Univerzita Pardubice

Fakulta Ekonomicko-správní

Aplikační rozhraní a standardy pro spolupráci částí informačních systémů

Roman Netolický

Bakalářská práce

2023

Univerzita Pardubice
Fakulta ekonomicko-správní
Akademický rok: 2022/2023

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Roman Netolický**
Osobní číslo:	**E20144**
Studijní program:	**B0688A140004 Informatika a systémové inženýrství**
Specializace:	**Informatika ve veřejné správě**
Téma práce:	**Aplikační rozhraní a standardy pro spolupráci částí informačních systémů**
Zadávající katedra:	**Ústav systémového inženýrství a informatiky**

Zásady pro vypracování

Cílem práce je vytvoření přehledu aplikačních rozhraní a standardů běžně používaných pro integraci částí informačních systémů, jejich porovnání a doporučení využití v modelovém příkladu.

Osnova:
- Zpracování přehledu aplikačních rozhraní a standardů.
- Porovnání na základě vybraných charakteristik.
- Doporučení využití pro vybraný modelový příklad.

Rozsah pracovní zprávy: **cca 35 stran**
Rozsah grafických prací:
Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

ASHBY, D., JENSEN, C.T. *APIs for dummies.* IBM, 2018.
MADDEN, N. *API Security in Action.* Shelter Island : Manning Publications Co. , 2020.
PATNI, S. *Pro RESTful APIs.* Birmingham : Packt Publishing, 2017.

Vedoucí bakalářské práce: **RNDr. Ing. Oldřich Horák, Ph.D.**
Ústav systémového inženýrství a informatiky

Datum zadání bakalářské práce: **1. září 2022**
Termín odevzdání bakalářské práce: **30. dubna 2023**

L.S.

prof. Ing. Jan Stejskal, Ph.D. v.r.
děkan

RNDr. Ing. Oldřich Horák, Ph.D. v.r.
vedoucí ústavu

V Pardubicích dne 1. září 2022

Prohlášení autora

Prohlašuji:

Práci s názvem Aplikační rozhraní a standardy pro spolupráci částí informačních systémů jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

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

Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 7/2019 Pravidla pro odevzdávání, zveřejňování a formální úpravu závěrečných prací, ve znění pozdějších dodatků, bude práce zveřejněna prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 30.4.2023 Roman Netolický

Poděkování

Rád bych touto cestou vyjádřil upřímné poděkování panu doktorovi Oldřichovi Horákovi za jeho vstřícnost, cenné rady a ochotu věnovat mi svůj čas při vypracovávání mé bakalářské práce. Jeho odborné znalosti a zkušenosti byly pro mě velkým přínosem a pomohly mi v mnoha ohledech.

Dále bych chtěl poděkovat svým kolegům v práci a přátelům za jejich podporu a povzbuzení během celého studia. Bez jejich podpory bych nedokázal překonat mnohé výzvy, které bakalářská práce přinesla.

Je mi ctí, že jsem mohl pracovat s tak skvělými lidmi a jsem jim velmi vděčný za vše, co pro mě udělali.

ANOTACE

V dnešní době se široce rozšířil trend využívání API pro integraci aplikací, které umožňují aplikacím komunikovat mezi sebou. Hlavním cílem této práce je ukázat, jak vytvořit strukturu dat, která vytváří kontext, který by bez ní neexistoval. Práce se zabývá standardy API (REST a SOAP), XML, JSON, zabezpečením API a Business Intelligence.

KLÍČOVÁ SLOVA

Aplikační rozhraní, webové služby, RESTful API, SOAP protokol, formát JSON, formát XML, ochrana API, Business Intelligence

TITLE

Application programming interfaces and standards for collaboration between parts of information systems

ANNOTATION

In today's world, there is a widely adopted trend of utilizing APIs for integrating applications, as they enable applications to communicate with each other. The main goal of this thesis is to demonstrate how to create a data structure that forms a context that would not exist without it. The thesis discusses the standards of API (REST and SOAP), XML, JSON, API security, and Business Intelligence.

KEYWORDS

Application programming interface, web services, RESTful API, SOAP protocol, JSON format, XML format, API security, Business Intelligence

Obsah

SEZNAM ZKRATEK

API	Application Programming Interface
REST	Representational State Transfer
SOAP	Simple Object Access Protocol
JSON	JavaScript Object Notation
XML	Extensible Markup Language
HTTP	Hypertext Transfer Protocol
WAAP	Web Application and API Protection
WAF	Web Application Firewall
RASP	Runtime Application Self-Protection
CSPM	Cloud Security Posture Management
Root	Root Directory
IP	Internet Protocol
BI	Business Intelligence
SQL	Structured Query Language

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

Úvod

V dnešní době je široce rozšířený trend využívat technologie na řešení různorodých otázek a problémů. Je standartní, že firmy nebo jedinci využívají technologické prostředky a aplikace pro jejich dobro. Databáze, webové služby, obchodně zaměřené programy, všechny tyto prostředky využívají dat, které je třeba si navzájem vyměnit. Zmíněné aplikace však nejsou vytvářeny způsobem, který dovoluje bezbariérovou komunikaci nehledě na příjemce a odesílatele. Pro tento důvod se u jednotlivých aplikací tvoří takzvané aplikační rozhraní. Aplikační rozhraní je součástí softwarové architektury, která umožňuje komunikaci mezi různými aplikacemi. Má definovanou sadu pravidel, protokolů a nástrojů pro vývojáře, které umožňují komunikaci různých aplikací a služeb. Pro zjednodušení této komunikace jsou vytvořené standardy formátu dat. Jedná se například o Extensible Markup Language (XML) nebo JavaScript Object Notation (JSON).

1. Standarty používané pro API

Tato kapitola se zabývá návrhy a způsoby provedení integrace různých aplikačních rozhraní. Popisuje standarty, které se využívají k přístupu k poskytovaným službám nebo zdrojům. Kapitola se nejvíce zaměří na strukturu orientovanou na zdroje z několika důvodů, které budou postupem práce vyznačeny.

1.1. REST architektura

Architektura REST (Representational State Transfer) je využívaná pro webové služby. Tato architektura byla vyvinuta ve stejný okamžik, kdy vznikl protokol HTTP, který se stal přenosovým médiem pro RESTful systémy. Hlavním znakem RESTu je určení cesty k datům. Všechny zdroje mají vlastní cestu, kudy se k nim dostat. Jelikož se využívá HTTP, k identifikaci cesty slouží http URL (Uniform Resource Identifier)[6].

1.1.1. Webový architektonický styl

Webový architektonický styl je způsob, jakým jsou navrženy a organizovány webové aplikace a služby. Každý styl má vytvořené omezení a pravidla, podle kterých musí vývojář postupovat. Pro REST se jedná o tyto omezení[6]:

1.1.1.1. Client-Server

Oddělení zájmů je hlavním tématem omezení klient-server na webu. Web je systémem založeným na klient-serveru, ve kterém mají klienti a servery odlišné role.

Mohou být implementovány a nasazovány nezávisle na sobě, používajíce jakýkoli jazyk nebo technologii, pokud splňují jednotné rozhraní webu.

1.1.1.2. Uniform Resource Interface

Interakce mezi komponentami webu - tedy jeho klienty, servery a intermediáry na bázi sítě - závisí na jednotnosti jejich rozhraní.

Komponenty webu spolupracují konzistentně uvnitř čtyř omezení jednotného rozhraní, která identifikoval Fielding:

- Identifikace zdrojů
- Manipulace zdroji prostřednictvím reprezentací
- Samopopisné zprávy
- Hypermedia jako motor aplikačního stavu (HATEOAS)

1.1.1.3. Vrstevnatý systém

Obecně platí, že intermediář založený na síti bude zachytávat komunikaci klient-server pro konkrétní účel.

Intermediáři založení na síti se běžně používají pro vynucování bezpečnosti, cachování odpovědí a vyvažování zátěže.

Omezení vrstevnatého systému umožňují intermediářům založeným na síti, jako jsou proxy a brány, aby byly transparentně nasazeny mezi klientem a serverem s využitím jednotného rozhraní webu.

1.1.1.4. Cachování

Cachování je jedním z nejdůležitějších omezení webové architektury. Omezení cachování přikazují webovému serveru deklarovat schopnost cachování každé odpovědi.

Cachování dat odpovědí může pomoci snížit vnímanou latenci klienta, zvýšit celkovou dostupnost a spolehlivost aplikace a ovládat zátěž webového serveru. Jedním slovem, cachování snižuje celkové náklady na web.

1.1.1.5. Bezstavovost

Toto omezení určuje, že webový server není povinen si pamatovat stav svých klientů. V důsledku toho musí každý klient zahrnout veškeré kontextové informace, které považuje za relevantní v každé interakci s webovým serverem.

Webové servery žádají klienty o správu složitosti komunikace jejich aplikačního stavu tak, aby webový server mohl obsloužit mnohem větší počet klientů. Toto kompromisní řešení je klíčovým přispěvatelem k škálovatelnosti architektury webu.

1.1.1.6. Code-on-Demand

Omezení umožňuje webovým serverům dočasně přenášet spustitelné programy, jako jsou skripty nebo plug-iny, na klienty.

Kód na požádání obvykle vytváří technologické svázání mezi webovými servery a jejich klienty, protože klient musí být schopen porozumět a spustit kód, který stáhne na požádání ze serveru. Z tohoto důvodu je kód na požádání jediným omezením, které se považuje za volitelné.

1.1.1.7. Hypermedia As The Engine Of Application (HATEOAS)

Toto omezení stanoví použití takzvaných Hypermédií. Jedná se o rozšíření hypertextu - textu zobrazeného na obrazovce počítače, který může odkazovat na další text nebo odkazy. Hypermédia nemusí být pouze ve formě textu, ale i grafiky, audia nebo videa.

Jedním z použití hypermedií a hypertextových odkazů je kompozice komplexních souborů informací z různých zdrojů. Informace by mohly být v privátním cloudu společnosti nebo v různých zdrojích veřejného cloudu.

1.1.2. Komunikace skrze RESTové rozhraní

Jak už bylo zmíněno v kapitole [rest architektura], hlavním způsobem komunikace je skrze protokol HTTP. Tento protokol umožňuje přenos hypertextových dokumentů na World Wide Webu. HTTP umožňuje klientovi požádat o zdroje, které se nachází na serveru. Server následně klientovi odpoví na požadavek. Komunikace probíhá pomocí textových zpráv, které jsou organizovány do hlaviček a těla. Na rozdíl od jiných architektur, REST nepotřebuje nový formát zpráv, proto se využívá metody CRUD (Create, Retrieve, Update, and Delete) [6].

- GET
  - Získání informace ze serveru. Výsledkem volání této metody by měla být odpověď obsahující tázané informace
- POST
  - Žádost na server k vytvoření nového zdroje. Obsah zdroje je specifikován v těle zprávy. Výsledkem by měla být odpověď serveru o úspěšném provedení
- PUT
  - Nahrazení již vytvořeného zdroje na serveru. Výsledkem by měla být informace o aktualizaci zdroje
- PATCH
  - Funguje na podobném principu jako metoda PUT, pouze se jedná o částečnou úpravu
- DELETE
  - Smazání již existujícího zdroje na serveru. Výsledkem by měla být informace o smazání zdroje
- HEAD
  - Identický princip jako u metody GET, kromě jedné výjimky. V odpovědi na dotaz se neodesílá tělo zprávy
- OPTIONS

- Umožňuje klientovi získat informace o nakládání se zdrojem. Může se jednat o hlavičky, konfiguraci nebo podporované metody

1.1.2.1. HTTP hlavičky

HTTP hlavičky jsou důležitou součástí HTTP požadavků a odpovědí. Tyto hlavičky přidávají doplňující informace o tom, jak by měl být zdroj zpracován službou nebo klientem. Existuje mnoho různých typů HTTP hlaviček, které jsou rozděleny do dvou kategorií - hlavičky požadavků a hlavičky odpovědí.

Hlavičky požadavků obsahují doplňující informace o požadovaném zdroji a jeho zpracování. Nejčastěji používané jsou Authorization nebo Content-Type. Hlavičky odpovědí zase obsahují informace o samotné odpovědi serveru. Jednou z nejdůležitějších je statusový kód, který vrací informaci o úspěšnosti požadavku.

Kromě standardních hlaviček je také možné nadefinovat si vlastní hlavičky, které se typicky označují prefixem X-. Tyto vlastní hlavičky mohou být užitečné pro specifické potřeby aplikace a mohou obsahovat jakékoliv informace, které jsou potřebné pro zpracování požadavku nebo odpovědi[3]. Často používané hlavičky jsou popsány v navazující tabulce.

Tabulka 1 Často využívané HTTP hlavičky

Hlavička	Popis	Příklad
Authorization	Autentizace klienta vůči serveru	Authorization: Basic YWxhZGRpbjpvcGVuc2VzYW1l
Content-Type	Formát dat v těle zprávy	Content-Type: application/json

Cache-Control	Specifikuje cachování odpovědi	Cache-Control: max-age=3600
User-Agent	Informace o klientské aplikaci nebo nástroji	User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.110 Safari/537.36
Accept	Typy dat, které klient zpracovává	Accept: application/json
Content-Length	Udává délku těla zprávy v bytech	Content-Length: 349
Origin	Specifikuje zdrojovou doménu	Origin: http://example.com
Referer	URL stránky, ze které byl odeslán požadavek	Referer: http://example.com/page.html

Zdroj: Vlastní zpracování podle [3]

1.1.3. Zpracování chyb

Ošetřování chyb je důležitou částí práce s API. Dobrý návrh chyb je důležitý pro každého účastníka v procesu integrace skrze API, jelikož dodává potřebný kontext k momentální situaci.

Když to zjednodušíme, existují v podstatě pouze 3 výsledky interakce mezi aplikací a API[6]:

- Vše fungovalo - úspěch
- Aplikace udělala něco špatně - chyba na straně klienta
- API udělalo něco špatně - chyba na straně serveru

1.1.3.1. Statusový kód

HTTP kódy mají syntax diskrétního třímístného čísla a popisu. Když se odkážeme na minulou podkapitolu, naše 3 kódy budou mít tento tvar[6]:

- 2xx – úspěch
- 4xx - chyba na straně klienta
- 5xx – chyba na straně serveru

Zde je ukázka několika různých kódů, který se často vyskytují při práci s API:

- 200 – OK
- 201 – Created
- 404 – Not Found
- 401 – Unauthorized
- 403 – Forbidden
- 500 – Internal Server Error
- 502 – Bad Gateway
- 503 – Service Unavailable

1.2. SOAP architektura

SOAP (Simple Object Access Protocol) je protokol pro výměnu zpráv mezi webovými službami. Je navržen pro poskytování rozhraní, aby se mohly aplikace naprogramované v různých programovacích jazycích komunikovat a spolupracovat na společných úlohách.

SOAP je nezávislý na protokolu přenosu, což znamená, že může být použit na různých protokolech, jako jsou HTTP, SMTP nebo FTP. Tento protokol byl standardizován organizací W3C a existují dvě verze - SOAP 1.1 a SOAP 1.2[14].

1.2.1. Formát SOAP zprávy

Zpráva SOAP je XML dokument skládající se ze tří částí – obálky, hlavičky a těla[14].

Obálka

Obálka SOAP zprávy je základní struktura, která v sobě obsahuje informace potřebné pro komunikaci mezi webovými službami.

Hlavičky

Hlavičky zprávy obsahují metadata a další doplňující informace k tělu zprávy, které jsou využívány pro správnou manipulaci s daty. Jedná se o nepovinný element.

Tělo

Tělo obsahuje vlastní data, která jsou přenášena mezi webovými službami. Nemá stanovený protokol přenosu, je ve formě XML a je povinný.

Následující ukázka představuje volání metody pomocí protokolu HTTP:

```
POST /webservicex.asmx HTTP/1.1

Host: www.webservicex.net

Content-Type: application/soap+xml; charset=utf-8

Content-Length: length

<?xml version="1.0" encoding="utf-8"?>

<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"

xmlns:xsd="http://www.w3.org/2001/XMLSchema"

xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
```

```
soap:Body

<GetWeather xmlns="http://www.webserviceX.NET">

<CityName>Prague</CityName>

<CountryName>Czech Republic</CountryName>

</GetWeather>

</soap:Body>

</soap:Envelope>
```

1.3. Porovnání architektur

Tabulka 2 Vlastnosti RESTu a SOAPu

VLASTNOST	REST	SOAP
Architektura	Client-server	Client-server, peer-to-peer
Komunikační protokol	HTTP, HTTPS	HTTP, HTTPS, SMTP, FTP
Data formát	JSON, XML, HTML, YAML, atd.	XML
Transportní vrstva	Pouze transportní vrstva	Transportní a aplikační vrstva
Bezestavovost	Bezestavový	Může být stavový
Cachování	Podporováno	Nepodporováno
Volání procedur	Nepodporováno	Podporováno

Bezpečnost	Standardní protokoly jako OAuth nebo JWT	Rozšířená bezpečnost pomocí WS-Security
Popularita	Velmi populární pro webové služby	Méně populární, používán většinou v podnikových prostředích

Zdroj: Vlastní zpracování

1.3.1. Důvody růstu popularity REST

- Jednoduchost a snadná použitelnost
  - REST API je navrženo tak, aby bylo jednoduché a snadno použitelné pro vývojáře. Na rozdíl od SOAP API není potřeba používat speciální nástroje pro generování a čtení zpráv. REST API využívá standardní HTTP metody jako GET, POST, PUT a DELETE pro manipulaci s daty.
- Menší režie
  - REST API nevyžaduje přenos složitých XML zpráv, což znamená menší režii při přenosu dat. REST API využívá formát JSON, který je kompaktní a rychle se zpracovává.
- Flexibilita
  - REST API je velmi flexibilní, což umožňuje vývojářům navrhovat vlastní URL adresy a definovat vlastní formáty zpráv. To umožňuje vývojářům navrhnout API přesně podle potřeb aplikace.
- Podpora pro různé platformy
  - REST API je nezávislé na platformě a lze ho použít s různými programovacími jazyky. To znamená, že může být používáno na široké škále zařízení a platforem.
- Rychlost

- REST API je obecně rychlejší než SOAP API, protože má menší režii a je navrženo pro použití v moderních webových aplikacích.

- Snadnější ladění
  - REST API je snadnější na ladění a testování, protože používá standardní HTTP metody, které lze snadno testovat pomocí běžných nástrojů pro testování API.

2. Formáty dat pro API

Tato kapitola se zaměřuje na základní koncepty datových formátů XML a JSON a následně je porovná.

2.1. XML

Extensible Markup Language (XML) je značkovací jazyk, který se využívá k reprezentaci strukturovaných dat. Jedná se o univerzální formát, který umožňuje přenos dat mezi aplikacemi nezávisle na platformě a programovacím jazyce[6].

2.1.1. Syntaxe

Data v XML jsou organizována pomocí značek, které popisují strukturu a význam jednotlivých elementů. Těmto značkám se taktéž říká „markup". Chovají se jako štítky, které přidávají identifikační nálepku na kus dat. Důležitým pravidlem je, že značky jsou plně zahnízděné. Zde je příklad XML kódu, který reprezentuje údaje o osobě:

```
<person>
  <name>John Doe</name>
  <age>30</age>
  <address>
    <street>Main St.</street>
    <city>New York</city>
    <zip>34346</zip>
  </address>
</person>
```

V XML mohou být tagy vybaveny atributy, což jsou dvojice jména a hodnoty, které se používají k poskytnutí dalšího popisu nebo informací o daném elementu. Atributy se vkládají do zápisu tagu jako páry "název=hodnota". Každý element může mít nula až mnoho atributů. Atributy se používají k poskytnutí dalších informací o elementu, jako například kód jazyka, formát data, velikost souboru. Zde je ukázka minulého kódu doplněný o atributy:

```
<person gender="male">
  <name>John Doe</name>
  <age>30</age>
  <address>
    <street house number="34/2">Main St.</street>
    <city part="South-West">New York</city>
    <zip>34346</zip>
  </address>
</person>
```

2.1.2. Využitelnost XML

XML je v dnešní době využívaný z několika důvodů. Umožňuje tvořit uživatelsky definované dokumenty, které lze ukládat a přenášet na Webu, tak mimo něj[6].

2.1.3. Výhody

- Čitelnost
- Kontrola chyb pomocí schémat a DTD
- Univerzálnost Unicodu
- Velká dostupnost nástrojů pro tvorbu a parsování

- Hierarchická uspořádanost

2.1.4. Nevýhody

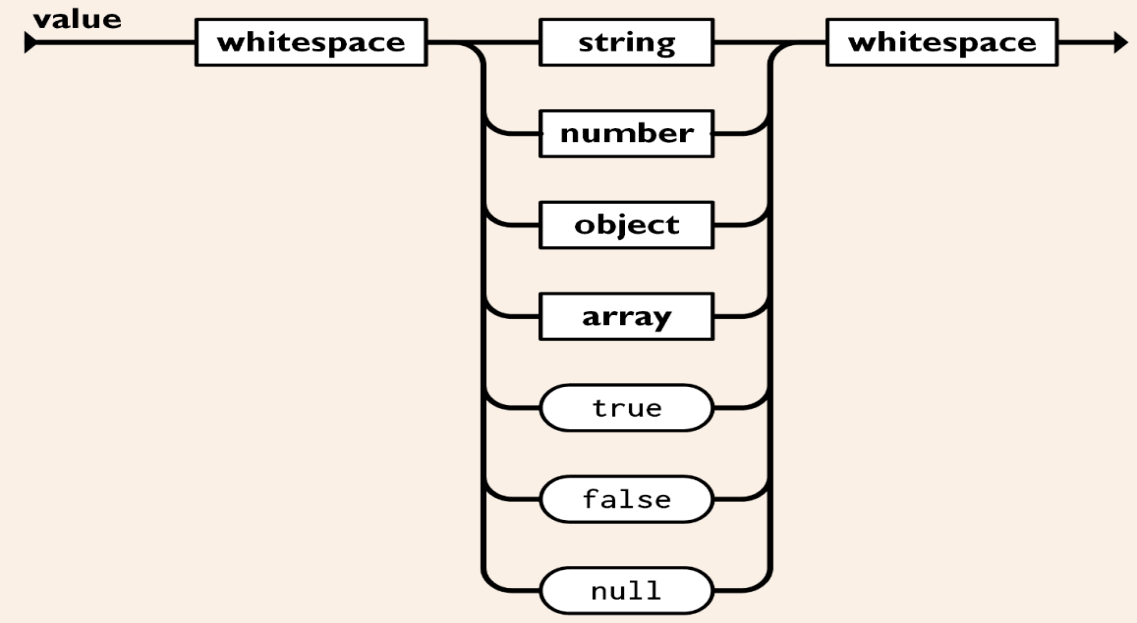
- Objemný text
- Vytváření a parsování je zátěžové na procesor klienta
- Běžně používané znaky nemusí být povolené při zpracování textu
- Obrázky a další binární data jsou složitější na kódování

2.2. JSON

JSON (JavaScript Object Notation) je lehký formát pro výměnu dat, který je snadno čitelný pro lidi i stroje. Je založen na syntaxi jazyka JavaScript, ale lze jej použít s libovolným programovacím jazykem. JSON umožňuje reprezentovat kolekce dat v hierarchické struktuře, která se skládá z objektů (sada párových klíč-hodnota) a pole hodnot[6].

2.2.1. Syntaxe

Syntaxe JSONu je velmi jednoduchá a intuitivní. Data jsou zapisována v tzv. párech "název : hodnota", oddělených čárkou a uzavřených do složených závorek {}. Názvy jsou vždy uvedeny v uvozovkách a mohou obsahovat pouze písmena, číslice a podtržítko[8].

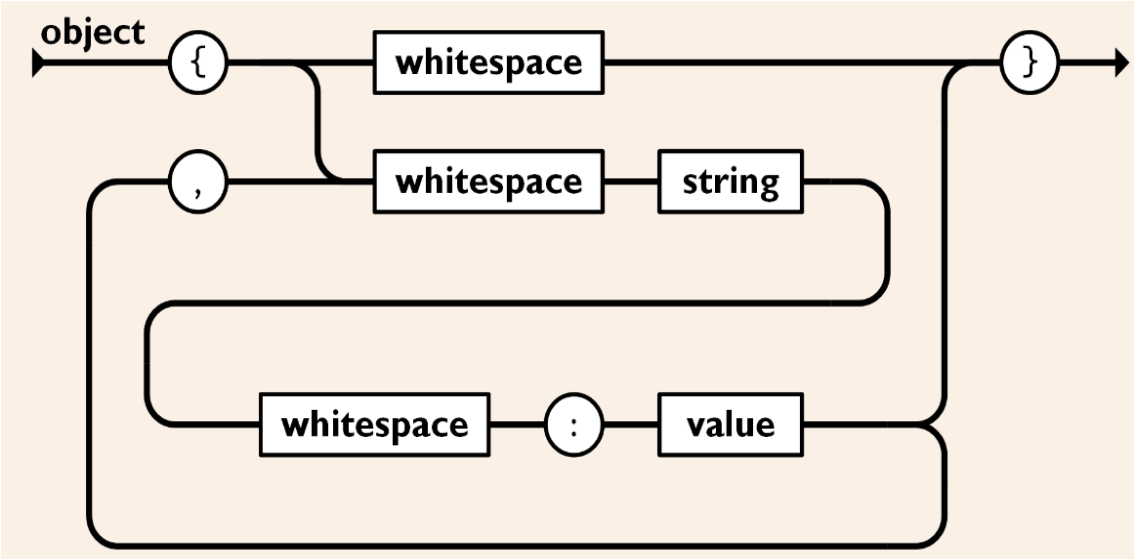


Obrázek 1: Syntaktický diagram hodnoty VALUE

Zdroj: [8]

Hodnoty můžou nabývat několika typů[8]:

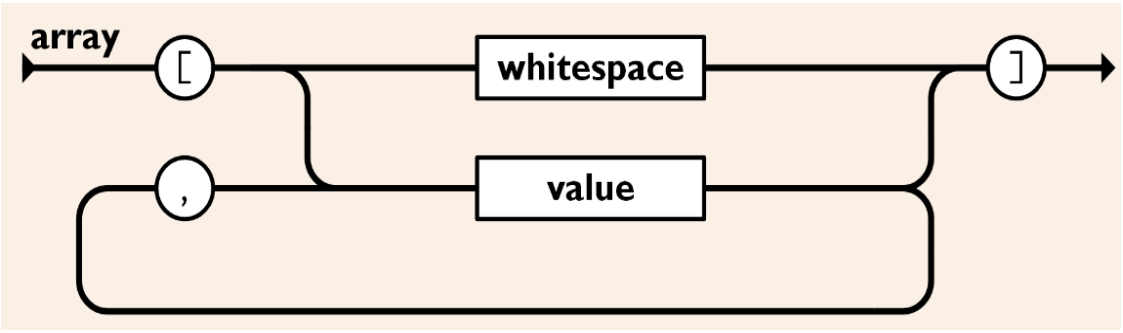
- Textový řetězec (string)
  - Může obsahovat libovolný text, včetně speciálních znaků. Je uzavřený v uvozovkách (" ")
- Číslo (number)
  - Mohou být zapsána jako celá nebo desetinná čísla. Desetinné čísla se oddělují tečkou. Záporná čísla se zapisují s mínusovým znaménkem před číslem
- Object (object)
  - Objekt je tvořen párem název-hodnota, který je uzavřen v složených závorkách { }. Názvy jsou řetězce a hodnoty mohou být libovolný datový typ, včetně jiných objektů a polí



Obrázek 2: Syntaktický diagram hodnoty OBJECT

Zdroj: [8]

- Pole (array)
  - Pole jsou kolekce hodnot uzavřené v hranatých závorkách [ ]. Prvky pole mohou být libovolný datový typ, včetně jiných polí a objektů



Obrázek 3: Syntaktický diagram hodnoty ARRAY

Zdroj: [8]

- Logické hodnoty (true / false)
  - Taktéž jinak flag nebo boolean, protikladné hodnoty
- Prázdná hodnota (null)
  - reprezentuje prázdnou hodnotu nebo chybějící hodnotu

2.2.2. Využitelnost JSON

Dalo by se říct, že JSON plní funkci XML, ale o dost lépe. V dnešní době velkou většinu programátorů čistě a jednoduše nebaví dokola parsovat. XML dokáže ty samé data reprezentovat několika způsoby, což bude klást odpor při parsování skrze aplikaci. Stále se musí procházet DOM strom nebo se musí sám postavit. JSON tyto problém naprosto přeskakuje, jelikož má čistý jednoduchý a jednotný zápis[6].

- JSON je jednodušší
- má lepší strukturu
- Je rychlejší z důvodu menší velikosti
- Na rozdíl od XML není rozšiřitelný, protože není CO rozšířit. JSON není značkovací jazyk, žádné nové tagy se nepotřebují definovat. JSON je dobrý tak jak je.
- Samozřejmě sdílí dobré vlastnosti XML
  - Je otevřený
  - Podporuje Unicode
  - Lehce se procesuje

2.2.3. Výhody

- Čitelný
- Stručný
- Rozšířený standart

2.2.4. Nevýhody

- Nelze pracovat s binárními daty
- Nelze využít komentářů
- Není možné vynutit nebo ověřovat strukturu/schéma dat
- Data musí být rozdělena na jednotlivé objekty

3. Zabezpečení API

Tato kapitola se zabývá zabezpečením API. Hlavním cílem bude bezpečnostní standard OAuth 2.0 a výčet jednotlivých možností, které lze použít pro zabezpečení aplikačního rozhraní.

3.1. OAuth 2.0

OAuth 2.0 (Open Authorization) je protokol pro autorizaci třetí strany, který umožňuje uživatelům sdílet svá data z jednoho webu nebo aplikace s jiným webem nebo aplikací bez nutnosti poskytovat své přihlašovací údaje. Tento protokol se používá v situacích, kdy chce uživatel povolit přístup k jeho datům jiné aplikaci nebo službě, ale nechce poskytovat své přihlašovací údaje. Namísto toho OAuth umožňuje aplikacím získat přístup k uživatelovým datům pomocí vytvoření autorizačního tokenu, který může být použit ke komunikaci s API poskytovatele služby a získání potřebných dat[7].

3.1.1. Role

OAuth 2.0 definuje několik rolí, které spolupracují na vytvoření autorizačního tokenu, který je dále využit pro udělení přístupu k datům pro daného uživatele. Jedná se o tyto role[7]:

- Vlastník zdroje
  - Uživatel, který má kontrolu nad chráněným zdrojem
- Zdrojový server
  - Poskytuje přístup k chráněným zdrojům
- Client
  - Mobilní, webová či jiná aplikace, která žádá přístup k chráněným zdrojům. Využívá údaje o uživateli pro jeho autentizaci
- Autorizační server

- Server, který poskytuje přístupový token pro uživatele. Token dovolí uživateli přistupovat k chráněnému zdroji

3.1.2. Token

OAuth 2.0 token je náhodně vygenerovaný textový řetězec, který umožňuje přístup k určitým informacím nebo akcím v rámci API. Tyto tokeny jsou vydávány autorizačním serverem a slouží k ověření identity uživatele a jeho oprávnění pro přístup k chráněným zdrojům.

OAuth 2.0 rozlišuje dva typy tokenů[7]:

- Přístupový token
  - Umožňuje chránit data uživatele před přístupem třetí strany
  - Nachází se v hlavičce požadavku
  - Má omezenou životnost, která je stanovena autorizačním serverem
- Obnovovací token
  - Vydá se společně s přístupovým tokenem, ale neodesílá se klientovi
  - Úlohou tohoto tokenu je obnovit přístupový token v moment co skončí jeho platnost

3.2. Nástroje pro zabezpečení

V dnešní době je bezpečnost API trochu složitější než jen využití tokenu a určitého standardu. Nestačí bránit samotné API, ale jakýkoliv způsob, který k němu útočníka dovede. Nejčastějšími útoky jsou úniky dat, ať už kvůli kompromisovanému koncovému bodu, prošlému bezpečnostnímu certifikátu nebo nedostatečné úrovni šifrování. Mezi další populární druhy útoků patří skenování zranitelností, které se dají dále zneužít nebo automatizované útoky za pomocí botů. Pro tyto účely byli vytvořené různé nástroje, které mají těmto hrozbám zabránit.

Mezi nejčastěji používané nástroje patří:

- WAAP
  - Web Application and API Protection slouží k zabezpečení webových aplikací a rozhraní API proti útokům
  - Většinou se jedná o kolekci nástrojů jako IAM (správa identit a přístupu), víceúrovňové ověřovací faktory nebo WAF
- WAF
  - Web Application Firewall je typ firewallu, který je speciálně navržen pro ochranu webových aplikací před různými druhy útoků
- API Gateway
  - Jedná se o centrální prvek, který spravuje a monitoruje požadavky na back-end služby
  - Může nahradit funkci autorizačního serveru při vydávání tokenů
  - Většinu času není využíván sám
- Skenery zranitelností
  - Nástroj, který skenuje často vyskytující se nedostatky, zranitelnosti a slabiny, které útočník může zneužít
  - Navrhuje opatření vůči těmto nedostatkům
  - V některých případech dokáže nedostatky napravit sám
- RASP
  - Runtime Application Self-Protection je technologie, která se používá k zajištění bezpečnosti aplikací v reálném čase
  - RASP umožňuje identifikovat a blokovat útoky na aplikace, zatímco aplikace běží
  - RASP dokáže detekovat a zabránit útokům bez nutnosti zásahu člověka nebo dalšího systému
- Bot management

- Bot management se zabývá identifikací, analýzou a ochranou webových aplikací proti botům = softwarové aplikace, které prochází webové stránky, sbírají z nich informace a provádějí akce, které tvoří spam nebo overload stránek
- Bot management se zaměřuje na identifikaci a blokování škodlivých botů
- Bere v úvahu legální vyhledávací boty, které slouží například k vyhledávání na webu

4. Business Intelligence nástroje

BI (Business Intelligence) nástroj je software, který umožňuje organizacím analyzovat data z různých zdrojů a získat z nich užitečné informace pro lepší rozhodování. BI nástroje poskytují uživatelům možnost vytvářet vizuální reporty a dashboardy, které zobrazují data v intuitivní podobě, což umožňuje rychlé a efektivní posuzování informací.

BI nástroje jsou důležité pro řízení firem a organizací, protože poskytují přehled o výkonnosti a výsledcích podnikání. Funkce BI nástrojů dovolí rozhodovat na základě faktů namísto odhadů. To umožňuje rychlé a efektivní rozhodování na všech úrovních organizace, což vede k lepšímu využití zdrojů a snížení rizik.

BI nástroje také mohou pomoci identifikovat trendy a vzorce v datech, což umožňuje vytvořit strategie a plány pro budoucí růst a rozvoj podnikání.

Jeden z příkladů BI nástrojů je platforma Qlik. Společnost Qlik byla založena v roce 1993 ve Švédsku. Jejím zakladatelem je Björn Berg a spoluzakladateli je Staffan Gestrelius a Peter Elfström. První verze produktu QlikView byla uvedena na trh v roce 1994 a rychle se stala jedním z nejpopulárnějších BI nástrojů. V roce 2010 se společnost Qlik Technologies Inc. stala veřejně obchodovatelnou a od té doby se stala celosvětově uznávaným lídrem v oblasti vizualizace dat a business intelligence[9].

Jejich platforma nabízí různé funkce, například[9]:

- Intuitivní uživatelské rozhraní
- Zpracování dat v reálném čase
- Kombinování dat z různých zdrojů
- Rozsáhlé možnosti vizualizace a analýzy dat

5. IMPLEMENTACE REST API

V minulých kapitolách jsme si rozebrali API z teoretického hlediska. V této kapitole si ukážeme krok za krokem integraci webových aplikací a BI systému Qlik. Pro otestování komunikace, autentizace a výsledných dotazů bude využito zdarma dostupného API klienta Insomnia od společnosti Kong.

Rozhodnutí zvolit aplikace Netbox, Skyhawk a Cynet pro moji bakalářskou práci nebylo náhodné. Mým cílem bylo vybrat nástroje, které jsou uživatelsky přívětivé, jednoduché na použití a poskytují rozsáhlou dokumentaci pro usnadnění integrace do existující infrastruktury dat, kde může být její kontext vykreslen za pomoci BI nástroje.

5.1. Insomnia

Insomnia je desktopová aplikace pro testování API, která umožňuje uživatelům odesílat HTTP požadavky na webové servery a zobrazovat odpovědi. Aplikace umožňuje uživatelům snadno vytvářet a organizovat různé projekty API, vytvářet a ukládat požadavky, konfigurovat HTTP hlavičky, nastavovat autorizaci a sledovat odpovědi. Insomnia nabízí intuitivní uživatelské rozhraní a mnoho užitečných funkcí, které pomáhají vývojářům testovat a ladit jejich API s lehkostí.

5.2. API root

API root je základní adresa API, která slouží k identifikaci a začátku interakce s API. Tato adresa je v obecném případě uvedena v dokumentaci API jednotlivých aplikací.

6. Netbox

NetBox je open-source webová aplikace pro správu IP adres a síťových zařízení. Je navržena pro organizace různých velikostí, které potřebují spravovat velké množství sítí a zařízení, a umožňuje uživatelům snadno dokumentovat, sledovat a spravovat síťové prvky a jejich atributy.

6.1. Debugging Netbox API

Uživatel začne vyplněním Api root adresy a zvolí si v předvolbě možnost GET, jelikož cílem je získat zpětnou odpověď s daty. Po vyplnění adresy lze dotaz odeslat tlačítkem Send.

Obrázek 4: Dotaz na zdrojový server

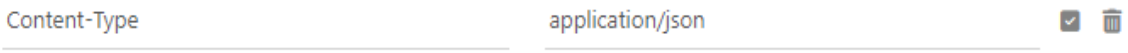
Zdroj: Vlastní zpracování

Tímto krokem pomocí výsledného http kódu 200 server potvrzuje, že spojení bylo úspěšně navázáno a požadavek byl úspěšně zpracován.

Obrázek 5: Statusový kód 200-OK

Zdroj: Vlastní zpracování

Po ověření dostupnosti serveru je třeba nastavit několik HTTP hlaviček z důvodů kompatibility a hlavně autorizace. V záložce Hlavičky se nastaví parametr content-type, který specifikuje formát, ve kterém bude server, jak odesílat, tak přijímat data. Proto je tento parametr hlavně využit pro účely metody POST. V tomto případě se jedná o formát JSON.



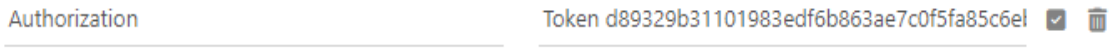
Obrázek 6: Definice formátu dat skrze HTTP hlavičku

Zdroj: Vlastní zpracování

Další věc, kterou je třeba nastavit je autorizační token. Jedná se o digitální identifikátor, který ověřuje identitu uživatele a jeho oprávnění pro přístup k API. Netbox autorizační kódy se vztahují k uživatelskému účtu. Uživatel si ověřovací kód může vygenerovat v aplikaci.

Obrázek 7: Vygenerovaný autorizační token

Zdroj: Vlastní zpracování



Obrázek 8: Definice autorizačního tokenu skrze HTTP hlavičku

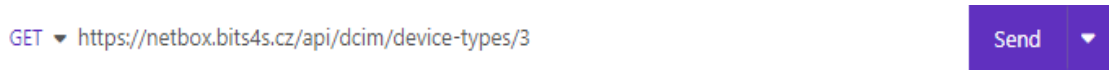
Zdroj: Vlastní zpracování

Úspěch autorizace lze ověřit jednoduchým dotazem na server. Při správné autorizaci server vrátí odpověď s požadovanými daty. Do cílové adresy přidáme Api/. Takovouto úpravou se uživatel pohybuje skrze adresáře. Příklad odpovědi serveru:


```
{
      "circuits": "https://netbox.bits4s.cz/api/circuits/",
      "dcim": "https://netbox.bits4s.cz/api/dcim/",
      "extras": "https://netbox.bits4s.cz/api/extras/",
      "ipam": "https://netbox.bits4s.cz/api/ipam/",
      "plugins": "https://netbox.bits4s.cz/api/plugins/",
```

```
    "status": "https://netbox.bits4s.cz/api/status/",

    "tenancy": "https://netbox.bits4s.cz/api/tenancy/",

    "users": "https://netbox.bits4s.cz/api/users/",

    "virtualization":
    "https://netbox.bits4s.cz/api/virtualization/",

    "wireless": "https://netbox.bits4s.cz/api/wireless/"}
```


Server odpověděl na dotaz, ale teprve nabízí různé kategorie, na které se lze dál zeptat. Přidání kategorie do kořenové adresy otevře její nabídky. Kategorie se takto budou větvit do té doby, dokud nedorazíme k cílovým datům. Příklad:



Obrázek 9: Specifikace dotazu na zdrojový server

Zdroj: Vlastní zpracování

Odpověď na specifikovaný dotaz:

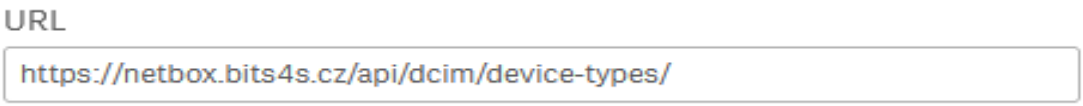

```
{

    "id": 3,

    "url":"https://netbox.bits4s.cz/api/dcim/devicetypes/3/",

    "display": "WiFi Router",

    "manufacturer": {

        "id": 7,

        "url":
"https://netbox.bits4s.cz/api/dcim/manufacturers/7/",

        "display": "ASUS",
```

```
            "name": "ASUS",
            "slug": "asus"
        },
        "model": "WiFi Router",
        "slug": "wifi-router",
        "part_number": "",
        "u_height": 0.0,
        "is_full_depth": true,
        "subdevice_role": null,
        "airflow": null,
        "front_image": null,
        "rear_image": null,
        "comments": "",
        "tags": [],
        "custom_fields": {},
        "created": "2021-06-30T00:00:00Z",
        "last_updated": "2021-06-30T14:24:29.023664Z",
        "device_count": 1
}
```


Po úspěšném debuggovém procesu tento princip lze aplikovat v cílovém BI nástroji.

6.2. Integrace v Qliku

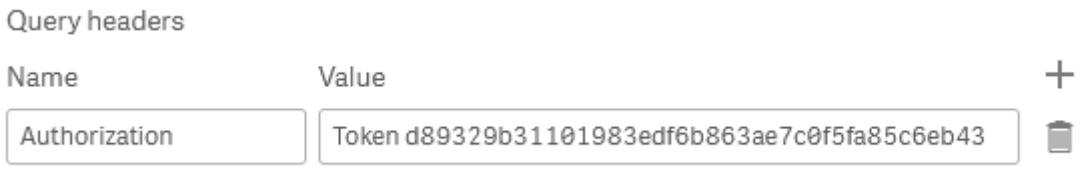
Prvním krokem je vyplnění API root adresy:



Obrázek 10: Dotaz na zdrojový server v Qliku

Zdroj: Vlastní zpracování

Dále se vyplní http hlavičky. Potřeba je vyplnit pouze parametr pro autorizaci. Parametr content-type zde není třeba, jelikož Qlik nepodporuje specifikaci formátu při použití metody GET.



Obrázek 11: Definice autorizačního tokenu skrze HTTP hlavičku

Zdroj: Vlastní zpracování

Poté lze provést ověření úspěšné komunikace:

Obrázek 12: Statusový kód 200-OK

Zdroj: Vlastní zpracování

Vytvoří se přípojka pomocí SQL, která definuje, na které API připojení se má Qlik odkázat:

```
LIB CONNECT TO 'REST_httpsnetbox.bits4s.czapidcimdevice-types3 (ec2amaz-1l6ufgi_netolickyroman)';
```

Obrázek 13: API přípojka

Zdroj: Vlastní zpracování

Je potřeba vytvořit provizorní tabulku, do které jsou dočasně uložená importovaná data. V tabule RestConnectorMasterTable se ale nachází data ve formátu JSON neboli v objektovém zápisu. Tyto data je třeba interpretovat v tabulkovém zápisu, jelikož BI nástroje využívají relačních tabulek pro ukládaní dat. K tomuto slouží příkaz: „JSON (wrap on)“. Viz.: Příloha [A].

Vícehodnotové atributy se dále rozdělí z provizorní tabulky do tabulek samostatných. Vytvoříme tabulku pro atribut manufacturer:

```
1. [manufacturer]:
2. LOAD    [id],
       [url],
       [display],
       [name],
       [slug],
       [__FK_manufacturer] AS [__KEY_root]
3. RESIDENT RestConnectorMasterTable
4. WHERE NOT IsNull([__FK_manufacturer]);
```

Vytvoříme tabulku pro atribut tags:

```
1. [tags]:
2. LOAD    [@Value],
      [__FK_tags] AS [__KEY_root]
3. RESIDENT RestConnectorMasterTable
4. WHERE NOT IsNull([__FK_tags]);
```

Vytvoříme tabulku pro atribut custom_fields:

```
1. [custom_fields]:

2. LOAD    [__FK_custom_fields] AS [__KEY_root]

3. RESIDENT RestConnectorMasterTable

4. WHERE NOT IsNull([__FK_custom_fields]);
```

K 3 předešlým tabulkám se přidá hlavní tabulka obsahující ostatní atributy. Nakonec je provizorní tabulka rozpuštěna.

```
1. [root]:

2. LOAD    [id_u0] AS [id_u0],

      [url_u0] AS [url_u0],

      [display_u0] AS [display_u0],

      [model],

      [slug_u0] AS [slug_u0],

      [part_number],

      [u_height],

      [is_full_depth],

      [subdevice_role],

      [airflow],

      [front_image],

      [rear_image],
```

```
      [comments],

      [created],

      [last_updated],

      [device_count],

      [__KEY_root]

3. RESIDENT RestConnectorMasterTable

4. WHERE NOT IsNull([__KEY_root]);

5. DROP TABLE RestConnectorMasterTable;
```

7. Skyhawk CSPM

SkyHawk Cloud Security Posture Management (CSPM) je platforma pro správu bezpečnosti v cloudovém prostředí. Poskytuje automatizovanou kontrolu konfigurace a vytváří seznamy prioritních úkolů pro řešení problémů v reálném čase.

Platforma SkyHawk CSPM je navržena pro využití v cloudech AWS, Azure a Google Cloud. Využívá umělé inteligence a strojového učení ke zlepšení výkonu a automatizace procesů, což umožňuje snížení počtu chyb a zvýšení efektivity.

7.1. Debugging Skyhawk API

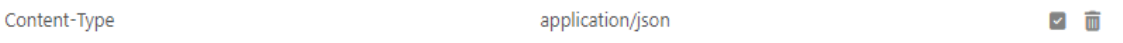
Častější varianta autentizace je formou tokenu, který odesílá server jako odpověď na dotaz. V dnešní době se nejčastěji autentizace pomocí tokenu řeší ve formátu JWT (JSON Web Token). Po odeslání uživatelským přihlašovacích údajů pomocí metody POST, server ověří jeho totožnost a v úspěšném případě vrátí v těle odpovědi token pro autentizaci.

První krok je vytvořit dotaz na ověřovací adresu:

Obrázek 14: Dotaz na autorizační server

Zdroj: Vlastní zpracování

Dále specifikovat potřebné hlavičky:



Obrázek 15: Definice formátu skrze HTTP hlavičku

Zdroj: Vlastní zpracování

Podle specifikovaného formátu dat v hlavičce se do děla dotazu zadají přihlašovací údaje uživatele:

```
{"username": "jméno","password": "heslo"}
```

Při úspěšném pokusu server navrátí v obsahu těla zprávy token:

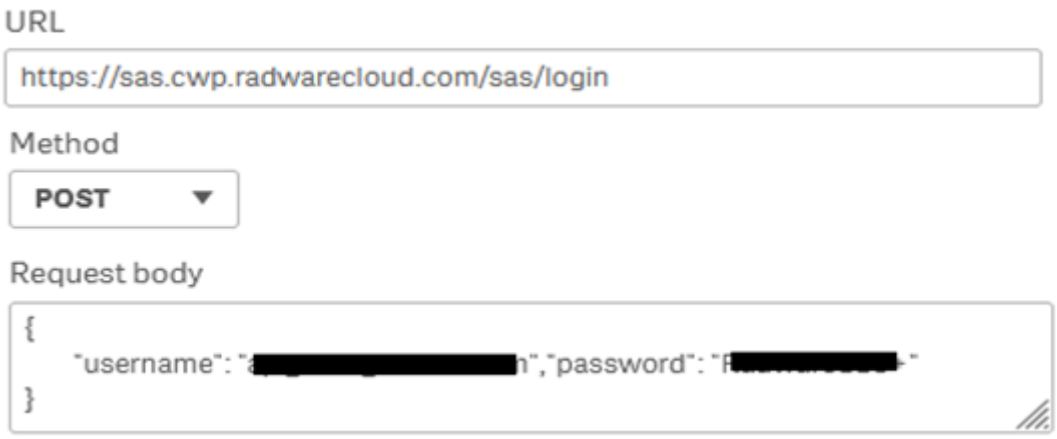
```
{"token":"eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJjdXN0b21lcklkIjoiNTE3OCIsIkN1c3RvbWVySWQiOiI1MTc4IiwidXNlck5hbWUiOiJhcGlfdXNlckBiaXRzMmIuY29tIiwiY3VzdG9tZXJfaWQiOiI1MTc4IiwiZXhwIjoxNjgwMDE1MTc1LCJ1c2VySWQiOiJfQml0czRzX18iLCJhcGlVc2VyIjp0cnVlfQ.U4L-ZKb-2cg38qQQqxNDiNP6GYqxVR5RZDbPNIvnPQc"}
```

Tyto tokeny mají omezenou dobu životnosti. Většinu času se tato doba pohybuje v rozmezí minut až několika hodin. Jedním z kvalitativních požadavků na data je jejich aktuálnost. Běžným principem je data neustále nahrávat a zálohovat na různá místa. Proto aby mohla data zůstat aktuální, musí se vytvořit cyklický proces nahrávání těchto dat. V tom ale zabraňuje životnost samotného tokenu, proto je dobré tento proces automatizovat.

7.2. Integrace v Qliku

Po úspěšném debuggovém procesu se podíváme zpět do BI nástroje Qlik.

Prvně se vytvoří API přípojka pro integraci = Specifikuje se ověřovací adresa, vybere se metoda POST a vyplní se tělo dotazu:



Obrázek 16: Dotaz na ověřovací server v Qliku

Zdroj: Vlastní zpracování

Nesmí se zapomenout na specifikaci potřebných hlaviček:

Obrázek 17: Definice formátu skrze HTTP hlavičku

Zdroj: Vlastní zpracování

Pro plnou automatizaci dotazu se vytvoří SQL dotaz pro zpracování tokenu, aby mohl být v dalších dotazech využit znovu. Stejně jako v přípojce se specifikují jednotlivé proměnné, společně s jejich hodnotou:

```
set vContentType = 'application/json';

set vURL = 'https://sas.cwp.radwarecloud.com/sas/login';

set vRequestBody = '{"username": "jméno","password":
"heslo"}';
```

Samotný dotaz pro token má tuto podobu:

```
RestConnectorMasterTable:

SQL SELECT

      "token"

FROM JSON (wrap on) "root"

WITH CONNECTION (

URL "$(vURL)",

HTTPHEADER "Content-Type" "$(vContentType)");
```

Token se nahraje do vytvořené tabulky LOGIN, kde se bude moct dále využít:

```
[LOGIN]:

LOAD  [token]

RESIDENT RestConnectorMasterTable;

DROP TABLE RestConnectorMasterTable;



LET LOGIN_ROW = NoOfRows('LOGIN');

Trace "Load INDEX Start $(LOGIN_ROW)";



For j = 0 to LOGIN_ROW -1

      Let TOKEN = peek('token', $(j), 'LOGIN');

      Trace "Load INDEX Start $(TOKEN)";

Next
```

V případě užití se token vloží do SQL jako nová proměnná, která se bude volat při tvorbě dočasné tabulky:

```
set Authorization = 'BEARER $(TOKEN)'
```

8. Cynet XDR

Cynet XDR je bezpečnostní řešení založené na detekci hrozeb, které integruje funkce EDR, NGAV a SIEM do jedné platformy. Nabízí rozsáhlé sledování chování a analýzu dat, umožňuje detekovat pokročilé hrozby a zasahuje proti nim včetně automatického řešení. Cynet XDR také poskytuje možnosti správy incidentů a sběru logů.

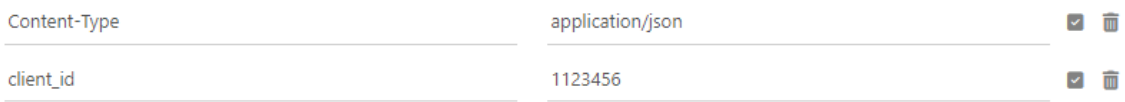
8.1. Debugging Cynet API

Dotaz na ověřovací adresu:

Obrázek 18: Dotaz na autorizační server

Zdroj: Vlastní zpracování

Další zajímavý způsob pro zabezpečení může být validační hodnota. Funguje na podobném principu jako token, pouze ho negeneruje autorizační server, ale je přidělen každému zákazníkovi, ať už manuálně nebo automaticky. V případě Cynetu se jedná o klientské ID. V případě takovýchto opatření je důležité dodržovat pokyny dokumentace, jelikož nejsou pro uživatele zřejmé.



Obrázek 19: Definice HTTP hlaviček

Zdroj: Vlastní zpracování

Do těla zprávy se vloží ověřovací údaje:

```
{"username": "jméno","password": "heslo"}
```

Autorizační server poskytne odpověď:

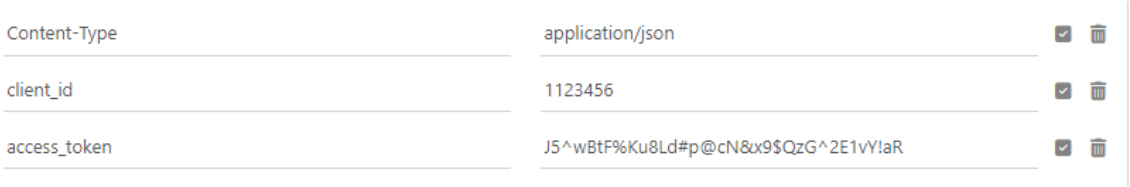
```
{"token":"TOKEN"}
```


Po získání tokenu se uživatel odkáže na API root adresu a zvolí si požadované zdroje:

Obrázek 20: Dotaz na zdrojový server

Zdroj: Vlastní zpracování

Nastaví se požadované hlavičky:



Obrázek 21: Definice HTTP hlaviček

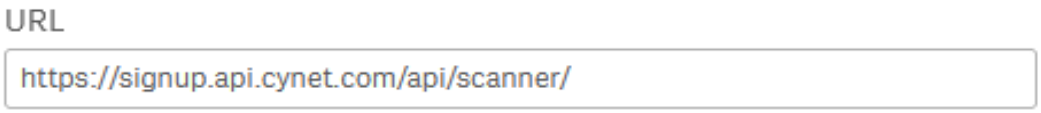
Zdroj: Vlastní zpracování

Zdrojový server odešle odpověď:


```
[{
      "DateIn": "2023-03-20T17:17:01+00:00",
      "HostName": "DESKTOP-SA3GARN",
      "ScanStart": "2023-03-20T17:17:01+00:00",
      "ScanEnd": "2023-03-21T15:06:20+00:00",
      "Group": {
            "ScanGroupName": "Manually Installed Agents",
            "DistributionType": "MSI",
            "PlatformType": "Windows"
            }}]
```


8.2. Integrace v Qliku

Uživatel vytvoří dotaz na zdrojový server:



Obrázek 22: Dotaz na zdrojový server

Zdroj: Vlastní zpracování

Specifikuje potřebné hlavičky:

Obrázek 23: Definice HTTP hlaviček

Zdroj: Vlastní zpracování

Při tvorbě přípojky a rozdělení tabulek se postupuje stejně jako v již uvedených případech. Příklad vytvořených dat:

DateIn	HostName	ScanStart	ScanEnd
2023-03-20T17:17:01+00:00	DESKTOP-SA3GARN	2023-03-20T17:17:01+00:00	2023-03-21T15:06:20+00:00
2023-03-21T12:45:26+00:00	DESKTOP-O4HITH4	2023-03-21T12:45:26+00:00	2023-04-02T10:33:58+00:00
2023-03-21T15:09:10+00:00	VIRT-DESKTOP-RIS	2023-03-21T15:09:10+00:00	2023-03-23T09:15:14+00:00

Obrázek 24: Ukázka importovaných dat

Zdroj: Vlastní zpracování

Na závěr by měl BI nástroj automaticky vytvořit relační propojení mezi daty, z čehož vznikne finální struktura dat. Viz.: Příloha [B].

Závěr

Hlavním záměrem této závěrečné práce bylo vytvořit praktický model integrace aplikací na základě myšlenky plného kontextu dat. Nejlepší příklad pro potvrzení této myšlenky se nachází v kybernetické bezpečnosti. V současnosti je v organizacích běžným stavem, že se vyskytuje více nezávislých nástrojů pro plnění potřeb v kybernetické bezpečnosti. Jedná se například o nástroje typu antivirus, log management, síťový monitoring, ipam, skener zranitelností, firewally.

Tyto nástroje jsou často vzájemně neintegrované, což sebou nese množství neduhů, např:

- Některé incidenty nejsou pro administrátora patrné, protože v žádném nástroji nepřekročí práh detekce, ale v komplexním pohledu by patrné byly
- Administrátoři přestávají některé nástroje používat, protože má každý jinou konzoli, a zvláště ve veřejné sféře jsou omezené personální prostředky, a tedy i možnosti starosti o tolik rozdílných systémů
- Z důvodu roztříštěnosti dochází k postupnému zakrnění některých nástrojů, až se dostanou do stavu naprosté ne-udržovanosti, a jsou tak zmařeny prostředky na nákup
- Při případném incidentu administrátoři nevědí, kde začít hledat a jak fungují návaznosti

Zároveň se v organizacích stále více používají nástroje pro datovou analytiku (BI nástroje). Tyto nástroje mají potenciál vyřešit či zmírnit problémy uvedené výše. Je však nezbytné, aby do nich byly bezpečnostní nástroje připojeny skrze API, a to bezpečnou konfigurací. Poté může BI nástroj sloužit jako střešní systém, aby administrátoři a manažeři řešili na denní bázi kybernetickou bezpečnost, nikoli specifika konzolí jednotlivých výrobců nástrojů pro kybernetickou bezpečnost.

Závěr této práce je, že organizace mohou optimalizovat svou práci s daty, což umožní využít data do jejich plného potenciálu. Pokud organizace budou postupovat podle této práce a zaintegrují si svá data, jejich práce bude jednodušší a otevřou se jim nové možnosti.

Reference

[1] Dennis Ashby, Claus T. Jensen. *APIs for dummies.* místo neznámé : IBM, 2018.

[2] Madden, Neil. *API Security in Action.* Shelter Island : Manning Publications Co. , 2020.

[3] Mozilla. HTTP headers. *Mozilla Developer Network.* [Online] 01. 03 2022. https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers.

[4] Swagger. The OpenAPI Specification. *Swagger.* [Online] 25. 2 2023. https://docs.swagger.io/spec.html.

[5] List of HTTP status codes. *Wikipedia: The Free Encyclopedia.* [Online] 26. 3 2023. https://en.wikipedia.org/wiki/List_of_HTTP_status_codes.

[6] Patni, Sanjay. *Pro RESTful APIs.* Birmingham : Packt Publishing, 2017.

[7] Authentication API. *Auth0 Documentation.* [Online] 28. 3 2023. https://auth0.com/docs/api/authentication.

[8] Crockford, Douglas. JSON. *json.org.* [Online] 25. 2 2023. https://www.json.org/json-en.html.

[9] Qlik Sense Cloud Services. *help.qlik.com.* [Online] 23. 3 2023. https://help.qlik.com/en-US/cloud-services/Content/Sense_Helpsites/Home.htm.

[10] Skyhawk security. *Resources.* [Online] 23. 3 2021. https://skyhawk.security/resources/.

[11] Cynet 360 AutoXDR. *Platform.* [Online] 23. 3 2023. https://www.cynet.com/platform/.

[12] Welcome to Cynet API Referenece. *API Reference.* [Online] 23. 3 2023. https://help.api.cynet.com/docs/cynet/iah0dsbmetd0e-welcome-to-cynet-api-reference.

[13] NetBox Documentation. *NetBox Documentation.* [Online] 23. 3 2023. https://docs.netbox.dev/en/stable/.

[14] SOAP Version 1.2 Part 1: Messaging Framework (Second Edition). World Wide Web Consortium (W3C) [online] 23. 3. 2023. Dostupné z: https://www.w3.org/TR/soap12/

Přílohy

[A] Provizorní tabulka – Netbox

```
RestConnectorMasterTable:
SQL SELECT
    "id" AS "id_u0",
    "url" AS "url_u0",
    "display" AS "display_u0",
    "model",
    "slug" AS "slug_u0",
    "part_number",
    "u_height",
    "is_full_depth",
    "subdevice_role",
    "airflow",
    "front_image",
    "rear_image",
    "comments",
    "created",
    "last_updated",
    "device_count",
    "__KEY_root",
    (SELECT
        "id",
        "url",
        "display",
        "name",
        "slug",
        "__FK_manufacturer"
    FROM "manufacturer" FK "__FK_manufacturer"),
    (SELECT
        "@Value",
        "__FK_tags"
    FROM "tags" FK "__FK_tags" ArrayValueAlias "@Value"),
    (SELECT
        "__FK_custom_fields"
    FROM "custom_fields" FK "__FK_custom_fields")
FROM JSON (wrap on) "root" PK "__KEY_root";
```

[B] Finální datová struktura