FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Validátor CSV souborů dle W3C doporučení CSV on the Web
Student:	Bc. Vojtěch Malý
Vedoucí:	RNDr. Jakub Klímek, Ph.D.
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce zimního semestru 2019/20

Pokyny pro vypracování

W3C doporučení CSV on the Web určuje, jak mají být pomocí JSON-LD deskriptorů popisovány CSV soubory publikované na Webu.
Cílem této práce je implementovat validátor CSV souborů založený na sadě doporučení CSV on the Web [1][2].
Přestože k doporučení existují referenční implementace [3][4], nejsou dostatečně udržovány a jsou obtížné na použití.

Postupujte v těchto krocích:
- Seznamte se s CSV on the Web [1][2][3] a JSON-LD [5]
- Prozkoumejte stávající referenční implementace [3][4]
- Navrhněte architekturu validátoru tak, aby byl snadno rozšiřitelný o další validační pravidla
- Implementujte validátor jako knihovnu v jazyce Java, která bude obsahovat rozumnou podmnožinu validačních pravidel, ideálně všechna
- Implementujte spouštěč z příkazové řádky a webovou službu
- Reprezentujte výsledky validace v RDF a CSV
- Vyhodnoťte implementovaný validátor na existující sadě testů [4] vzhledem k ostatním implementacím

Seznam odborné literatury

[1] Model for Tabular Data and Metadata on the Web, W3C, https://www.w3.org/TR/2015/REC-tabular-data-model-20151217/
[2] Metadata Vocabulary for Tabular Data, W3C, https://www.w3.org/TR/2015/REC-tabular-metadata-20151217/
[3] csvlint.io, Open Data Institute, https://github.com/theodi/csvlint.rb
[4] CSVW Implementation Report, W3C, http://w3c.github.io/csvw/publishing-snapshots/PR-earl/earl.html
[5] JSON for Linking Data, https://json-ld.org/

Ing. Michal Valenta, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 31. srpna 2018

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práce

Validátor CSV souborů dle W3C doporučení CSV on the Web

Bc. Vojtěch Malý

Katedra softwarového inženýrství
Vedoucí práce: RNDr. Jakub Klímek, Ph.D.

9. května 2019

Poděkování

Chtěl bych poděkovat panu RNDr. Jakubu Klímkovi, Ph.D. za odborné vedení práce, cenné rady a pomoc při zpracování této práce.

Prohlášení

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

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

V Praze dne 9. května 2019 .....

České vysoké učení technické v Praze
Fakulta informačních technologií

Odkaz na tuto práci

Malý, Vojtěch. *Validátor CSV souborů dle W3C doporučení CSV on the Web.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2019.

Abstrakt

Tato práce se zaměřuje na tvorbu validátoru *Comma-separated values* (CSV) souborů podle W3C doporučení *CSV on the Web*. Zmíněné doporučení definuje, jakým způsobem se mají ukládat tabulková data ve formátu CSV a jak k nim mohou být přikládány dodatečná metadata v podobě *JavaScript Object Notation for Linked Data* (JSON-LD) deskriptorů. Validátor je knihovna napsaná v programovacím jazyku Java. Součástí práce je i konzolová aplikace, webová služba a webová aplikace, které využívají implementovanou knihovnu pro spouštění validací. Práce se zabývá celým vývojovým procesem validátoru: analýzou, návrhem, implementací s ohledem na budoucí vývoj, testováním a dokumentací. Na závěr je vyvinutý validátor vyhodnocen vzhledem k již existujícím validátorům.

Klíčová slova validátor, webová služba, webová aplikace, validace CSV souborů, CSV on the Web, JSON-LD, Java

Abstract

The thesis deals with creating a validator for *Comma-separated values* (CSV) files according to the W3C recommendation *CSV on the Web*. This recommendation defines how to save tabular data in the CSV format and how to attach

additional metadata to this data in the form of *JavaScript Object Notation for Linked Data* (JSON-LD) descriptors. The validator is a Java library written in the Java programming language. A console application, a web service and a web application which use the implemented library are also part of the thesis. The thesis covers all parts of the development process: analysis, design, implementation with regards to future development, testing and documentation. Finally, the implemented validator is compared to other existing validators.

Keywords validator, web service, web application, validation of CSV files, CSV on the Web, JSON-LD, Java

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Otevřená data jsou strukturované a strojově čitelné informace, které jsou bezplatné a volně dostupné na internetu. Datové sady je dobré publikovat v jednoduchém formátu s volně dostupnou specifikací – např. ve formátu CSV. Těmto sadám by mělo být přiřazeno datové schéma, jenž popisuje syntaktickou strukturu dat. Vyjádřit schéma pro data ve formátu CSV lze pomocí jazyku *JSON Table Schema* [6] či *Metadata Vocabulary for Tabular Data* [7] z W3C doporučení *CSV on the Web*.

Je vhodné, aby si vydavatelé dat ověřovali jejich validitu – zda formát CSV splňuje specifikaci a zda je k němu přiložené schéma validní. Již je možné se setkat s validátory pro soubory CSV, které ověřují správnost formátu dat. Pro schémata z doporučení *CSV on the Web* existují tři referenční implementace validátorů, které jsou však obtížné na použití a jsou nedostatečně udržovány.

Práce se zabývá vývojem validátoru CSV souborů dle doporučení *CSV on the Web*. Validátor je vyvíjen jako knihovna, která je následně použita v konzolové aplikaci, webové službě a webové aplikaci, jež jsou také součástí práce. Především webová aplikace má umožnit skrz snadné používání jednoduchou validaci souboru, kterou využijí především vydavatelé otevřených dat, např. úřad Ministerstva vnitra.

V první kapitole se práce zabývá vysvětlením tabulkových dat a způsoby, kterými je lze ukládat do formátu CSV.

Následně jsou stručně vysvětleny základy formátu JSON-LD, neboť dle doporučení *CSV on the Web* jsou schémata pro datové sady připojována k CSV souborům v podobě JSON-LD deskriptorů.

Práce se dále zabývá doporučením *CSV on the Web* a vysvětluje pojem anotovaný model tabulkových dat a způsob, kterým tento model vznikne za pomoci CSV souborů a jejich schémat.

Další část obsahuje porovnání již existujících řešení validátorů dle *CSV on the Web* a je zde vysvětleno, proč stávající řešení nejsou dostačující a jaké jsou jejich problémy.

Navazující kapitoly popisují vývojový proces validátoru od analýzy přes návrh, implementaci, testování až k dokumentaci. Návrh je popsán s ohledem na budoucí vývoj, neboť mezi hlavní požadavky validátoru patří možnost jednoduchého přidání nových validačních pravidel.

Tuto práci uzavírá vyhodnocení zhotoveného validátoru vzhledem k již existujícím řešením, a to jak po stránce funkčnosti, tak i výkonnosti.

Kapitola 1

Úvod do tabulkových dat

Tabulková data jsou na webu běžně k vidění v různých formátech. Nejčastěji se jedná o CSV (*Comma-separated values*, česky hodnoty oddělené čárkami), TSV (*Tab-separated values*, česky tabulátorem oddělené hodnoty), HTML (*Hypertext Markup Language*) tabulky, tabulkové procesory či výpis SQL (*Structured Query Language*) databází.

Doporučení *CSV on the Web* [1] popisuje datový model pro tabulková data a také pro ně definuje metadata, která umožňují jejich validaci, zobrazení nebo převod do jiného formátu. Zmíněné doporučení mimo jiné poskytuje tzv. osvědčené postupy (*best practices*), jak publikovat tabulková data pomocí CSV.

Tato kapitola se zabývá tím, co přesně jsou tabulková data, jak je lze uložit ve formátu CSV dle doporučení *CSV on the Web* a čím se odlišuje formát CSV daný tímto doporučením od formátu CSV dle RFC 4180 [4].

1.1 Definice tabulkových dat

Tabulková data (*tabular data*) jsou řádkově strukturovaná data, která by měla dodržovat relační model. Soubor s tabulkovými daty obsahuje tabulku nebo-li relaci v relačním modelu. Tabulka má nějaké vlastnosti tzv. atributy. Každý jeden atribut je tvořen jedním sloupcem tabulky. Atributy mají definovaný určitý typ a množinu hodnot, kterých mohou nabývat – tzv. doménu. Relaci R nad n atributy poté definujeme jako podmnožinu kartézského součinu domén jednotlivých atributů. Řádky tabulky pak tvoří prvky této relace, neboli jsou to již konkrétní n-tice s vyplněnými hodnotami. [8]

Tabulková data se na webu běžně vyskytují ve formátu **CSV**, pomocí kterého lze relaci jednoduše zapsat. Řádky CSV souboru tvoří prvky relace a jednotlivé hodnoty atributů (sloupce) jsou odděleny oddělovačem (čárkou). Pokud je však potřeba popsat relační model přesněji a definovat např. domény atributů nebo integritní omezení, tak formát CSV již nestačí. Z tohoto důvodu

zavádí doporučení *CSV on the Web* metadata pro tabulková data, která tyto informace obsahují, viz Kapitola 3.3.

1.2 Druhy CSV

Na web se vyskytuje mnoho CSV souborů, a jejich formáty jsou různorodé. Některé z nich mají jako oddělovače čárku, jiné středník nebo tabulátor. Problémem je, že lidé a dokonce i počítačové programy nedodržují definované CSV standardy. Nejrozšířenějším standardem pro formát CSV je RFC 4180 [4]. Doporučení *CSV on the Web* poskytuje tzv. osvědčené postupy (*best practices*), jak publikovat tabulková data pomocí CSV. [1] V této kapitole jsou oba tyto formáty popsány a porovnány.

1.2.1 CSV dle RFC 4180

CSV formát se používá již dlouho, ale jeho první formální popis přišel až s dokumentem RFC 4180. Podle tohoto dokumentu by každý řádek CSV souboru měl obsahovat stejný počet hodnot oddělených čárkou (znak `,`). Tyto řádky jsou od sebe odděleny koncem řádku – znaky `CRLF`. Poslední řádek může, ale nemusí končit znakem koncem řádku. [4]

Hodnoty obsahující čárku nebo konec řádku musí být uzavřeny v uvozovkách (znak `"`). Pokud hodnota sama obsahuje uvozovku, měla by být zdvojena, tj. místo `"` bude zapsána jako `""`. [4]

RFC 4180 zaregistrovalo **typ internetového média** nebo-li ***media-type*** pro CSV soubor jako `text/csv`. [4] Pokud je CSV soubor přenášen přes HTTP (*Hypertext Transfer Protocol*), pak by HTTP odpověď měla obsahovat HTTP hlavičku `Content-type` s hodnotou `text/csv`.

První řádek CSV souboru se nazývá **hlavička** a měla by obsahovat názvy sloupců oddělené čárkou. Na tyto názvy se nevztahuje žádné omezení. Pokud CSV soubor neobsahuje hlavičku, mělo by to být specifikováno v typu internetového média v parametru `header`. Podporované hodnoty `header` parametru jsou `present` a `absent`. [4]

CSV soubory by měly být kódovány pomocí **US-ASCII**. Pokud soubor není kódován v US-ASCII, mělo by být kódování uvedeno v parametru `charset` v HTTP hlavičce `Content-type` [4] viz:

```
Content-Type: text/csv;charset=ISO-8859-1
```

Gramatika se syntaxí CSV souboru je zapsána v ABNF [9] (*Augmented Backus–Naur Form*, česky Rozšířená Backusova–Naurova forma) [4]:

```
file = [header CRLF] record *(CRLF record) [CRLF]
header = name *(COMMA name)
record = field *(COMMA field)
```

```
name = field
field = (escaped / non-escaped)
escaped = DQUOTE *(TEXTDATA / COMMA / CR / LF / 2DQUOTE) DQUOTE
non-escaped = *TEXTDATA
COMMA = %x2C
CR = %x0D
DQUOTE =  %x22
LF = %x0A
CRLF = CR LF
TEXTDATA =  %x20-21 / %x23-2B / %x2D-7E
```

1.2.2 CSV dle CSV on the Web

Autoři *CSV on the Web* vydali sadu omezení pro CSV soubory. [1] Řádky CSV souboru opět musí obsahovat stejný počet hodnot oddělených čárkou (znak `,`). Z minulé kapitoly víme, že RFC 4180 podporuje jako konce řádků jen znaky `CRLF`. Podle *CSV on the Web* lze použít pro konce řádků nejen znaky `CRLF`, ale i samotný znak `LF`. Poslední řádek je ukončen koncem řádku.

Hodnoty obsahující čárku nebo konce řádků musí být stále uzavřeny v uvozovkách (znak `"`). Pokud hodnota sama obsahuje uvozovku, měla by být zdvojena. Před a za uvozovkami by neměly být bílé znaky (anglicky *white spaces*). [1]

Za typ internetového média se využívá již registrovaný `text/csv` a tato hodnota by měla být při přenosu souboru přes HTTP obsažena v hlavičce `Content-type`. [1]

První řádek (hlavička) by měl obsahovat názvy sloupců oddělené čárkou. Pokud CSV soubor hlavičku neobsahuje, tak je zapotřebí tuto skutečnost specifikovat v parametru `header` v typu internetového média. [1]

CSV soubory by měly být kódovány pomocí **UTF-8** (*Unicode Transformation Format*) a měly by být v normálové formě C (*Unicode Normal Form C*) definované v Unicode technickém standardu [10]. Pokud soubor není v UTF-8, mělo by být kódování uvedeno v parametru `charset` v HTTP hlavičce `Content-type`. [1]

Gramatika syntaxe je zapsána v EBNF (*Extended Backus–Naur Form*, česky Rozvinutá Backusova–Naurova forma) dle XML 1.0 [11] a je pouhým zobecněním RFC 4180 gramatiky [1]:

```
csv  ::=  header record+
header  ::=  record
record  ::=  fields #x0D? #x0A
fields  ::=  field ("," fields)*
field  ::=  WS* rawfield WS*
rawfield  ::=  '"' QCHAR* '"' |SCHAR*
QCHAR  ::=  [^"] |'""'
```

```
SCHAR  ::=  [^",#x0A#x0D]
WS  ::=  [#x20#x09]
```

1.2.3 Srovnání CSV

Srovnání dvou standardů CSV souborů RFC 4180 a *CSV on the Web* obsahuje Tabulka 1.1. Oba dva porovnávané formáty jsou si velmi podobné, což je viditelné již z důvodu, že CSV soubory dle RFC 4180 splňují formát CSV dle *CSV on the Web*.

Oba formáty používají jako oddělovač čárku. Jestliže samotné hodnoty obsahují čárku, je nezbytné, aby tato hodnota byla uvedena v uvozovkách. A pokud hodnota obsahuje uvozovku, tak musí být uvozovka zdvojena.

Největším rozdílem těchto dvou formátu je podpora znaků pro konce řádků. Oba formáty definují jako konec řádku znaky `CRLF`. Nicméně *CSV on the Web* podporuje jako konec řádku i samotný znak `LF`. Dalším rozdílem je fakt, že dle RFC 4180 může, ale i nemusí poslední řádek tabulky končit znaky pro konec řádku. Zatímco dle *CSV on the Web* těmito znaky musí končit i poslední řádek.

Oba formáty používají `text/csv` jako typ internetového média včetně volitelných parametrů `header` a `charset`. Parametr `header` udává, zda tabulka obsahuje hlavičku, a parametrem `charset` určuje kódování souboru. Základní kódování formátu se liší, neboť RFC 4180 považuje jako defaultní kódování US-ASCII, ale *CSV on the Web* UTF-8.

	RFC 4180	CSV on the Web
Oddělovač hodnot	čárka (`,`)	čárka (`,`)
Uvozovací znak	uvozovka (`"`)	uvozovka (`"`)
Escapovací znak	uvozovka (`"`)	uvozovka (`"`)
Konce řádků	`CRLF`	`LF`, `CRLF`
Typ internetového média	`text/csv`	`text/csv`
Hlavička	Nepovinná	Nepovinná
Základní kódování	US-ASCII	UTF-8
Konec řádku v posledním řádku	Nepovinný	Povinný

Tabulka 1.1: Porovnání formátu CSV mezi definicí dle RFC 4180 [4] a dle *CSV on the Web* [1]

Kapitola 2

Formát JSON-LD

Metadata, definovaná dle *CSV on the Web*, jsou uložena ve formátu JSON-LD (*JavaScript Object Notation for Linked Data*, česky JavaScriptový objektový zápis pro Linked Data) [12]. V této kapitole je popsaná základní myšlenka formátu JSON-LD, tak aby bylo možné pochopit práci s metadaty. Ve druhé části jsou definovány omezení formátu JSON-LD, která se na tyto metadata vztahují.

2.1 Definice IRI

Základem formátu JSON-LD jsou IRI (*Internationalized Resource Identifier*, česky internacionalizovaný identifikátor zdroje) [13] a URI (*Uniform Resource Identifier*, česky jednotný identifikátor zdroje) [14], tudíž je zapotřebí tyto dva pojmy definovat a od sebe odlišit.

URI je sekvence znaků, se kterou lze identifikovat zdroje. Do množiny znaků tvořící URI spadají ASCII znaky (*American Standard Code for Information Interchange*) ale i nějaké další znaky, mezi které např. patří `:`, `/`, aj. Ostatní znaky, jež nespadají do této množiny, je nutné zakódovat pomocí URL kódování (*Percent-encoding*).

IRI je rozšíření URI o univerzální kódovanou znakovou sadu (anglicky *Universal Coded Character Set*, UCS), která by měla zahrnovat všechny doposud známe znaky. V JSON-LD formátu se s IRI můžeme setkat ve třech různých tvarech – absolutní IRI, relativní IRI nebo kompaktní IRI.

Definice 1 ***Absolutní IRI*** *je IRI obsahující celou cestu i schéma např.* `http://example.com/`*. [13]*

Definice 2 ***Relativní IRI*** *je relativní vzhledem k absolutní IRI. Nechť existuje dokument s absolutní IRI* `http://example.com/about/`*, poté* `../` *lze nazvat relativní IRI, která se přeloží vzhledem k absolutní IRI na* `http://example.com/`*. [13]*

Definice 3 ***Kompaktní IRI*** *je způsob vyjádření IRI za pomocí předpony a přípony oddělenými dvojtečkou. Předponou je výraz, který je na určitém místě definován jako přezdívka pro absolutní IRI. Příkladem může být výraz* `foaf`*, jež definujeme jako zkratku pro Friend-of-a-Friend slovník s IRI* `http://xmlns.com/foaf/0.1/`*. Přípona pak dodefinovává přesnou adresu. Kompaktní IRI foaf:name se tedy přeloží jako* `http://xmlns.com/foaf/0.1/name`*. [12]*

2.2 Princip JSON-LD

Formát JSON-LD je jeden ze způsobů serializace RDF (*Resource Description Framework*, česky systém popisu zdrojů) a vychází z velmi rozšířeného formátu JSON (*JavaScript Object Notation*, česky JavaScriptový objektový zápis) [15].

2.2.1 Základní koncept

JSON-LD dokument je validní JSON dokument dle RFC 4627 [15]. Syntaxe formátu JSON není v práci popsaná, jelikož se jedná o velmi rozšířený formát na webu. Důležité je jen zmínit základní JSON kostrukty, mezi které patří JSON objekt, pole, textový řetězec, číslo, true nebo false a null. JSON dokument se skládá z vlastností (tzv. *properties*), což jsou páry klíč-hodnota např. `"year": 1995`.

Klasický JSON má jednu velkou nevýhodu pro použití na webu – nepodporuje hyperlinky, které jsou základním stavebním kamenem webu. Příklad 2.1 obsahuje ukázku jednoduchého JSON dokumentu.

Příklad 2.1: Jednoduchý JSON dokument

```
{
  "name": "Vojtech Maly",
  "homepage": "http://vojtech.maly.cz/",
  "image": "http://vojtech.maly.cz/obrazky/vojta.png"
}
```

Člověk může – někdy ale poněkud obtížně – pomocí své intuice zjistit, že data z Příkladu 2.1 popisují osobu se jménem Vojtech Maly a že vlastnost *homepage* obsahuje jeho domovskou stránku. Stroje však onu lidskou schopnost nemají. Použitím jednoznačných identifikátorů místo různých výrazů, jako je *name* či *homepage*, tento problém vyřešíme.

Propojená data – a web celkově – používají IRI pro jednoznačnou identifikaci. Idea je taková, že pomocí IRI jednoznačně identifikujeme určitá data, která mohou být použita ostatními vývojáři, což je užitečné pro výrazy jako *name* či *homepage*. Pokud se tyto výrazy přeloží na IRI, je každému vývojáři jasné, co tímto výrazem popisujeme. Zamezíme tím víceznačnému chápání.

Navíc stroje nebo lidé mohou tuto IRI použít (např. otevřením ve webovém prohlížeči), jít tak na definici daného výrazu a zjistit, co přesně znamená. Tento proces se nazýva dereference IRI. [12]

Použitím slovníku *schema.org* [16] lze Příklad 2.1 přepsat jako:

Příklad 2.2: Jednoduchý JSON dokument používající IRI místo výrazů

```
{
  "http://schema.org/name": "Vojtech Maly",
  "http://schema.org/url": {
    "@id": "http://vojtech.maly.cz/"
  },
  "http://schema.org/image": {
    "@id": "http://vojtech.maly.cz/obrazky/vojta.png"
  }
}
```

V Příkladu 2.2 jsou všechny vlastnosti jednoznačně definovány pomocí IRI. Všechny hodnoty, které reprezentují IRI, jsou označeny klíčovým slovem `@id`. Tento JSON dokument je již JSON-LD dokument, ovšem velmi specifický. Vzhledem k tomu, že by se s ním uživatelům pravděpodobně špatně pracovalo, zavádí JSON-LD tzv. **kontext**, který je vysvětlen v následující Kapitole 2.2.2.

2.2.2 Kontext

Komunikují-li spolu dva lidé, je jejich konverzace zasazená v určitém prostředí, tzv. „kontextu konverzace“. Tento společný kontext umožňuje účastníkům konverzace používat různé výrazy, jako např. přezdívky kamarádů, což vede k rychlé komunikaci bez ztráty přesnosti. Stejným způsobem se chová **kontext** v JSON-LD. [12]

Jednoduše řečeno **kontext** slouží k překladu krátkých výrazů na IRI. Vezmeme-li jednoduchý dokument z Příkladu 2.2, vypadal by kontext jako v Příkladu 2.3.

V kontextu z Příkladu 2.3 jsou vidět tři definice výrazu – *name*, *image* a *homepage*. **Výraz** je tedy krátký řetězec, který se rozšíří na IRI, přičemž definice tohoto rozšíření je definována v kontextu. Výraz nesmí být roven některému z klíčových slov, nesmí být prázdným řetězcem a také nesmí začínat znakem `@`. [12]

Z příkladu lze vidět, že hodnotou **definice výrazu** může být buď řetězec, anebo JSON objekt. Je-li hodnotou řetězec, pak se výraz překládá na IRI, např. vlastnost *name* má jako hodnotu řetězec `http://schema.org/name`, což znamená, že *name* je zkratka pro IRI `http://schema.org/name`.

Pokud je hodnotou JSON objekt, nazývá se **rozšířenou definicí výrazu**. Vezmeme-li z příkladu rozšířenou definici výrazu *image*, poté vlastností *@id*

s hodnotou `http://schema.org/image` vyjadřujeme, že *image* je zkratka pro IRI `http://schema.org/image`, a vlastností `@type` s hodnotou `@id` říkáme, že řetězcové hodnoty pro vlastnost `image` budeme prezentovat jako IRI. [12]

Kontext může být v dokumentu definován přímo, jako v Příkladu 2.3, nebo se lze na něj odkázat pomocí IRI.

Příklad 2.3: Kontext pro JSON dokument z Příkladu 2.2

```
{
  "@context":
  {
    "name": "http://schema.org/name",
    "image": {
      "@id": "http://schema.org/image",
      "@type": "@id"
    },
    "homepage": {
      "@id": "http://schema.org/url",
      "@type": "@id"
    }
  }
}
```

2.3 JSON-LD pro metadata

Metadatový slovník pro tabulková data používá formát založený na JSON-LD s těmito omezeními [7]:

- všechny výrazy použité v metadatech musejí být definovány v *CSVW Namespace Vocabulary Terms* [17];
- hodnotou jakékoliv vlastnosti `@id` nebo `@type` nesmí být blank node – speciální typ viz definice v JSON-LD specifikaci [12];
- v dokumentu se nesmí přídávat nový kontext nebo upravovat hlavní kontext jinak, než je povoleno v Kapitole 3.3.3;
- hodnota vlastnosti `@type` musí být výraz, kompaktní IRI s výrazem v předponě nebo absolutní IRI;
- hodnoty nesmí být listem (`@list`) nebo množinou (`@set`);
- objekty nesmí obsahovat klíčová slova `@graph` a `@context`, výrazy a blank node.

Kapitola 3

CSV on the Web

Pro vytvoření validátoru je nutné definovat anotovaný model tabulkových dat z článku *Model for Tabular Data and Metadata on the Web* (česky model pro tabulková data a metadata na webu) [1]. Tento model vzniká na základě tabulkových dat a dodatečných metadat, která mají formát a strukturu popsanou v článku *Metadata Vocabulary for Tabular Data* (česky metadatový slovník pro tabulková data) [7] a jsou uložena v podobě JSON-LD deskriptorů.

Článek *Model for Tabular Data and Metadata on the Web* i článek *Metadata Vocabulary for Tabular Data* jsou součástí doporučení *CSV on the Web*, které bylo vydáno v prosinci roku 2015 skupinou W3C (*World Wide Web Consortium*) s cílem poskytnout technologie pro aplikace na Webu, jež využívají jako zdroje dat CSV soubory či podobné formáty.

Anotovaný model tabulkových dat je definovaný v Podkapitole 3.2. Formát i struktura metadatových dokumentů (JSON-LD deskriptorů) a jejich využití při tvorbě anotovaného modelu tabulkových dat je dále popsáno v Podkapitola 3.3 [1]

3.1 Důvod vzniku doporučení

V Kapitole 1.1 jsou popsána tabulková data, která lze dobře ukládát ve formátu CSV. Problémem formátu CSV je skutečnost, že v něm společně s tabulkovými daty již nelze uložit žádné dodatečné informace. Mezi tyto dodatečné informace patří např. typy sloupců (atributů) nebo integritní omezení tabulky – primární nebo cizí klíče.

K validaci, konverzi, zobrazení nebo hledání tabulkových dat na webu jsou však tyto dodatečné informace nezbytné. A právě *CSV on the Web* definuje způsob, jak tyto dodatečná metadata k CSV připojit pomocí JSON-LD deskriptoru používající slovník *Metadata Vocabulary for Tabular Data* [7]. Zmíněná metadata mohou popisovat vše, od několika tabulek a jejich vzájemných vztahů až po popis jednotlivých buněk v tabulce. [7]

Příklad 3.1 obsahuje tabulková data ve formátu CSV:

Příklad 3.1: Tabulková data ve formátu CSV dostupná na http://example.org/tree-ops.csv [7]

```
GID,On Street,Species,Inventory Date
1,ADDISON AV,Celtis australis,10/18/2010
2,EMERSON ST,Liquidambar styraciflua,6/2/2010
```

Člověk by tato data mohl považovat za dobře čitelná, dokonce by mohl zjistit typy jednotlivých sloupců, a to zejména v případě, že by k datům dostal i jejich dokumentaci. Pro stroje by zjištění významu těchto dat bylo velmi složité, až téměř nemožné, proto se dle standardu *CSV on the Web* k tabulkovým datům přikládají strojově čitelná metadata ve formátu JSON-LD – jako na Příkladu 3.2 – která umožňují interpretaci tabulkových dat strojům a přidávají dodatečné informace o datech. [7]

Metadatové soubory mohou být využity různými aplikacemi. Jedná se především o [7]:

- **Prohlížeče**, jež umožňují lépe čitelný a jednodušší pohled na tabulková data. Mohou zobrazovat různé tabulky, grafy, vztahy mezi nimi ad.
- **Nástroje pro vkládání dat** používající metadata k tomu, aby zajistily vkládání správných dat ve správném formátu do souboru s tabulkovými daty.
- **Validátory**, které porovnávají metadatový soubor se souborem s tabulkovými daty a kontrolují, zda se názvy sloupců v obou souborech navzájem rovnají, zda hodnoty ve sloupcích mají správný datový typ či formát, a mnoho dalšího.
- **Konvertory** využívající metadata k převodu tabulkových dat (např. CSV) do jiných formátů, např. JSON, RDF nebo XML (*eXtensible Markup Language*).

CSV on the Web také popisuje způsob práce s tabulkovými daty, tedy jak je správně parsovat, kde k ním najít metadatové soubory, aj. Viz Podkapitola 3.4.

Příklad 3.2: JSON-LD deskriptor obsahující strojově čitelná metadata pro tabulková data z Příkladu 3.1 [7]

```
{
  "@context": ["http://www.w3.org/ns/csvw", {
     "@language": "en"}],
  "url": "tree-ops.csv",
  "dc:title": "Tree Operations",
  "dcat:keyword": ["tree", "street", "maintenance"],
  "dc:publisher": {
    "schema:name": "Example Municipality",
    "schema:url": {"@id": "http://example.org"}
  },
  "dc:modified": {"@value": "2010-12-31", "@type":
     "xsd:date"},
  "tableSchema": {
    "columns": [{
      "name": "GID",
      "titles": ["GID", "Generic Identifier"],
      "datatype": "string",
       "required": true
    }, {
      "name": "on_street",
       "titles": "On Street",
       "datatype": "string"
    }, {
      "name": "species",
      "titles": "Species",
      "dc:description": "The species of the tree.",
      "datatype": "string"
    }, {
      "name": "inventory_date",
      "titles": "Inventory Date",
      "dc:description": "The date of the operation
         that was performed.",
      "datatype": {"base": "date", "format":
         "M/d/yyyy"}
    }],
  "primaryKey": "GID",
  }
}
```

3.2 Model tabulkových dat

Anotovaný model tabulkových dat (tzv. ***annotated tabular data model***) je dle doporučení *CSV on the Web* modelem pro tabulky, které jsou anotované, respektive obsahují nějaká metadata. Model se skládá z několika komponent (např. tabulka, řádek, sloupec) viz Podkapitola 3.2.1. Každá z komponent může obsahovat **anotace**, jež poskytují informace o komponentě, ke které jsou přiřazeny. Hodnoty těchto anotací mohou být jednoduché (řetězce, čísla), strukturované (objekty) anebo se může jednat o seznam těchto hodnot.

Anotace rozdělujeme na **základní** (tzv. *core annotations*), mezi které patří např. názvy sloupců, hodnoty buněk, a na **doplňující**, jež jsou definovány společnými vlastnostmi z Kapitoly 3.3.3. [1]

3.2.1 Komponenty anotovaného modelu

Na Obrázku 3.1 je zobrazen anotovaný model tabulkových dat zachycený v doménovém modelu. Třídy doménového modelu představují komponenty anotovaného modelu a atributy těchto tříd jsou jejich základními anotacemi.

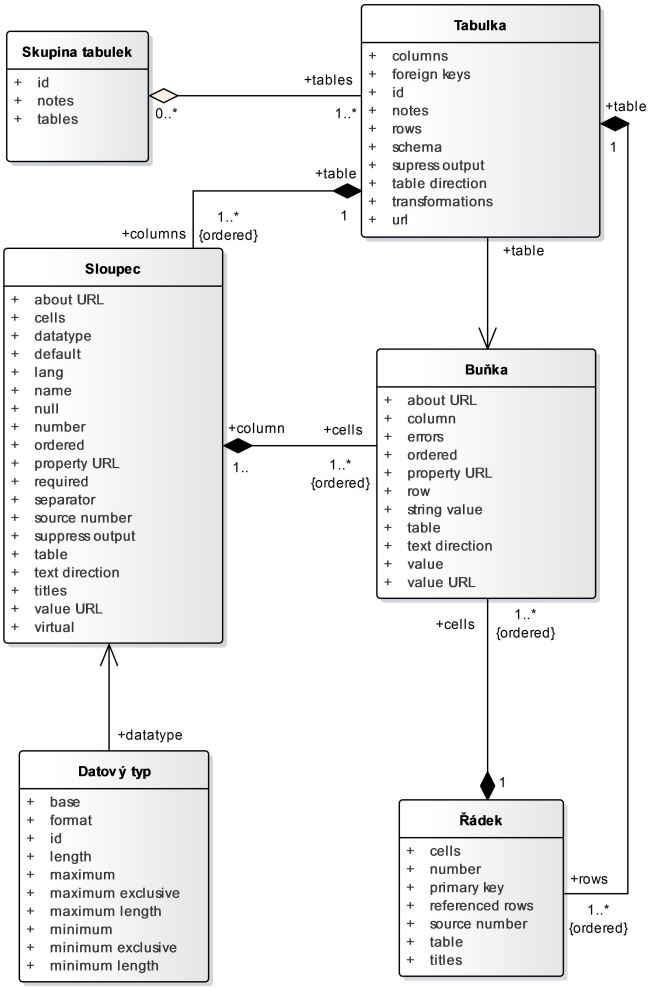
V práci jsem základním anotacím zanechal původní anglické názvy, neboť je jejich popis velmi rozsáhlý. Více se o anotacích dozvíte v Kapitole 4 článku *Model for Tabular Data and Metadata on the Web* [1]. Komponenty anotovaného modelu mohou mít libovolný počet doplňujících anotací. Z tohoto důvodu nejsou v doménovém modelu zobrazeny.

Základní komponentou je tabulka, která obsahuje dvě důležité anotace – sloupce (*columns*) a řádky (*rows*). Tabulka musí obsahovat alespoň jeden sloupec a jeden řádek. Pořadí sloupců ale i řádků je důležité a musí být dodrženo. Kromě těchto dvou anotací tabulka obsahuje i další jako např. definice cizích klíčů (*foreign keys*), URL zdroje tabulkových dat této tabulky (*URL*) ad.

Hlavní anotací sloupce jsou buňky (*cells*), které k danému sloupci patří. Pro každý řádek tabulky musí mít sloupec jednu buňku, z čehož vyplývá nutnost zachovávat pořadí buněk. Důležitou anotací je také datový typ (*datatype*), který definuje očekávaný typ hodnot buněk ve sloupci. Mezi další anotace sloupce patří např. *name*, jež definuje jméno sloupce, nebo *required*, která určuje, zda buňky ve sloupci mohou být prázdné.

Řádky stejně jako sloupce obsahují anotaci buňky (*cells*). Řádek obsahuje buňku pro každý ze sloupců, a tedy musí buňkám opět zachovávat stejné pořadí. Další důležitou anotací je primární klíč (*primary key*), jež obsahuje množinu buněk, které dohromady v daném řádku tvoří primární klíč.

Buňka reprezentuje pole tabulky, které vzniklo protnutím řádku a sloupce. Originální hodnota buňky – řetězec z CSV souboru – je uložena v anotaci *string value*. Anotace *value* umožňuje validátoru vnímat sémantickou hodnotu buňky např. jako číslo nebo datum. Buňka dále obsahuje anotace pro tabulku (*table*), sloupec (*column*) a řádek (*row*), ve kterém je obsažena.



Obrázek 3.1: Diagram anotovaného modelu dle *CSV on the Web* [1]

Buňky sloupce nabývají hodnot určitého typu, který vznikne parsováním řetězce uloženého v buňce. Více se o datových typech, jež se používají v anotovaném datovém modelu, dozvíte v následující Kapitole 3.2.2.

Poslední komponentou je skupina tabulek (*table group*), jež v anotaci *tables* obsahuje množinu tabulek, které do této skupiny patří. V anotaci musí být alespoň jedna tabulka.

3.2.2 Datové typy

Datové typy používané v anotovaném modelu jsou podmnožinou datových typů definovaných dle *W3C XML Schema* [18]. Model tabulkových dat omezuje množinu datových typů na ty, jež jsou zobrazeny na Obrázku 3.2. Od těchto datových typů se však mohou odvozovat nové datové typy pomocí anotace *base*. [1]

Anotace *base* obsahuje absolutní URL libovolného datové typu z Obrázku 3.2. Pomocí ostatních anotací lze tento typ specifikovat. Příkladem je anotace formát (*format*), jež slouží k popisu formátu hodnot datového typu, který je následně využit při parsování hodnoty z řetězce buňky. [1]

Dalšími anotacemi lze omezit délku hodnot nebo jejich množinu. U textových a binárních datových typů lze omezit délku hodnot např. definováním maximální možné délky. U datových typů pro čísla, datum, čas a dobu trvání lze omezit hodnoty, kterých mohou nabývat. [1]

Pro různé datové typy musí aplikace podporovat i jejich různé formáty [1]:

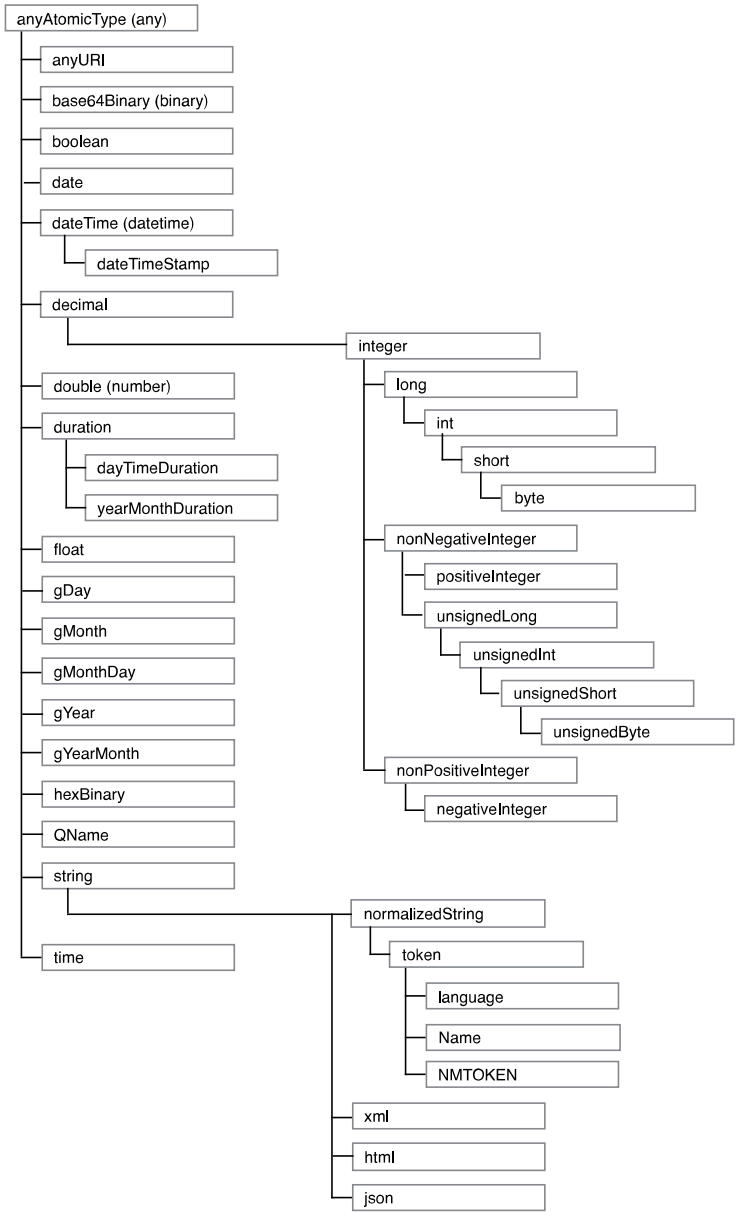
- **Formát pro numerické typy** by měl splňovat formát definovaný dle *XML Schema* [18]. Není běžné v tabulkových datech formátovat čísla pro dobrou čitelnost – např. seskupení číslic po řádech, přidávání znaménka pro procenta a další – ale je potřeba s tímto počítat. Proto je nutné v anotovaném modelu umět rozpoznávat číselný formát dle *Number Format Patterns* [10].

- **Formát pro boolean** lze definovat mnoha způsoby, nejčastěji se však setkáte s `0` a `1` či `false` a `true`. Boolean formát lze zapsat řetězcem – hodnotu pro `true` následovanou hodnotou pro `false`, oddělené znakem `|`. Tedy např. `ANO|NE`, kde `ANO` značí hodnotu `true` a `NE` hodnotu `false`.

- **Formát pro datum a čas** by měl být vyjádřen dle *XML Schema* [18], nicméně v tabulkových datech jsou čas i datum často reprezentovány různými způsoby. Z tohoto důvodu je podporován formát se symboly z *Date Field Symbol Table* [10]. Seznam všech povolených formátu najdete v Kapitole 6.4.4 v článku *Model for Tabular Data and Metadata on the Web* [1]

- **Formát pro dobu trvání** musí být ve formátu definovaném v *XML Schema* [18], který používá formát ISO 8601 `-?PnYnMnDTnHnMnS`. Příkladem může být `P1Y1D`, jež značí rok a den.

- Je-li datový typ jiný než některý z výše zmíněných, lze jeho formát definovat ve formě regulárního výrazu dle jazyku *ECMAScript* [19].



Obrázek 3.2: Podporované datové typy hodnot vycházející z W3C XML schématu [2]

3.3 Metadata pro tabulková data

Metadata uvedena v této kapitole slouží k získání anotací a vytvoření anotovaného modelu tabulkových dat z Kapitoly 3.2. Metadata jsou uložena ve formátu JSON-LD se speciálním dialektem, který byl definován v Kapitole 2.3. Tyto JSON-LD dokumenty se skládají z tzv. popisujících objektů, které jsou definovány dle [7] následovně:

Definice 4 *„**Popisující objekt** je JSON objekt, jež popisuje jednu z komponent anotovaného modelu tabulkových dat (např. tabulku) a obsahuje jednu nebo více vlastností, z kterých lze vytvořit anotace této komponenty.“*

V Příkladu 3.3 je ukázka popisujícího objektu sloupce tabulky. Obsahuje vlastnosti *name*, *titles* a *datatype*, které slouží k vytvoření stejně pojmenovaných anotací sloupce. Vlastnost *dc:description* je tzv. společná vlastnost. Společné vlastnosti jsou vysvětleny v Kapitole 3.3.3 a slouží ke vzniku doplňujících anotací. [7]

Příklad 3.3: Ukázka popisujícího objektu sloupce

```
{
  "name": "birth_date",
  "titles": "Date of Birth",
  "dc:description": "The date of birth.",
  "datatype": {
    "base": "date",
    "format": "M/d/yyyy"
  }
}
```

Pomocí definice popisujícího objektu, lze definovat metadatový dokument [7]:

Definice 5 *„**Metadatový dokument** je JSON dokument obsahující jeden objekt. Tímto objektem je popisující objekt skupiny tabulek nebo jedné tabulky. Takový dokument však může mít i další vnořené nebo pomocí URI referencované popisující objekty (např. pro sloupce). Jeho součástí jsou i jiné JSON objekty, které nepopisují komponenty anotovaného modelu. Tyto objekty popisují např. definice cizích klíčů a jsou použity k vytvoření anotací v anotovaném modelu.“*

3.3.1 Typy vlastností

Nyní budou představeny typy JSON vlastností definovaných dle *CSV on the Web*. Každá z vlastností má jinou množinu povolených hodnot a aplikace musí tyto vlastnosti správně interpretovat.

Klasické JSON vlastnosti jsou dle *CSV on the Web* nazývány jako **atomické vlastnosti**. Do této skupiny zařazujeme čísla (celá nebo desetinná), boolean hodnoty (`true` nebo `false`), textové řetězce, objekty nebo pole těchto hodnot.

Dalším typem vlastností jsou **objektové vlastnosti**. Ty obsahují buď jeden vnořený popisující objekt, anebo řetězec s URL odkazující na tento objekt. Prostřednictvím této vlastnosti lze definovat popisující objekt na určité URL adrese. Tento objekt poté lze používat v různých metadatových dokumentech, a to pomocí odkazu na tuto URL. Tímto tedy odpadá nutnost definovat popisující objekt stále dokola. Příkladem může být vlastnost *dialect*, definována buď jako reference (Příklad 3.4), nebo jako vnořený objekt (Příklad 3.5). [7]

Příklad 3.4: Reference na popisující objekt dialektu

```
"dialect": "http://example.org/tab-separated-values"
```

Příklad 3.5: Vlastnost `dialect` s vnořeným popisujícím objektem dialektu

```
"dialect": {
  "delimiter": "\t",
  "encoding": "utf-8"
}
```

Array vlastnosti v sobě obsahují pole popisujících objektů. Typickým příkladem je popisující objekt skupiny tabulek a jeho array vlastnost *tables*, ve které se nachází popisující objekty tabulek patřící do této skupiny.

Má-li popisující objekt sloupce definovanou vlastnost *name*, pak lze její hodnotu brát jako identifikátor sloupce v dané tabulce. Tímto identifikátorem lze na sloupec odkazovat. **Column reference vlastnosti** obsahují jeden či více identifikátorů sloupce. [7]

Link vlastnosti v sobě uchovávají odkaz na zdroj identifikovatelný pomocí URL. Jejich hodnota je textový řetězec, který se přeloží na URL vzhledem k tzv. *base URL metadatové dokumentu*. Zmíněný překlad je definován v RFC 3986 [14]. *Base URL* je detailněji vysvětlena Kapitole 3.3.3.

Natural language vlastnosti obsahují textové řetězce v přirozeném jazyce. Jejich hodnoty jsou řetězce nebo pole řetězců v defaultním jazyce (viz Kapitola 3.3.3). Hodnotou natural language vlastnosti může být i objekt, jehož vlastnosti jsou jazykové kódy dle BCP47 [20] a hodnotami jsou řetězce či pole řetězců. Ukázka popisovaného objektu je znázorněna na Příkladu 3.6. [7]

Posledním typem vlastností jsou tzv. **URI template vlastnosti**, jež obsahují URI template (česky URI šablonu) definovanou dle RFC 6570 [21]. Z těchto šablon se generují URI. Generování probíhá v kontextu každé řádky kombinací šablony s proměnnými. Mezi tyto proměnné patří např. hodnota

aktuální buňky, číslo aktuálního sloupce nebo řádku, ad. URI template i ostatní vlastnosti jsou specifikovány v článku *Metadata Vocabulary for Tabular Data* [7], ve kterém lze nalézt i situace, v nichž má validátor vytvořit chybu či varování. [7]

Příklad 3.6: Natural language vlastnost s hodnotou ve formě objektu

```
"titles": {
  "en": [ "Project title", "Project" ],
  "fr": "Titre du projet"
}
```

3.3.2 Typy popisujících objektů

V metadatových dokumentech existují následující popisující objekty [7]:

- skupiny tabulek,
- tabulky,
- sloupce,
- datového typu,
- schématu a
- dialektu.

Všechny tyto popisující objekty, vyjma schématu a dialektu, slouží k vytváření anotací pro stejně pojmenované komponenty anotovaného modelu.

Oproti tomu schéma definuje formát tabulky. Jedná se např. o její sloupce, cizí klíče a primární klíče. Tento formát může být společný pro vícero tabulek. Dialekt poskytuje parserům tabulkových dat informace o jejich formátu, neboť *CSV on the Web* se snaží pokrýt většinu používaných formátů i přesto, že k ním neexistují standardy. V dialektu lze tedy specifikovat oddělovač buněk, konce řádků, kódování, uvozovací znak a další vlastnosti, které jsou potřeba pro správné parsování. [7]

Každý z těchto popisujících objektů je složen z několika vlastností jejichž názvy a popisy lze nalézt v Kapitole 5 v článku *Metadata Vocabulary for Tabular Data* [7]. Tyto vlastnosti mají nadefinované hodnoty, kterých mohou nabývat, a také kdy a za jakých podmínek má validátor vytvořit varování či chybu.

3.3.3 Speciální vlastnosti

Kromě jednotlivých typů vlastností z Kapitoly 3.3.1 lze v metadatových dokumentech objevit i speciální vlastnosti. Mezi tyto vlastnosti lze zařadit hlavní vlastnosti metadat, společné vlastnosti a dědičné vlastnosti.

Dle definice metadatového dokumentu je tento dokument tvořen jedním JSON objektem. Tento objekt společně s popisujícími objekty, jež jsou referencované pomocí URL, musí obsahovat kontext z kapitoly 2.2.2. Kontext je tvořen vlastností *@context*. Hodnotou této vlastnosti musí být řetězec `http://www.w3.org/ns/csvw` nebo pole o dvou prvkách, kde první prvek je zmíněný řetězec a druhý prvek je objekt s dvěma vlastnostmi [7]:

- **@base** – řetězec definující *base URL*, vůči níž se ostatní URL z metadatového souboru vyhodnocují. Není-li vlastnost specifikována, pak se za *base URL* považuje umístění metadatového dokumentu.

- **@language** – řetězec definující defaultní jazyk pro hodnoty řetězců v přirozeném jazyce. Musí splňovat podmínky jazykového kódu podle BCP47 [20].

Popisující objekty skupiny tabulek, tabulky, sloupce i schématu mohou obsahovat tzv. **společné vlastnosti**, jejichž název (klíč) je absolutní nebo kompaktní URL. Popis tabulky může např. obsahovat vlastnosti *dc:description*, *dcat:keyword* a *schema:copyrightHolder* k poskytnutí popisu tabulky, klíčových slov a držitele autorských práv, tak jak je definováno ve slovnících *Dublin Core Terms* [22], *DCAT* [23] a *schema.org* [16]. Hodnoty společných vlastností jsou jakékoliv JSON-LD hodnoty. V kontextu *CSVW Namespace Vocabulary Terms* [17] jsou definovány prefixy pro slovníky, které lze použít. Tento kontext není možné přepisovat, tudíž je-li potřeba použít jiný slovník, definuje se vlastnost absolutní URL místo kompaktní URL. Ze společných vlastností vznikají doplňující anotace pro anotovaný model. [7]

Sloupce a řádky mohou mít přiřazeny anotace podle vlastností popisujících objektů (skupiny tabulek, tabulek nebo schématu), ke kterým náleží. Tyto vlastnosti jsou nazývány **dědičnými vlastnostmi**. Pokud dědičná vlastnost není definována v popisujícím objektu sloupce, převezme se její definice z popisujícího objektu schématu nebo z tabulky, případě ze skupiny tabulek. [7]

3.4 Zpracování tabulkových dat a metadat

Pracuje-li aplikace s tabulkovými daty a vytváří z nich společně s metadaty anotovaný model, musí aplikace metadata nejdříve lokalizovat, poté je normalizovat a nakonec porovnat jestli jsou kompatibilní s tabulkovými daty. V následujících podkapitolách jsou tyto jednotlivé části stručně vysvětleny.

3.4.1 Lokalizace metadat

V předchozích kapitolách bylo řečeno, že tabulková data jsou nejčastěji uložena ve formátu CSV. CSV formát je velmi jednoduchý, z toho důvodu se metadata k tabulkovým datům musí přikládat v podobě JSON-LD deskriptoru, viz Kapitola 3.3. Aplikace pracující s tabulkovými daty musí tento deskriptor lokalizovat.

Existují čtyři metody pro získání metadat a jsou seřazeny postupně dle pořadí, v jakém se mají vykonávat [1]:

1. metadata jsou dodána přímo uživatelem – viz Kapitola 3.4.1.1;
2. při přístupu k tabulkovým datům na webu přes HTTP, je v HTTP odpovědi vyplněna hlavička **Link**, jež odkazuje na soubor, který je JSON-LD deskriptorem – viz Kapitola 3.4.1.2;
3. metadata lze najít z odkazů získaných pomocí tzv. **site-wide** konfigurace – viz Kapitola 3.4.1.3;
4. metadata jsou dodávána rovnou v souboru tabulkových dat. V tomto případě se metadata nazývají **embedovaná**, viz Kapitola 3.4.1.4.

3.4.1.1 Uživatelem vložená metadata

Aplikace by měly uživateli poskytnout možnost vložit jejich vlastní metadata. Tato metadata mohou být poskytnuta např. [1]:

- nastavením aplikace (např. argumentem v příkazovém řádku);
- přes GUI (*Graphical User Interface*, česky grafické uživatelské rozhraní);
- výběrem souboru s metadaty (lokální nebo vzdálený soubor).

3.4.1.2 HTTP hlavička Link

Pokud uživatel nevloží svá vlastní metadata a přijímá tabulková data z webu přes HTTP, musí aplikace zkontrolovat HTTP odpověď, zda neobsahuje hlavičku `Link` s následujícími parametry [1]:

- `rel="describedby"`,
- `type="application/csvm+json"`, `type="application/ld+json"` nebo `type="application/json"`.

Metadatový soubor získaný z hlavičky `Link` je **validní**, pokud obsahuje referenci na tabulková data. Je-li metadatový soubor nevalidní, musí se ignorovat. [1]

3.4.1.3 Site-wide konfigurace

Nedostaneme-li od uživatele metadata přímo a nemáme-li ani validní metadata soubor v hlavičce `Link`, musí se aplikace pokusit najít metadatový dokument pomocí tzv. site-wide konfigurace.

V tomto případě musí aplikace nejdříve najít site-wide konfiguraci. Ta by měla být uložena na *well-known URI* [24] `/.well-known/csvm`. Tento soubor musí obsahovat na každém řádku URI template dle RFC 6570 [21]. Počínaje prvním řádkem aplikace provádí [1]:

1. rozšíření URI template tak, že za proměnnou `url` dosadí URL souboru s tabulkovými daty;
2. přeložení výsledné URL vzhledem k URL souboru s tabulkovými daty;
3. získání metadatového dokumentu z finální URL;
4. pokud není nalezen validní metadatový dokument, pak se tyto kroky provedou pro další URI template.

Jestliže soubor na URI `/.well-known/csvm` neexistuje (např. HTTP kód odpovědi je `4xx` nebo `5xx` – tedy chybový kód), pak se použijí jen dvě defaultní URI template z Příkladu 3.7.

Příklad 3.7: Základní URI template pro lokalizaci metadat

```
{+url}-metadata.json
csv-metadata.json
```

3.4.1.4 Embedovaná metadata

Většina ze syntaxí zápisu tabulkových dat umožňuje i zápis embedovaných metadat. Mělo by však být definováno, jak syntaxi namapovat na anotovaný model. Například parsováním základního CSV souboru lze získat názvy sloupců (anotaci *titles*) z hlavičky. [1] Tyto metadata však nejsou na rozdíl od JSON-LD deskriptoru strojově čitelná.

3.4.2 Kompatibilita metadat

Předtím, než aplikace použije nalezená metadata, se musí ujistit, zda jsou kompatibilní s tabulkovými daty. Pokud metatada nejsou kompatibilní, musí aplikace pokračovat v jejich lokalizaci dle Kapitoly 3.4.1. [1]

Pro zjištění kompatibility je potřeba z tabulkových dat získat embedovaná metadata, a poté se o kompatibilitě rozhoduje dle následujících pravidel.

Popisující objekty dvou skupin tabulek nazveme kompatibilní, pokud v každé skupině existuje alespoň jedna tabulka, jež obsahuje stejnou normalizovanou

hodnotu vlastnosti *url*, a pokud jsou zároveň tyto tabulky kompatibilní. Tabulky jsou kompatibilní pokud mají stejnou normalizovanou hodnotu vlastnosti *url* a zároveň mají kompatibilní schéma. Dvě schémata jsou kompatibilní, jestliže mají stejný počet popisů nevirtuálních sloupců a zároveň pokud jsou tyto popisy na stejných indexech navzájem kompatibilní. [7] Popisy sloupců jsou kompatibilní za následujících podmínek [7]:

- Oba sloupce nemají definovanou vlastnost *name* ani *titles*.
- Pokud se řetězce z vlastností *name* obou sloupců shodují, a to včetně velikosti písmen.
- Existuje-li průnik mezi hodnotami vlastnosti *titles*. Dva řetězce z různých vlastností *titles* se shodují, pokud si jsou řetězce rovny včetně velikosti písmen a zároveň se rovnají i jejich jazyky. Jazyky se shodují, pokud jsou si rovny po zkrácení definovaném v *Tags for Identifying Languages* [20].

3.4.3 Normalizace metadat

Předtím než budeme vytvářet anotace je nutné získaná metada normalizovat. Důvodem jsou různé možnosti zápisu metadat, např. popisující objekty mohou být referencované pomocí URL. Tyto objekty se z dané URL musí během normalizace získat a správně vložit do JSON-LD deskriptoru. Cely postup normalizace je popsán v Kapitole 6 článku *Metadata Vocabulary for Tabular Data* [7].

Po normalizaci metadat jsou vlastnosti *@base* a *@language* obsažené v kontextu irelevantní, tudíž jakákoliv normalizovaná metadata mohou mít vlastnost *@context* nastavenou na řetězec `"http://www.w3.org/ns/csvw"`. [7]

3.4.4 Normalizace URL

Nalezení metadat a jejich kompatibilita závisí na porovnávání URL, proto je důležité používat normalizované URL. Aplikace musí používat *Syntax-Based* normalizace a *Scheme-Based* normalizace pro schémata HTTP (80) a HTTPS (443). Vyjmenované normalizace jsou definovány v RFC 3986 [14]. V Příkladu 3.8 jsou dvě URL, které si jsou po *Syntax-Based* normalizaci rovny. V Příkladu 3.9 jsou čtyři URL, jež se rovnají po *Scheme-Based* normalizaci. [1].

Příklad 3.8: Ekvivalentní URL po Syntax-Based normalizaci [14]

```
example://a/b/c/%7Bfoo%7D
eXAMPLE://a/./b/../b/%63/%7bfoo%7d
```

Příklad 3.9: Ekvivalentní URL po Scheme-Based normalizaci [14]

```
http://example.com
http://example.com/
http://example.com:/
http://example.com:80/
```

Kapitola 4

Existující řešení

Následující kapitola obsahuje popis již existujících řešení validátorů dle doporučení *CSV on the Web*. Nejprve jsou popsány referenční implementace validátorů a jejich vyhodnocení vzhledem k referenční sadě testů. Ve druhé části je zanalyzována webová aplikace a webová služba csvlint.io [3]. Nakonec jsou funkční existující řešení porovnány mezi sebou.

Validátor dle *CSV on the Web* testuje, zda tabulková data dodržují strukturu definovanou ve schématu a zda je schéma vůbec validním metadatovým dokumentem dle Kapitoly 3.3. Hlavním úkolem validátorů je varovat uživatele porušují-li tabulková data či metadata pravidla z doporučení *CSV on the Web* [1][7]. Výstupem validátoru je seznam chyb a varování, kde chyba značí velké porušení pravidel, zatímco varování má jen uživatele upozornit na drobné problémy.

4.1 Referenční validátory CSV on the Web

Existují celkem tři referenční implementace validátoru. Jedná se o CSVLint [25], pycsvw [26] a RDF::Tabular [27]. V následujících podkapitolách jsou všechny tři implementace popsány a otestovány.

4.1.1 Validátor CSVlint

CSVlint je knihovna vytvořená v programovacím jazyku Ruby (tzv. *ruby gem*). Knihovna podporuje validování CSV podle specifikace *CSV on the Web* a lze ji využít i v jiném Ruby programu či jako samostatnou konzolovou aplikaci. [25]

Mezi hlavní funkce CSVlint patří [25]:

- validace formátu lokálního CSV souboru nebo CSV souboru získaného z URL;
- validace různých CSV dialektů;

- validace schémat tabulkových dat podle standardů *JSON Table Schema* a *CSV on the Web*.

CSVlint dle výsledků v originální testovací sadě *CSVW Validation Tests* podporuje celé doporučení, neboť prochází všemi 281 testovacími scénáři. [5]

Ve zdrojovém kódu CSVLint kompletně chybí jeho dokumentace. V GitHub repositáři knihovny existuje alespoň uživatelská dokumentace, ve které jsou uvedeny příklady, jak s knihovnou pracovat. [25] Instalace je jednoduchá. Je-li na stroji nainstalovaný jazyk Ruby, tak se CSVlint nainstaluje příkazem:

```
gem install csvlint
```

Poté lze spustit validaci CSV souboru:

```
csvlint URL
```

`URL` je cesta k CSV souboru na lokálním počítači, či na webu. Pro validaci schématu se používá přepínač `--schema`:

```
csvlint --schema=URL
csvlint URL1 --schema=URL2
```

Přestože je instalace i použití jednoduché, má aplikace mnoho chyb. Program nelze spustit na operačním systému Windows 10, protože k jeho spuštění je zapotřebí nástroj cURL, který není součástí operačního systému. Po nainstalování nástroje cURL for Windows zůstala situace nezměněna. Testování tedy probíhalo na systému macOS 10.14 . Při spuštění jakékoliv validace hlásí CSVlint problém o nevalidním kódováním souboru a také jiné chyby související s kódováním. Domnívám se, že hlavním zdrojem těchto chyb je zastaralost použitých Ruby knihoven, neboť již Ruby kompilátor upozorňuje na použití starých (tzv. *deprecated*) funkcí. Příklad 4.1 obsahuje chybu programu vypsanou při snaze validovat CSV soubor.

Příklad 4.1: Chyba při validaci CSV souboru pomocí CSV Lint

```
csvlint http://www.w3.org/2013/csvw/tests/test245.csv
NOTE: Csvlint::Schema.load_from_json is deprecated;
   use load_from_uri instead. It will be removed on
   or after 2018-01-01.

http://www.w3.org/2013/csvw/tests/test245.csv is
   INVALID
1. wrong_content_type
2. invalid_encoding. Row: 1. 1
1. no_encoding
```

4.1.2 Validátor Pycsvw

Pycsvw (*Python implementation of the W3C CSV on the Web specification*, česky Python implementace specifikace W3C *CSV on the Web*) je již dle názvu aplikace napsaná v programovacím jazyku Python podporující validování CSV podle doporučení *CSV on the Web*. [26]

Jedná se o nejjednodušší implementaci z popisovaných validátorů, co lze vypozorovat i z výsledků testů z originální testovací sady. Pycsvw splňuje pouze 158 testovacích scénářů z celkového počtu 281. Většina chybných testů je zapříčiněna chybějící podporou formátu čísel dle UAX35 [10], omezení délky a hodnoty datových typů a několika dalších validací. [5]

Velkým nedostatkem je rovněž absence dokumentace, a to jak dokumentace zdrojového kódu, tak především uživatelské dokumentace. [26]

Stav této aplikace je katastrofální. Většina problémů může být jen důsledkem chybějící dokumentace, což však neeliminuje chyby aplikace. Aplikace je implementována v jazyku Python verze 2.7, která je již zastaralá a bude pro ni brzo ukončena podpora. Pro spuštění aplikace je zapotřebí nejdříve přejmenovat soubor `/pycsvw/main.py` na `/pycsvw/__main__.py`. Následně je nutná instalace knihovny *simplejson*:

```
python -m pip install simplejson --user
```

Poté lze již instalovat samotnou aplikaci pycsvw z jejího adresáře:

```
python setup.py install
```

Po skončení instalaci lze aplikaci spustit příkazem:

```
python pycsvw
```

Vzhledem k chybějící dokumentaci není jasné, jaké parametry jsou nezbytné pro spuštění validování souborů.

Pokud soubor `/pycsvw/main.py` není přejmenován na soubor s názvem `/pycsvw/__main__.py`, lze knihovnu pycsvw importovat do interaktivního Pythonu v konzoli. I přesto se však validaci nepodařilo spustit, neboť volání funkce očekávalo více parametrů než jen URL validovaného souboru. Bez dokumentace však nelze zjistit, jaké konkrétní parametry jsou zapotřebí.

4.1.3 Validátor RDF::Tabular

Další z knihoven vytvořených v programovacím jazyku Ruby (tzv. *ruby gem*) je RDF::Tabular. Ta podporuje nejenom validaci CSV podle doporučení *CSV on the Web*, ale také převod CSV do formátu JSON-LD a RDF za pomoci *CSV on the Web* metadat. [27] Jako jediná implementace splňuje všechny scénáře všech referenčních testů (*CSVW JSON tests*, *CSVW RDF Tests* a *CSVW Validation Tests*) – tedy nejen validačních, ale i konverzních. [5]

Aplikace je výborně zdokumentována včetně zdrojového kódu. Jedním z důvodů kvality RDF::Tabular může být i fakt, že hlavním vývojářem je Gregg Kellogg, jeden z autorů doporučení *CSV on the Web*. [27]

Instalace je rovněž jako u CSVlint jednoduchá. Je-li na stroji nainstalovaný jazyk Ruby, nainstaluje se nástroj příkazy:

```
gem install linkeddata
gem install rdf-tabular
```

Poté lze spustit validaci CSV souboru příkazem:

```
rdf validate URL
```

`URL` je cesta k souboru s tabulkovými daty na lokálním počítači či na webu.

Validaci se vstupem JSON-LD deskriptoru dle *CSV on the Web* spustíme přidáním přepínače `--format=tabular`:

```
rdf validate --format=tabular URL
```

Výstupem programu RDF::Tabular je informace, zda je soubor validní či nikoliv. Zároveň jsou uvedeny všechny chyby a varování, které nastaly během validace. Vzhledem k tomu, že program opět nelze zprovoznit na operačním systému Windows 10, probíhalo testování na systému macOS 10.14. Při testování souborů z referenčních testů *CSV on the Web* se zdálo, že RDF::Tabular je jedinou funkční referenční implementací. Bohužel validátor umí pracovat pouze s URI nikoliv s IRI. Pokud validátor spustíme s IRI, jež obsahuje např. českou diakritikou (`http://dev.nkod.opendata.cz/soubor/datov%C3%A9-sady.csv-metadata.json`), validace neproběhne a program skončí s chybou, a to právě kvůli neznámým znakům v IRI.

4.2 Webové rozhraní csvlint.io

Webová aplikace a webová služba csvlint.io jsou k dispozici na adrese `https://csvlint.io` respektive `https://csvlint.io/package.json`. Jejich účelem je poskytnout snadnou validaci CSV souborů, a to za pomoci Ruby knihovny CSV Lint z minulé Kapitoly 4.1.1.

Kromě CSV souborů zvládne csvlint.io validovat i ZIP archivy obsahující CSV soubor. Validátor není nutně omezen na formát CSV, neboť mu lze nastavit dialekt – oddělovač hodnot, znaky pro konce řádků a uvozovací znak. K tabulkovým souborům lze přiložit schéma, jedná se však o JSON Table Schema [6], což je kromě doporučení *CSV on the Web* jiný způsob definování schématu tabulkových dat. Schémata dle doporučení *CSV on the Web* nejsou podporována. [3]

Validování kontroluje zadané schéma a základní chyby v CSV souborech, kterými jsou např. jiné kódování než UTF-8, nedefinovaná hlavička, prázdné

řádky, ad. Po skončení validace vypíše *csvlint.io* seznam chyb a varování. Dokonce lze stáhnout upravený CSV soubor, ve kterém budou vyřešeny jednoduché chyby např. nevalidní kódování. [3]

4.2.1 Webová služba

Webová služba je dostupná na adrese `https://csvlint.io/package.json` a umožňuje validovat libovolné množství CSV souborů publikovaných na webu včetně JSON Table schémat. Validaci CSV souboru lze spustit pomocí cURL, viz Příklad 4.2.

Příklad 4.2: Validace CSV souboru pomocí webové služby csvlint.io [3]

```
curl -L --data "urls[]=
  http://theodi.github.io/hot-drinks/hot-drinks.csv"
http://csvlint.io/package.json
```

Pro validovaci CSV soubor spolu s jeho schématem, je nutné požadavek upravit tak, jako je zobrazeno v Příkladu 4.3.

Příklad 4.3: Validace CSV souboru s JSON Table Schema pomocí webové služby csvlint.io [3]

```
curl -L --data "urls[]=
http://theodi.github.io/hot-drinks/hot-drinks.csv
  &schema=1
  &schema_url=http://example.com/some_schema.json"
http://csvlint.io/package.json
```

Webová služba vytvoří asynchronní validační úlohu s identifikátorem, který je odeslán uživateli v JSON odpovědi, viz Příklad 4.4. Po skončení úlohy, lze poslat požadavek na URL adresu, jež byla vrácena v této odpovědi. Tím se získají data ve formátu JSON, která obsahují např. výsledek validace, nalezené chyby nebo varování. [3]

Příklad 4.4: Odpověď webové služby csvlint.io na požadavek z Příkladu 4.2 [3]

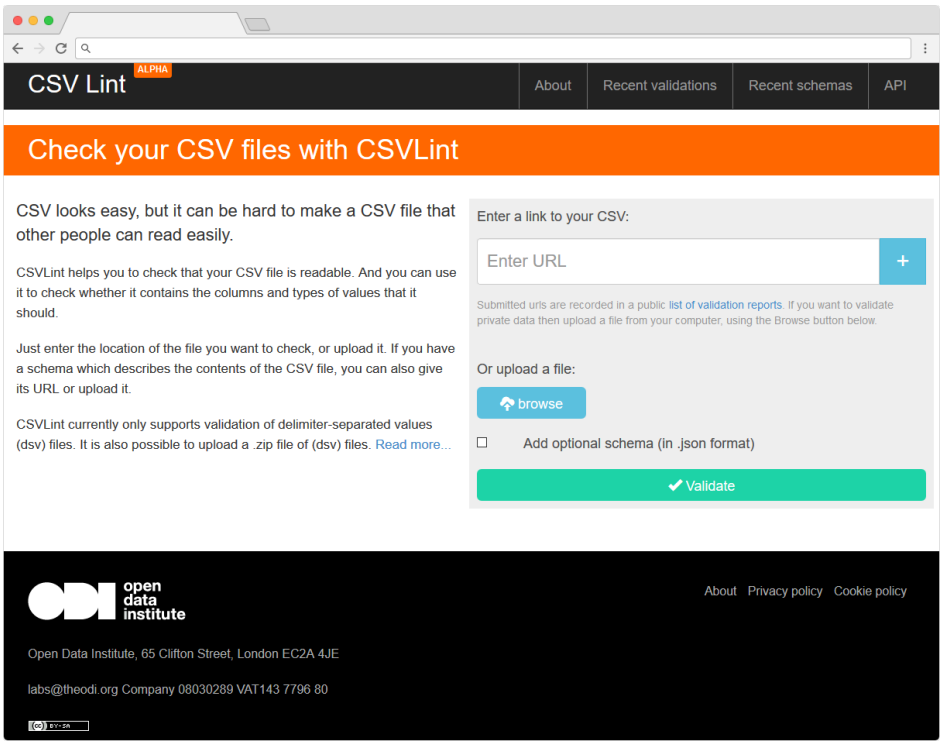
```
{
  "package": {
    "url":"http://csvlint.io/package/53a150336373764"
  }
}
```

Bohužel ani webová služba csvlint.io nepodporuje IRI.

4.2.2 Webová aplikace

Webovou aplikaci csvlint.io lze nalézt na adrese `https://csvlint.io`. Jedná se o jednoduché webové rozhraní, které využívá webovou službu z Kapitoly 4.2.1.

Validace se spustí vyplněním formuláře z Obrázku 4.1. Do formuláře lze vložit libovolný počet CSV souborů – pomocí URL adresy nebo nahráním souboru z disku. K validaci lze přidat i JSON Table schémata, a to také jako URL nebo nahráním z disku. Po stisknutí tlačítka `Validate`, se odešle požadavek webové službě. Když je validace dokončena, je uživatel přesměrován na obrazovku s výsledkem, viz Obrázek 4.2. [3]

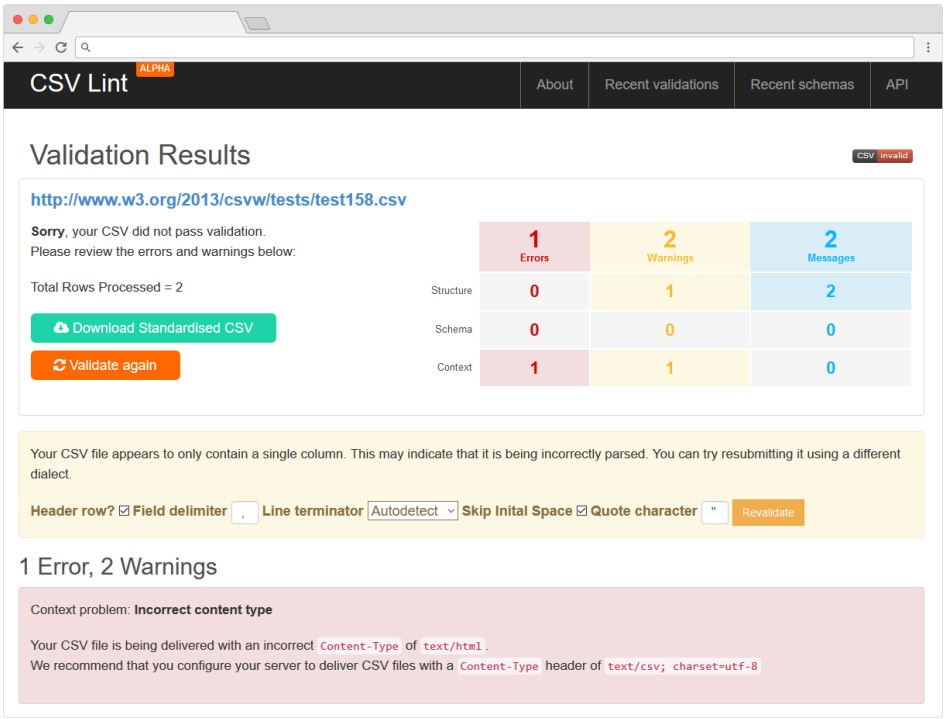


Obrázek 4.1: Validační stránka webové aplikace csvlint.io [3]

Obrazovka s výsledkem validace obsahuje [3]:

- výsledek validace – zda je soubor validní či nikoliv;
- celkový počet zkontrolovaných řádků CSV souboru;
- statistiky vzniklých chyb a varování;
- tlačítko ke stažení opraveného CSV souboru;

- tlačítko sloužící k opakované validaci;
- formulář pro definici dialektu CSV souboru;
- seznam všech chyb a varování i s popisem, který uživatele informuje, jak daný problém vyřešit.



Obrázek 4.2: Stránka s výsledkem validace webové aplikace csvlint.io [3]

Webová aplikace obsahuje kromě možnosti validace i popis aplikace a webové služby, a to včetně návodu, jak je používat. [3]

4.3 Porovnání validátorů

Tato kapitola obsahuje porovnání existujících řešení. Jelikož validátor CSVLint a Pycsvw nebylo možné během testování zprovoznit, budou v konečném srovnání vynechány. Porovnávat se tedy budou jen funkční validátory – csvlint.io a RDF::Tabular. Shrnutí těchto dvou validátoru naleznete v Tabulce 4.1.

RDF::Tabular je čistě konzolová aplikace, zatímco csvlint.io je nadstavba nad validátorem CSV Lint. Tato nadstavba obsahuje webovou službu a webovou aplikace, což vede k jednoduchému používání a k nezávislosti na operačním systému.

Funkcionalita	csvlint.io	RDF::Tabular
Validace lokálních CSV	Ano	Ano
Validace CSV z URL	Ano	Ano
Podpora CSV on the Web schémat	Ne	Ano
Podpor JSON Table schémat	Ano	Ne
Transformace tabulkových dat do RDF	Ne	Ano
Transformace tabulkových dat do JSON	Ne	Ano
Konzolová aplikace	Ne	Ano
Webová služba	Ano	Ne
Webová aplikace	Ano	Ne
Podpora IRI	Ne	Ne
Podpora OS Windows 10	Ano	Ne
Podpora OS macOS 10	Ano	Ano

Tabulka 4.1: Porovnání jednotlivých funkcionalit mezi validátory csvlint.io a RDF::Tabular

Oba testované validátory dokáží validovat lokální i vzdálené CSV soubory, a to dokonce v různých dialektech. Velkým nedostatkem obou validátorů ale je neschopnost používat IRI jako identifikátor souborů. Oba totiž podporují jen URI s ASCII znaky.

Mezi největší nevýhody csvlint.io patří absence podpory pro schémata dle *CSV on the Web*. Dokáže přijímat pouze JSON Table Schema [6]. Zatímco RDF::Tabular splňuje celé doporučení *CSV on the Web* a to jak validace, tak i převody do formátu JSON a RDF.

Kapitola 5

Analýza validátoru

Z pohledu jednotlivých funkcionalit validátoru *CSV on the Web*, lze jeho práci rozdělit do těchto kroků:

1. parsování a validace souboru tabulkových dat (především souborů CSV);
2. lokalizace schématu pro tabulková data;
3. parsování, validace a normalizace JSON-LD metadatového souboru obsahující schéma;
4. porovnání kompatibility mezi souborem tabulkových dat a jeho schématem;
5. vytvoření anotovaného modelu tabulkových dat ze souboru tabulkových dat a jeho schématu;
6. validace vzniklého anotovaného modelu;

Pohled na tyto jednotlivé kroky usnadňuje definování jednotlivých požadavků validátoru v Kapitole 5.1 a případy užití v Kapitole 5.2.

5.1 Analýza požadavků

Účel a funkčnosti validátoru jsou definovány v zadání práce a v doporučení *CSV on the web*. S vedoucím práce byly však některé požadavky dospecifikovány.

5.1.1 Funkční požadavky

Mezi funkční požadavky patří:

F1 Podpora IRI. Validátor bude schopen pracovat s IRI dle RFC 3987 [13] a zároveň je musí normalizovat dle RFC 3986 [14].

F2 Práce s lokálními soubory. Validátor bude schopen nalézt soubory na lokálním disku a následně je číst.

F3 Práce se vzdálenými soubory. Validátor bude schopen stáhnout soubory z webu, které budou definovány pomocí IRI.

F4 Parsování tabulkových dat a extrahování vnořených metadat. Validátor dokáže parsovat tabulková data a vyextrahovat z nich vnořená metadata dle algoritmu z Kapitoly 8 *Parsing Tabular Data* článku *Model for Tabular Data and Metadata on the Web* [1].

F5 Validace CSV souboru. Validátor bude kontrolovat podmnožinu nejčastějších chyb CSV souborů definovaných na webu otevřených dat [28]. Konkrétně se budou kontrolovat tyto chyby:

- Jiné kódování než UTF-8.
- Jiný oddělovač než čárka.
- Chybné escapování čárek a uvozovek.
- Mezery použité pro zarovnání (tzv. *leading* a *trailing spaces*).
- Hodnoty null.
- Chybějící schéma.
- Chybná hlavička HTTP Content-Type u CSV souboru.

Poruší-li CSV soubor některé z těchto pravidel, tak musí validátor vytvořit varování.

F6 Lokalizace schématu. Začíná-li validátor validaci jen s tabulkovými daty, musí být pro ně schopen najít metadatový dokument podle postupu z Kapitoly 3.4.1.

F7 Parsování metadatového dokument. Validátor musí být schopen parsovat JSON-LD deskriptor, jež je ve formátu popsaném v Kapitole 3.3.

F8 Porovnání kompatibility. Validátor bude schopen kontrolovat kompatibilitu mezi tabulkovými daty a nalezeným schématem podle Kapitoly 3.4.2.

F9 Normalizace metadatového dokumentu. Metadatový dokument musí být před použitím normalizován podle Kapitoly 3.4.3.

F10 Validace metadatového dokumentu. Validátor musí validovat také formát medatadového dokumentu.

F10.1 Validátor musí vygenerovat chybu a zastavit zpracovávání, pokud dokument s metadaty nepoužívá validní JSON syntaxi, nedodržuje JSON-LD dialekt z Kapitoly 2.3, nebo pokud není definována povinná vlastnost.

F10.2 Validátor musí vytvořit varování v případě, že objeví vlastnosti, které *CSV on the Web* nespecifikuje (nepočítaje společné vlastnosti).

F10.3 Validátor musí vytvořit varování v případě, kdy má vlastnost nevalidní hodnotu. Má-li vlastnost defaultní hodnotu, použije se namísto té nevalidní. Pokud defaultní hodnotu nemá, musí se validátor chovat, jako kdyby vlastnost nebyla definována.

F11 Vytvoření anotovaného modelu. Validátor vytvoří anotovaný model tabulkových dat podle doporučení *CSV on the Web* – z tabulkových dat vznikne základ modelu, který se doplní anotacemi pomocí metadatového dokumentu.

F12 Validace anotovaného modelu. Po vytvoření anotovaného modelu musí validátor tento model validovat.

F12.1 Validátor vytvoří chybu, pokud metadatový dokument není kompatibilní s embedovanými metadaty vyextrahovanými ze souboru tabulkových dat.

F12.2 Validátor vytvoří chybu, pokud existuje více jak jeden řádek se stejným primárním klíčem.

F12.3 Validátor vytvoří chybu pro každý cízí klíč, pro který není referencovaná řádka unikátní v dané tabulce.

F12.4 Validátor vytvoří chybu pro každou chybu buňky z anotace *errors*. Tato anotace vzniká při parsování hodnot buněk podle Kapitoly 6.4 *Parsing Cells* článku *Model for Tabular Data and Metadata on the Web* [1].

F13 Konzolová aplikace. Validátor bude možné spouštět z příkazového řádku a zároveň bude schopen přijímat lokální soubory i vzdálené soubory definované pomocí IRI.

F14 Webová služba. Validátor se bude moci spouštět přes webovou službu, která bude přijímat URL validovaných souborů. Webová služba bude také podporovat hromadnou validaci více souborů.

F15 Webová aplikace. Bude implementována jednoduchá webová aplikace využívající validátor, která umožní nahrávat soubory nebo zadávat IRI souborů a následně na nich spouštět validaci.

F16 Lokalizace. Webová aplikace bude obsahovat podporu více jazyků. V aktuální verzi bude přeložena v českém a anglickém jazyce.

F17 Persistence výsledků. Validace, které byly spuštěny přes webovou službu nebo webovou aplikaci, budou ukládány do databáze, aby mohli být kdykoliv znovu zobrazeny.

F18 Výsledky validace ve formátu CSV. Výsledky validace bude možné reprezentovat ve formátu CSV.

F19 Výsledky validace ve formátu RDF. Výsledky validace bude možné reprezentovat ve formátu RDF.

5.1.2 Nefunkční požadavky

Mezi nefunkční požadavky patří:

N1 Programovací jazyk Java. Validátor má být implementovaný v programovacím jazyku Java.

N2 Architektura umožnující snadné rozšíření. Validátor nemusí obsahovat všechna validační pravidla dle *CSV on the Web*, ale jeho součástí musí být správně navržená architektura, která bude umožňovat jednoduché přidávání dalších pravidla.

N3 Jednoduché nasazení. Validátor bude možné rychle a jednoduše nasadit a spustit.

N4 Podpora IRI schémat. Validátoru stačí podporovat IRI se schématy HTTP, HTTPS a file.

N5 Formát tabulkových dat. Validátor nemusí umět parsovat tabulková data dle různých dialektů z Kapitoly 8 *Parsing Tabular Data* článku *Model for Tabular Data and Metadata on the Web* [1]. Stačí zpracovávat tabulková data ve formátu dle RFC 4180 a dle *CSV on the Web*, viz Kapitola 1.2.

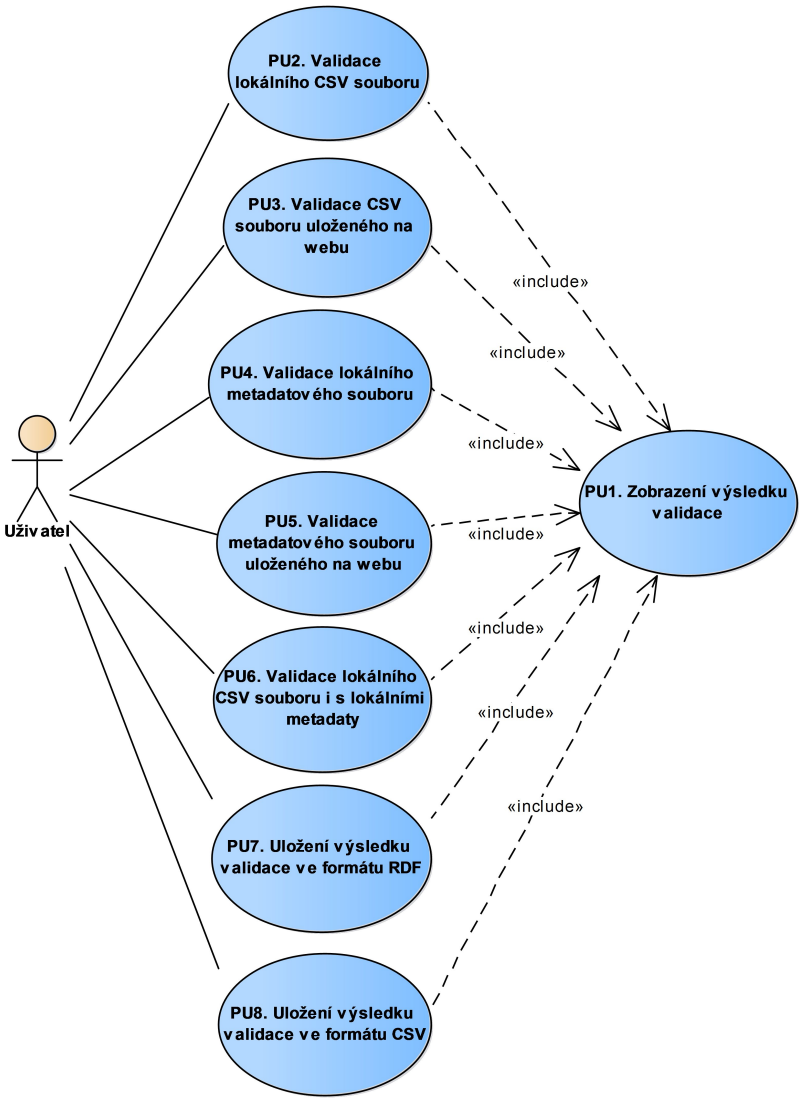
N6 Formát výsledků validace v CSV. Výsledky generované ve formátu CSV budou splňovat standard RFC 4180 [4].

N7 Formát výsledků validace v RDF. Výsledky generované ve formátu RDF budou v serializaci Turtle [29].

5.2 Model případů užití

Model případů užití webové aplikace se nachází na Obrázku 5.1 a pod ním jsou dále rozepsány jednotlivé scénáře těchto případů užití.

Jako aktér vystupuje ve webové aplikaci obecný **Uživatel**, neboť aplikace neobsahuje žádné role a ani její použití není omezeno např. přihlášením.



Obrázek 5.1: Model případů užití webové aplikace

PU1 Zobrazení výsledku validace.

Validátor dokončil validaci zadaných souborů a je možné si v aplikaci zobrazit výsledek validace.

PU2 Validace lokálního CSV souboru.

Uživatel chce validovat CSV soubor, který má uložený lokálně na svém počítači.

Základní cesta:

1 Uživatel spustí webovou aplikaci a vybere si validování lokálního CSV souboru.

2 Aplikace zobrazí formulář pro vložení CSV souboru.

3 Uživatel zvolí soubor k validaci a spustí ji.

4 Aplikace parsuje a validuje CSV soubor.

5 *Include*: Zobrazení výsledku validace

6 Aplikace přesměruje uživatele na stránku s výsledkem validace.

PU3 Validace CSV souboru uloženého na webu

Uživatel chce validovat CSV soubor identifikovatelný pomocí IRI.

Základní cesta:

1 Uživatel spustí webovou aplikaci a vybere si validování CSV souboru.

2 Aplikace zobrazí formulář pro vložení IRI metadatového souboru.

3 Uživatel vloží IRI a spustí validaci.

4 Aplikace stáhne CSV soubor z IRI.

5 Aplikace parsuje a validuje CSV soubor.

6 Aplikace lokalizuje kompatibilní metadatový soubor pro tabulková data.

7 Aplikace parsuje a validuje metadatový soubor.

8 Aplikace vytvoří a validuje anotovaný model tabulkových dat.

9 *Include*: Zobrazení výsledku validace

10 Aplikace přesměruje uživatele na stránku s výsledkem validace.

PU4 Validace lokálního metadatového souboru

Uživatel chce validovat metadatový soubor, který má uložený lokálně na svém počítači.

Základní cesta:

1 Uživatel spustí webovou aplikaci a vybere si validování lokálního metadatového souboru.

2 Aplikace zobrazí formulář pro vložení metadatového souboru.

3 Uživatel zvolí soubor k validaci a validaci spustí.

4 Aplikace parsuje a validuje metadatový soubor soubor.

5 *Include*: Zobrazení výsledku validace

6 Aplikace přesměruje uživatele na stránku s výsledkem validace.

PU5 Validace metadatového souboru uloženého na webu

Uživatel chce validovat metadatový soubor identifikovatelný pomocí IRI.

Základní cesta:

1 Uživatel spustí webovou aplikaci a vybere si validování metadatového souboru.

2 Aplikace zobrazí formulář pro vložení IRI metadatového souboru.

3 Uživatel vloží IRI a spustí validaci.

4 Aplikace stáhne metadatový soubor z IRI.

5 Aplikace parsuje a validuje metadatový soubor.

6 Aplikace stáhne tabulková data, která jsou popsána v metadatovém souboru.

7 Aplikace parsuje a validuje tabulková data.

8 Aplikace vytvoří a validuje anotovaný model tabulkových dat.

9 *Include*: Zobrazení výsledku validace

10 Aplikace přesměruje uživatele na stránku s výsledkem validace.

PU6 Validace lokálního CSV souboru i s lokálními metadaty

Uživatel chce validovat lokální CSV soubor s lokálním metadatovým souborem.

Základní cesta:

1 Uživatel spustí webovou aplikaci a vybere si validování lokálních souborů.

2 Aplikace zobrazí formulář pro vložení CSV i metadatového souboru.

3 Uživatel vloží oba soubory a spustí validaci.

4 Aplikace parsuje a validuje CSV soubor.

5 Aplikace parsuje a validuje metadatový soubor.

6 Aplikace porovná, zda jsou tabulková data a metadata kompatibilní.

7 Aplikace vytvoří a validuje anotovaný model tabulkových dat.

8 *Include*: Zobrazení výsledku validace

9 Aplikace přesměruje uživatele na stránku s výsledkem validace.

PU7 Uložení výsledku validace ve formátu RDF

Uživatel si zobrazí výsledek validace, který si následně uloží jako soubor ve formátu RDF.

PU8 Uložení výsledku validace ve formátu CSV

Uživatel si zobrazí výsledek validace, který si následně uloží jako soubor ve formátu CSV.

Kapitola 6

Návrh a implementace

Kapitola se zaměřuje na popis návrhu i implementace validátoru. Nejdříve jsou uvedeny technologie použité k implementaci, následně je představena architektura validátoru a jeho rozdělení do jednotlivých komponent a nakonec je popsána struktura projektu včetně nejdůležitějších tříd a rozhraní.

6.1 Použité technologie

Pro realizaci bylo třeba využít programovacího jazyku Java. Použita byla verze 8, kterou lze aktuálně nazývat standardem pro tvorbu Java aplikací a knihoven. Výhodou Java jazyku je velký počet a různorodost *open source* knihoven, které lze použít ať již pro rychlejší a pohodlnější programování, tak i pro funkčnost validátoru jako např. parsování CSV a JSON souborů.

Jazyk Java je velmi „upovídaný“ a rutinní např. při generování tzv. *getterů* a *setterů* či při definování metod `hashCode` a `equals`. Z tohoto důvodu je použita knihovna Project Lombok [30], která práci v jazyku Java urychluje. Místo zbytečného generování kódu lze používat jednoduché Java anotace, které jsou při kompilaci automaticky nahrazeny rutinním kódem. Příkladem je třídní anotace `@Data`, která pro třídu automaticky vygeneruje *gettery* (pro všechny atributy), *settery* (pro všechny *non-final* atributy), konstruktor pro *final* atributy a metody `equals`, `hashCode` a `toString`. [30]

Spring Framework [31] byl použit kvůli jeho implementaci *Inversion of Control* (též *dependency injection*), což je proces, v němž si jednotlivé objekty určí závislosti na jiných objektech. Tyto závislosti pak zaručí dosazení správného objektu při jeho vytvoření. Spring Boot [32] umožňuje vytvořit z validátoru samostatnou serverovou aplikaci, která je spuštěna na vnořeném serveru (např. Tomcat), bez nutnosti nasazení *Web application Archive* (WAR) souboru. Spring se také používá pro vytvoření REST webové služby.

Pro persistenci výsledků validací bylo nutné použít databázi. Vybrána byla databáze H2 Database Engine [33], neboť se jedná o jednoduchou a rychlou Java SQL databázi. Výhodou je snadná integrace se Spring Boot aplikací za

využití Spring Data JPA. Databázi H2 lze spustit v klasickém režimu, kdy je pro databázi vytvořen soubor na disku, a nebo v *embedded* módu, kdy se databáze uchovává jen v operační paměti. [33]

Spring Boot lze spojit s knihovnou Vaadin [34], jež slouží k vytváření uživatelského rozhraní pro webové aplikace. Uživatelské rozhraní vzniká za pomoci jazyku Java, který je knihovnou Vaadin automaticky přetransformován do jazyku JavaScript. [34]

K testování jednotlivých částí kódu (tzv. *unit* testování) i k integračním testům byla použita knihovna Spock [35]. Spock je výkonná alternativa k tradiční testovací knihovně jUnit. Hlavním rozdílem je využití jazyku Groovy, jenž je jako Java jazyk postavený na JVM (*Java Virtual Machine*), tudíž jsou tyto dva jazyky interoperabilní. Možnosti jazyku Groovy a knihovny Spock celkově nabízí velké zjednodušení testů a především parametrických nebo-li *data driven* testů.

O sestavení aplikace a o provázání jednotlivých knihoven se stará Apache Maven [36]. Maven je tzv. *build tool*, tedy nástroj pro usnadnění kompilování a sestavení velkých aplikací. Základem je soubor `pom.xml`, ve kterém jsou definovány základní údaje o projektu např. jméno nebo verze. Dále se v tomto souboru určí závislosti na ostatních knihovnách. Maven při sestavování aplikace tyto knihovny stáhne ze svého repositáře a importuje do projektu.

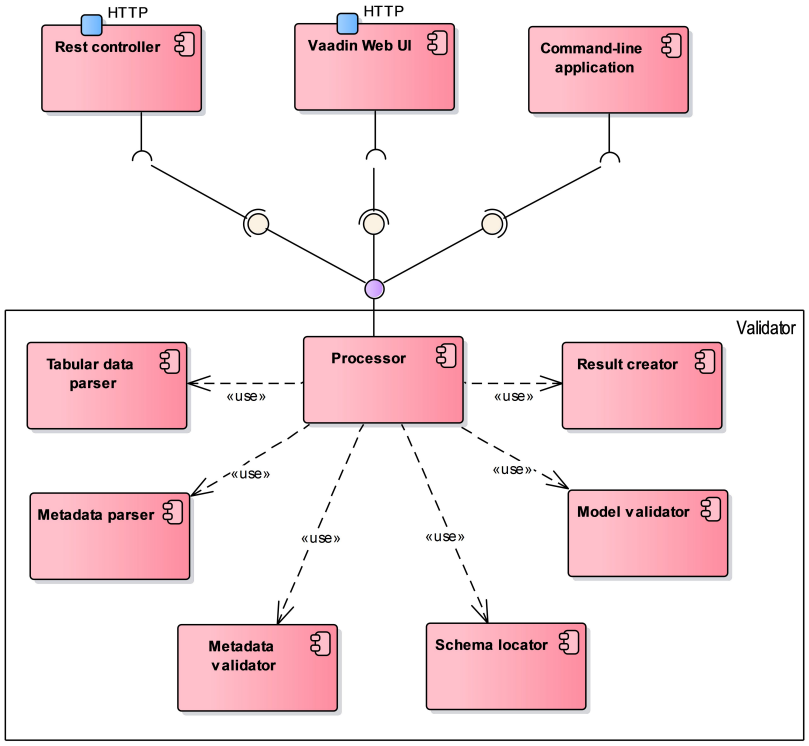
Projekt byl vyvíjen v programu IntelliJ IDEA [37] od firmy JetBrains, jelikož se jedná o výborné vývojové prostředí pro projekty v jazyku Java a obsahuje podporu Spring Framework, Maven, Groovy či verzování pomocí systému správy verzí GIT, který byl použit k zálohování projektu. Repositář se zdrojovým kódem validátoru je uložen na stránce vývojové platformy GitHub, viz `https://github.com/malyvoj3/csvw-validator`.

6.2 Architektura

V Kapitole 5 byla práce validátoru rozdělena do jednotlivých kroků. Validační chyby nebo varování mohou vzniknout v kterémkoliv z uvedených kroků. V některých případech validační chyba nebrání dalšímu zpracování, ale jindy je potřeba proces validace zastavit a již nepokračovat navazujícím bodem.

O každý z definovaných kroků by se měla starat samostatná komponenta, neboť se jedná o nezávislé logické celky. Na Obrázku 6.1 lze vidět logické rozdělení systému do komponent.

Hlavní částí validátoru je *Processor*, který slouží jako návrhový vzor Fasáda. Obsahuje několik komponent rozdělených podle principu *single responsibility*, který říká, že každá komponenta by měla mít jen jednu funkci. Tyto jednotlivé funkce dohromady vytváří celou logiku validátoru, přičemž provázání mezi nimi je komplexní. Fasáda zajišťuje orchestraci těchto komponent a navenek nabídne jen zjednodušené rozhraní.



Obrázek 6.1: Architektura validátoru obsahující základní komponenty

Již bylo řečeno, že jednotlivé komponenty by měly dodržovat *single responsibility* princip. To však v této architektuře není úplně zaručeno. Bohužel v některých případech je nutné omezit architekturu programu vzhledem k jeho efektivitě. Příkladem by byla komponenta *Model creator*, která by sloužila k vytvoření anotovaného modelu z Kapitoly 3.2. Takto vzniklý model by se následně předal komponentě *Model validator*, která by celý model validovala. Udržovat si však v operační paměti celý model je paměťově neefektivní a u velkých souborů by to mohlo vést k problémům s velikostí haldy. Z tohoto důvodu je zapotřebí validovat CSV soubory postupně po řádcích, které budou postupně validovány.

Na tomto postupné zpracování jsou založeny komponenty *CSV parser* a *Model validator*. *CSV parser* má na starosti parsování a validaci CSV souborů. Validuje se např. zda je soubor validní, zda neobsahuje špatný formát CSV, aj. Přečtená data jsou dále uložena do dočasného souboru, který se využívá při validace modelu v komponentě *Model validator*. Komponenta *Model validator* kontroluje datové typy buněk, primární klíče, ad.

Schema locator je použit, pokud validace začíná jen se souborem tabulkových dat a během ní je potřeba najít metadatový soubor se schématem. Komponenta využívá postupy popsané v kapitole 3.4.1.

Parsování metadatových souborů je hlavní funkcionalitou komponenty *Metadata parser*, která mimo parsování také validuje formát JSON-LD souborů. Jedná se např. o názvy JSON vlastností a jejich povolené typy, neznámé JSON vlastnosti, aj. Chceme-li validovat metadatový soubor s větší logikou, např. zda virtuální sloupce následují až po těch nevirtuálních, využijeme *Metadata validator*. *Metadata validator* má již k dispozici celý JSON-LD deskriptor, který na rozdíl od tabulkových dat lze mít celý uložený v paměti.

Nakonec *Result creator* ze shromážděných dat a validačních chyb, získaných během celého procesu, vytvoří finální výsledek validátoru. Validační chyby jsou přeloženy do jazyka, který byl zvolen jako parametr validace.

Processor vystavuje jednoduché API pro spouštení validací, které může využít jak webová služba tvořená komponentou *Rest controller*, tak webové uživatelské rozhraní (nebo-li UI z angl. *user interface*) *Vaadin Web UI*, nebo jen *Konzolová aplikace*.

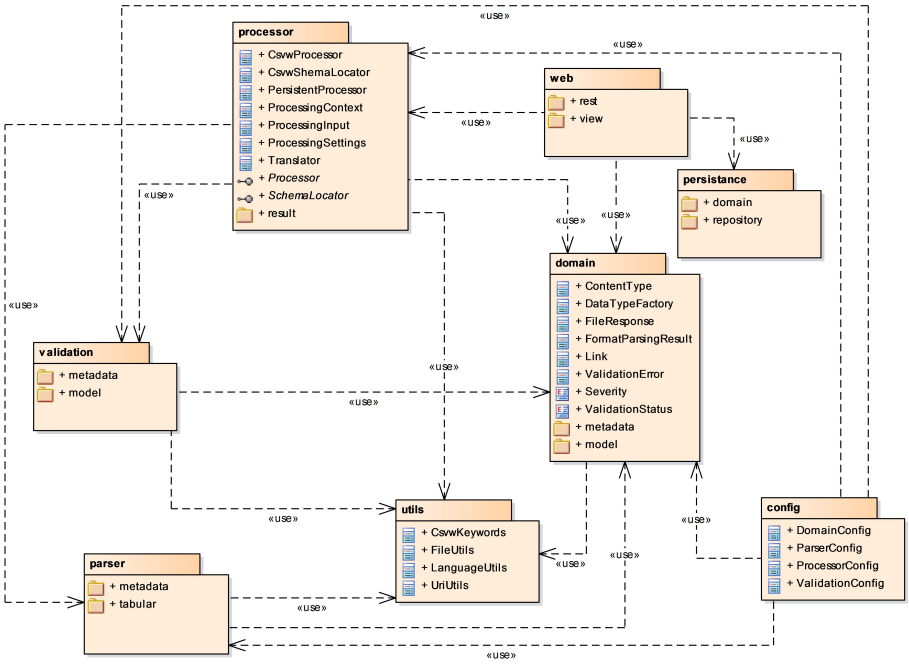
6.3 Struktura projektu

Vzhledem k velikosti projektu a jeho složení z několika logických celků, je vhodné rozdělit jednotlivé třídy do balíčků (tzv. *java packages*), jež seskupují třídy se společným záměrem. Výsledná struktura je zobrazena na Obrázku 6.2 včetně závislostí mezi jednotlivými balíčky.

Projekt je na nejvyšší úrovni tvořen balíčkem `com.malyvoj3.csvwvalidator`, který slouží jako jmenný prostor pro třídy zajišťující chod serverové i konzolové aplikace a pro těchto dalších osm vnořených balíčků:

- `config` – balíček obsahující konfigurace knihovny Spring, které definují tzv. *spring beans*. *Spring bean* je zjednodušeně třída s anotací `@Bean`, která je uložena ve Spring kontejneru a používá se pro již zmíněné *dependency injection*.
- `domain` – rozhraní a třídy definující základní objekty, tedy anotované tabulky, sloupce, popisující objekty a jejich vlastnosti.
- `parser` – implementace parserů tabulkových dat a metadat.
- `processor` – hlavní logika validátoru. Patří sem třídy tvořící komponenty *Schema locator*, *Result creator* a *Processor*.
- `repository` – rozhraní a třídy pro práci s databází.
- `utils` – balíček obsahující tzv. *utility* třídy, tedy třídy bez stavu, jejichž metody jsou statické a poskytují nějakou funkcionalitu skrz více jiných tříd.

- `validation` – definuje validační chyby, pravidla a validátory používající tato pravidla.
- `web` – implementace webové službu nebo webové aplikace.



Obrázek 6.2: Struktura projektu – rozdělení do balíčků

Kromě dělení projektu na balíčky je projekt rozdělen i do Maven modulů. Základem pro nástroj Apache Maven je soubor `pom.xml`. Těchto souborů lze však mít více, pokud využijeme modulární schopnosti knihovny Maven. V takovém případě budeme mít jeden hlavní `pom.xml` soubor pro rodičovský modul `csvw-validator`, který – jako hlavní modul – může mít libovolný počet tzv. submodulů s vlastním souborem `pom.xml`. Pro každý ze submodulů je při klasické Maven instalaci vygenerován samostatný archiv JAR (Java Archive) nebo WAR (Web Archive), což umožňuje používat každý modul samostatně.

Projekt je rozdělen do tří submodulů:

- `csvw-validator-lib` – modul obsahující kompletní logiku validátoru včetně parsování. Skládá se z balíčků `config`, `domain`, `parser`, `processor`, `utils` a `validation`.
- `csvw-validator-web-app` – modul zahrnující Spring Boot aplikaci, která obsahuje REST webovou službou a webovou aplikaci. Modul má tedy závislost na modulu `csvw-validator-lib`. Skládá se z balíčku `web`, `repository` a z třídy pro spuštění aplikačního serveru `CsvwValidatorWebApplication`. Navíc také rozšiřuje balíček `processor`.
- `csvw-validator-cli-app` – modul obsahující jen jednoduchou třídu `CsvwValidatorConsoleApplication`, která spouští validace z příkazového řádku. Modul je také závislý na modulu `csvw-validator-lib`.

6.4 Popis jednotlivých balíčků

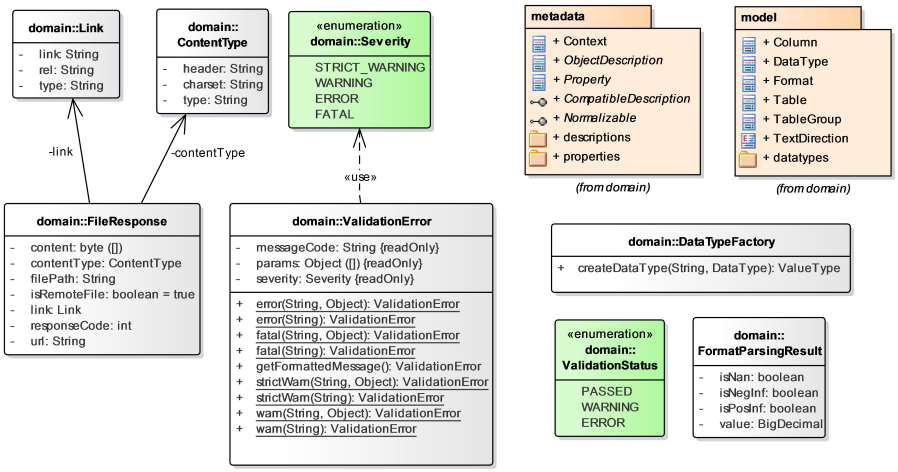
Nyní budou představeny jednotlivé balíčky projektu a jejich hlavní obsah. Je-li dále nějaký balíček pojmenován, používá se vždy jen název v rámci kontextu. Tedy balíček `model` v kapitole o balíčku `domain` bude znamenat `com.malyvoj3.csvwvalidator.domain.model` a podobně.

6.4.1 Balíček domain

Obsah balíčku `domain` je zobrazen na Obrázku 6.3, který obsahuje základní rozhraní a třídy pro práci validátoru. Kromě dvou vnořených balíčků `model` a `metadata` zde najdeme dva výčtové typy `ValidationStatus`, definující výsledek validátoru, a `Severity`, udávající vážnost chyby validátoru. Chybu validátoru představuje třída `ValidationError`, jež obsahuje textový řetězec popisující chybu a její severitu. `ValidationError` obsahuje vzor *static factory methods* – statické metody, které slouží k vytváření chyb s určitou severitou.

Třída `FileResponse` reprezentuje libovolný soubor s dalšími užitečnými atributy. Mezi tyto atributy patří např. `ContentType` a `Link`, které v sobě uchovávají stejnojmenné HTTP hlavičky v případě, že byl soubor stažen přes HTTP.

`DataTypeFactory` je třída implementující návrhový vzor Factory, kdy pro zadaný textový řetězec a definici datového typu dle Kapitoly 3.2.2 vytvoří správný objekt s datovým typem. `FormatParsingResult` se používá jako výsledek při parsování řetězce do numerického datového typu.



Obrázek 6.3: Obsah balíčku `domain`

6.4.1.1 Balíček model

V balíčku `model` lze nalézt třídy, které reprezentují část anotovaného modelu z Kapitoly 3.2. Jsou zde třídy pro tabulku (`Table`), sloupce (`Column`) i pro další komponenty. Kvůli častému vytváření objektů těchto tříd byl pro většinu z nich implementován návrhový vzor Builder. Implementace tohoto návrhového vzoru je díky knihovně Project Lombok velmi jednoduchá, neboť stačí přidat ke třídě anotaci `Builder` z Lombok knihovny. Složitějším balíčkem je až vnořený balíček `datatypes`, viz Obrázek 6.4.

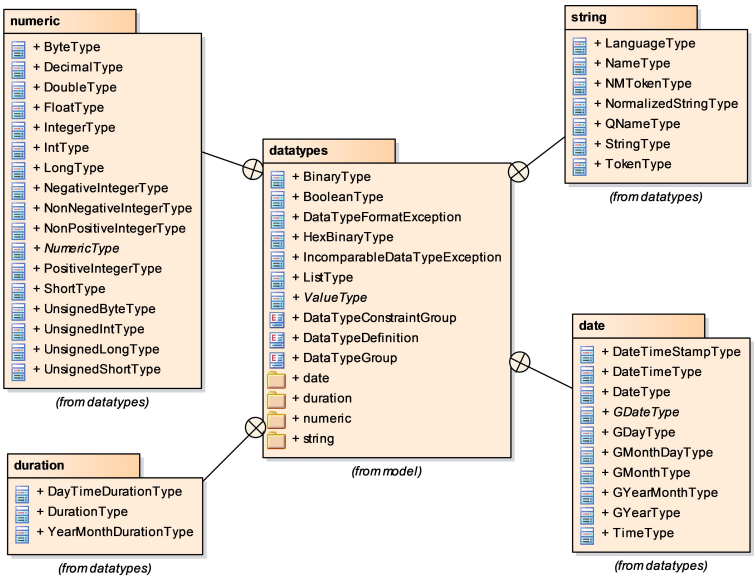
Balíček `datatypes` se zabývá datovými typy z Kapitoly 3.2.2 – reprezentací datových typů v programovacím jazyku Java, jejich vytvořením a podporou různých formátů.

Datové typy lze rozdělit do tři kategorií:

- datový typ obsahující hodnotu, kterou lze porovnávat, např. čísla;
- datový typ obsahující hodnotu, která má definovanou délku, např. textové řetězce;
- datový typ, který neobsahuje ani jedno z předchozích dvou, např. boolean.

Rozlišovat mezi těmito typy je potřeba dynamicky za běhu, neboť validátor předem neví, jakého typu buňky v tabulkových datech budou. Toho je docíleno

abstraktní třídou `ValueType`, která obsahuje metody pro všechny druhy datových typů. Ovšem až podtřídy třídy `ValueType`, implementující určitý datový typ, tyto funkce definují. Každá z těchto podtříd se stará i o správné vytvoření sémantické hodnoty buňky z textového řetězce, který se získal v buňce tabulky. Lze tedy říci, že každý datový typ, který je podporován validátorem, má svoji podtřídu rozšiřující třídu `ValueType`. Příklady jsou třídy `IntegerType` pro hodnoty typu `xsd:integer`, `DateType` pro hodnoty typu `xsd:date`, atd.

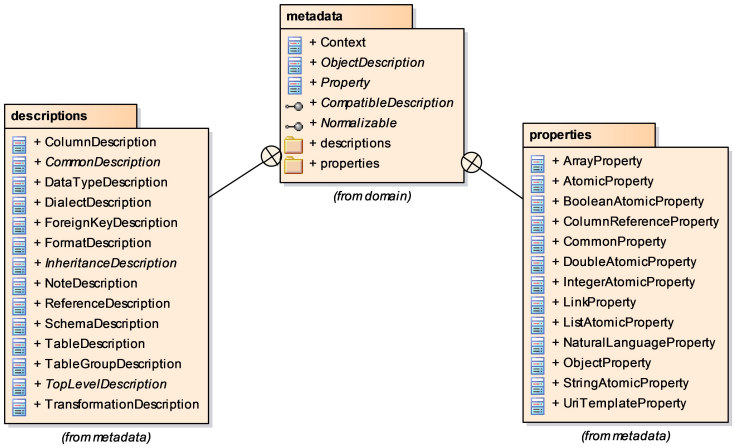


Obrázek 6.4: Balíček `domain.model.datatypes` a jeho vnořené balíčky

Výčtové typy `DataTypeConstraintGroup`, `DataTypeDefinition` a `DataTypeGroup` slouží k rozlišování typů a druhů datových typů. Výjimky `DataTypeFormatException` a `IncomparableDataTypeException` jsou kontrolované výjimky, které jsou vytvořeny, pokud z textové řetězce a formátu datového typu nelze vytvořit sémantickou hodnotu, resp. pokud hodnotu neporovnatelného datového typu chceme porovnávat.

6.4.1.2 Balíček metadata

V balíčku `metadata` se definují třídy pro metadatové vlastnosti a popisující objekty podle Kapitoly 3.3, z nichž se skládají soubory s metadaty. Na Obrázku 6.5 lze vidět obsah tohoto balíčku.



Obrázek 6.5: Balíček `domain.metadata` a jeho vnořené balíčky

Třída `Context` definuje kontext podle Kapitoly 3.3.3. Tato třída je následně použita v rozhraní `Normalizable`, jež definuje objekty, které lze normalizovat dle Kapitoly 3.4.3 za pomocí metody `normalize(Context context)`.

Abstraktní třída `Property` implementuje rozhraní `Normalizable` a reprezentuje JSON vlastnost. V balíčku `properties` jsou podtřídy třídy `Property` a každá z těchto podtříd představuje jeden typ metadatových vlastností z Kapitoly 3.3.1. Příkladem může být třída `ObjectProperty`, která reprezentuje objektovou vlastnost a zároveň obsahuje metodu pro korektní normalizaci objektové vlastnosti.

Následující abstraktní třída `ObjectDescription` také implementuje rozhraní `Normalizable` a reprezentuje popisující objekt z Kapitoly 3.3. Rozhraní `CompatibleDescription` definuje popisující objekty, u nichž je potřeba zjišťovat, zda jsou kompatibilní s jiným popisujícím objektem stejného typu. Kompatibilitu lze zajistit implementací funkce `isCompatibleWith(T other)`, kde `T` je generický objekt typu `ObjectDescription`.

Konkrétní popisující objekty, jako např. `ColumnDescription` pro popisující objekt sloupce, jdou definovány v balíčku `descriptions`. Kromě konkrétních popisujících objektů se zde nacházejí i abstraktní předci. `CommonDescription` přidává podporu pro společné vlastnosti, `InheritanceDescription` obsahuje dědičné vlastnosti a `TopLevelDescription` je předkem pro hlavní popisující objekty – tabulku či skupinu tabulek.

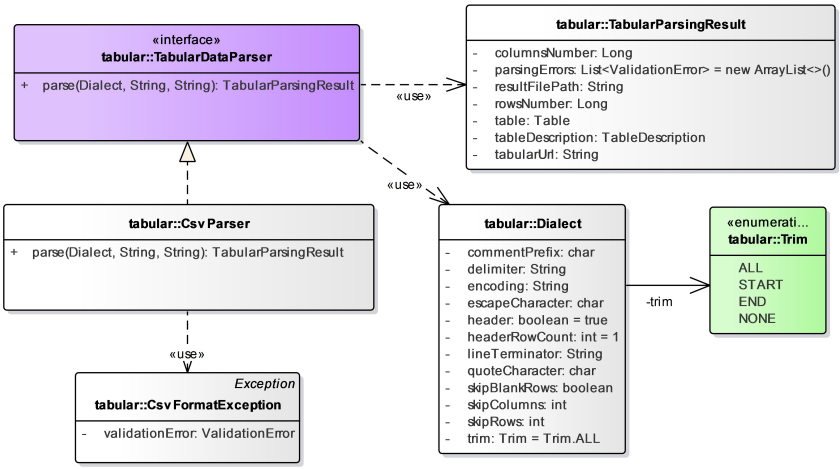
6.4.2 Balíček parser

V následující části je postupně popsán obsah balíčku `parser`, jež obsahuje třídy zabývající se parsováním tabulkových a metadatových souborů.

6.4.2.1 Balíček tabular

Balíček `tabular` obsahuje vše, co je potřebné k parsování a validaci souborů tabulkových dat. Na Obrázku 6.6 jsou zobrazeny třídy z balíčku. Třída `Dialect` reprezentuje dialekt pro parsování různých formátů tabulkových dat. Hlavní část balíčku tvoří rozhraní `TabularParser` a jeho zatím jediná implementace `CsvParser`. Rozhraní definuje metodu `parse()`, jež slouží ke čtení a validování tabulkových dat. Výsledkem této metody je objekt třídy `TabularParsingResult`, jež obsahuje chyby, které při parsování nastaly, URL adresu parsovaného souboru a popisující objekt tabulky, tedy tzv. embedovaná metadata získaná dle algoritmu z Kapitoly 8 *Parsing Tabular Data* článku *Model for Tabular Data and Metadata on the Web* [1].

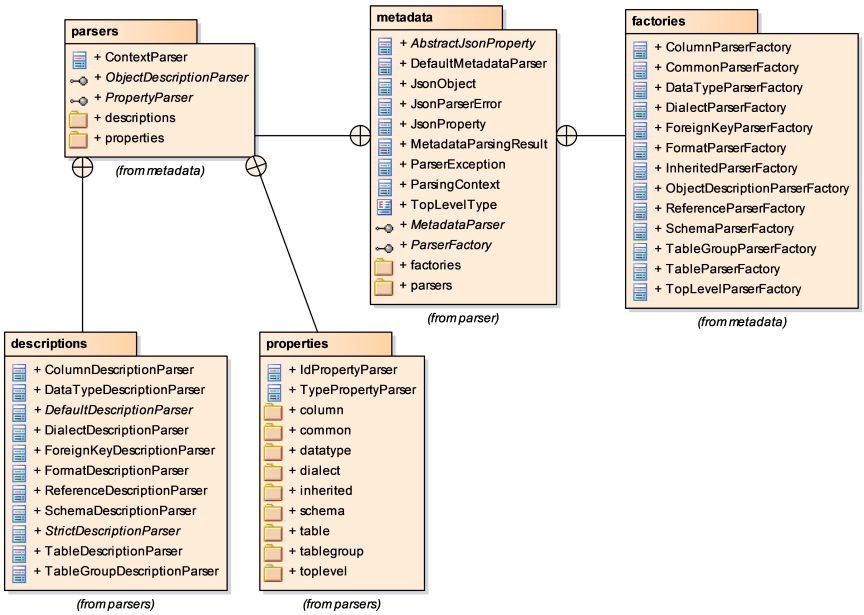
`CsvParser` slouží k parsování tabulkových data ve formátech z Kapitoly 1.2 a vůbec tedy nepoužívá dialekt. Dokáže však rozeznat i jiné oddělovače buněk, než je jen čárka. Tato schopnost je zajištěna použitím knihovny univocity-parsers [38], která dokáže nejen parsovat CSV soubory ve formátu RFC 4180 a *CSV on the Web*, ale umí i automaticky rozpoznávat různé konce řádků a oddělovače buněk. Parsování navíc probíhá rychleji než s nejznámější konkurenční knihovnou Apache Commons CSV [39].



Obrázek 6.6: Obsah balíčku `parser.tabular`

6.4.2.2 Balíček metadata

Tento balíček, zabývající se parsováním metadatových souborů, je velmi rozsáhlý. Základem je knihovna Jackson [40] – nejrozšířenější Java knihovna pro prací s formátem JSON. Vzhledem k formátu metadatových souborů by bylo lepší použít knihovnu podporující formát JSON-LD. Bohužel referenční JSON-LD knihovna je teprve ve verzi 0.12.3 a zdaleka nepodporuje funkcionality, které jsou pro práci potřeba.



Obrázek 6.7: Balíček `parser.metadata` a jeho vnořené balíčky

Hlavní proces parsování je definován v rozhraní `MetadataParser` a implementován v třídě `DefaultMetadataParser`. Tato třída používá knihovnu Jackson k uložení celého metadatového souboru do objektu typu `JsonNode`. `JsonNode` je abstraktní typ, z něhož dědí např. `ObjectNode` pro JSON objekty, `ArrayNode` pro JSON pole, `TextNode` pro řetězce atd. Následně musí parser rozeznat, zda získaný objekt je typu `ObjectNode` a zda je to objekt popisující tabulku nebo skupinu tabulek. Podle jeho rozhodnutí se zavolá metoda `parse(objectNode)` třídy `TableDescriptionParser` či `TableGroupDescriptionParser` vracející `TableDescription` respektive `TableGroupDescription`.

Nyní budou představena tři důležitá rozhraní. `PropertyParser` definuje

parsování jedné vlastnosti popisujícího objektu, který je reprezentován třídou `ObjectDescription`. Ukázka implementace tohoto rozhraní – třídy `UrlPropertyParser` – je na Příkladu 6.1. Třída se snaží z objektu `JsonNode` vytvořit objekt reprezentující vlastnost *url* z popisujícího objektu tabulky. Pokud hodnota v JSON souboru není textového typu, musí parser vytvořit varování o nevalidním typu. Jednotlivé implementace pro každou vlastnost jsou uloženy ve vnořeném balíčku `properties`.

Příklad 6.1: Ukázka z třídy `UrlPropertyParser`

```
@Override
public void parsePropertyToDescription(@NonNull T
   description, @NonNull JsonProperty jsonProperty) {
    JsonNode property = jsonProperty.getJsonValue();
    LinkProperty url;
    if (property.isTextual()) {
      url = new LinkProperty(property.textValue());
    } else {
      jsonProperty.addError(
        JsonParserError
          .invalidPropertyType(jsonProperty.getName())
      );
      url = null;
    }
    description.setUrl(url);
}
```

Druhým důležitým rozhraním je `ParserFactory`, které slouží k získání `ProperyParser` z názvu JSON vlastnosti a z generického typu rozšiřujícího `ObjectDescription`. Jedná se znovu o návrhový vzor Factory a na Příkladu 6.2 je zobrazena implementace daného rozhraní ve třídě `ObjectDescriptionParserFactory`.

Poslední rozhraní `ObjectDescriptionParser` slouží k parsování celého popisujícího objektu. Jeho logikou by měl být průchod přes všechny vlastnosti zadaného JSON objektu a pomocí `ParserFactory` získat k vlastnostem ekvivalentní `PropertyParser`. Zavoláním všech takto získaných `PropertyParser` vznikne finální popisující objekt. Jednotlivé implementace pro všechny popisující objekty jsou uloženy ve vnořeném balíčku `descriptions`.

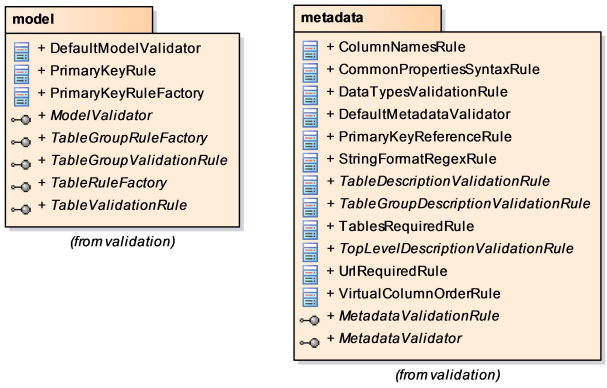
Kombinací těchto tří rozhraní získáme možnost jednoduše přidávat do aplikace podporu jak pro další vlastnosti, tak i pro popisující objekty.

Příklad 6.2: Ukázka z třídy `ObjectDescriptionParserFactory`

```
public PropertyParser<T> createParser(String
    propertyName) {
  switch (propertyName) {
    case CsvwKeywords.ID_PROPERTY:
      return new IdPropertyParser<>();
    case CsvwKeywords.TYPE_PROPERTY:
      return new TypePropertyParser<>();
    default:
      return null;
  }
}
```

6.4.3 Balíček validation

Balíček `validation` má na starosti validace dle *CSV on the Web*, které lze provádět již na celém metadatovém schématu (vnořený balíček `metadata`) nebo na vytvořeném anotovaném modelu (vnořený balíček `model`).



Obrázek 6.8: Obsah balíčku `validation`

6.4.3.1 Balíček metadata

Balíček `metadata` obsahuje rozhraní `MetadataValidationRule` a jeho abstraktní implementace `TablepDescriptionValidationRule` a `TableGroupDescriptionValidationRule`, které představují pravidlo pro validování po-

pisujícího objektu tabulky resp. skupiny tabulek. Z těchto abstraktních tříd dědí již opravdová validační pravidla – `UrlRequiredRule` kontrolující, zda popis tabulky obsahuje vlastnost url, či `DataTypesValidationRule` validující datové typy.

Validace lze spustit zavoláním metod na rozhraní `MetadataValidator`. Rozhraní obsahuje metody pro validaci popisu tabulky a validaci popisu skupiny tabulek. Jeho implementace `DefaultMetadaValidator` využívá knihovnu Spring, neboť všechna validační pravidla pro popis tabulek a skupiny tabulek získá do atributu třídy jen pomocí anotace `@Autowired`. Zjednodušeně lze říci, že anotace atributu `@Autowired` dosadí do daného atributu správnou referenci – *spring bean*, jež je stejného datového typu. Je-li atributem listem, pak jsou do něj vloženy všechny nalezené *spring beans*. [31] Pokud tedy chceme přidat další validační pravidlo pro metadata, stačí implementovat novou třídu dědící z `TableGroupDescriptionValidationRule` nebo `TablepDescriptionValidationRule`, přidat ji do konfigurace jako *spring bean* a validátor metadat si ji automaticky přidá mezi validační pravidla.

6.4.3.2 Balíček model

Balíček `model` je odlišný od balíčku `metadata`, neboť anotovaný model nelze mít v paměti nahraný celý. Validace tedy musí probíhat po jednotlivých řádcích. Rozhraní `ModelValidator` obsahuje metody pro validaci tabulky nebo skupiny tabulek. Implementace tohoto rozhraní `DefaulModelValidator` taktéž využívá anotace `@Autowired` z knihovny Spring, pomocí které získá všechny implementované *TableRuleFactory* a *TableGroupRuleFactory*. Tyto Factory třídy vytvoří pro aktuální validaci pravidla, které se provolávají pro každý řádek. Na konci se z těchto pravidla získají výsledky. Ukázkou implementace konkrétního pravidla je např. `PrimaryKeyRule` – pravidlo validující unikátnost primárních klíčů.

6.4.4 Balíček utils

V balíčku utils nalezneme již řečené *utility* třídy. Pro vytvoření takovéto třídy stačí přidat nad třídu Java anotaci `@UtilityClass` z knihovny Lombok.

Balíček obsahuje celkem čtyři třídy. `FileUtils` dodává funkcionalitu pro práci se soubory. Obsahuje metodu pro stažení souboru pomocí IRI (s HTTP/HTTPS či file schématem) a metodu ověřující to, zda je soubor zakódován v kódování UTF-8.

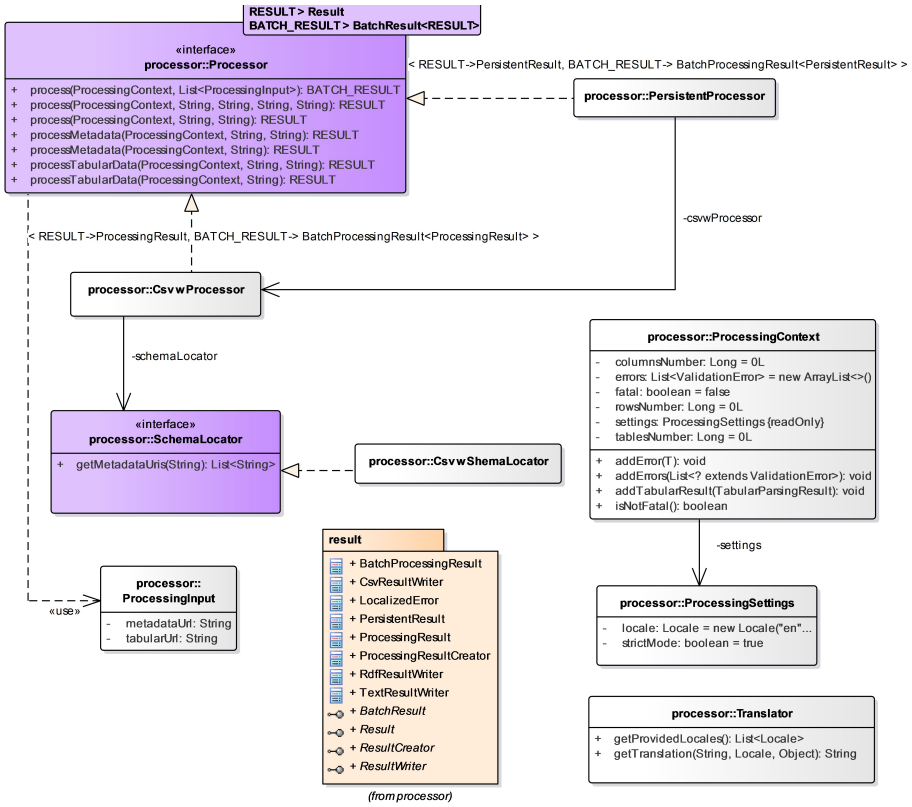
Třída `LanguageUtils` se zabývá jazykovými kódy dle BCP47 [20]. Umožňuje validovat vložený textový řetězec a potvrdit, zda je platným jazykovým kódem. Dále obsahuje metodu pro zkracování a následné porovnání jazykových kódů.

`CsvwKeywords` má v sobě uloženy jen konstanty pro klíčová slova (např. názvy výrazů), která jsou definována doporučením *CSV on the Web*.

Poslední a největší utility třídou je `UriUtils`, která se zabývá validací, normalizací a porovnáváním IRI a překladem relativních IRI na absolutní podle standardu RFC 3968 [14]. Jazyk Java obsahuje třídu `java.net.URI` pro práci s URI, ale tato referenční implementace podporuje jen starý standard pro URI RFC 2396. Práci s IRI je usnadněna díky použití knihovny urllib [41]. Dále `UriUtils` obsahuje metodu pro rozpoznání společné vlastnosti, přeložení společné vlastnosti na absolutní IRI, a rozšíření URI template dle RFC 6570 [21]. Pro práci s URI template je použita knihovna Handy URI Templates [42].

6.4.5 Balíček processor

Balíček `processor` tvoří hlavní logiku validátoru. Jeho obsah lze vidět na Obrázku 6.9.



Obrázek 6.9: Obsah balíčku `processor`

Vnořený balíček `result` má za cíl definovat a reprezentovat výsledky validace. Výsledek validace je definován rozhraním `Result`, které obsahuje jen metodu pro získání výsledku validace `ValidationStatus`. Výsledek hromadné validace definuje rozhraní `BatchResult`, obsahující list objektů implementujících rozhraní `Result`. Rozhraní `ResultCreator` definuje metody pro vytváření objektů typu `Result` a `BatchResult`. V knihovním modulu je použito jen základních implementací těchto rozhraní a to konkrétně `ProcessingResult`, `BatchProcessingResult` a `ProcessingResultCreator`. V modulu webové aplikace a webové služby je navíc přidána implementace `PersistentResult`, která rozšiřuje *ProcessingResult* o identifikátor výsledku.

`ResultWriter` je rozhraní, jež definuje zápis výsledku do souboru (pole bytů). Jak a v jakém formátu se výsledek zapíše, je určeno až v jeho implementacích. Aktuálně existují tři implementace:

- `CsvResultWriter` – ukládá výsledek do formátu CSV dle RFC 4180 [4]. Tento soubor obsahuje dva sloupce – *Error severity* (závažnost chyby) a *Error message* (popis chyby).
- `RdfResultWriter` – ukládá výsledek do formátu RDF v serializaci Turtle. Pro práci s RDF je použita knihovna Apache Jena [43]. K popisu výsledků validace je použit slovník *Evaluation and Report Language* (EARL) [44].
- `TextResultWriter` – ukládá výsledek v textovém formátu, viz Příklad 6.3.

Příklad 6.3: Výsledek zapsaný pomocí `TextResultWriter`

```
Tabular URL:
   http://www.w3.org/2013/csvw/tests/test286.csv
Metadata URL:
   http://www.w3.org/2013/csvw/tests/test286-
   metadata.json
Result: ERROR
Strict mode: false
Total errors: 1
Warning errors: 0
Error errors: 1
Fatal errors: 0
Errors:
  ERROR: Cell (row 2 column 1) cannot be formatted as
     'integer' datatype.
```

Rozhraní `Processor` definuje sedm metod pro spouštění validace tabulkových dat a/nebo metadat. Výsledkem validace je generický typ implementující `Result` (v případě hromadné validace `BatchResult`). Základní implementací je třída `CsvwProcessor`, která pro výsledky používá třídy `ProcessingResult` a `BatchProcessingResult`. Tato třída dodržuje návrhový vzor Fasáda

– komplexní logiku mezi několika komponentami obaluje navenek do jednoduchého rozhraní.

Logika zpracování ve třídě `CsvwProcessor` záleží na zvolené metodě. Stačí si však popsat jen případ, kdy validujeme soubor tabulkových dat, neboť ostatní případy si jsou již velmi podobné.

Je-li vstupem validace pouhé IRI souboru tabulkových dat, je tento soubor stažen pomocí `utility` třídy `FileUtils`. Stažený soubor se přečte a vyextrahují se z něj vnořená metadata pomocí `TabularDataParser`.

Následně je potřeba pro tabulková data najít jejich schéma podle Kapitoly 3.4.1. Tuto funkcionalitu definuje rozhraní `SchemaLocator`, které implementuje `CsvwSchemaLocator`, vracející seznam IRI, na nichž se schéma může vyskytovat. Z těchto adres následně `CsvwProcessor` zkusí stáhnout JSON-LD deskriptor, parsovat a normalizovat jej. Z takhle vzniklých metadat se hledají taková, která jsou kompatibilní s vyextrahovanými vnořenými metadaty.

Pokud validátor získal kompatibilní soubor s metadaty, je potřeba jej validovat. O validaci se stará `MetadataValidator` z balíčku `validation`, viz Kapitola 6.4.3.

Máme-li zkontrolovány data ze souboru tabulkových dat a i metadatový soubor popisující schéma tabulkových dat, začne se postupně validovat anotovaný model za použití `ModelValidator` z balíčku `validation`.

V každém z těchto kroků se mohou objevit validační chyby, které se postupně sbírají a které na konci procesu tvoří výslednou množinu validačních chyb. Seznam těchto chyb je poslán třídě `ResultCreator`, která z nich vytvoří konečný výsledek validace.

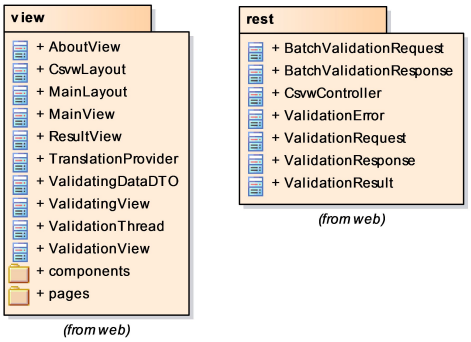
Pro modul s webovou službou a aplikací vznikla nová implementace rozhraní `Processor` – `PersistentProcessor`. Tato třída tvoří návrhový vzor Proxy, neboť implementuje rozhraní `Processor`, ale obsahuje v sobě *CsvwProcessor*, na který přesměrovává volání. Výsledky z *CsvwProcessor* zachycuje a ukládá je do databáze pomocí rozhraní `ResultRepository` – viz následující Kapitola 6.4.6.

6.4.6 Balíček repository

Balíček `repository` obsahuje třídy spjaté s databází. Nachází se zde objekt `ResultEntity` – JPA (*Java Persistence API*) entita pro objektově-relační mapování výsledku validace. Tento objekt čteme z databáze a zapisujeme do ní pomocí rozhraní `ResultRepository`.

6.4.7 Balíček web

Balíček `web` vystavuje navenek rozhraní, přes která lze spouštět validace. Jak lze vidět na Obrázku 6.10, tak se rozděluje na balíčky `rest`, implementující REST webovou službu a balíček `view`, který implementuje webovou aplikace.



Obrázek 6.10: Obsah balíčku `web`

6.4.7.1 Balíček rest

Ve vnořeném balíčku `rest` jsou především třídy definující tzv. *data transfer object* (DTO) – objekty pro přenos dat mezi procesy (např. mezi webovými službami). Tyto třídy se serializují do formátu JSON a jsou posílány skrze webovou službu. Hlavní logika je uložena ve třídě `CsvwController`, která zjednodušeně vypadá jako na Příkladu 6.4.

Nechť máme aplikaci spuštěnou pomocí knihovny Spring Boot, poté anotace `@RestController` z knihovny Spring zjednodušeně říká, aby *singleton* (česky jedináček) této třídy poslouchal HTTP požadavky a zavolal příslušnou metodu podle druhu požadavku .

HTTP požadavek typu `POST` na adrese `/validate` volá metodu `validate()`. Tato metoda obsahuje anotaci `PostMapping`, která říká, že metoda je zavolána při POST HTTP požadavku. Proměnné této anotace specifikují parametry požadavku. Proměnná `path` určuje cestu URL, `consumes` a `produces` nastavují HTTP hlavičky `accept` a `content-type`.

V metodě se již pracuje s Java třídami, neboť Spring automaticky deserializuje a serializuje objekty z/do formátu JSON. Rozhraní webové služby je popsáno v dokumentaci, viz Kapitola 8.

Příklad 6.4: Implementace třídy `CsvwController`

```
@RestController
public class CsvwController {

  @Autowired
  private final CsvwProcessor csvwProcessor;

  @PostMapping(path = "/validate",
      consumes = MediaType.APPLICATION_JSON_VALUE,
      produces = MediaType.APPLICATION_JSON_VALUE)
  public ResponseEntity<ValidationResponse> validate(
    @RequestBody ValidationRequest request) {
    // logic
  }

  @PostMapping(path = "/validateBatch",
      consumes = MediaType.APPLICATION_JSON_VALUE,
      produces = MediaType.APPLICATION_JSON_VALUE)
  public ResponseEntity<BatchValidationResponse>
     validateBatch(
    @RequestBody BatchValidationRequest request) {
    // logic
  }
}
```

6.4.7.2 Balíček web

Balíček `view` používá knihovnu Vaadin k vytvoření webového UI (tzv. frontendu aplikace). Tato knihovna umožňuje psát fronted aplikace přímo v jazyku Java, není tedy nutné řešit např. komunikaci. Použitím běžného frontedu, napsaného nejběžněji v jazyce JavaScript, by bylo nutné řešit komunikaci takové aplikace s webovou službou. Jelikož je použita knihovna Vaadin přímo v jazyku Java, nemusí komunikace probíhat skrz webovou službu, ale stačí volat rovnou metody libovolných tříd, viz jednoduchý Příklad 6.5.

V tomto příkladu je vytvářeno tlačítko pomocí třídy `Button`, které obsahuje text `Validate`, jenž byl předán v konstruktoru. Následně je přidána akce, které se zavolá při kliknutí na tlačítko. Právě tato akce je definována v jazyku Java, v němž lze dělat vše – v tomto případě se zobrazí notifikaci a začne validování metadata.

Vaadin podporuje i lokalizaci UI, o kterou se v projektu stará třída `TranslationProvider`. Grafické komponenty reagují na změnu jazyka a této třídy se ptají na správný překlad.

Příklad 6.5: Implementace tlačítka pomocí knihovny Vaadin

```
Button validationButton = new Button("Validate");
validationButton.addClickListener(e -> {
  Notification.show("Validating file");
  csvwProcessor.processMetadata(null, url);
});
```

Dále Vaadin obsahuje technologii WebSocket – obousměrný komunikační kanál přes TCP (*Transmission Control Protocol*) připojení. Navíc je tato technologie jednoduchá na použití a webová aplikace ji využívá při čekání na validaci. Pokud je poslán z webové aplikace požadavek na validaci souboru, je uživatel přesměrován na obrazovku s indikátorem průběhu a aplikace na pozadí vytvoří nové vlákno, v němž probíhá tato validace. Jakmile vlákno skončí výpočet a uloží se výsledek do databáze, informuje server uživatelské rozhraní pomocí WebSocketu, a to tak zobrazí daný výsledek.

6.4.8 Balíček csvwvalidator

Hlavní balíček `csvwvalidator` obsahuje také tři třídy. Spring Boot třídu `CsvwValidatorWebApplication` s hlavní Java metodou `main()`, pomocí které se spustí aplikační server s webovou službou i aplikací. Po doběhnutí této metody máme aplikaci spuštěnou na adrese `http://localhost:8080`. Výhodou knihovny Spring Boot je vestavěný Tomcat server a není tedy nutné nasazovat WAR (Web archive) soubor. Samozřejmě, že knihovna Spring Boot nabízí spoustu možností – lze měnit např. port aplikace, nechat si vygenerovat WAR soubor a jiné. [32]

Třída `CsvwValidatorConsoleApplication` implementuje spouštěč validátoru z příkazového řádku. Tato třída je uložena v samostatném modulu `csvw-validator-cli-app`, který má závislost na knihovním modulu `csvw-validator-lib`. Jelikož i knihovní modul využívá knihovnu Spring pro *dependency injection*, je potřeba před použitím knihovny načíst Spring kontext, viz Příklad 6.6. Třída `ProcessorConfig` je hlavní konfigurační soubor modulu `csvw-validator-lib`, a tak je ji třeba importovat a načíst z ní Spring kontext. Poté z načteného kontextu máme přístup ke všem *Spring beans*, definovaným v knihovním modulu, např. k `CsvwProcessor`.

Po nastartování konzolové aplikace se spustí logika třídy, která spočívá v přečtení argumentů z příkazového řádku, spuštění validace souborů, vytvoření výsledků a vypnutí aplikace. O čtení argumentů příkazového řádku se stará knihovna Commons CLI [45]. V dokumentaci v Kapitole 8.3.1 lze najít přesné použití konzolové aplikace.

Příklad 6.6: Import Spring kontextu z knihovního modulu

```
public static void main(String[] args) {
  ApplicationContext ctx = new
     AnnotationConfigApplicationContext(
        ProcessorConfig.class
     );
  CsvwProcessor processor =
     ctx.getBean(CsvwProcessor.class);
  // logika
}
```

Kapitola 7

Testování

Cílem testování je ověřit funkčnost aplikace, přičemž v případě refaktoringu kódu testy říkají, že chování aplikace nebylo změněno. V této kapitole jsou popsány automatizované testy, jež rozlišujeme na tzv. **unit** (jednotkové) **testy** a **integrační testy**. Pro oba typy testů byla použita již zmíněná knihovna Spock.

7.1 Unit testy

Unit testy slouží k otestování funkčnosti jednotlivých částí aplikace. Jednotlivé funkčnosti jsou testovány izolovaně, a jestliže testovaná část aplikace komunikuje s jinými komponentami, jsou tyto komponenty zastoupeny tzv. *mocky* – objekty simulující chování zastupující části.

V aktuální verzi validátoru nepokrývají unit testy mnoho kódu, neboť majoritní část funkcionalit je testována integračními testy. Unit testy pokrývají většinu tříd z balíčku `utils` ale i některé třídy z balíčku `domain.metadata`. Celkově se jedná o 199 unit testů.

Na Příkladu 7.1 je ukázka parametrizovaného Spock testu. Parametrizovaný test znamená, že se jedná pouze o jeden test, který se spouští několikrát s různými parametry (vstupy).

Každý Spock test rozšiřuje třídu `Specification`. Test se poté definuje klíčovým slovem `def`, za nímž následuje název testu a jeho tělo s logikou. Parametrizovaný test lze aktivovat klíčovým slovem `where` a tabulkou s hodnotami nacházející se pod tímto slovem. Názvy parametrů tvoří první řádek tabulky. V příkladu se jedná o parametry `firstTag`, `secondTag` a `expectedValue`. Tyto názvy lze v testu používat jako proměnné. Následně je test spuštěn tolikrát, kolik tabulka obsahuje řádků s hodnotami parametrů (zde třikrát). Anotace `@Unroll` nad testem zapíná možnost použití hodnot parametrů i v názvu testu.

Příklad 7.1: Unit testy ve třídě `LanguageUtilsTest`

```
class LanguageUtilsTest extends Specification {

  @Unroll
  def "Language tags equals #expectedValue -
     '#firstTag' and '#secondTag'."() {
    when: "Receive two strings"
    boolean areEquals =
       LanguageUtils.equalsLanguageTags(firstTag,
       secondTag)

    then: "Compare them if their smallest truncated
       versions by BCP47 are equal"
    areEquals == expectedValue

    where:
    firstTag | secondTag         | expectedValue
    "und"    | "a-bad-language" | false
    "en"     | "en"              | true
    "en"     | "en-US"           | true
  }
}
```

7.2 Integrační testy

Integrační testy oproti unit testům slouží k otestování aplikace jako celku, tedy toho, zda jednotlivé komponenty správně pracují, zároveň mezi sebou komunikují a dohromady tvoří funkční celek.

Knihovna Spoc obsahuje podporu i pro integrační testy. Testovací třída musí opět rozšiřovat třídu `Specification`, ale navíc je zapotřebí přidat třídní anotaci `@SpringBootTest`, díky níž se před integračními testy spustí Spring Boot aplikace.

Aktuálně existují v projektu dva parametrizované integrační testy – první pro webovou službu (třída `CsvwController`) a druhý pro třídu `CsvwProcessor`, který je testován pomocí referenčních testů *CSV on the Web* [5]. Pokrytí kódu těmito integračními testy ukazuje Tabulka 7.1.

Třídy	**Metody**	**Řádky**
239/250	831/1130	3921/4909
95,6 %	73,5 %	79,8 %

Tabulka 7.1: Pokrytí zdrojového kódu integračními testy

Naměřené pokrytí kódu testy bylo získáno pomocí vývojového prostředí IntelliJ Idea. Z naměřených dat vychází, že je integračními testy pokryto okolo $79,8\,\%$ řádků kódu.

Kapitola 8

Dokumentace

Dokumentace je rozdělena do tří kategorií, podle tří skupin uživatelů, jež budou validátor používat. Konkrétně se jedná o administrátorskou, programátorskou a uživatelskou dokumentaci.

8.1 Administrátorská dokumentace

Administrátorská dokumentace se zabývá:

- požadavky validátoru na prostředí, ve kterém je validátor spouštěn;
- instalací validátoru;
- spuštěním webové služby a aplikace.

8.1.1 Instalace

Před samotnou instalací validátoru csvw-validator je potřeba mít v systému nainstalovány aplikace:

- Java verze alespoň 8,
- Apache Maven verze alespoň 3.3.9.

Jsou-li tyto požadavky splněny, lze validátor stáhnout, např. na URL `https://github.com/malyvoj3/csvw-validator`. Po stažení je nutné otevřít hlavní složku validátoru a nainstalovat jej příkazem:

```
mvn clean install
```

Po skončení instalace jsou vygenerovány tři JAR soubory – pro každý z modulů jeden:

- `csvw-validator-cli-app-1.0.0-SNAPSHOT.jar`,
- `csvw-validator-lib-1.0.0-SNAPSHOT.jar`,
- `csvw-validator-web-app-1.0.0-SNAPSHOT.jar`.

Kde `1.0.0` je aktuální verze validátoru definována v souborech `pom.xml` – může se tedy měnit.

8.1.2 Spuštění webové služby a aplikace

Obvykle je pro zprovoznění webové služby nebo aplikace potřeba nastartovat aplikační server a nasadit na něj webový archiv. Validátor však používá knihovnu Spring Boot, která dokáže spustit embedovaný aplikační server Tomcat a zároveň na něj nasadit jak webovou službu, tak webovou aplikaci.

```
java -jar csvw-validator-web-app-1.0.0-SNAPSHOT.jar
```

Po nastartování aplikace bude webová služba i aplikace k dispozici na URL `http://localhost:8080`. Defaultně se server nahraje na port 8080. Pomocí parametru spuštění `--server.port` však lze cílový port definovat:

```
java -jar csvw-validator-web-app-0.0.1-SNAPSHOT.jar
--server.port={port}
```

8.2 Programátorská dokumentace

Programátorská dokumentace je věnovaná programátorům, kteří chtějí používat implementovaný validátor. Dokumentace obsahuje návod popisující jak importovat a použít validátor jako knihovnu v jiném projektu. V dokumentaci se také nachází, které jazyky byly použity pro vytvoření dokumentace webové služby a zdrojového kódu, a kde lze tuto dokumentaci nalézt.

8.2.1 Použití validátoru jako knihovny

Validátor lze použít jen jako knihovnu, a to pomocí submodulu `csvw-validator-lib`. Chce-li programátor importovat knihovnu do svého projektu, musí buď přidat do závislostí balíček `csvw-validator-lib-1.0.0-SNAPSHOT.jar`, nebo správně nastavit Maven.

Je-li použit Maven, musí se nejdříve přidat do souboru `pom.xml` plugin `maven-dependency-plugin`, který je povinný kvůli závislosti validátoru na knihovně Apache Jena, viz Příklad 8.1.

Příklad 8.1: Nastavení Maven závislosti na `csvw-validator-lib` – krok 1

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-dependency-plugin</artifactId>
      <dependencies>
        <dependency>
          <groupId>org.apache.felix</groupId>
          <artifactId>maven-bundle-plugin</artifactId>
          <version>2.4.0</version>
          <type>maven-plugin</type>
        </dependency>
      </dependencies>
      <extensions>true</extensions>
    </plugin>
  </plugins>
</build>
```

Následně již lze vložit závislost na `csvw-validator-lib`, viz Příklad 8.2.

Příklad 8.2: Nastavení Maven závislosti na `csvw-validator-lib` – krok 2

```
<dependencies>
  <dependency>
    <groupId>com.malyvoj3</groupId>
    <artifactId>csvw-validator-lib</artifactId>
    <version>1.0.0-SNAPSHOT</version>
  </dependency>
</dependencies>
```

Jak bylo rečeno v Kapitole 6.4.8 i knihovní modul využívá knihovnu Spring pro *dependency injection*. Pro používání knihovny je důležité mít v projektu závislost na modul Core knihovny Spring. Následně lze získat Spring kontext podle Příkladu 6.6. Pokud *dependency injeciton* není vyžadována, není nutné kontext načítat. V takovém případě si musí programátor sám dodefinovat jednotlivé závislosti mezi všemi třídami validátoru.

8.2.2 Dokumentace webové služby

Dokumentace RESTové webové služby je uložena ve složce `csvw-validator-web-app` v souboru `validator.raml`. Pro popis REST API (*Application Programming Interface*) je použit jazyk RAML (*RESTful API Modeling Language*) [46], který je dobře čitelný lidmi, ale i stroji.

Z formátu RAML lze vygenerovat HTML stránku programem raml2html. Výsledná stránka obsahuje přehlednou a dobře čitelnou dokumentaci. Popis

webové služby je publikován na GitHub pages validátoru: `https://malyvoj3.github.io/csvw-validator/restApi.html`.

8.2.3 Dokumentace zdrojového kódu

Zdrojový kód obsahuje dokumentaci, kterou lze převést do HTML formátu pomocí generátoru Javadoc. Generování HTML lze spustit následujícím příkazem v adresáři validátoru nebo v adresáři kteréhokoliv ze submodulů:

```
mvn javadoc:javadoc
```

Dokončením příkazu se dokumentace vygeneruje do složky `/target/site`. Aktuální verzi dokumentace lze najít na GitHub pages stránkách projektu `https://malyvoj3.github.io/csvw-validator/sourceCode.html`.

8.3 Uživatelská dokumentace

Uživatelská dokumentace uvádí příklady použití konzolové aplikace, webové služby i webové aplikace. U webové aplikace se prochází jednotlivé scénáře případů užití z Kapitoly 5.2.

8.3.1 Konzolová aplikace

Modul `csvw-validator-cli-app` se stará o spouštění validátoru pomocí příkazového řádků. Validaci lze z příkazového řádků spustit následovně:

Příklad 8.3: Nápověda k validování příkazovým řádkem

```
java -jar csvw-validator-lib-0.0.1-SNAPSHOT.jar
[-f <FILE>] [-s <SCHEMA>] [-o <OUTPUT>]
[--strict] [-h] [--rdf] [--csv]

 -f,--file <FILE>
 -s,--schema <SCHEMA>
 -o,--output <OUTPUT>
    --strict
    --csv
    --rdf
 -h,--help
```

Nápovědu ke konzolové aplikace obsahuje Příklad 8.3. V příkladu se nachází několik přepínačů, s nimiž lze program spustit. Všechny přepínače jsou volitelné, nicméně je nutné použít alespoň jeden z následujících: `-f`, `--file`, `-s` nebo `--schema`.

Přepínač `-f` nebo `--file` očekává jako parametr IRI tabulkového souboru, který se má validovat. Naproti tomu přepínač `-s` nebo `--schema` očekává jako parametr také IRI, ale metadatového souboru. Přepínači `-o` nebo `--output` se vkládá do parametru název souboru, ve kterém uloží výsledky. Není-li přepínač `-o` nebo `--output` nastaven, zvolí se jako název řetězec `result`. Přípony se výslednému souboru dodatečně přidávají podle formátu (`.txt`, `.csv` nebo `.ttl`).

Výsledek se defaultně vypisuje na výstup v textovém formátu, jai již bylo uvedeno v Kapitole 6.4.5, a ukládá se do souboru s příponou `.txt`. Vygenerovat výsledek lze i v jiném formátu – v CSV s příponou `.csv` nebo v RDF serializaci Turtle s příponou `.ttl`. Pro tyto výsledky je nutné použít přepínač `--csv`, respektive `--rdf`.

Přepínačem `--not-strict` lze vypnout striktní mód validace, a tudíž se nebude kontrolovat formát CSV souboru. Přepínači `-h` nebo `--help` lze zobrazit nápovědu.

Ukázka spuštění validátoru je prezentována v Příkladu 8.4 s výstupem zobrazeným v Příkladu 6.3.

Příklad 8.4: Ukázka příkazu ke spuštění validace z příkazového řádku

```
java -jar csvw-validator-lib-0.0.1-SNAPSHOT.jar
 -f http://www.w3.org/2013/csvw/tests/test286.csv
 -s http://www.w3.org/2013/csvw/tests/test286
 -metadata.json --rdf --csv
```

8.3.2 Webová služba

V aktuální verzi webová služba obsahuje tři tzv. *endpointy* – URL adresy, na nichž webová služba komunikuje.

První endpoint `/validationResult/{resultId}` slouží k získání výsledku validace. URI parametr *resultId* je jednoznačný identifikátor daného výsledku, který je vždy vracen při dokončení validace pomocí webové služby nebo webové aplikace. Existuje-li výsledek s daným identifikátorem, pak je vrácen v HTTP odpovědi stavový kód 200 společně s výsledkem uloženém ve formátu JSON – viz Příklad 8.5. Neexistuje-li výsledek identifikovatelný pomocí `resultId`, pak je v HTTP odpovědi nastaven stavový kód na 404.

Příklad 8.5: Odpověď endpointu `/validationResult/{resultId}`

```
{
  "id": "resultId",
  "validationStatus": "ERROR",
  "tabularUrl":
     "http://www.w3.org/2013/csvw/tests/test286.csv",
  "metadataUrl": "http://www.w3.org/2013/csvw/tests/
                  test286-metadata.json",
  "validationErrors": [
    {
      "severity": "ERROR",
      "message": "Cell (row 2 column 1) cannot be
         formatted as 'integer' dataype."
    }
  ],
  "warningCount": 0,
  "errorCount": 1,
  "fatalCount": 0,
  "totalErrorsCount": 1,
  "rowsNumber": 2,
  "columnsNumber": 1,
  "tablesNumber": 1
}
```

Výsledný JSON z Příkladu 8.5 obsahuje tyto vlastnosti:

- *validationStatus* – výsledek validace. Jeden z řetězců `PASSED`, `WARNING` nebo `ERROR`;
- *tabularUrl* – IRI validovaného souboru tabulkových dat;
- *metadatUrl* – IRI validovaného souboru metadat;
- *validationErrors* – pole obsahující dvojice chyba (její popis) a závažnost chyby (`WARNING`, `ERROR`, `FATAL`);
- *warningCount* – počet `WARNING` chyb;
- *errorCount* – počet `ERROR` chyb;
- *fatalCount* – počet `FATAL` chyb;
- *totalErrorsCount* - celkový počet všech chyb;
- *rowsNumber* - celkový počet zpracovaných řádků během validace;

- *columnsNumber* - celkový počet zpracovaných sloupců během validace;
- *tablesNumber* - celkový počet zpracovaných tabulek během validace.

Druhým endpointem je `/validate`, který slouží k validaci jednoho souboru – CSV souboru, JSON-LD deskriptoru či jejich kombinací. Na tomto endpointu se očekává POST HTTP požadavek, odesílající a přijímající typ `application/json`. Tělo požadavku je JSON ve formátu, který lze vidět na Příkladu 8.6.

Příklad 8.6: Tělo požadavku pro endpoint `/validate`

```
{
    "tabularUrl": "http://www.w3.org/2013/csvw/tests/
        test286.csv",
    "metadataUrl":
       "http://www.w3.org/2013/csvw/tests/
        test286-metadata.json",
    "strictMode": false,
    "language": "en"
}
```

Vlastnost *tabularUrl* obsahuje textový řetězec, který reprezentuje IRI tabulkových dat. Ve vlastnosti *metadataUrl* se očekává IRI pro metadatový dokument a vlastností *strictMode* lze aktivovat nebo deaktivovat striktní mód. Jazyk výsledných popisů chyb lze nastavit vlastností *language*, jež očekává jazykový kód dle BCP47 [20]. Aktuálně je podporovány jen český a anglický jazyk. Žádná z vlastností není povinná, je však nutné mít vyplněnu jednu z vlastností *tabularUrl* či *metadataUrl*. Není-li specifikována vlastnost *strictMode* nebo *language*, použijí se jejich základní hodnoty `false` a `"en"`.

Odpověď na požadavek z Příkladu 8.6 je totožná s odpovědí z Příkladu 8.5.

Poslední endpoint webové služby `/validateBatch` umožňuje spouštět hromadnou validaci. Rovněž se jedná o POST HTTP požadavek, odesílající a přijímající typ `application/json`. Ukázkové tělo požadavku je zobrazeno v Příkladu 8.7 a skládá se z následujících JSON vlastností:

- *filesToProcess* – pole párů *tabularUrl* a *metadataUrl*, kde jeden pár je jeden soubor z hromadné validace;
- *filesResults* – boolean, který informuje server o tom, jestli má být v odpovědi připojen výsledek pro každý z validovaných souborů, a to ve formě odpovědi z endpointu `/validate`;
- *strictMode* – boolean sloužící k aktivaci nebo deaktivaci striktního módu;

- *language* – řetězec obsahující jazykový kód dle BCP47 [20].

Příklad 8.7: Tělo požadavku pro endpoint `/validateBatch`

```
{
  "filesToProcess": [
    {
        "tabularUrl":
           "http://www.w3.org/2013/csvw/tests/
            test282.csv",
        "metadataUrl":
           "http://www.w3.org/2013/csvw/tests/
            test282-metadata.json"
    },
    {
        "tabularUrl":
           "http://www.w3.org/2013/csvw/tests/
            test283.csv"
    },
    {
        "metadataUrl":
           "http://www.w3.org/2013/csvw/tests/
            test284-metadata.json"
    }
  ],
  "strictMode": false,
  "filesResults": false,
  "language": "en"
}
```

Ukázka odpovědi na požadavek pro validaci více souborů je na Příkladu 8.8 a mezi její JSON vlastnosti patří:

- *filesCount* – počet validovaných souborů;
- *passedFilesCount* – počet souborů bez chyb;
- *warningFilesCount* – počet souborů s varováním;
- *errorFilesCount* – počet souborů s chybou;
- *filesResults* – pole dvojic identifikátoru výsledku a samotného výsledku. Zachovávají pořadí, v jakém byly uloženy ve vlastnosti *filesToProcess*.

Příklad 8.8: Odpověď na požadavek endpointu `/validateBatch`

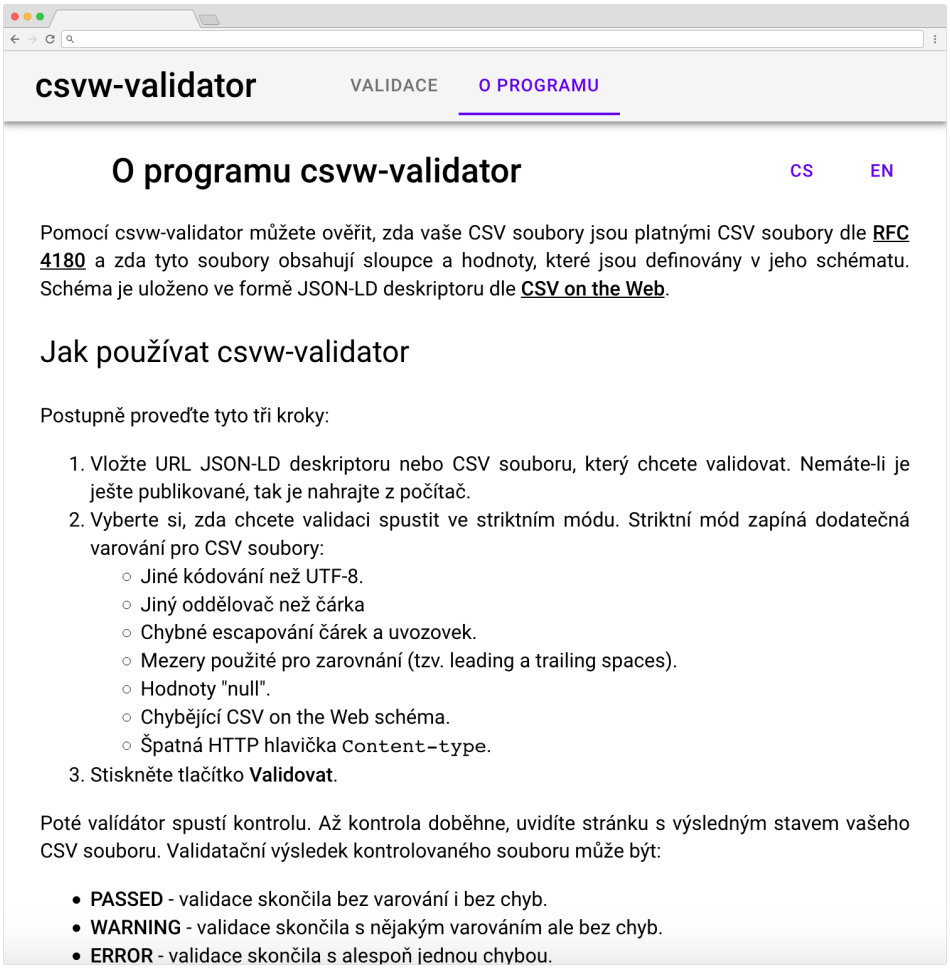
```
{
  "filesCount": 3,
  "passedFilesCount": 3,
  "warningFilesCount": 0,
  "errorFilesCount": 0,
  "filesResults": [
      {
        "id": "resultId1"
      },
      {
        "id": "resultId2"
      },
      {
        "id": "resultId3"
      }
  ]
}
```

8.3.3 Webová aplikace

Webová aplikace obsahuje tři obrazovky – hlavní obrazovku s formulářem pro zadání validace, obrazovku s výsledkem validace a obrazovku s informacemi o validátoru. Text webové aplikace je lokalizován ve dvou jazycích – v českém a anglickém – mezi kterými lze přepínat pomocí tlačítek umístěných v levém horním rohu.

Obrazovka popisující validátor obsahuje základní informace potřebné pro práci s ním. Uživatel se zde dozví, co a proč validátor kontroluje, a jak lze validace spouštět. Podoba této obrazovky je zachycena na Obrázku 8.1.

V následujících podkapitolách bude vysvětleno, jak splnit jednotlivé scénáře případů užití z Kapitoly 5.2.



Obrázek 8.1: Webová aplikace – obrazovka s popisem validátoru

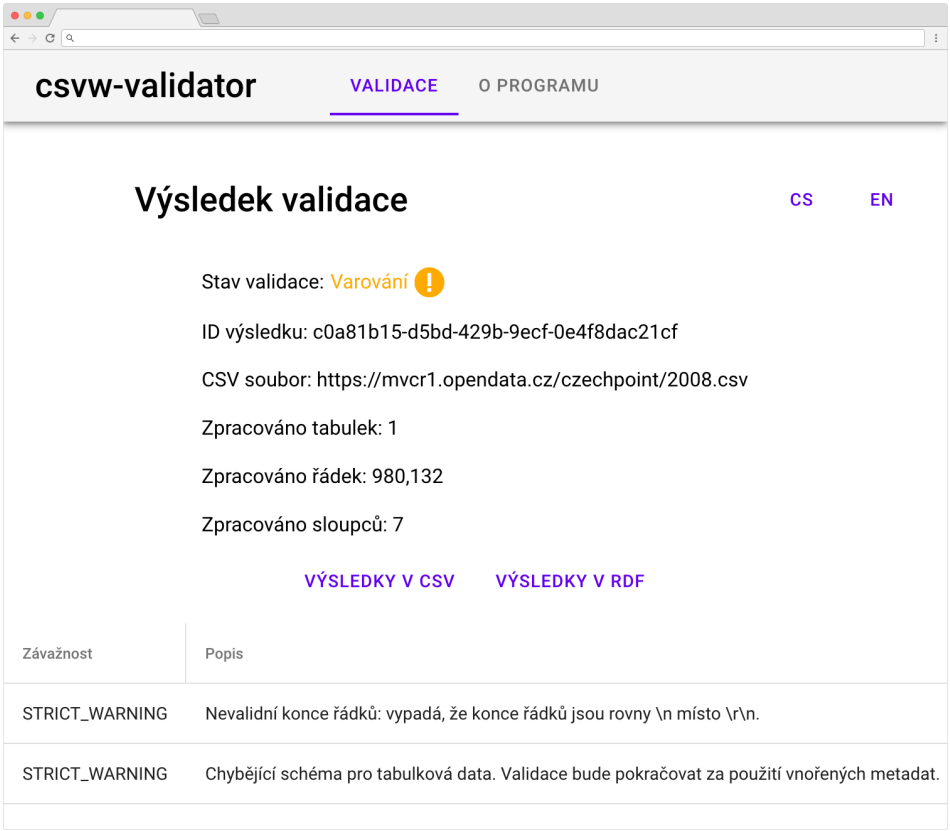
8.3.3.1 PU1 Zobrazení výsledku validace

Má-li uživatel identifikátor pro výsledek validace, který chce vidět, pak musí otevřít webovou aplikaci na adrese `/result/{resultId}`, kde *resultId* je zmíněný identifikátor. Ukázka obrazovky s výsledkem validace samotného CSV souboru je na Obrázku 8.2.

Obrazovka s výsledkem validace obsahuje tyto části:

- stav validace – indikuje výsledek validace pomocí barevného názvu stavu a ikonky;
- id výsledku – identifikátor výsledku, podle kterého lze výsledek vždy dohledat a zobrazit;

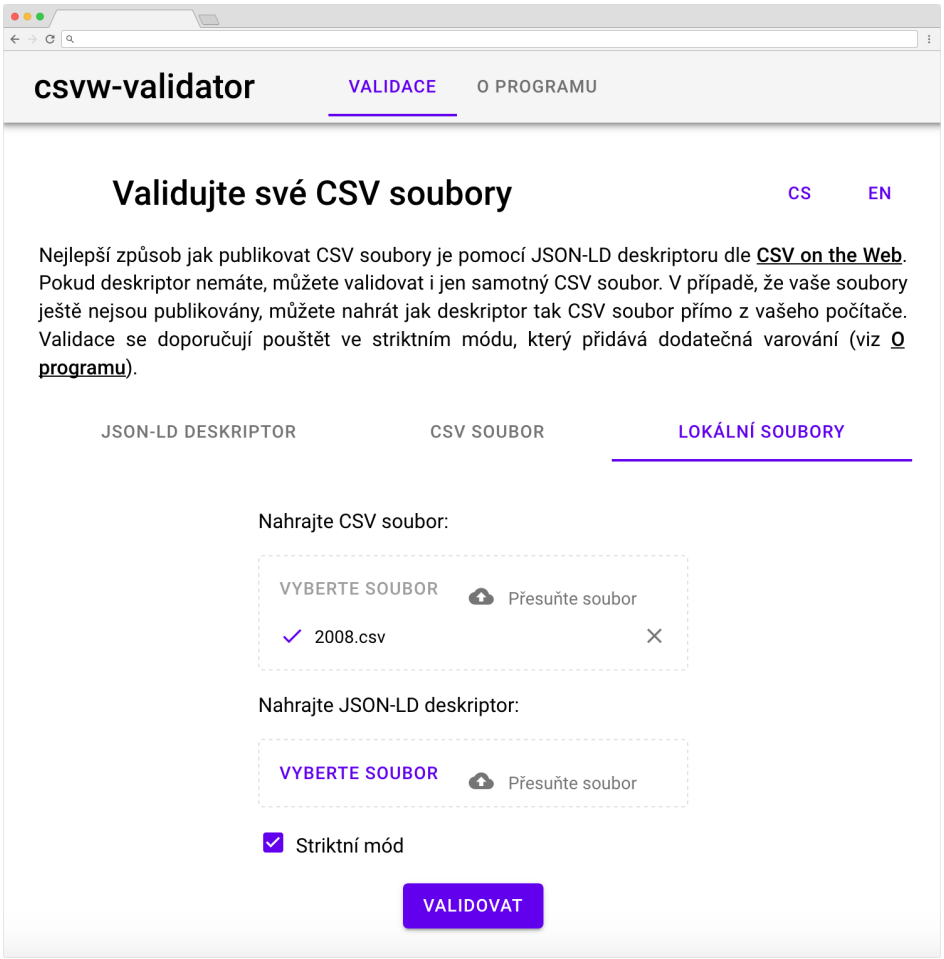
- CSV soubor – IRI validovaného souboru;
- informace o tom, kolik bylo během validace celkově zpracováno tabulek, řádek a sloupců;
- tabulka s chybami;
- tlačítka pro stažení výsledku ve formátu CSV a RDF.



Obrázek 8.2: Webová aplikace – obrazovka s výsledkem validace

8.3.3.2 PU2 Validace lokálního CSV souboru

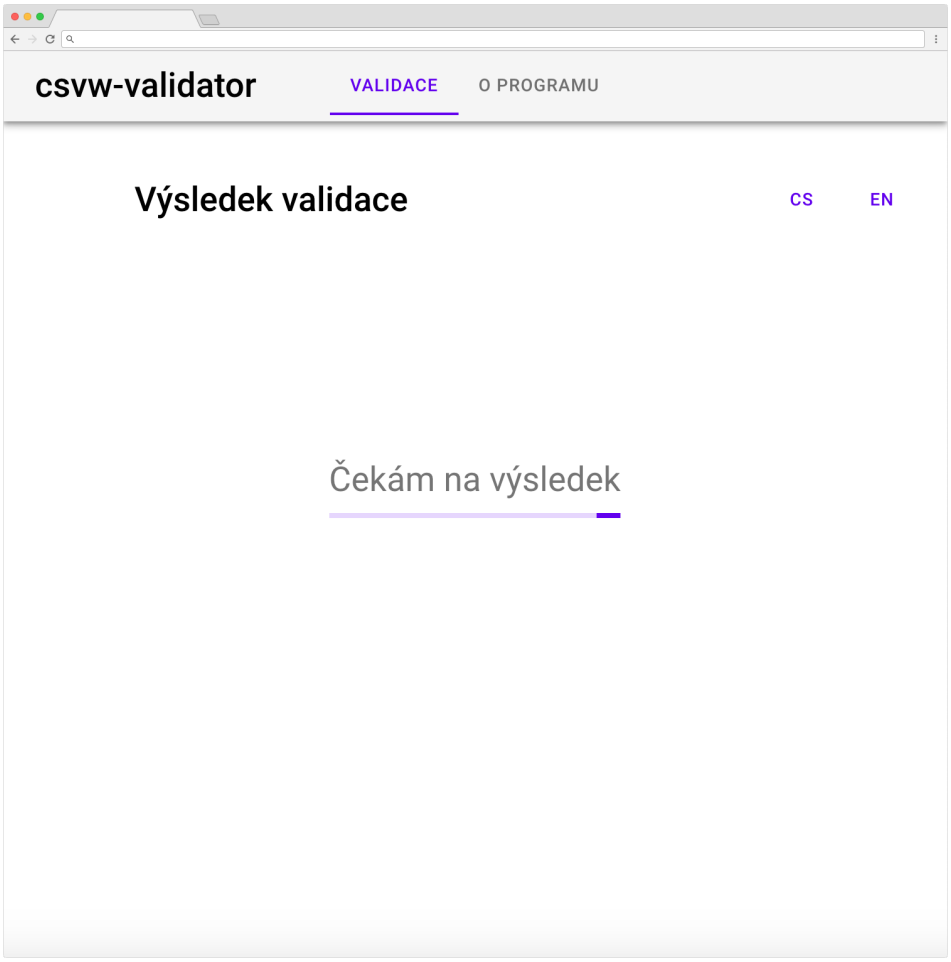
Uživatel otevře hlavní stránku aplikace – zadání validace. Následně ve formuláři vybere tabulku *Lokální soubory*, kde se uživateli nabídne možnost nahrát lokální soubor. Uživatel tedy přesune soubor do vyznačeného místa pro CSV soubor, nebo klikne na tlačítko *Vyberte soubor*. Pokud uživatel vybere soubor, tak je následně nahrán do formuláře, viz Obrázek 8.3.



Obrázek 8.3: Webová aplikace – validace lokálního CSV souboru

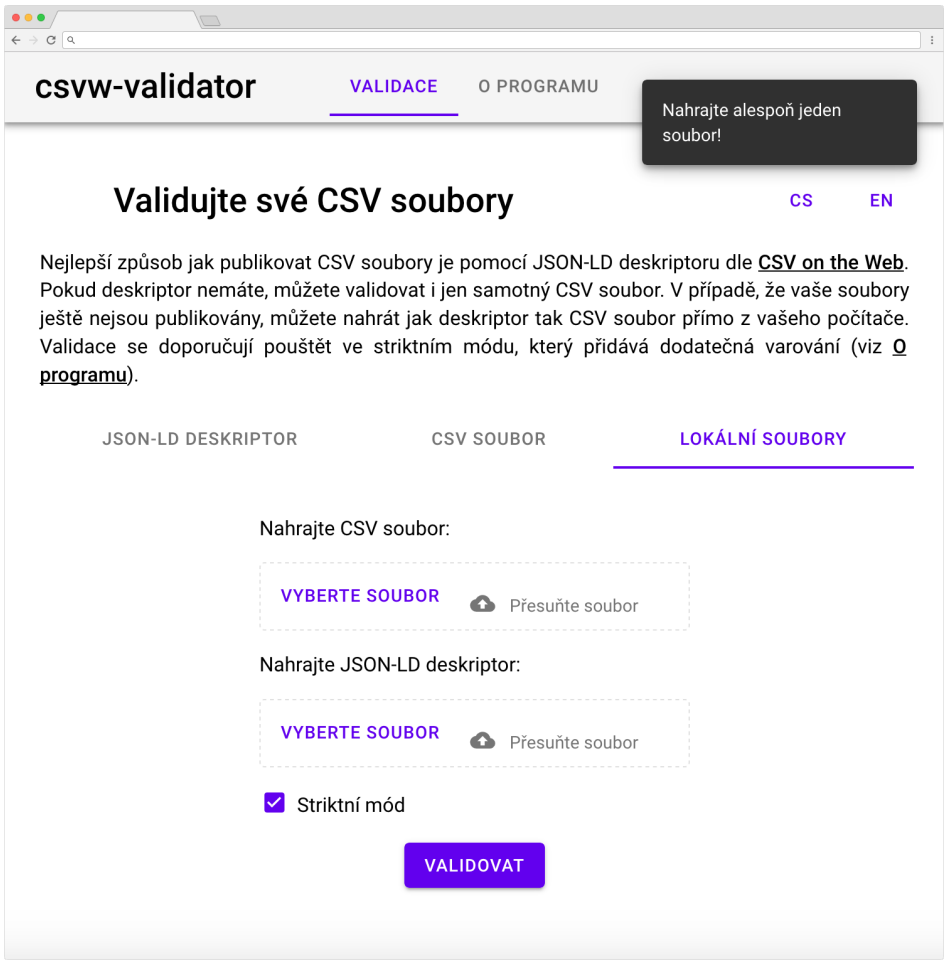
Následně již může uživatel stisknout tlačítko *Validovat*, které ho přesměruje na dočasnou stránku z Obrázku 8.4, kde je uživateli zobrazen indikátor průběhu validace.

Validátor při nahrání samotného CSV souboru z disku provádí jen kontrolu CSV souboru, neboť se server k disku uživatele nedostane a nemůže tedy lokalizovat metadata. V okamžiku, kdy je validace dokončena, je uživatel automaticky přesměrován na obrazovku s výsledkem validace – viz Kapitola 8.3.3.1.



Obrázek 8.4: Webová aplikace – čekání na výsledek validátoru

Pokud uživatel nevybere ani jeden soubor a klikne na tlačítko *Validovat*, je uživatele zobrazena notifikace z Obrázku 8.5, ve které je řečeno, aby nahrál alespoň jeden soubor k validaci.



Obrázek 8.5: Webová aplikace – varovná notifikace při nevyplnění souborů

8.3.3.3 PU3 Validace CSV souboru uloženého na webu

Uživatel otevře hlavní stránku aplikace – zadání validace. Následně ve formuláři vybere tabulku *CSV soubor*, viz Obrázek 8.6. Tím se uživateli nabídne možnost napsat IRI vzdáleného CSV souboru.

Obrázek 8.6: Webová aplikace – validace CSV souboru uloženého na webu

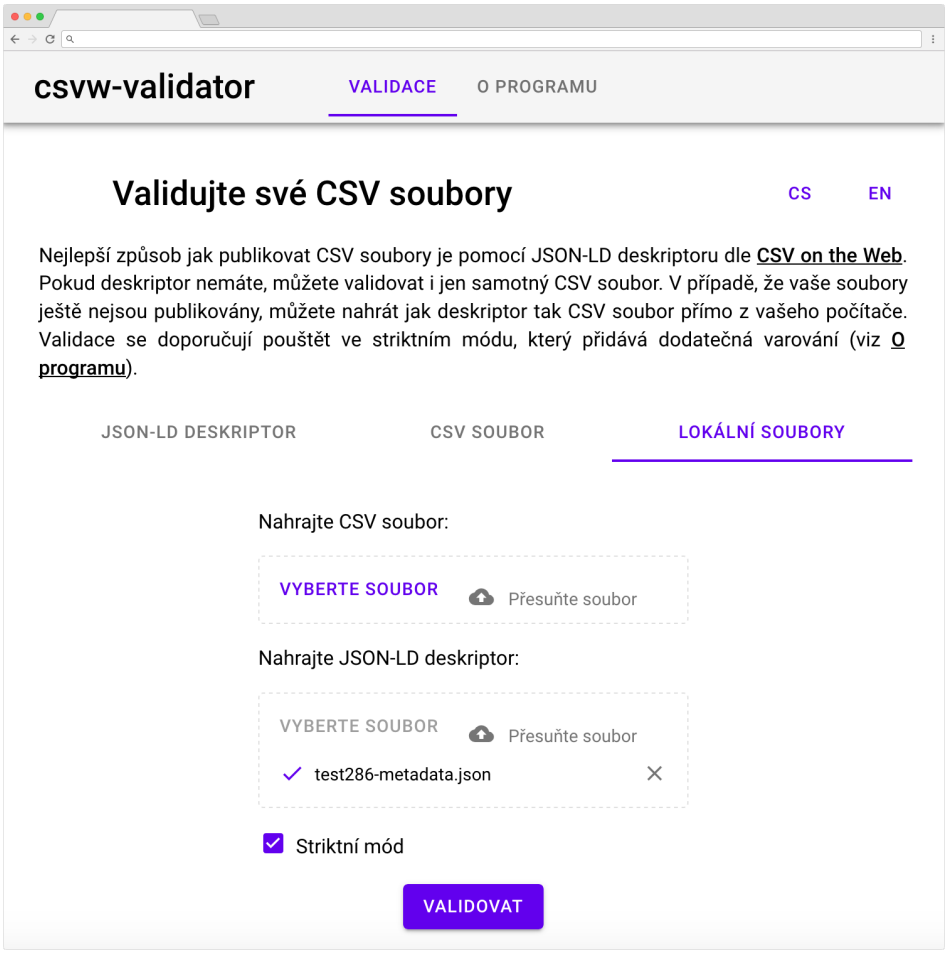
Vloží-li uživatel IRI, může stisknout tlačítko *Validovat*, které ho přesměruje na dočasnou stránku z Obrázku 8.4, kde je uživateli zobrazen indikátor průběhu validace. Textové pole pro IRI je povinné, tudíž s prázdným polem nelze stisknout tlačítko *Validovat*.

Validátor se snaží získat CSV soubor, parsovat ho, lokalizovat pro něj schéma a následně vytvořit anotovaný model, který se zkontroluje. V okamžiku, kdy je validace dokončena, je uživatel automaticky přesměrován na obrazovku s výsledkem validace z Kapitoly 8.3.3.1.

8.3.3.4 PU4 Validace lokálního metadatového souboru

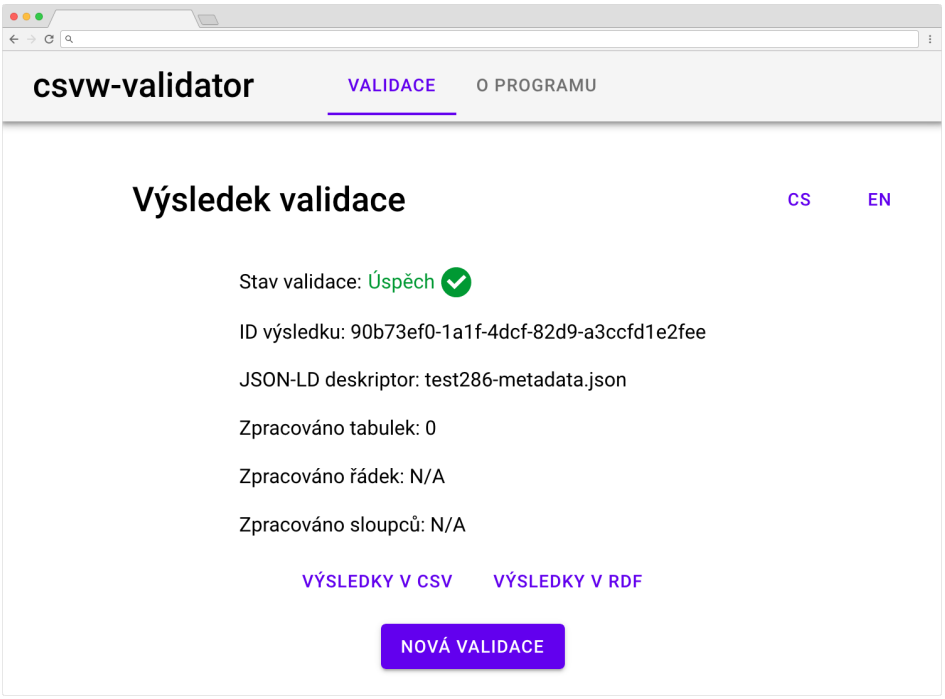
Uživatel otevře hlavní stránku aplikace – zadání validace. Následně ve formuláři vybere tabulku *Lokální soubory*, kde se uživateli nabídne možnost nahrát lokální soubor. Uživatel tedy přesune soubor do vyznačeného místa pro JSON-LD deskriptor, nebo stlačí tlačítko *Vyberte soubor*. Pokud uživatel vybere soubor, tak je následně nahrán do formuláře – viz Obrázek 8.7.

Poté již může uživatel stisknout tlačítko *Validovat*, které ho přesměruje na dočasnou stránku z Obrázku 8.4, kde je uživateli zobrazen indikátor průběhu validace. Pokud uživatel nezvolil žádný ze souborů a klikne na tlačítko *Validovat*, je uživateli opět zobrazena notifikace z Obrázku 8.5.



Obrázek 8.7: Webová aplikace – validace lokálního JSON-LD deskriptoru

Validátor při nahrání samotného JSON-LD deskriptoru z disku provádí jen kontrolu formátu metadatové dokumentu. V okamžiku, kdy je validace dokončena, je uživatel automaticky přesměrován na obrazovku s výsledkem validace. Na Obrázku 8.8 je výsledek validace samotného JSON-LD deskriptoru.



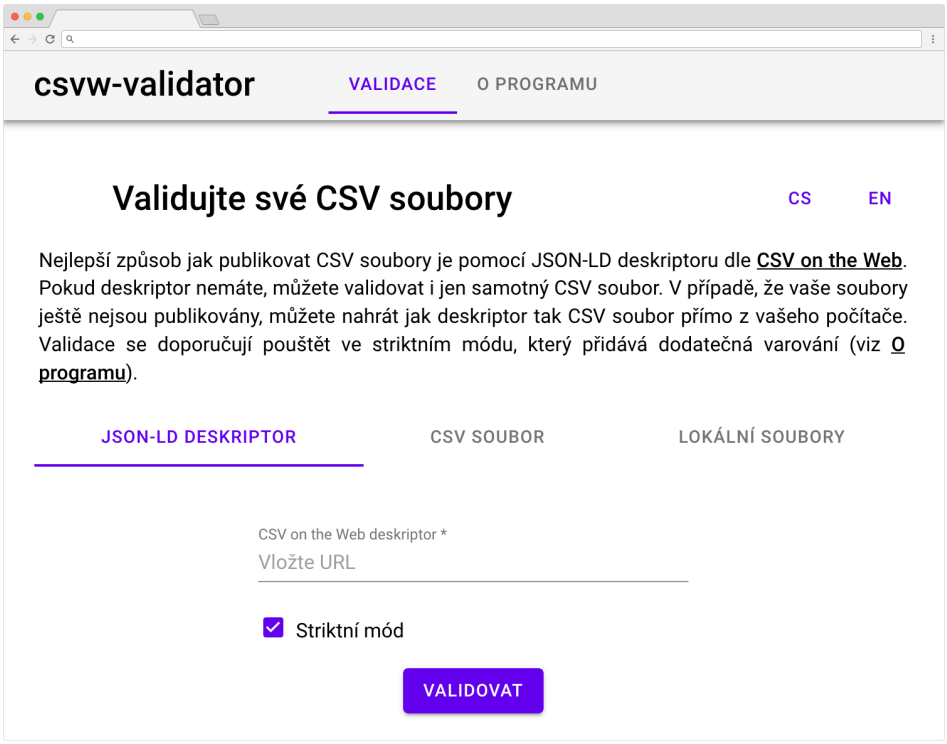
Obrázek 8.8: Webová aplikace – výsledek validace samotného JSON-LD deskriptoru

8.3.3.5 PU5 Validace metadatového souboru uloženého na webu

Uživatel otevře hlavní stránku aplikace. Následně ve formuláři vybere tabulku *JSON-LD deskriptor*, kde se uživateli nabídne možnost napsat IRI vzdáleného JSON-LD deskriptoru, jak lze vidět na Obrázku 8.9.

Zadá-li uživatel IRI, může stisknout tlačítko *Validovat*, které ho přesměruje na dočasnou stránku z Obrázku 8.4, kde je uživateli zobrazen indikátor průběhu validace. Textové pole pro IRI je povinné, tudíž s prázdným polem nelze stisknout tlačítko *Validovat*.

Validátor se snaží stáhnout metadatový dokument, parsovat ho, získat tabulková data, která popisuje, a následně vytvořit anotovaný model, který se zkontroluje. V okamžiku, kdy je validace dokončena, je uživatel automaticky přesměrován na obrazovku s výsledkem validace – viz Kapitola 8.3.3.1.



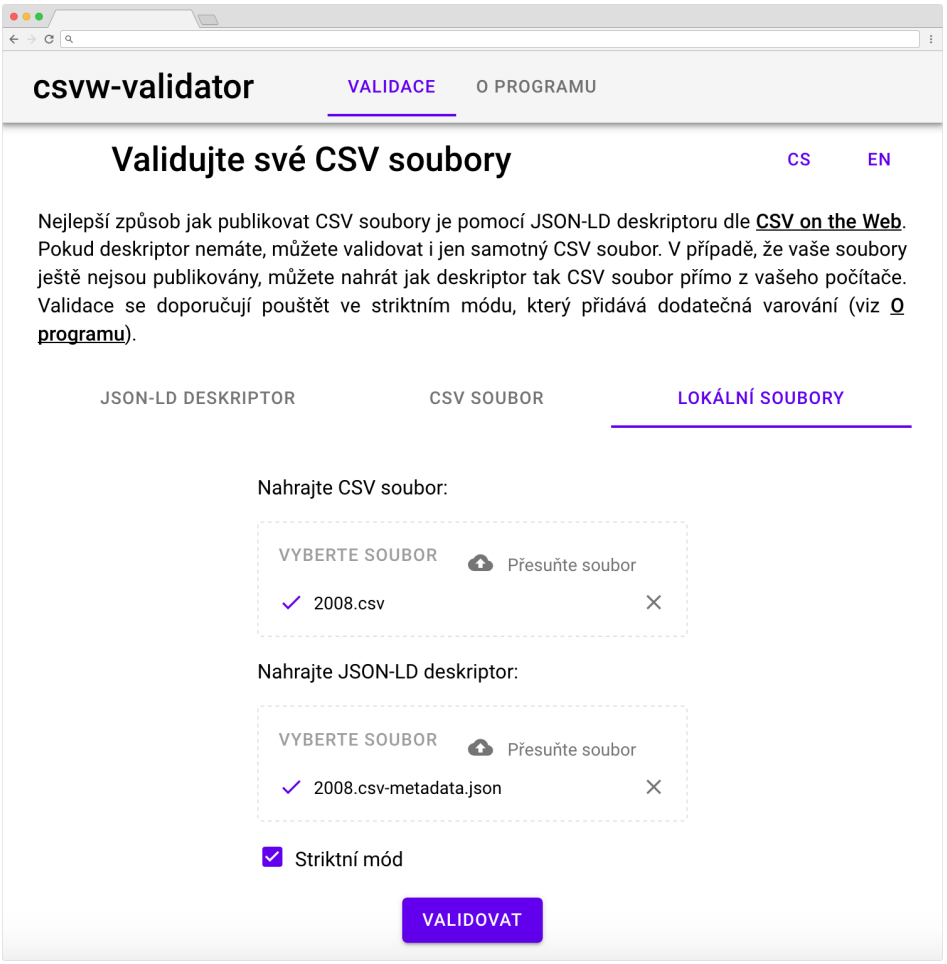
Obrázek 8.9: Webová aplikace – validace JSON-LD deskriptoru uloženého na webu

8.3.3.6 PU6 Validace lokálního CSV souboru i s lokálními metadaty

Uživatel otevře hlavní stránku aplikace. Následně ve formuláři vybere tabulku *Lokální soubory*, kde se uživateli nabídne možnost nahrát lokální soubor. Uživatel tedy přesune soubor do vyznačeného místa pro CSV soubor i JSON-LD deskriptor, nebo klikne na tlačítko *Vyberte soubor* a soubor vybere. Má-li uživatel ve formuláři nahrány oba soubory, viz Obrázek 8.10, může již poté stisknout tlačítko *Validovat*.

Následně je přesměrován na dočasnou stránku z Obrázku 8.4, kde se uživateli zobrazí indikátor průběhu validace.

Validátor při nahrání CSV souboru i JSON-LD deskriptoru z disku provádí jak kontrolu CSV souboru, tak i kontrolu modelu, který vznikne z tabulkových dat a schématu. V okamžiku, kdy je validace dokončena, je uživatel automaticky přesměrován na obrazovku s výsledkem validace z Kapitoly 8.3.3.1.



Obrázek 8.10: Webová aplikace – validace lokálního CSV souboru i s JSON-LD deskriptorem

8.3.3.7 PU7 Uložení výsledku validace ve formátu RDF

Uživatel si zobrazí existující výsledek validace, viz Kapitola 8.3.3.1. Poté co mu je zobrazena obrazovka s výsledkem, stiskne uživatel tlačítko *Výsledky v RDF*. Následně se uživateli otevře stahování daného výsledku ve formátu RDF.

8.3.3.8 PU8 Uložení výsledku validace ve formátu CSV

Uživatel si zobrazí existující výsledek validace, viz Kapitola 8.3.3.1. Poté co mu je zobrazena obrazovka s výsledkem, stiskne uživatel tlačítko *Výsledky v CSV*. Následně se uživateli otevře stahování daného výsledku ve formátu CSV.

Kapitola 9

Vyhodnocení validátoru

Validátor csvw-validator je v následující kapitole porovnán s již existujícími referenčními validátory. Nejdříve se porovnává funkčnost a splnitelnost validačních pravidel dle doporučení *CSV on the Web*. Následně je otestován výkon validátoru – konzolové aplikace a webové služby. V poslední části jsou nastíněny možná vylepšení validátoru, která lze v budoucnu dodělat.

9.1 Vyhodnocení vzhledem k referenčním implementacím

V Kapitole 4.1 byly porovnány tři referenční implementace validátoru *CSV on the Web*, jež byly vyhodnoceny na referenčních testech pro validátory [5]. V Tabulce 9.1 jsou jména jednotlivých validátoru a jejich úspěšnost v těchto testech.

CSVlint	**pycsvw**	**RDF::Tabular**
281/281	158/281	281/281
100 %	56, 2 %	100 %

Tabulka 9.1: Úspěšnost tří referenčních implementací validátoru v referenčních testech [5]

Z výsledků lze vyčíst, že validátor `pycsvw` je s jeho nízkou úspěšností téměř nepoužitelný. Vše podtrhuje nulová dokumentace a problémy se starou verzí jazyka Python, v němž je validátor programován.

CSVlint a RDF::Tabular již dle testů podporují celé doporučení *CSV on the Web*, ovšem i tak mají problémy, neboť referenční testy nepokrývají všechny možnosti a typy vstupů. Příkladem je IRI `https://dev.nkod.opendata.cz/soubor/datov%C3%A9-sady.csv-metadata.json`, kterou ani jeden z referenčních validátoru nezpracuje správně.

Má implementace validátoru, **csvw-validator**, prochází 262 testy z celkového počtu 281v viz porovnání v Tabulce 9.2

CSVlint	**pycsvw**	**RDF::Tabular**	**csvw-validator**
281/281	158/281	281/281	262/281
100 %	56, 2 %	100 %	93, 2 %

Tabulka 9.2: Úspěšnost validátoru csvw-validator v referenčních testech oproti referenčním implementacím

Vyvíjenému validátoru csvw-validator chybí podpora pro tři větší oblasti. Jedná se o dialekty, transformace a cizí klíče.

Transformace jsou objekty uloženy v popisujícím objektu tabulku, které pomáhají při převodu tabulkových dat do formátu JSON nebo RDF. [7] Pro transformace nebyla implementována žádná funkcionalita.

Validátor aktuálně nepoužívá dialekt pro parsování tabulkových dat, umí však parsovat objekt popisující dialekt a validovat jeho vlastnosti. Tabulková data je ovšem schopen parsovat jen ve formátu dle Kapitoly 1.2.

Dále validátor zvládne parsování objektů popisujících cizí klíče. Následně je ale potřeba cizí klíče validovat, zda odkazují na existující sloupec a tabulku, zda hodnoty tvořící cizí klíč jsou v tabulce unikátní aj. Tuto funkcionalitu prozatím csvw-validator nepodporuje.

Mimo tyto vyjmenované oblasti, by se měl validátor chovat podle doporučení *CSV on the Web*. Může ale stále obsahovat nějaké chyby, neboť ani referenční testy, ani integrační testy nepokrývají všechny možnosti použití validátoru.

Mezi devatenácti referenčními testy, které csvw-validator špatně vyhodnotí, patří sedm testů cizím klíčů, pět testů transformacím, jeden test dialektu, dva testy lokalizaci schématu dle HTTP hlavičky Link a zbylé čtyři chybné testy jsou důsledkem drobných chyb, které vznikly z důvodu jiného pochopení doporučení. Validátor by měl podporovat lokalizaci schématu dle hlavičky Link, ovšem referenční testy tuto hlavičku nevyplňují a validátor nepodporuje zadávání této hlavičky. Detailní rozpis jednotlivých testů a jejich výsledků je připojen v příloze v Tabulce D.1.

I když csvw-validator neobsahuje celou množinu validačních pravidel dle *CSV on the Web*, je stejně obstojným soupeřem ostatních validátorů.

Jeho architektura umožňuje jednoduché dokončení chybějících validační pravidel. Rozdělení do tří modulů uvítají uživatelé. Chtějí-li validátor jen jako knihovnu potřebují modul `csvw-validator-lib`, pokud chtějí webovou službu, použijí modul `csvw-validator-web-app`.

Ať se jedná o webovou službu, konzolovou aplikaci nebo jen knihovnu, používá se validátor velmi snadno oproti ostatním validátorům, neboť je k němu napsána podrobná dokumentace kódu i celé aplikace (viz Kapitola 8). Možnost

jednoduchého nasazení na server a používání webové služby či webového UI je velmi důležitou funkcionalitou, neboť nasadí-li se validátor na veřejnou adresu, mohou uživatelé jednoduše validovat své soubory.

Použití jazyku Java umožňuje využít validátor jinými projekty, ať už se jedná o aplikace programované v jazyku Java, Scala, Groovy či jiným jazykem postaveným na JVM.

9.2 Výkonnostní testy

Tato kapitola se zabývá vyvinutým validátorem a jeho výkoností. Jediným funkčním nástrojem je validátor RDF::Tabular, s kterým bude srovnána konzolová aplikace používající csvw-validator. Srovnání výkonu webových služeb csvlint.io a csvw-validator není možné, neboť nevíme na jakém prostředí je csvlint.io nasazen, tudíž by nešly splnit rovnoměrné podmínky pro obě služby. Nicméně byla alespoň otestována webová služba samotná.

Testování probíhalo na stroji MacBook Pro 2017 s konfigurací podle Tabulky 9.3.

Processor	Intel Core i5 2,3 GHz (2 jádra / 4 vlákna)
RAM	16 GB 2133 MHz
Disk	256 GB SSD
Operační systém	macOS Mojave 10.14.4
Java	1.8.0_171
Velikost haldy	4 GB

Tabulka 9.3: Popis testovacího prostředí pro výkonnostní testy

9.2.1 Výkon konzolové aplikace

V testech konzolových aplikací bude porovnávan csvw-validator s validátorem RDF::Tabular. Testovat se bude rychlost zpracování validace. Validovat se bude CSV soubor `2007.csv` s IRI `https://mvcr1.opendata.cz/czechpoint/2007.csv` a metadatový soubor `2007.json` s IRI `https://mvcr1.opendata.cz/czechpoint/2007.json`. CSV souboru budou tři verze, kdy první verze je onen soubor. Druhá `2007v2.csv` a třetí verze `2007v3.csv` obsahuje dvojnásobný resp. trojnásobný počet řádků, které vznikly duplikací prvního souboru. Všechny soubory budou k dispozici lokálně na disku, aby nedocházelo ke zkreslení vlivem internetového připojení.

Výsledky jsou zobrazeny v Tabulce 9.5. Naměřené časy jsou průměrné hodnoty z deseti validací, které byly naměřeny Unixovým programem time.

Z výsledku lze vypozorovat rychlost, kterou csvw-validator kontroluje samotné CSV soubory bez schématu. U testovaných souborů se jedná jen o jed-

Název souboru	Počet řádek	Počet sloupců	Velikost
2007.csv	54000	7	6,78 MB
2007v2.csv	107999	7	13,57 MB
2007v3.csv	161998	7	20,36 MB

Tabulka 9.4: Popis souborů použitých pro výkonnostní testy

CSV soubor	Schéma	Čas csvv-validator	Čas RDF::Tabular
2007.csv	NE	4,8 s	146,1 s
2007v2.csv	NE	5,2 s	347,3 s
2007v3.csv	NE	5,3 s	578,8 s
2007.csv	ANO	34,4 s	114,9 s
2007v2.csv	ANO	41,7 s	267,3 s
2007v3.csv	ANO	56,5 s	428,2 s

Tabulka 9.5: Výkonnostní srovnání csvw-validator a RDF::Tabular

notky vteřin. RDF::Tabular validuje samotné CSV soubory pomalu. U největšího testovaného CSV souboru je validace samotného CSV formátu validátorem csvw-validator více jak stokrát rychlejší než validátorem RDF::Tabular.

Připojí-li se k CSV souboru metadata, tak se csvw-validátor výrazně zpomalí. Pořád je však rychlejší než RDF::Tabular – u největšího souboru je sedmkrát rychlejší. Zvláštním zjištěním je skutečnost, že RDF::Tabular validuje rychleji CSV soubor se schématem, než jen samotný CSV soubor. Validátor csvw-validator má však opačný problém a validace modelu (zde datových typů a primárního klíče) prodlužuje validační čas.

9.2.2 Výkon webové služby

Na výkon webové služby nebyly kladeny požadavky, tudíž cílem testu bylo otestovat, že server poběží stabilně i při několika současně připojených uživatelích. REST API bylo otestováno pomocí programu Apache JMeter [47].

Jako testovací soubor byl znovu vybrán `2007.json` s IRI `https://mvcr1.opendata.cz/czechpoint/2007.json`, neboť se jedná o středně velký soubor, který je JSON-LD deskriptorem. Tudíž se bude testovat nejenom validace CSV ale i schématu, které csvw-validator validuje pomaleji – viz předchozí Kapitola 9.2.1.

Testování probíhalo ve třech scénářích:

1. 8 uživatelů spustí validaci najednou a po dokončení se validace spustí znova (celkem 5krát).

Konkurenčních uživatelů	Propustnost požadavků za minutu
8	6,5
50	6,1
100	5,2

Tabulka 9.6: Propustnost webové služby

2. 50 uživatelů spustí validaci najednou a po dokončení se validace spustí ještě jednou.

3. 100 uživatelů spustí validaci najednou.

Všechny scénáře skončily úspěšně, bez problémů a se správným výsledkem. Aplikace byla při testování stabilní a nedocházelo k vyčerpání velikosti haldy. Měřila se propustnost tzv. *throughput*, která definuje počet vyřízených požadavků za minutu. Naměřené propustnosti jsou zobrazeny v Tabulce 9.6.

V Kapitole 9.2.1 validátoru csvw-validator trvalo průměrně 34,4 vteřin validovat soubor `2007.json`. Vezme-li v potaz počet vláken procesoru, čas určený na stažení souboru s tabulkovými daty, čas na režii vláken a komunikaci skrz HTTP, tak poté propustnost odpovídá skutečnosti. Neboť teoreticky s časem validování souboru 34,4 vteřin lze za minutu zpracovat 1,74 požadavků. Použijeme-li 4 vlákna, dostaneme se na propustnost téměř 7 požadavků za minutu.

9.3 Budoucí vývoj

Z hlediska implementace je plánováno přidat zbývající validační pravidla z doporučení *CSV on the Web*.

Tabulková data mohou mít různý dialekt, který aktuálně není podporován. V budoucnu by se měl vytvořit parser pro tyto dialekty. Parser musí dědit z rozhraní `TabularDataParser`.

JSON-LD parser již zvládne správně parsovat cizí klíče, tudíž zbývá dodělat jejich logika. Při validaci modelu v `DefaultModelValidator` je potřeba tyto cizí klíče zpracovat a vytvořit pro ně odpovídající implementace validačních pravidel, viz Kapitola 6.4.3.

Pro transformace není v aktuální verzi validátoru žádná podpora. Nejdříve je potřeba upravit JSON-LD parser podle Kapitoly 6.4.2.2 a poté přidat další validační pravidla z Kapitoly 6.4.3.

Mimo validační pravidla, by bylo dobré dopsat zbývající unit a integrační testy. Důvodem je, že ani referenční testy z doporučení *CSV on the Web* nepokrývají všechny vstupy a možnosti. Určitě by se našli i části kódu, které by šli refaktorovat, a tím by se dosáhlo čistějšího kódu.

Z výkonnostního testování vyšlo najevo, že nejpomalejší část je validace modelu. Ta funguje iterativně po jednotlivých řádcích tabulky a šlo by ji snadno paralelizovat. Pokud bude potřeba zvýšit výkon webové aplikace či webové služby, doporučil bych vytvoření nové více uzlové aplikace. Tedy aplikace, která by si validace řadila do fronty, z které by je více výpočetních jednotek vybíralo a zpracovávalo.

Závěr

Cílem této práce bylo vytvoření validátoru CSV souborů dle W3C doporučení *CSV on the Web*. Zmíněné doporučení definuje, jak ukládat tabulková data ve formátu CSV a jak k nim připojit datové schéma v podobě JSON-LD deskriptorů. Validátor má za úkol kontrolovat formát CSV souborů, najít pro ně datové schéma a zkontrolovat, zda tabulková data splňují nalezené schéma. Je-li porušeno některé z pravidel formátů nebo doporučení, musí o nich validátor informovat.

Již existují referenční implementace validátorů dle doporučení *CSV on the Web*, nicméně se jedná o neudržované a pro použití obtížné aplikace. Cílem práce byla tedy také rešerše stávajících řešení, v níž bylo zjištěno, že žádný z existujících validátorů není schopen používat IRI s mezinárodními znaky (např. s českou diakritikou), většinu z nich nelze zprovoznit na operačním systému Windows 10 a jediná webová aplikace, jež umožňuje validování CSV souborů, nepodporuje schémata z doporučení *CSV on the Web*.

Validátor bylo nutné implementovat jako knihovnu v programovacím jazyku Java. Cílem návrhu validátoru byla architektura, jež umožní jednoduché přidávání dalších validačních pravidel. Kromě knihovny byly vytvořeny spouštěče validací, které používají zmíněnou knihovnu. Konkrétně se jedná o spouštěč z příkazového řádku, REST webové služby a webové aplikace. Architektura, tři možnosti spouštění validací a jejich podrobná dokumentace mají za následek funkční validátor, který je postaven na moderní technologii a který lze snadno používat na různých platformách.

Knihovna je připravená na případná následná vylepšení. Z doporučení *CSV on the Web* by bylo příp. možné implementovat parsování různých dialektů tabulkových dat. Validátor aktuálně podporuje jen tabulková data ve formátu CSV dle RFC 4180 a dle *CSV on the Web*. Z validačních pravidel schématu chybějí jen dvě oblasti – cizí klíče a transformace – které však lze vzhledem k architektuře validátoru v budoucnu jednoduše doimplementovat. Validátor splňuje 262 z 281 referenčních testů doporučení *CSV on the Web*.

Literatura

[1] Tennison, J.; Kellogg, G.; Herman, I.: Model for Tabular Data and Metadata on the Web. [online], Prosinec 2015, [cit. 2019-01-10]. Dostupné z: https://www.w3.org/TR/2015/REC-tabular-data-model-20151217/

[2] Tennison, J.; Kellogg, G.; Herman, I.: Diagram showing the built-in datatypes, based on XML Schema. [online], Prosinec 2015, [cit. 2019-05-01]. Dostupné z: https://www.w3.org/TR/2015/REC-tabular-data-model-20151217/datatypes.svg

[3] CSVLint. [online], [cit. 2019-05-03]. Dostupné z: https://csvlint.io/

[4] Shafranovich, Y.: Common Format and MIME Type for Comma-Separated Values (CSV) Files. [online], Říjen 2005, [cit. 2019-01-10]. Dostupné z: https://tools.ietf.org/html/rfc4180/

[5] Kellogg, G.; Associates, K.: CSVW Implementation Report. [online], Říjen 2015, [cit. 2019-02-02]. Dostupné z: http://w3c.github.io/csvw/publishing-snapshots/PR-earl/earl.html

[6] Walsh, P.; Pollock, R.: Table Schema. [online], [cit. 2019-05-03]. Dostupné z: https://frictionlessdata.io/specs/table-schema/

[7] Pollock, R.; Tennison, J.; Kellogg, G.; aj.: Model for Tabular Data and Metadata on the Web. [online], Prosinec 2015, [cit. 2019-01-10]. Dostupné z: https://www.w3.org/TR/2015/REC-tabular-metadata-20151217/

[8] Skřivan, J.: Datové modely a návrhy relačních schémat. [online], Listopad 2008, [cit. 2019-04-30]. Dostupné z: https://sites.ff.cuni.cz/uisk/wp-content/uploads/sites/62/2016/01/Datov%C3%A9-modely-a-n%C3%A1vrhy-rela%C4%8Dn%C3%ADch-sch%C3%A9mat_Sk%C5%99ivan.pdf

[9] Crocker, D.; Overell, P.: Augmented BNF for Syntax Specifications: ABNF. [online], Listopad 1997, [cit. 2019-04-30]. Dostupné z: https://tools.ietf.org/html/rfc2234

[10] Davis, M.: Unicode Locale Data Markup Language (LDML). [online], Březen 2013, [cit. 2019-01-10]. Dostupné z: http://www.unicode.org/reports/tr35/tr35-31/tr35.html

[11] Bray, T.; Paoli, J.; Sperberg-McQueen, C. M.; aj.: EBNF Notation. [online], Prosinec 2008, [cit. 2019-01-10]. Dostupné z: http://www.w3.org/TR/xml

[12] Sporny, M.; Longley, D.; Kellogg, G.; aj.: JSON-LD 1.0. [online], Leden 2014, [cit. 2019-01-17]. Dostupné z: https://www.w3.org/TR/json-ld/

[13] Duerst, M.; Suignard, M.: Internationalized Resource Identifiers (IRIs). [online], Leden 2005, [cit. 2019-01-17]. Dostupné z: https://www.ietf.org/rfc/rfc3987.txt

[14] Berners-Lee, T.; Fielding, R.; Masinter, L.: Uniform Resource Identifier (URI): Generic Syntax. [online], Leden 2005, [cit. 2019-01-15]. Dostupné z: https://tools.ietf.org/html/rfc3986

[15] Crockford, D.: The application/json Media Type for JavaScript Object Notation (JSON). [online], Červen 2006, [cit. 2019-01-17]. Dostupné z: https://www.ietf.org/rfc/rfc4627.txt

[16] schema.org. [online], [cit. 2019-02-09]. Dostupné z: https://schema.org/

[17] Kellogg, G.: CSVW Namespace Vocabulary Terms. [online], Červen 2017, [cit. 2019-01-18]. Dostupné z: https://www.w3.org/ns/csvw

[18] Peterson, D.; Gao, S.; Malhotra, A.; aj.: W3C XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes. [online], Duben 2012, [cit. 2019-01-14]. Dostupné z: https://www.w3.org/TR/xmlschema11-2/

[19] ECMAScript Language Specification. [online], Leden 2019, [cit. 2019-01-16]. Dostupné z: https://tc39.github.io/ecma262/

[20] Phillips, A.; Davis, M.: Tags for Identifying Languages. [online], Září 2009, [cit. 2019-01-13]. Dostupné z: https://tools.ietf.org/html/bcp47

[21] Gregorio, J.; Fielding, R.; Hadley, M.; aj.: URI Template. [online], Březen 2012, [cit. 2019-01-15]. Dostupné z: https://tools.ietf.org/html/rfc6570

[22] DCMI Metadata Terms. [online], Červen 2012, [cit. 2019-02-09]. Dostupné z: http://dublincore.org/documents/dcmi-terms/

[23] Maali, F.; Erickson, J.: Data Catalog Vocabulary. [online], Leden 2014, [cit. 2019-02-09]. Dostupné z: https://www.w3.org/TR/vocab-dcat/

[24] Nottingham, M.; Hammer-Lahav, E.: Defining Well-Known Uniform Resource Identifiers (URIs). [online], Duben 2010, [cit. 2019-01-15]. Dostupné z: https://tools.ietf.org/html/rfc5785

[25] CSV Lint. [online], [cit. 2019-02-02]. Dostupné z: https://github.com/theodi/csvlint.rb

[26] Neumaier, S.; Umbrich, J.; Li, M.: Python implementation of the W3C CSV on the Web specification. [online], [cit. 2019-02-02]. Dostupné z: https://github.com/sebneu/csvw-parser

[27] Kellogg, G.: Tabular Data RDF Reader and JSON serializer. [online], [cit. 2019-02-02]. Dostupné z: https://github.com/ruby-rdf/rdf-tabular

[28] Klímek, J.: Formát CSV. [online], Únor 2019, [cit. 2019-04-18]. Dostupné z: https://opendata.gov.cz/standardy:csv

[29] Beckett, D.; Berners-Lee, T.; Prud'hommeaux, E.; aj.: RDF 1.1 Turtle. [online], Únor 2014, [cit. 2019-04-18]. Dostupné z: https://www.w3.org/TR/turtle/

[30] Project Lombok. [online], [cit. 2019-04-18]. Dostupné z: https://projectlombok.org/

[31] Spring. [online], [cit: 2019-04-18]. Dostupné z: http://spring.io/

[32] Spring Boot. [online], [cit: 2019-04-18]. Dostupné z: https://spring.io/projects/spring-boot

[33] H2 Database Engine. [online], [cit: 2019-05-08]. Dostupné z: https://www.h2database.com/html/main.html

[34] Vaadin. [online], [cit: 2019-04-18]. Dostupné z: https://vaadin.com/

[35] Spock. [online], [cit: 2019-04-18]. Dostupné z: http://spockframework.org/

[36] Apache Maven. [online], Duben 2019, [cit: 2019-04-18]. Dostupné z: https://maven.apache.org/

[37] IntelliJ IDEA. [online], [cit: 2019-04-18]. Dostupné z: https://www.jetbrains.com/idea/

[38] univocity-parsers. [online], [cit: 2019-04-18]. Dostupné z: https://www.univocity.com/pages/univocity_parsers_tutorial

[39] Apache Commons CSV. [online], Leden 2019, [cit: 2019-04-18]. Dostupné z: https://commons.apache.org/proper/commons-csv/

[40] Jackson Project. [online], [cit: 2019-04-18]. Dostupné z: https://github.com/FasterXML/jackson

[41] Edens, E.: urllib. [online], [cit: 2019-04-18]. Dostupné z: https://github.com/EricEdens/urllib

[42] Handy URI Templates. [online], [cit: 2019-04-18]. Dostupné z: https://github.com/damnhandy/Handy-URI-Templates

[43] Apache Jena. [online], [cit: 2019-04-18]. Dostupné z: https://jena.apache.org/

[44] Evaluation and Report Language. [online], Únor 2017, [cit: 2019-04-18]. Dostupné z: https://www.w3.org/TR/EARL10-Schema/

[45] Commons CLI. [online], Únor 2019, [cit: 2019-04-18]. Dostupné z: https://commons.apache.org/proper/commons-cli/

[46] RESTful API Modeling Language (RAML). [online], [cit: 2019-04-22]. Dostupné z: https://raml.org/

[47] Apache JMeter. [online], [cit: 2019-05-08]. Dostupné z: https://jmeter.apache.org/

Příloha A

Seznam použitých zkratek

API Application Programming Interface

ASCII American Standard Code for Information Interchange

CSV Comma-Separated Values

GUI Graphical User Interface

HTML Hypertext Markup Language

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol Secure

IRI Internationalized Resource Identifier

JSON JavaScript Object Notation

JSON-LD JavaScript Object Notation for Linked Data

JVM Java Virtual Machine

RDF Resource Description Framework

SQL Structured Query Language

TCP Transmission Control Protocol

TSV Tab-Separated Values

UI User Interface

URI Uniform Resource Identifier

URL Uniform Resource Locator

UTF Unicode Transformation Format

W3C World Wide Web Consortium

XML Extensible Markup Language

Příloha B

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `DP_Maly_Vojtech_2019.pdf` ..... text práce ve formátu PDF

Příloha C

Ukázka výsledku reprezentovaného v RDF

Příklad C.1: Výsledek validátoru reprezentovaný v RDF v serializaci Turtle

```
@prefix doap:  <http://usefulinc.com/ns/doap#> .
@prefix rdf:
   <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xsd:   <http://www.w3.org/2001/XMLSchema#> .
@prefix earl:  <http://www.w3.org/ns/earl#> .
@prefix rdfs:
   <http://www.w3.org/2000/01/rdf-schema#> .
@prefix foaf:  <http://xmlns.com/foaf/0.1/> .
@prefix dc:    <http://purl.org/dc/elements/1.1/> .

[ earl:assertedBy
   <https://github.com/malyvoj3/csvw-validator> ;
  earl:result  [ a             earl:TestResult ;
                 dc:date
                    "2019-04-21T23:19:25.007Z"
                 ^^xsd:dateTime ;
                 earl:info     "Cell (row 2 column 1)
                    cannot be formatted as 'integer'
                    datatype."@en ;
                 earl:mode     earl:automatic ;
                 earl:outcome  earl:failed
               ] ;
  earl:result  [ a              earl:TestResult ;
                 dc:date
                    "2019-04-21T23:19:25.007Z"
                 ^^xsd:dateTime ;
                 dc:description "Summary result of
```

```
                    validation by csvw-validator."@en
                    ;
                  dc:title       "Validation
                    result"@en ;
                  earl:mode      earl:automatic ;
                  earl:outcome   earl:failed
                ] ;
  earl:subject
     <https://github.com/malyvoj3/csvw-validator> ;
  earl:test    <http://www.w3.org/2013/csvw/tests/
  test286-metadata.json> ,
     <http://www.w3.org/2013/csvw/tests/test286.csv>
] .

<https://github.com/malyvoj3/csvw-validator#fatal>
  a              earl:Fail ;
  dc:description "Failed which caused stopping of
     validating"@en ;
  dc:title       "Fatal"@en .

<https://github.com/malyvoj3/csvw-validator#warning>
  a              earl:Pass ;
  dc:description "Warning"@en ;
  dc:title       "Warning"@en .

<https://github.com/malyvoj3/csvw-validator>
  a                  earl:Software , earl:TestSubject
     , earl:Assertor , doap:Project ;
  dc:creator         <https://github.com/malyvoj3> ;
  dc:description     "Java implementation of the W3C
     CSV on the Web validator."@en ;
  dc:title           "csvw-validator"@en ;
  doap:bug-database
     <https://github.com/malyvoj3/csvw-validator/issues>
     ;
  doap:description   "Java implementation of the W3C
     CSV on the Web validator."@en ;
  doap:developer     <https://github.com/malyvoj3> ;
  doap:documenter    <https://github.com/malyvoj3> ;
  doap:download-page
     <https://github.com/malyvoj3/csvw-validator> ;
  doap:homepage
     <https://github.com/malyvoj3/csvw-validator> ;
  doap:implements
     <http://www.w3.org/TR/tabular-data-model/> ,
```

```
      <http://www.w3.org/TR/tabular-metadata/> ;
  doap:license
      <https://choosealicense.com/licenses/mit/> ;
  doap:maintainer    <https://github.com/malyvoj3> ;
  doap:name          "csvw-validator"@en ;
  foaf:maker         <https://github.com/malyvoj3> .

<https://github.com/malyvoj3>
  a             foaf:Person ;
  foaf:homepage <https://github.com/malyvoj3> ;
  foaf:name     "Vojtech Maly" ;
  foaf:title    "Implementor"@en .

<https://github.com/malyvoj3/
    csvw-validator#strict-warning>
  a              earl:Pass ;
  dc:description "Warning in strict mode"@en ;
  dc:title       "Strict warning"@en .

<https://github.com/malyvoj3/
    csvw-validator#strict-mode>
  a              earl:TestMode ;
  dc:description "This mode enables STRICT_WARNING,
     which validates format of CSV againt RFC
     4180"@en ;
  dc:title       "Strict mode"@en .
```

Příloha D

Výsledky referenčních testů

Tabulka D.1: Výsledky referenčních validátorů a csvw-validator v referenční testech [5]

Test	CSVlint	pycsvw	RDF::Tabular	csvw-validator
Test 001	PASS	PASS	PASS	PASS
Test 005	PASS	PASS	PASS	PASS
Test 006	PASS	PASS	PASS	PASS
Test 007	PASS	PASS	PASS	PASS
Test 008	PASS	PASS	PASS	PASS
Test 009	PASS	PASS	PASS	PASS
Test 010	PASS	PASS	PASS	PASS
Test 011	PASS	PASS	PASS	PASS
Test 012	PASS	PASS	PASS	PASS
Test 013	PASS	PASS	PASS	PASS
Test 014	PASS	PASS	PASS	PASS
Test 015	PASS	PASS	PASS	PASS
Test 016	PASS	PASS	PASS	PASS
Test 017	PASS	PASS	PASS	PASS
Test 018	PASS	PASS	PASS	PASS
Test 023	PASS	PASS	PASS	FAIL
Test 027	PASS	PASS	PASS	PASS
Test 028	PASS	PASS	PASS	PASS
Test 029	PASS	PASS	PASS	PASS
Test 030	PASS	PASS	PASS	PASS
Test 031	PASS	PASS	PASS	PASS
Test 032	PASS	PASS	PASS	PASS
Test 033	PASS	PASS	PASS	PASS
Test 034	PASS	PASS	PASS	FAIL
Test 035	PASS	PASS	PASS	FAIL

Tabulka D.1: Výsledky referenčních validátorů a csvw-validator v referenčních testech [5]

Test	CSVlint	pycsvw	RDF::Tabular	csvw-validator
Test 036	PASS	PASS	PASS	PASS
Test 037	PASS	PASS	PASS	PASS
Test 038	PASS	PASS	PASS	PASS
Test 039	PASS	PASS	PASS	PASS
Test 040	PASS	PASS	PASS	PASS
Test 041	PASS	PASS	PASS	PASS
Test 042	PASS	PASS	PASS	PASS
Test 043	PASS	PASS	PASS	PASS
Test 044	PASS	PASS	PASS	PASS
Test 045	PASS	PASS	PASS	PASS
Test 046	PASS	PASS	PASS	PASS
Test 047	PASS	PASS	PASS	PASS
Test 048	PASS	PASS	PASS	PASS
Test 049	PASS	PASS	PASS	PASS
Test 059	PASS	PASS	PASS	PASS
Test 060	PASS	PASS	PASS	PASS
Test 061	PASS	PASS	PASS	PASS
Test 062	PASS	PASS	PASS	FAIL
Test 063	PASS	PASS	PASS	PASS
Test 065	PASS	PASS	PASS	PASS
Test 066	PASS	PASS	PASS	PASS
Test 067	PASS	PASS	PASS	PASS
Test 068	PASS	PASS	PASS	PASS
Test 069	PASS	PASS	PASS	PASS
Test 070	PASS	PASS	PASS	PASS
Test 071	PASS	PASS	PASS	PASS
Test 072	PASS	PASS	PASS	PASS
Test 073	PASS	PASS	PASS	PASS
Test 074	PASS	PASS	PASS	PASS
Test 075	PASS	PASS	PASS	PASS
Test 076	PASS	PASS	PASS	PASS
Test 077	PASS	PASS	PASS	PASS
Test 078	PASS	PASS	PASS	PASS
Test 079	PASS	PASS	PASS	PASS
Test 080	PASS	PASS	PASS	PASS
Test 081	PASS	FAIL	PASS	PASS
Test 082	PASS	FAIL	PASS	FAIL
Test 083	PASS	PASS	PASS	PASS
Test 084	PASS	PASS	PASS	PASS
Test 085	PASS	PASS	PASS	PASS

Tabulka D.1: Výsledky referenčních validátorů a csvw-validator v referenčních testech [5]

Test	CSVlint	pycsvw	RDF::Tabular	csvw-validator
Test 086	PASS	PASS	PASS	PASS
Test 087	PASS	FAIL	PASS	PASS
Test 088	PASS	FAIL	PASS	FAIL
Test 089	PASS	PASS	PASS	PASS
Test 090	PASS	PASS	PASS	PASS
Test 092	PASS	PASS	PASS	PASS
Test 093	PASS	PASS	PASS	PASS
Test 094	PASS	PASS	PASS	PASS
Test 095	PASS	PASS	PASS	FAIL
Test 096	PASS	PASS	PASS	PASS
Test 097	PASS	PASS	PASS	PASS
Test 098	PASS	PASS	PASS	PASS
Test 099	PASS	PASS	PASS	FAIL
Test 100	PASS	PASS	PASS	PASS
Test 101	PASS	PASS	PASS	PASS
Test 102	PASS	PASS	PASS	PASS
Test 103	PASS	FAIL	PASS	PASS
Test 104	PASS	FAIL	PASS	PASS
Test 105	PASS	PASS	PASS	PASS
Test 106	PASS	PASS	PASS	PASS
Test 107	PASS	PASS	PASS	PASS
Test 108	PASS	FAIL	PASS	PASS
Test 109	PASS	PASS	PASS	PASS
Test 110	PASS	PASS	PASS	PASS
Test 111	PASS	PASS	PASS	PASS
Test 112	PASS	PASS	PASS	PASS
Test 113	PASS	PASS	PASS	PASS
Test 114	PASS	PASS	PASS	PASS
Test 115	PASS	PASS	PASS	PASS
Test 116	PASS	PASS	PASS	PASS
Test 117	PASS	PASS	PASS	PASS
Test 118	PASS	PASS	PASS	PASS
Test 119	PASS	PASS	PASS	PASS
Test 120	PASS	PASS	PASS	FAIL
Test 121	PASS	PASS	PASS	PASS
Test 122	PASS	PASS	PASS	FAIL
Test 123	PASS	PASS	PASS	PASS
Test 124	PASS	FAIL	PASS	PASS
Test 125	PASS	FAIL	PASS	PASS
Test 126	PASS	FAIL	PASS	PASS

Tabulka D.1: Výsledky referenčních validátorů a csvw-validator v referenčních testech [5]

Test	CSVlint	pycsvw	RDF::Tabular	csvw-validator
Test 127	PASS	FAIL	PASS	PASS
Test 128	PASS	PASS	PASS	PASS
Test 129	PASS	PASS	PASS	PASS
Test 130	PASS	PASS	PASS	PASS
Test 131	PASS	PASS	PASS	PASS
Test 132	PASS	PASS	PASS	PASS
Test 133	PASS	FAIL	PASS	PASS
Test 134	PASS	FAIL	PASS	PASS
Test 135	PASS	FAIL	PASS	PASS
Test 136	PASS	FAIL	PASS	PASS
Test 137	PASS	FAIL	PASS	PASS
Test 138	PASS	FAIL	PASS	PASS
Test 139	PASS	FAIL	PASS	PASS
Test 140	PASS	FAIL	PASS	PASS
Test 141	PASS	FAIL	PASS	PASS
Test 142	PASS	FAIL	PASS	PASS
Test 143	PASS	FAIL	PASS	PASS
Test 144	PASS	FAIL	PASS	PASS
Test 145	PASS	FAIL	PASS	PASS
Test 146	PASS	FAIL	PASS	PASS
Test 147	PASS	FAIL	PASS	PASS
Test 148	PASS	FAIL	PASS	FAIL
Test 149	PASS	PASS	PASS	PASS
Test 150	PASS	PASS	PASS	PASS
Test 151	PASS	PASS	PASS	PASS
Test 152	PASS	PASS	PASS	PASS
Test 153	PASS	PASS	PASS	PASS
Test 154	PASS	FAIL	PASS	PASS
Test 155	PASS	PASS	PASS	PASS
Test 156	PASS	PASS	PASS	PASS
Test 157	PASS	FAIL	PASS	PASS
Test 158	PASS	PASS	PASS	PASS
Test 159	PASS	PASS	PASS	PASS
Test 160	PASS	FAIL	PASS	PASS
Test 161	PASS	FAIL	PASS	PASS
Test 162	PASS	FAIL	PASS	PASS
Test 163	PASS	FAIL	PASS	PASS
Test 164	PASS	FAIL	PASS	PASS
Test 165	PASS	FAIL	PASS	PASS
Test 166	PASS	FAIL	PASS	PASS

Tabulka D.1: Výsledky referenčních validátorů a csvw-validator v referenčních testech [5]

Test	CSVlint	pycsvw	RDF::Tabular	csvw-validator
Test 167	PASS	FAIL	PASS	PASS
Test 168	PASS	PASS	PASS	PASS
Test 169	PASS	FAIL	PASS	PASS
Test 170	PASS	PASS	PASS	PASS
Test 171	PASS	PASS	PASS	PASS
Test 172	PASS	FAIL	PASS	PASS
Test 173	PASS	FAIL	PASS	PASS
Test 174	PASS	FAIL	PASS	PASS
Test 175	PASS	FAIL	PASS	PASS
Test 176	PASS	FAIL	PASS	PASS
Test 177	PASS	FAIL	PASS	PASS
Test 178	PASS	FAIL	PASS	PASS
Test 179	PASS	FAIL	PASS	PASS
Test 180	PASS	FAIL	PASS	PASS
Test 181	PASS	FAIL	PASS	PASS
Test 182	PASS	FAIL	PASS	PASS
Test 183	PASS	PASS	PASS	PASS
Test 184	PASS	PASS	PASS	FAIL
Test 185	PASS	FAIL	PASS	PASS
Test 186	PASS	FAIL	PASS	PASS
Test 187	PASS	PASS	PASS	PASS
Test 188	PASS	PASS	PASS	PASS
Test 189	PASS	PASS	PASS	PASS
Test 190	PASS	PASS	PASS	PASS
Test 191	PASS	FAIL	PASS	FAIL
Test 192	PASS	FAIL	PASS	PASS
Test 193	PASS	PASS	PASS	PASS
Test 194	PASS	FAIL	PASS	PASS
Test 195	PASS	PASS	PASS	PASS
Test 196	PASS	FAIL	PASS	PASS
Test 197	PASS	FAIL	PASS	PASS
Test 198	PASS	FAIL	PASS	PASS
Test 199	PASS	FAIL	PASS	PASS
Test 200	PASS	FAIL	PASS	PASS
Test 201	PASS	FAIL	PASS	PASS
Test 202	PASS	PASS	PASS	PASS
Test 203	PASS	FAIL	PASS	PASS
Test 204	PASS	FAIL	PASS	PASS
Test 205	PASS	FAIL	PASS	PASS
Test 206	PASS	FAIL	PASS	PASS

Tabulka D.1: Výsledky referenčních validátorů a csvw-validator v referenčních testech [5]

Test	CSVlint	pycsvw	RDF::Tabular	csvw-validator
Test 207	PASS	FAIL	PASS	PASS
Test 208	PASS	FAIL	PASS	PASS
Test 209	PASS	PASS	PASS	PASS
Test 210	PASS	FAIL	PASS	PASS
Test 211	PASS	FAIL	PASS	PASS
Test 212	PASS	FAIL	PASS	PASS
Test 213	PASS	FAIL	PASS	PASS
Test 214	PASS	FAIL	PASS	PASS
Test 215	PASS	FAIL	PASS	PASS
Test 216	PASS	FAIL	PASS	PASS
Test 217	PASS	FAIL	PASS	PASS
Test 218	PASS	FAIL	PASS	PASS
Test 219	PASS	FAIL	PASS	PASS
Test 220	PASS	FAIL	PASS	PASS
Test 221	PASS	FAIL	PASS	PASS
Test 222	PASS	FAIL	PASS	PASS
Test 223	PASS	FAIL	PASS	PASS
Test 224	PASS	FAIL	PASS	PASS
Test 225	PASS	FAIL	PASS	PASS
Test 226	PASS	FAIL	PASS	PASS
Test 227	PASS	FAIL	PASS	PASS
Test 228	PASS	PASS	PASS	PASS
Test 229	PASS	PASS	PASS	PASS
Test 230	PASS	FAIL	PASS	PASS
Test 231	PASS	PASS	PASS	PASS
Test 232	PASS	FAIL	PASS	PASS
Test 233	PASS	PASS	PASS	PASS
Test 234	PASS	FAIL	PASS	PASS
Test 235	PASS	PASS	PASS	PASS
Test 236	PASS	PASS	PASS	PASS
Test 237	PASS	PASS	PASS	PASS
Test 238	PASS	PASS	PASS	PASS
Test 242	PASS	PASS	PASS	PASS
Test 243	PASS	FAIL	PASS	PASS
Test 244	PASS	FAIL	PASS	PASS
Test 245	PASS	PASS	PASS	PASS
Test 246	PASS	PASS	PASS	PASS
Test 247	PASS	FAIL	PASS	PASS
Test 248	PASS	PASS	PASS	PASS
Test 249	PASS	PASS	PASS	PASS

Tabulka D.1: Výsledky referenčních validátorů a csvw-validator v referenčních testech [5]

Test	CSVlint	pycsvw	RDF::Tabular	csvw-validator
Test 250	PASS	PASS	PASS	PASS
Test 251	PASS	FAIL	PASS	FAIL
Test 252	PASS	FAIL	PASS	FAIL
Test 253	PASS	FAIL	PASS	FAIL
Test 254	PASS	PASS	PASS	PASS
Test 255	PASS	PASS	PASS	PASS
Test 256	PASS	PASS	PASS	PASS
Test 257	PASS	FAIL	PASS	FAIL
Test 258	PASS	FAIL	PASS	FAIL
Test 259	PASS	PASS	PASS	PASS
Test 260	PASS	PASS	PASS	PASS
Test 261	PASS	FAIL	PASS	PASS
Test 263	PASS	PASS	PASS	PASS
Test 264	PASS	PASS	PASS	PASS
Test 266	PASS	PASS	PASS	PASS
Test 267	PASS	FAIL	PASS	PASS
Test 268	PASS	PASS	PASS	PASS
Test 269	PASS	FAIL	PASS	PASS
Test 270	PASS	PASS	PASS	FAIL
Test 271	PASS	FAIL	PASS	PASS
Test 272	PASS	FAIL	PASS	PASS
Test 273	PASS	PASS	PASS	PASS
Test 274	PASS	PASS	PASS	PASS
Test 275	PASS	PASS	PASS	PASS
Test 276	PASS	PASS	PASS	PASS
Test 277	PASS	PASS	PASS	PASS
Test 278	PASS	FAIL	PASS	PASS
Test 279	PASS	FAIL	PASS	PASS
Test 280	PASS	FAIL	PASS	PASS
Test 281	PASS	FAIL	PASS	PASS
Test 282	PASS	PASS	PASS	PASS
Test 283	PASS	PASS	PASS	PASS
Test 284	PASS	PASS	PASS	PASS
Test 285	PASS	PASS	PASS	PASS
Test 286	PASS	FAIL	PASS	PASS
Test 287	PASS	FAIL	PASS	PASS
Test 288	PASS	FAIL	PASS	PASS
Test 289	PASS	FAIL	PASS	PASS
Test 290	PASS	FAIL	PASS	PASS
Test 291	PASS	FAIL	PASS	PASS

Tabulka D.1: Výsledky referenčních validátorů a csvw-validator v referenčních testech [5]

Test	CSVlint	pycsvw	RDF::Tabular	csvw-validator
Test 292	PASS	FAIL	PASS	PASS
Test 293	PASS	FAIL	PASS	PASS
Test 294	PASS	FAIL	PASS	PASS
Test 295	PASS	FAIL	PASS	PASS
Test 296	PASS	FAIL	PASS	PASS
Test 297	PASS	FAIL	PASS	PASS
Test 298	PASS	FAIL	PASS	PASS
Test 299	PASS	FAIL	PASS	PASS
Test 300	PASS	FAIL	PASS	PASS
Test 301	PASS	FAIL	PASS	PASS
Test 302	PASS	FAIL	PASS	PASS
Test 303	PASS	FAIL	PASS	PASS
Test 304	PASS	FAIL	PASS	PASS
Test 305	PASS	PASS	PASS	PASS
Test 306	PASS	PASS	PASS	PASS
Test 307	PASS	PASS	PASS	PASS
Percentage	100.0%	56.2%	100.0%	93.2%