literatura

[1] Bank iD. *O Bank iD* [online]. [cit. 2024-04-18]. Dostupné z: https://www.bankid.cz/o-nas.

[2] Bertocci, V. *OAuth2 and OpenID Connect: The Professional Guide.* Auth0 [cit. 2024-04-18].

[3] Cantor, S. *Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0 – Errata Composite, Working Draft 07.* 2015 [cit. 2024-04-17]. Dostupné z: https://groups.oasis-open.org/higherlogic/ws/public/document?document_id=56776.

[4] Cantor, S. *Bindings for the OASIS Security Assertion Markup Language (SAML) V2.0 – Errata Composite. Working Draft 06.* 2015 [cit. 2024-04-17]. Dostupné z: https://www.oasis-open.org/committees/download.php/56779/sstc-saml-bindings-errata-2.0-wd-06.pdf.

[5] Cantor, S. *Metadata for the OASIS Security Assertion Markup Language (SAML) V2.0 – Errata Composite. Working Draft 05.* 2015. Dostupné z: https://www.oasis-open.org/committees/download.php/56785/sstc-saml-metadata-errata-2.0-wd-05.pdf.

[6] Cantor, S. *Profiles for the OASIS Security Assertion Markup Language (SAML) V2.0 – Errata Composite. Working Draft 07.* 2015. Dostupné z: https://www.oasis-open.org/committees/download.php/56785/sstc-saml-metadata-errata-2.0-wd-05.pdf.

[7] CESNET. *Česká akademická federace identit eduID.cz* [online]. [cit. 2024-04-18]. Dostupné z: https://www.eduid.cz/.

[8] eduGain. *Identity Federations and eduGAIN* [online]. [cit. 2024-04-18]. Dostupné z: https://wiki.geant.org/display/eduGAIN/Identity+Federations+and+eduGAIN.

[9] Hardt, D. *The OAuth 2.0 Authorization Framework* [RFC 6749]. RFC Editor, říjen 2012 [cit. 2024-04-17]. DOI: 10.17487/RFC6749. Dostupné z: https://www.rfc-editor.org/info/rfc6749.

[10] Jones, M. B. *OpenID Connect Back-Channel Logout 1.0 incorporating errata set 1* [online]. 2023 [cit. 2024-04-18]. Dostupné z: https://openid.net/specs/openid-connect-backchannel-1_0.html.

[11] Jones, M. B. *OpenID Connect Front-Channel Logout 1.0* [online]. 2023 [cit. 2024-04-18]. Dostupné z: https://openid.net/specs/openid-connect-backchannel-1_0.html.

[12] Jones, M. B., Bradley, J. a Sakimura, N. *JSON Web Signature (JWS)* [RFC 7515]. RFC Editor, květen 2015 [cit. 2024-04-17]. DOI: 10.17487/RFC7515. Dostupné z: https://www.rfc-editor.org/info/rfc7515.

[13] Jones, M. B., Bradley, J. a Sakimura, N. *JSON Web Token (JWT)* [RFC 7519]. RFC Editor, květen 2015 [cit. 2024-04-17]. DOI: 10.17487/RFC7519. Dostupné z: https://www.rfc-editor.org/info/rfc7519.

[14] Jones, M. B., Bradley, J. a Sakimura, N. *OpenID Connect Core 1.0 incorporating errata set 2* [online]. 2023 [cit. 2024-04-18]. Dostupné z: https://openid.net/specs/openid-connect-core-1_0.html.

[15] Jones, M. B., Bradley, J. a Sakimura, N. *OpenID Connect Discovery 1.0 incorporating errata set 2* [online]. 2023 [cit. 2024-04-18]. Dostupné z: https://openid.net/specs/openid-connect-discovery-1_0.html.

[16] Jones, M. B., Bradley, J., Sakimura, N., Medeiros, B. de a Agarwal, N. *OpenID Connect RP-Initiated Logout 1.0* [online]. 2023 [cit. 2024-04-18]. Dostupné z: https://openid.net/specs/openid-connect-discovery-1_0.html.

[17] Michalica, D. *Autentizační rámec pro webové aplikace* [online]. 2023 [cit. 2024-04-17]. Dostupné z: https://www.vut.cz/www_base/zav_prace_soubor_verejne.php?file_id=252442.

[18] Okta. *Implications Around PSD2 and Open Banking* [online]. [cit. 2024-04-20]. Dostupné z: https://www.okta.com/resources/whitepaper/implications-around-psd2-and-open-banking/.

[19] OpenID Foundation. *How OpenID Connect Works* [online]. [cit. 2024-04-18]. Dostupné z: https://openid.net/developers/how-connect-works/.

[20] Peyrott, S. *OAuth 2 in Action*. Auth0, 2018 [cit. 2024-04-18].

[21] REFEDS. *Research and Education FEDerations* [online]. [cit. 2024-04-18]. Dostupné z: https://wiki.refeds.org/.

[22] Reschke, J. *The 'Basic' HTTP Authentication Scheme* [RFC 7617]. RFC Editor, září 2015 [cit. 2024-04-17]. DOI: 10.17487/RFC7617. Dostupné z: https://www.rfc-editor.org/info/rfc7617.

[23] Richer, J. a Sanso, A. *OAuth 2 in Action*. 1. vyd. Manning, 2017 [cit. 2024-04-18]. ISBN 9781617293276.

[24] Wikipedia. *EduGain* [online]. [cit. 2024-04-18]. Dostupné z: https://en.wikipedia.org/wiki/EduGAIN.

[25] Wikipedia. *Federated identity* [online]. [cit. 2024-04-18]. Dostupné z: https://en.wikipedia.org/wiki/Federated_identity.

[26] Wikipedia. *Identity management* [online]. [cit. 2024-04-18]. Dostupné z: https://en.wikipedia.org/wiki/Identity_management.

[27] Wikipedia. *Identity management* [online]. [cit. 2024-04-18]. Dostupné z: https://en.wikipedia.org/wiki/SAML_2.0.

[28] Wikipedia. *Otevřené bankovnictví* [online]. [cit. 2024-04-20]. Dostupné z: https://cs.wikipedia.org/wiki/Otev%C5%99en%C3%A9_bankovnictv%C3%AD.

[29] Wilson, Y. a Hingnikar, A. *Solving Identity Management in Modern Applications: Demystifying OAuth 2.0, OpenID Connect, and SAML 2.0*. Apress, 2018 [cit. 2024-04-18]. ISBN 978-1-4842-8261-8.

[30] Wirantono, M. LocalStorage vs Cookies: All You Need To Know About Storing JWT Tokens Securely in The Front-End. [online]. 2020, [cit. 2024-04-18]. Dostupné z: https://dev.to/cotter/localstorage-vs-cookies-all-you-need-to-know-about-storing-jwt-tokens-securely-in-the-front-end-15id.

[31] Česká bankovní asociace. *Czech Standard for Banking IDentity Assurance* [online]. 2020 [cit. 2024-04-20]. Dostupné z: https://cbaonline.cz/czech-standard-for-banking-identity-assurance.