Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Automatické zpracování dat z meteostanic

Plzeň 2020 Tomáš Pašek

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta aplikovaných věd
Akademický rok: 2019/2020

ZADÁNÍ BAKALÁŘSKÉ PRÁCE
(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Tomáš PAŠEK**
Osobní číslo:	**A17B0319P**
Studijní program:	**B3902 Inženýrská informatika**
Studijní obor:	**Informatika**
Téma práce:	**Automatické zpracování dat z meteostanic**
Zadávající katedra:	**Katedra informatiky a výpočetní techniky**

Zásady pro vypracování

1. Prostudujte zpracování dat regulárními výrazy a jazykem XPath a formáty dat poskytovanými běžnými typy meteostanic.
2. Navrhněte metodu pro generování regulárních a XPath výrazů na základě dat z meteostanic.
3. Implementujte ve zvoleném jazyce tuto metodu a otestujte ji na reálných datech z meteostanic.
4. Dosažené výsledky zhodnoťte.

Rozsah bakalářské práce: **doporuč. 30 s. původního textu**
Rozsah grafických prací: **dle potřeby**
Forma zpracování bakalářské práce: **tištěná**

Seznam doporučené literatury:

Dodá vedoucí bakalářské práce.

Vedoucí bakalářské práce: **Ing. Martin Prantl**
Nové technologie pro informační společnost

Datum zadání bakalářské práce: **7. října 2019**
Termín odevzdání bakalářské práce: **7. května 2020**

Západočeská univerzita v Plzni
Fakulta aplikovaných věd
L.S.

Doc. Dr. Ing. Vlasta Radová
děkanka

Doc. Ing. Přemysl Brada, MSc., Ph.D.
vedoucí katedry

V Plzni dne 15. října 2019

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 6. května 2020

Tomáš Pašek

Abstract

The goal of this study is to research data processing using regular expressions and XPath language and create method for automatic generating of this expressions for automatic data processing from weather stations. Regular expressions are used to validation and parsing of strings. They are supported in many programming languages (for example Java, C or PHP). Their syntax is not same in all of them, but differences are not so big, so they can be often easily modified for different language. XPath language is used for for data processing from XML files. There is also support for XPath in many programming languages. Expressions in this study will be generated for PHP language.

Abstrakt

Cílem této bakalářské práce je prostudovat zpracování dat regulárními výrazy a jazykem XPath a vytvořit metodu na automatické generování těchto výrazů pro automatické zpracovávání dat z meteostanic. Regulární výrazy obecně slouží k validaci nebo parsování textových řetězců. Jsou podporovány v mnoha programovacích jazycích (např. Java, C nebo PHP). Jejich syntax se sice v jednotlivých jazycích liší, ale rozdíly nejsou příliš významné, tudíž se dají většinou lehce upravit i do jiného jazyka. Jazyk XPath slouží pro získávání dat z XML souborů. I pro něj je k dispozici podpora mezi mnoha programovacími jazyky. V této práci budou výrazy vytvářeny pro jazyk PHP.

Poděkování

Chtěl bych poděkovat panu Ing. Martinu Prantlovi, Ph.D. za odborné vedení, cenné rady a vstřícnost při konzultacích během vypracovávání bakalářské práce.

Obsah

1 Úvod **8**

2 Regulární výrazy **9**

3 XPath **12**

4 Analýza zpracování meteorologických dat **15**
4.1 Zpracování pomocí regulárních výrazů 15
4.2 Zpracování pomocí XPath 16

5 Zvolené řešení zpracování meteorologických dat **17**
5.1 Generování regulárních výrazů 17
5.2 Generování XPath výrazů 19
5.3 Aplikace pro generování výrazů 20
5.3.1 Balíček application 22
5.3.2 Balíček parsers 22
5.3.3 Balíček constants 23
5.4 Aplikace pro zobrazení dat 23

6 Výsledky **27**
6.1 Úvod 27
6.2 Regulární výrazy 27
6.2.1 Testování 27
6.2.2 Doby běhů 29
6.3 XPath výrazy 31
6.3.1 Testování 31
6.3.2 Doby běhů 34
6.4 Zpracování dat 36

7 Závěr **38**

Literatura **39**

Uživatelská příručka **40**

1 Úvod

Meteorologické stanice poskytují výstupní data v různých formátech. Většinou mají textovou podobu, nicméně se liší svým formátováním. Některé jsou v CSV formátu, jiné v JSON a další například v XML nebo HTML. Existují i některé amatérské stanice, které mají své vlastní formátování. Jako univerzální řešení, které by bylo schopné zpracovat textový soubor v libovolném formátu, se jeví použití regulárních nebo XPath výrazů.

Regulární výrazy slouží k validování nebo parsování textových řetězců. Používají k tomu zástupné znaky, pomocí kterých lze popsat strukturu libovolného textového řetězce. Díky tomu je možné definovat, jak má takový řetězec vypadat, případně získat z většího řetězce nějakou jeho podčást.

XPath je jazyk, pomocí kterého je možné přistupovat k elementům v XML souborech a pracovat s jejich hodnotami, případně atributy. Používá k tomu vlastní výrazy, pomocí kterých je možné daný element popsat a případně určit, jestli se má pracovat s hodnotou nějakého jeho atributu.

V této práci bude vytvořena aplikace pro automatické generování regulárních a XPath výrazů pro automatické zpracování dat z meteostanic. V této aplikaci uživatel otevře výstupní soubor své meteostanice, vybere údaje, které chce ze souboru pomocí výrazu automaticky extrahovat, a aplikace vygeneruje výraz, který daná data (např. teplotu vzduchu) extrahuje. Poté je možné tento výraz používat i pro všechny ostatní výstupní soubory z dané meteostanice za předpokladu, že daná meteostanice poskytuje data v konzistentní podobě a nemanipuluje například s pořadím jednotlivých elementů.

Jako další bude vytvořena PHP aplikace, ve které bude možné data zpracovaná výrazy zobrazit. Zde uživatel nahraje soubor s daty z meteostanice, poté regulární nebo XPath výraz a aplikace mu pomocí daného výrazu z dat zobrazí pouze požadované údaje.

Pro generování těchto výrazů je nutné vymyslet způsob, jak automaticky generovat regulární nebo XPath výraz tak, aby fungoval pro všechna data, která budou držet danou strukturu. Výraz musí fungovat navzdory různému počtu znaků u jednotlivých hodnot a nemůže se pouze opírat o index daného znaku v textu.

2 Regulární výrazy

„Regulární výraz je textový řetězec chápaný jako vzor (maska) pro vyhledávání v textových souborech. Syntaxe regulárních výrazů zahrnuje užití nejen zástupných znaků (např. hvězdička, otazník), ale i řadu dalších zvláštních znaků a pravidel, které umožňují zápis komplexních vyhledávacích požadavků."[10]

Regulární výrazy zavedl americký matematik Stephen Cole Kleene, kdy vytvořil koncept regulárního jazyka. Momentálně jsou regulární výrazy podporovány celou řadou programovacích jazyků (např. Java, C, PHP, Python). Většina programovacích jazyků má nicméně drobné úpravy pro jejich syntaxi, tudíž výraz vytvořený pro jeden programovací jazyk nemusí fungovat i v jazyce jiném.

Nejjednodušším regulárním výrazem je samotné písmeno, např. 'a'. Pokud necháme v textu vyhledávat podle 'a', budou vyhledány všechny výskyty písmene 'a'. Takovéto výrazy je možné za sebou řetězit a jelikož se obvykle nevyhledávají jednotlivé znaky, ale rovnou celá slova, je možné jako regulární výraz použít např. 'auto'. Tímto výrazem budou vráceny výskyty slova 'auto' v daném textu.

Jelikož někdy nemusíme nutně vědět, jak má určitá část ve vyhledávaném řetězci vypadat, lze použít znak tečky '.' . Pomocí ní lze vyhledat libovolný znak, kromě konce řádku. Pokud se tedy nechá vyhledat např. 'aut.' , výraz bude vracet jak 'auto', tak i 'auta' nebo 'auty'. Pokud ale víme, že nám ne úplně známý znak musí být písmeno, lze využít ještě hranatých závorek se kterými je možné určit, jaké znaky budou na daném místě akceptovány. Při zápisu '[aoy]' budou na daném místě akceptována písmena 'a', 'o' a 'y'. Pokud se do hranatých závorek napíše '[a-z]', bude akceptováno jakékoliv malé písmeno. V tabulce 2.1 jsou vypsány některé možnosti, jak jednotlivé znaky regulárními výrazy zapsat (pro jazyk PHP).[9]

Někdy nevíme, kolik daných znaků vyhledávaný řetězec obsahuje. Pokud potřebujeme například získat v textu nějakou číselnou hodnotu, nemůžeme si být jisti, kolik cifer bude dané číslo mít. Pro tyto příklady jsou užitečné kvantifikátory. Pomocí nich můžeme určit, kolik daných znaků má vyhledávaný řetězec mít. Když tedy zadáme výraz např. '\d+' , bude vráceno jak číslo 3, tak i číslo 333. V tabulce 2.2 jsou vypsány některé důležité kvantifikátory.

Regulární výrazy také umožňují skupinové konstrukce, kdy můžeme se skupinou znaků pracovat podobně, jako s jednotlivým znakem, tudíž je možné

Výraz	Funkce
.	jakýkoliv znak komě konce řádku
[abc]	znak a, b nebo c
[^abc]	jakýkoliv znak vyjma a, b a c
[a-z]	jakýkoliv znak od a do z
[^a-z]	jakýkoliv znak vyjma a až z
[a-zA-Z]	jakýkoliv znak od a do z a od A do Z
\s	jakýkoliv bílý znak (mezera, tabulátor, nový řádek)
\S	jakýkoliv znak kromě bílého
\d	jakákoliv číslice
\D	jakýkoliv znak kromě číslice
\w	jakýkoliv slovní znak (písmeno, číslice, podtržítko)
\W	jakýkoliv znak kromě slovního
\A	začátek řetězce
\Z	konec řetězce

Tabulka 2.1: Zástupné znaky v regulárních výrazech

Výraz	Funkce
a?	žádné nebo jedno a
a*	žádné a více a
a+	jedno a více a
a{3}	přesně 3 a
a{3,}	3 a více a
a{3,6}	3 až 6 a

Tabulka 2.2: Kvantifikátory v regulárních výrazech

Výraz	Funkce
(...)	vrací vše, co je uvnitř a vytvoří skupinu
(...)	vrací vše, co je uvnitř a vytvoří skupinu
(a|b)	vrací a nebo b
(?:...)	vrací vše, co je uvnitř, ale nevytváří skupinu
(?|...)	cokoliv uvnitř má stejné číslo skupiny
(?(1)true|false)	podmínka pokud skupina 1 existuje, provede se true, jinak false
a(?=Z)	pozitivní dopředné vyhledávání vrací a, pokud se ihned za ním vyskytuje Z
a(?!Z)	negativní dopředné vyhledávání vrací a, pokud se ihned za ním nevyskytuje Z
(?<=Z)a	pozitivní zpětné vyhledávání vrací a, pokud se ihned před ním vyskytuje Z
(?<!Z)a	negativní zpětné vyhledávání vrací a, pokud se ihned před ním nevyskytuje Z

Tabulka 2.3: Skupinové konstrukce v regulárních výrazech

na ně uplatňovat kvantifikátory nebo vracet různé řetězce v rámci jednoho regulárního výrazu. Pokud tedy pro text 'Auto narazilo do stromu.' použijeme výraz `Auto|strom[a-z]` , výraz nám vrátí 'Auto' i 'stromu'. Další důležitou konstrukcí je dopředné a zpětné vyhledávání. Pomocí tohoto vyhledávání je možné popsat, co následuje před nebo za popsaným řetězcem. Tato vyhledávání mohou být jak pozitivní (definují, co před nebo za řetězcem musí být), tak i negativní (definují, co tam být nesmí). Pokud bychom na větu 'Auto značky Volvo narazilo do stromu' použili výraz `(\w+)(?=\snarazilo)`, který využívá dopředné pozitivní vyhledávání, regulární výraz by vrátil 'Volvo'. Pokud by se totožný výraz použil pro 'Auto značky Mercedes narazilo do stromu', výraz by vrátil 'Mercedes'. Více skupinových konstrukcí se nachází v tabulce 2.3.[5]

Výše zmíněné podmínky a dopředné, případně zpětné vyhledávání, představují důležité nástroje při psaní regulárních výrazů. Lze s nimi popisovat text, který je okolo vyhledávaného řetězce, tudíž se hodí zejména ve chvíli, kdy nemáme úplně představu, jak daný výraz má vypadat, případně se kolem něj nachází mnoho podobných výrazů, které si ale nepřejeme vyhledat.

Pomocí všech výše zmíněných konstrukcí lze popsat prakticky jakýkoliv řetězec. Díky tomu lze daný řetězec vyhledat, případně zvalidovat. Regulární výrazy jsou velmi užitečné kupříkladu pro validaci položek ve formulářích, například emailových a webových adres.

3 XPath

Jazyk XPath slouží k adresování částí XML dokumentu. S jeho pomocí lze vybírat jednotlivé elementy a atributy v XML souborech a pracovat s nimi. XPath by definován organizací W3C (mezinárodní konsorcium, které společně s veřejností vyvíjí standardy pro world wide web). Pro jazyk XPath je podpora v mnoha programovacích jazycích a narozdíl od regulárních výrazů se jeho syntax v různých jazycích nemění, tudíž je možné stejný XPath výraz používat napříč různými jazyky. Na ukázce 3.1 se nachází příklad XML souboru.

```
<?xml version="1.0" encoding="UTF-8"?>
<autobazar>
    <auto>
        <model znacka="Volvo">XC90</model>
        <cena>250000</cena>
    </auto>
    <auto>
        <model znacka="Skoda">Octavia</model>
        <cena>150000</cena>
    </auto>
     <auto>
        <model znacka="Skoda">Fabia</model>
        <cena>90000</cena>
    </auto>
</autobazar>
```

Ukázka 3.1: XML soubor

Nejjednodušším XPath výrazem je název hledaného elementu. Pokud tedy jako XPath výraz použijeme "auto", budou vybrány všechny uzly s tímto jménem. V tabulce 3.1 je vypsáno několik důležitých XPath výrazů.

Důležitou součástí jazyka XPath jsou také predikáty. Predikáty slouží k nalezení specifického uzlu nebo uzlu se zadanou hodnotou. Píší se vždy do hranatých závorek. Predikáty je možné využít pro nalezení uzlu s určitým indexem, důležité je ale pamatovat na to, že v XPath začíná počáteční index na 1, nikoliv na 0. V tabulce 3.2 jsou ukázky některých predikátů, které jazyk XPath nabízí.[8]

Jazyk XPath se momentálně nachází ve verzích 1.0 (vydáno 1999), 2.0 (vydáno 2007), 3.0 (vydáno 2014) a 3.1 (vydáno 2017). V novějších verzích

Výraz	Funkce
nazevUzlu	vybere všechny uzly s tímto názvem
/	vybere uzel z kořenového uzlu (absolutní cesta)
//	vybere všechny uzly s požadovaným názvem bez ohledu na to, kde se nacházejí
.	vybere momentální uzel
..	vybere rodičovský uzel
@	vybere atributy

Tabulka 3.1: Základní XPath výrazy

Výraz	Funkce
/autobazar/auto[1]	vybere první uzel s názvem auto
/autobazar/auto[last()]	vybere poslední uzel s názvem auto
/autobazar[position()<3]	vybere první 2 uzly
//auto[@znacka]	vybere všechny elementy, které mají atribut znacka
//auto[@znacka=skoda]	vybere všechny elementy, které mají atribut znacka s hodnotou skoda
//autobazar/auto[cena<160000]	vybere všechny elementy auto, které mají cenu nižší než 160000
//autobazar/auto[cena<160000]	vybere všechny elementy auto, které mají cenu nižší než 160000
//auto[1]|//auto[2]	logické nebo, vrátí element auto 1 i auto 2

Tabulka 3.2: Predikáty v XPath

podporuje mnoho užitečných funkcí, např. má podporu regulárních výrazů. Díky tomu je možné pomocí XPath vracet i pouze určitou část textového řetězce uvnitř vyhledávaného elementu. Momentálně nicméně spousta programovacích jazyků má přímou podporu pouze pro XPath 1.0, mezi těmito jazyky je i PHP. Aby bylo možné v těchto jazycích používat XPath 2.0 a vyšší, je nutné využít nějaké externí knihovny nebo externího procesoru, jako je například Saxon[7], což ale v rámci některých aplikací nemusí být žádoucí.

4 Analýza zpracování meteorologických dat

4.1 Zpracování pomocí regulárních výrazů

Vzhledem k tomu, že se pomocí regulárních výrazů budou zpracovávat soubory, které nebudou mít žádnou jednotnou strukturu, bude nutné k nim přistupovat velmi obecně. Aby se zabránilo nějakým nejednoznačnostem, bude nutné prakticky vždy popsat celý vstupní soubor. Pouze takto by mělo být možné těmto nejednoznačnostem zabránit, jelikož se tím přesně popíše daný výraz a jeho okolí.

K popisu samotného zpracovávaného řetězce poslouží samotné zástupné znaky, ať již `\d` pro číslice nebo `\w` pro slovní znaky. Před speciální znaky, jako je např. tečka nebo plus, bude nutné přídat lomítko, aby nebyly brány jako znaky zástupné. Tyto znaky bude nutné doplnit také vhodnými kvantifikátory (Tabulka 2.2), pravděpodobně pomocí `+` , které označuje jeden a více výskytů daného znaku. Jeden a více je zapotřebí z důvodu, že daná hodnota je na tomto místě očekávaná (očekává se konzistence vstupních dat), tudíž není možné, aby se daná hodnota na tomto místě v textu nevyskytla, ale je možné, že bude obsahovat více znaků než hodnota, nad kterou byl daný regulární výraz vytvořen.

Dále bude nutné popsat okolí zpracovávaného řetězce. K tomu by se mohlo hodit dopředné a zpětné vyhledávání. Pomocí nich by bylo možné popsat, co následuje před a za zpracovávaným řetězcem. Pravděpodobně bude s jejich pomocí nutné popsat celý vstupní soubor, aby se zabránilo tomu, že se najde více řetězců s totožným okolím. Pokud se popíše celý soubor, bude zaručena jednoznačnost daného řetězce, jelikož s ním budou zároveň popsány i všechny řetězce před ním i za ním. Toto řešení bude sice vytvářet velmi dlouhé regulární výrazy, ale vzhledem k tomu, že vstupní soubory mohou mít nejrůznější formáty, bude užitečné mít způsob, který dokáže univerzálně fungovat pro všechny. Pokud by kvůli popisování celého souboru nastával nějaký výkonnostní problém, bylo by možné přistoupit na kompromis a pomocí dopředného a zpětného vyhledávání popisovat pouze okolí do určité vzdálenosti od zpracovávaného řetězce. Když ale výkonnostní problém nenastane, bude lepší kvůli jednoznačnosti popisovat celý soubor.

Vzhledem k tomu, že uživatel bude chtít většinou pomocí jednoho výrazu zpracovávat více řetězců najednou, bude nutné vytvářet výraz tak, aby

vracel více různých řetězců. K tomu poslouží logické nebo (OR), které se v případě regulárních výrazů značí jako `|`. S tímto bude možné jednotlivé řetězce napojit, tudíž výsledný výraz bude vypadat takto:
`regularni_vyraz_A|regularni_vyraz_B|regularni_vyraz_C`.

4.2 Zpracování pomocí XPath

Pro XML soubory bude vhodnější místo regulárních výrazů používat XPath. Vzhledem k tomu, že by každá stanice měla poskytovat data v konzistentní podobě (měřené veličiny budou vždy ve stejném počtu a pořadí), bude možné v podstatě pouze popsat cestu k danému uzlu a použít predikát, určující jeho pořadí. Pravděpodobně ani nebude nutné popisovat absolutní cestu pomocí `/`, ale bude stačit pouze vybrat uzel pomocí jeho názvu a poté určit pomocí kvantifikátoru jeho pořadí v zpracovávaném souboru. Uživatel bude mít opět možnost vybrat více uzlů najednou, tudíž se využije `|`, čímž se v XPath značí logické "OR". Výsledný výraz bude tedy vypadat přibližně takto:
`//uzel[1]|//uzel[2]"`

V případě, že uživatel vybere pouze určitou podčást uzlu, bude nutné využít i regulárních výrazů. XPath 2.0 nabízí několik funkcí, které umožňují jejich použití. Jako nejzajímavější se pro tento účel jeví funkce `replace(...)`[6], která umožňuje nahrazení znaků jinými. Takto by bylo možné nepotřebné znaky nahradit prázdným řetězcem, čímž by bylo možné získat požadovanou část uzlu. S opětovným využitím dopředného vyhledávání by například výraz, který z uzlu `<datum>19.02.2020</datum>`, získá hodnotu '02' mohl vypadat takto:
`replace((//datum)[1],'(\d+\.)(?=\d+\.\d+)|(\.\d+)(?=$)', '')`

Vzhledem k tomu, že PHP momentálně bohužel nemá přímou podporu pro XPath 2.0, bude nutné buď využít externí knihovny v PHP aplikaci, nebo v takovémto případě generovat regulární výraz místo XPath výrazu.

5 Zvolené řešení zpracování meteorologických dat

5.1 Generování regulárních výrazů

Pro automatické generování regulárních výrazů se ukázalo jako nejlepší přímočaré řešení, kdy se znak po znaku popíše celý řetězec, určený ke zpracování, a pak pomocí pozitivního dopředného vyhledávání i to, co za ním následuje. Původně se mělo popisovat i vše, co se vyskytuje před vybraným řetězce, nicméně vzhledem k tomu, že zpětné vyhledávání v PHP narozdíl od dopředného neumožňuje použití kvantifikátorů, nelze jej pro účel automatického generování použít. Kvantifikátory jsou zapotřebí k udržení funkčnosti výrazu při změně délky řetězců v datech, kdy se nelze spoléhat například na to, že rychlost větru bude mít vždy 2 cifry.

Algoritmus funguje tak, že se nejprve vezme startovní index zpracovávaného řetězce, ve vstupních datech se nalezne znak na tomto indexu a postupně se popíší všechny znaky až do koncového indexu zpracovávaného řetězce. Pro písmena se použije `\w`, pro číslice `\d` a pro bílé znaky `\s`. Před speciální znaky, jako je například tečka nebo pomlčka se navíc přidá `\`, aby nedošlo ke kolizím se syntaxí pro zástupné znaky v regulárních výrazech. K číslicím a písmenům se zároveň přidá kvantifikátor `+`, jelikož se jejich výskyt na daném místě očekává, nicméně nelze zaručit, že budou mít vždy stejnou délku, tudíž nelze přímo specifikovat jejich počet. Algoritmus si zároveň pamatuje typ posledního znaku v řetězci, tudíž se za sebe zbytečně neskládají stejné výrazy se stejnými kvantifikátory, tudíž místo `\w+\w+` se ve vygenerovaném výrazu vytvoří pouze `\w+`. K mezerám a speciálním znakům se žádné kvantifikátory nepřidávají, jelikož narozdíl od číslic a písmen se u nich očekává konstantní počet. Celý tento vygenerovaný výraz se obalí kulatými závorkami tak, aby vznikla skupina. Bude tedy vypadat například takto: `(\d+\.\d+)`.

Dále se v algoritmu popíší všechny znaky, které následují za zpracovávaným řetězcem. Tyto znaky se popíší stejným způsobem, jako se popisovaly pro samotný zpracovávaný řetězec. Když se popíší všechny následující znaky, tak se ještě na konec přidá `\z`, které značí konec řetězce. Tímto se zaručí to, že se popíše pouze jedno konkrétní místo v datech a není poté ani nutné použít zpětné vyhledávání, které v PHP neumožňuje použití kvantifikátorů. Celý tento řetězec se vloží do značek pro pozitivní dopředné vy-

hledávání, což je `(?=...)`. Tato část výrazu bude tedy vypadat například takto: `(?=\s\d+\z)`.

Na závěr se obě části poskládají za sebe. Výsledný výraz může vypadat jako `(\d+\.\d+)(?=\s\d+\z)`. Když bychom tedy tento výraz použili na řetězec: „15.4 4.1 20.5 10", získali bychom hodnotu 20.5. Pokud se má zpracovávat ještě další řetězec, tak se za tento výraz přidá `|` a začne se generovat výraz pro další vybraný řetězec.

Při testování se ovšem ukázalo, že způsob popisování znaků je určitých případech problematický. Problém nastává v případě, kdy se z kladné hodnoty stane záporná. V takovémto případě by nebylo v regulárním výrazu popsáno znamínko mínus, které se zde může, ale nutně nemusí vyskytnout. Kvůli této situaci bylo nutné před každý znak pro číslici přidat `\-{0,1}`. Tento zápis popíše znaménko mínus tak, že se na daném místě může, ale nutně nemusí vyskytovat, což umožní dané hodnotě nabývat jak kladných, tak i záporných hodnot. Jako další problém se ukázala situace, kdy je například nějaká hodnota momentálně na 0. V takovémto případě některé stanice přestávají uvádět za číslicí i případná desetinná místa, na která je daná veličina měřena. Toto se vyřešilo tím, že se zkontroluje, zda předchozí i následující znak je mezera (takto se z 99% rozdělují jednotlivé hodnoty v datech) a pro daný znak se popíše pomocí `\-{0,1}\d+\.{0,1}\d*`. Tímto může tato hodnota mít zároveň i libovolný počet desetinných míst a zároveň také nabývat jak kladných, tak i záporných hodnot. Mohlo by se sice zdát, že by se tímto způsobem daly popisovat libovolné číslicové znaky a nemuselo by se hlídat i to, zda je hodnota obklopena mezerami, nicméně poté by uživatel ztratil možnost například vybírat pouze určité části datumu. Když by se tímto popisovaly všechny číslice, v případě že by u datumu "23.04.2020" uživatel vybral jako řetězec pouze "04", tento výraz by mu vrátil "04.2020". Takovéto chování je nežádoucí, tudíž muselo dojít k omezení na pouze atomické znaky.

Další problém, který byl odhalen při testování, souvisí trochu i s předchozím případem. V datovém souboru byla například hodnota změny délky dne, kdy se před číslicí vyskytlo i znaménko "+", označující prodlužování dne. Zde se ovšem může také místo plus objevit i mínus, tudíž je nutné oba tyto znaky v regulárních výrazech popisovat jako `\-{0,1}\+{0,1}`. Tímto se zaručí, že na daném místě může, ale zároveň nemusí být jakákoliv z těchto hodnot. Aby se alespoň trochu generované výrazy zkrátily, není nutné před každou číselnou hodnotou dávat rovnou tuto konstrukci pro tlačítko plus i mínus. Předpokládá se, že pokud hodnota před sebou nemá žádné znaménko, aplikace takto značí kladnou hodnotu a znaménko plus před ní nikdy nedává. Číslo tedy může mít před sebou nanejvýš znaménko mínus, což stačí zajistit

pouze pomocí `\-{0,1}`.

Vzhledem k tomu, že se při zkoušení výsledného regulárního výrazu neukázal žádný výkonnostní problém kvůli jeho délce, nebylo nutné přistupovat k nějakému kompromisu, kdy by se například popisovala pouze určitá část textu za řetězcem. Délka výsledného výrazu se v podstatě odvíjí od vzdálenosti vybraného řetězce od konce celých vstupních dat, čím dále požadovaný řetězec od konce je, tím delší regulární výraz se pro něj vygeneruje. Vzhledem k tomu, že tento výraz popíše úplně vše (i konec dat), je výsledný výraz naprosto přesný a nemělo by docházet ke kolizím s jinými místy ve vstupních datech.

5.2 Generování XPath výrazů

U XPath výrazů se osvědčilo využití názvu hledaného elementu a poté zjištění pořadí, ve kterém se daný element v XML souboru vyskytuje. Algoritmus tedy funguje tak, že v datech vstoupí na startovní index požadovaného řetězce a poté se podívá na znak před startovním, kde zjistí, zda je shodný s `>`. Pokud ano, tak ještě zkontroluje, zda po koncovém indexu požadovaného řetězce následuje `</`. Když i toto souhlasí, tak algoritmus začne zjišťovat název elementu, ve kterým se daný řetězec vyskytuje, což zjistí tak, že se začne postupně po indexech posouvat od znaku `>` dozadu, než narazí na `<`. Slovo, které se nachází mezi `<` a `>` je název námi požadovaného elementu.

Když získáme název požadovaného elementu, začneme data procházet dále dozadu, kdy začneme v datech vyhledávat všechny elementy s tímto názvem. Když zjistíme počet stejných elementů, které předcházejí našemu požadovanému, tak již máme vše potřebné pro náš XPath výraz. Výsledný výraz vypadá jako `(//nazevElementu)[poradi]`. Pokud máme pomocí XPath zpracovat více řetězců najednou, lze za tento výraz přidat `|` a poté opět vygenerovat požadovaný výraz.

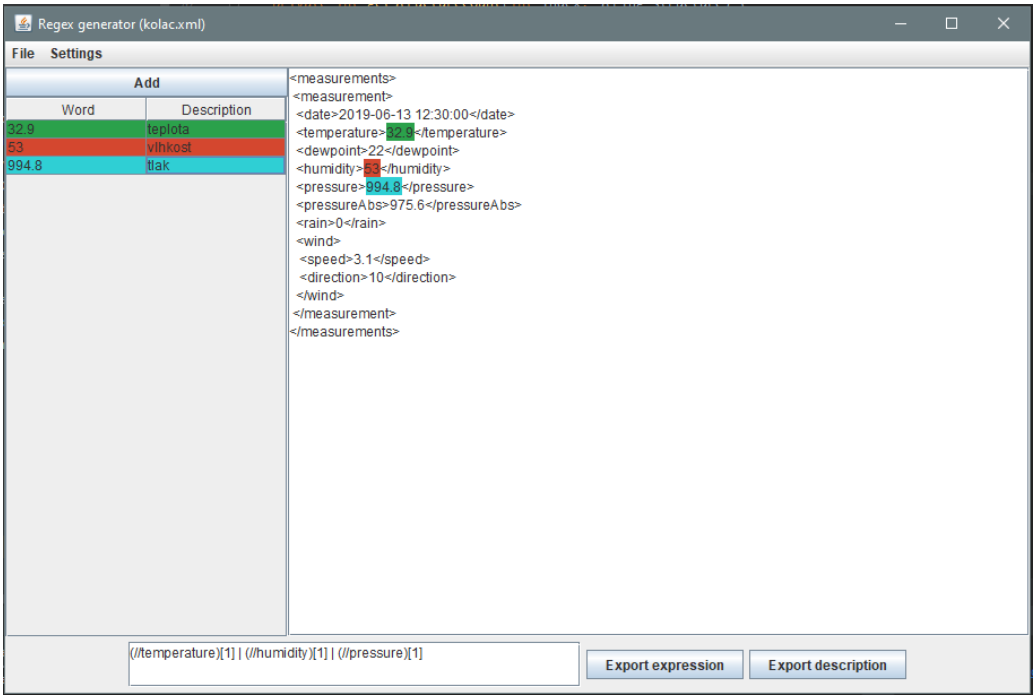
Může nastat situace, kdy uživatel bude chtít pouze určitou část uvnitř uzlu, nikoliv uzel celý. K tomu by bylo zapotřebí využít funkcí pro regulární výrazy, které jsou ale až od XPath 2.0. Vzhledem k tomu, že jazyk PHP momentálně nemá pro XPath 2.0 přímou podporu, tudíž by se daný výraz nedal využít napříč různými aplikacemi, použijí se v takovém případě pouze samotné regulární výrazy bez XPath.

Výsledný XPath výraz se může jevit na první pohled krátký, nicméně vzhledem k tomu, že podle pořadí přesně vybere požadovaný uzel, tak není zapotřebí například popisovat celou cestu k danému uzlu. Na vygenerovaném uzlu ani není poznat, že byl automaticky vygenerován, jelikož způsob, jelikož

tento způsob zápisu výrazu je nejsnazší i pro ruční napsání.

5.3 Aplikace pro generování výrazů

Vzhledem k tomu, že tato aplikace mohla být napsána v libovolném jazyce, tak jsem se rozhodl vzhledem k snadné přenositelnosti na jiná zařízení ji napsat v jazyce Java. Pro grafické zobrazení jsem využil javovské knihovny Swing. Aplikace dále využívá Maven[3], což je nástroj pro zprávu a automatizaci buildů. Na obrázku 5.1 lze vidět vzhled okna této aplikace.



Obrázek 5.1: Okno aplikace pro generování výrazů

V této aplikaci lze tedy označením vybrat požadovaný řetězec, který se po klepnutí na tlačítko 'Add' přidá do tabulky, ve které lze k němu doplnit popisek. Při přidání řetězce se daný řetězec obarví a stejnou barvou se obarví i řádek v tabulce. Jednotlivé řetězce se obarvují různými barvami, které jsou nadefinované v souboru 'colors.txt'. V tomto souboru jsou barvy popsány pomocí RGB, kdy každý řádek znamená jednu barvu a jednotlivé RGB hodnoty jsou odděleny středníkem. Je zde přednastaveno 10 barev, nicméně uživatel může soubor snadno upravit a barvy změnit, případně přidat další. V případě, že při vybírání řetězců nebudou stačit barvy nadefinované v souboru, aplikace použije náhodně vygenerované barvy. Tyto barvy jsou generovány tak, že mají pro každou RGB složku nastavenou minimální

hodnotu 100, čímž se zaručí to, že barvy nebudou příliš tmavé. Vzhledem k tomu, že možný rozsah jednotlivých barev je stále poměrně velký, nemělo by se příliš stávat, že by se barvy jednotlivých řetězců mnoho nelišily.

Po přidání do tabulky se zároveň přegeneruje výsledný výraz, který se zobrazuje ve spodní části okna. Vedle něj se nachází možnost exportovat výsledný výraz do souboru a dále také možnost exportovat popisky k daným řetězcům. Ty se exportují ve formátu, kdy jsou jednotlivé popisky v souboru odděleny středníkem a jsou poskládány za sebou v pořadí, v jakém se nacházejí v datovém souboru hodnoty, ke kterým patří. Díky tomu se mohou v PHP aplikaci k sobě snadno přiřadit, jelikož knihovny pro XPath i pro regulární výrazy vracejí hodnoty do polí v tomto pořadí. Vzhledem k jednoduchosti formátu je také možné soubor s popisky snadno zeditovat i ručně aniž by bylo nutné k tomu otevírat projekt v aplikaci.

Funkce jako otevření datového souboru, otevření rozpracovaného projektu nebo uložení projektu se nacházejí v horním menu pod položkou 'File'. Dále se zde nachází položka 'Settings', ve které je možné zapnout nebo vypnout zalamování řádek v textovém poli pro data. Tuto možnost je vhodné zapnout zejména pro dlouhé soubory, ve kterých se data nacházejí pouze na jednom řádku, jelikož se poté vejdou na obrazovku.

Ukládání rozpracovaného projektu se provádí tak, že se do souboru uloží vstupní data a informace o řetězcích, které uživatel vybral. K tomuto řešení jsem se rozhodl proto, že regulární výrazy v Javě nejsou úplně totožné s těmi v PHP, tudíž pokud by se do souboru ukládal místo řetězců regulární výraz, tak by bylo nutné ho konvertovat do podoby, se kterou Java dokáže pracovat. Vzhledem k tomu, že generované výrazy mohou být i velmi dlouhé, tak toto řešení se jeví jako efektivnější jak z hlediska výkonu aplikace, tak i z hlediska velikosti ukládaného souboru. Do souboru s uloženým projektem se zároveň ukládá i informace o tom, zda jsou data v XML nebo jiném formátu. Díky tomu se projekt otevírá výrazně snadněji a není zapotřebí zbytečně procházet vstupní data a zjišťovat, v jakém formátu jsou.

Uživateli je také umožněno nadefinovat výchozí složku pro průzkumníka souborů, přes kterého se otevírají datové soubory nebo projekty. Tuto složku lze změnit v souboru 'config.txt', ve kterém se k ní nachází cesta. Základní hodnotou je pro tuto složku v případě operační systému Windows složka 'Dokumenty' u operačního systému Linux je to složka 'Home'.

Aplikace dále umožňuje otevření vstupních dat z URL adresy. Pro tyto účely byla využita knihovna Commons IO od Apache[1], která má metodu `FileUtils.copyURLtoFile`, která zkopíruje obsah ze zadané adresy do lokálního souboru, se kterým je možné poté dále pracovat. Tato knihovna se stáhne automaticky při buildu aplikace pomocí Mavenu, který má na ní

přidanou závislost.

Aplikace je rozčleněna celkem do 3 balíčků, ve kterých se nachází celkem 11 tříd.

5.3.1 Balíček application

V tomto balíčku se nachází celkem 5 tříd, které jsou určené ke spuštění aplikace a celkovému vykreslení okna.

Třída `MainClass` je hlavní třídou celé aplikace. Vyskytuje se zde metoda `main`, která slouží k jejímu spuštění.

Třída `Window` slouží k vykreslení výsledného okna. Nacházejí se zde metody, které vykreslují jednotlivé komponenty a dále také metody, které zajišťují reakce na některé akce od uživatele, jako je například stisk tlačítka v aplikaci nebo stisk nějaké klávesy na klávesnici.

Třída `SelectedWord` je přepravka, která slouží k uložení důležitých informací o řetězci, který uživatel vybere. Ukládá se do ní tedy index začátku a konce výběru, vybrané slovo a barva, která je použita pro jeho výběr. Díky uchovávání všech těchto informací je poté možné i snáze ukládat celý projekt.

Třída `ColorPicker` slouží pro správu barev, určených pro zvýrazňování jednotlivých řetězců. Tato si v bezparametrovém konstruktoru nejprve načte barvy ze souboru 'colors.txt' a poté je možné metodou `pickColor` získávat barvy. Ve chvíli, kdy barvy ze souboru dojdou, metoda začne náhodně generovat barvy další.

Enum `FileType` je pouze výčet typů souborů, aby bylo možné snáze určit typ načteného datového souboru.

5.3.2 Balíček parsers

V tomto balíčku se nacházejí celkem 2 třídy a rozhraní, které se starají o generování jak regulárních, tak i XPath výrazů.

Rozhraní `IParser` zastřešuje zbývající 2 třídy. Je zde definována metoda `createExpression()`, kterou zbývající třídy v tomto balíčku implementují.

Třída `RegexParser` implementuje rozhraní `IParser` a stará se o generování regulárních výrazů. V konstruktoru této třídy se do ní předá text se vstupními daty a dále list s vybranými řetězci. Poté skrz překrytou metodu `createExpression()` vrací vygenerovaný regulární výraz.

Třída `XPathParser` implementuje také rozhraní `IParser` a stará se primárně o generování XPath výrazů. V konstruktoru se také předává text se vstupními daty a dále seznam s vybranými řetězci. V překryté metodě

`createExpression()` generuje výsledný XPath výraz, nicméně pokud je vybrána pouze určitá část uzlu, tak si vytvoří instanci `RegexParser` a pomocí něho vrátí místo XPath výrazu regulární. Čistě regulární výraz se musí vracet z důvodu, že výsledný výraz musí být kompatibilní s jazykem PHP, který má ovšem přímou podporu pouze pro XPath 1.0 a v případě vybrání části uzlu by bylo nutné využít funkcí, které jsou až v XPath 2.0.

5.3.3 Balíček constants

V balíčku `constants` se nacházejí pouze třídy s používanými konstantami. Pro konstanty jsou vytvářeny speciální třídy z důvodu, aby se k nim snadněji přistupovalo a vzhledem k tomu, že některé konstanty jsou využívány napříč různými třídami, není nutné je definovat v každé třídě zvlášť, případně dohledávat v jaké třídě je již použita.

V třídě `ASCIIConstants` jsou nadefinovány číselné hodnoty používaných ascii znaků, které se používají při generování regulárních výrazů.

V třídě `Messages` jsou nadefinovány texty upozornění nebo vyskakovacích oken, které se v aplikaci místy vyskytují.

V poslední třídě `RegexConstants` jsou nadefinovány konstanty regulárních výrazů, jako je například zástupný znak pro nový řádek nebo mezeru.

5.4 Aplikace pro zobrazení dat

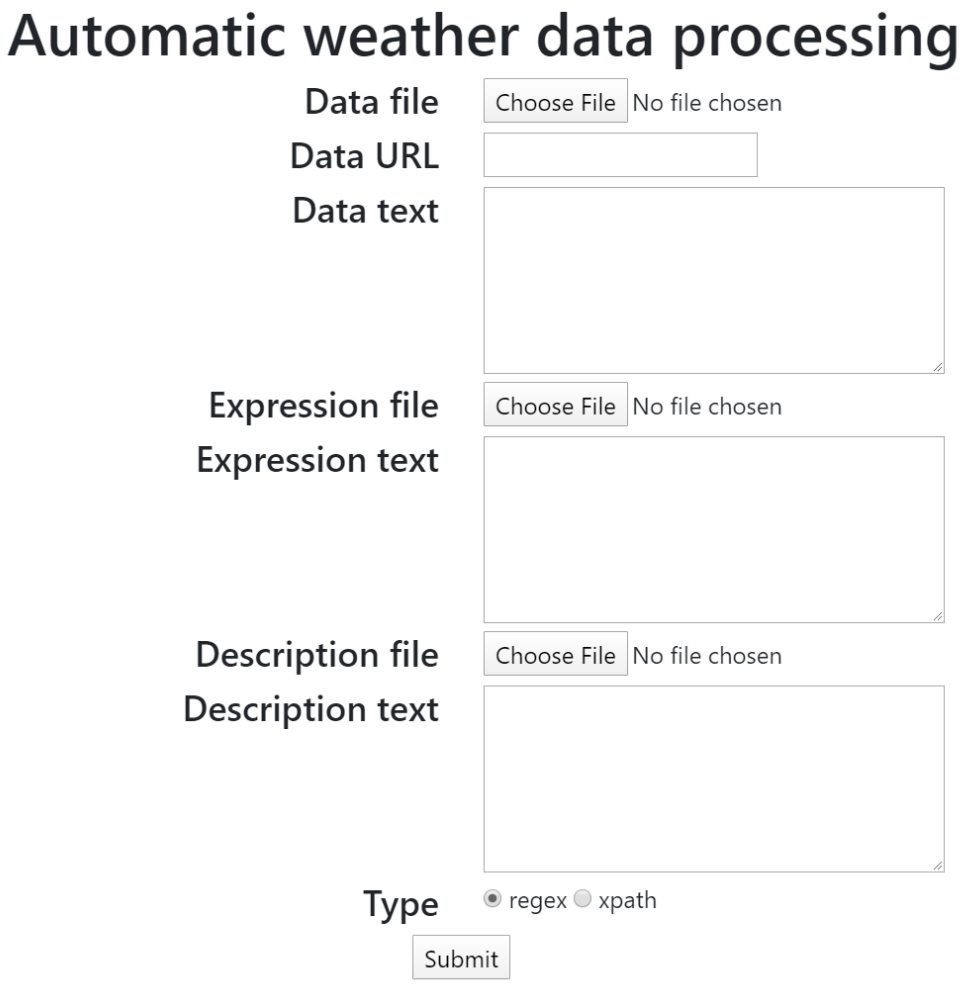
Tato aplikace byla napsána v jazyce PHP 7, pro který se v rámci Java aplikace generovaly výsledné regulární výrazy. Tato aplikace obsahuje formulář, do kterého je možné vložit některým z nabízených způsobů vstupní data, výraz, pomocí kterého se budou zpracovávat, a soubor s popisky. Vzhled tohoto formuláře je na obrázku 5.2.

Vstupní data se dají vložit 3 způsoby:

- výběr datového souboru z lokálního úložiště
- URL adresa k datovému souboru
- přímé vložení textu souboru do textového pole

Regulární a XPath výraz lze vložit buď jako soubor, nebo do textového pole jako text. U popisků jsou nabízené možnosti vložení stejné jako u výrazů. Po kliknutí na tlačítko "Submit"se formulář odešle a data se začnou pomocí vloženého výrazu zpracovávat.

Pro zpracovávání pomocí regulárních výrazů byla použita PHP funkce `preg_match_all(..)`[4]. Tato funkce při použití vygenerovaného výrazu



Obrázek 5.2: Formulář PHP aplikace

jako parametru výsledek zpracování nahraje do pole, kdy na každý index nahraje další pole. Na indexu 0 je vždy pole se všemi získanými výsledky zpracovávání pomocí regulárního výrazu v pořadí, jak jdou v textu za sebou. Od indexu 1 se poté nachází vždy pole, které má všechny položky null kromě těch, které se získaly pomocí dané části výrazu. Ve výrazech jsou jednotlivé části oddělovány pomocí |, což znamená logické "OR". Od indexu 1 lze tedy získat výstupní hodnoty v takovém pořadí, v jakém byly postupně získávány pomocí regulárního výrazu, a nikoliv pouze v pořadí, v jakém se vyskytují ve vstupních datech. V ukázce 5.1 je příklad použití této funkce v PHP a na ukázce 5.2 je výstup po provedení daného kódu.

```
preg_match_all('/(met)|(plamen)/', 'plamenomet', $matches);
var_dump($matches);
```

Ukázka 5.1: Ukázka použití funkce

```
array (size=3)
  0 =>
    array (size=2)
      0 => string 'plamen' (length=6)
      1 => string 'met' (length=3)
  1 =>
    array (size=2)
      0 => string '' (length=0)
      1 => string 'met' (length=3)
  2 =>
    array (size=2)
      0 => string 'plamen' (length=6)
      1 => string '' (length=0)
```

Ukázka 5.2: Výstup aplikace

U zpracovávání pomocí regulárních výrazů nastal jeden drobný problém v situaci, kdy datový soubor pro nový řádek používá pouze 'LF'. V případě PHP aplikace na Windows se ukázalo, že pokud se takovýto soubor nahraje napřímo, tak si tento znak pro nový řádek zachovává, nicméně pokud byl nahrán přes textové pole, tak se místo 'LF' použilo 'CRLF', což svým způsobem změnilo strukturu vstupních dat a způsobilo to, že regulární výraz přestal fungovat. Toto jsem vyřešil tak, že pokud funkce `preg_match_all(..)` nevrátí žádný výsledek, tak se ve vstupních datech všechna 'CRLF' nahradí 'LF' a poté se funkce nechá znovu provést, což v takovémto případě tento problém opraví. Tato situace může nastat i obráceně, když vstupní soubor obsahuje 'CRLF' a PHP aplikace se spustí pod Linuxem, tudíž v tomto případě se postupuje opět podobně a to tak, že se tentokrát všechna 'LF' ve vstupních datech nahradí pomocí 'CRLF'. Tímto by mělo být možné jednotlivé vstupní datové soubory a regulární výrazy, které jsou pro ně vygenerovány, používat navzájem mezi operačními systémy Windows a Linux.

Pro zpracování pomocí XPath byla použita PHP třída DOMXPath[2], která má funkci `query(...)`, pomocí které je možné nechat vstupní XML soubor zpracovat XPath výrazem. Tato funkce vrátí jednotlivé hodnoty do pole, ve kterém jsou seřazeny v pořadí, v jakém se nacházejí ve vstupním souboru. Vzhledem k tomu, jak XPath funguje, tak nebylo nutné řešit podobné problémy jako u regulárních výrazů, kdy bylo nutné zařizovat kompatibilitu mezi různými operačními systémy.

Po zpracování dat jak s pomocí regulárních výrazů, tak s XPath výrazy, se výsledné pole vypíše v tabulce společně s popisky k daným hodnotám. Aby se popisky a hodnoty získané pomocí zpracování výrazy správně přiřadily, jsou hodnoty získané pomocí zpracování výrazy vypisovány v pořadí, ve kterém se nacházejí ve vstupním datovém souboru. Popisky jsou již z ja-

vovské aplikace v tomto pořadí seřazeny, tudíž je poté možné podle indexu snadno zjistit, který popisek se má k dané hodnotě vypsat.

6 Výsledky

6.1 Úvod

Testování probíhalo na notebooku s procesorem Intel Core i5-6200U a RAM 8192 MB DDR3. Obecně dobu běhu programu ovlivňuje zejména velikost vstupního datového souboru a počet uživatelem vybraných řetězců pro zpracovávání. V závislosti na typu výrazu doby běhu ovlivňuje také pozice uživatelem vybraného řetězce ve vstupním souboru. Doby běhu generování byly měřeny pomocí javovské metody `System.currentTimeinMilis()`. Do dob běhů se vždy započítával pouze čas na vygenerování výrazu, nikoliv i na překreslení okna s přidaným záznamem v tabulce a zvýrazněným řetězcem.

6.2 Regulární výrazy

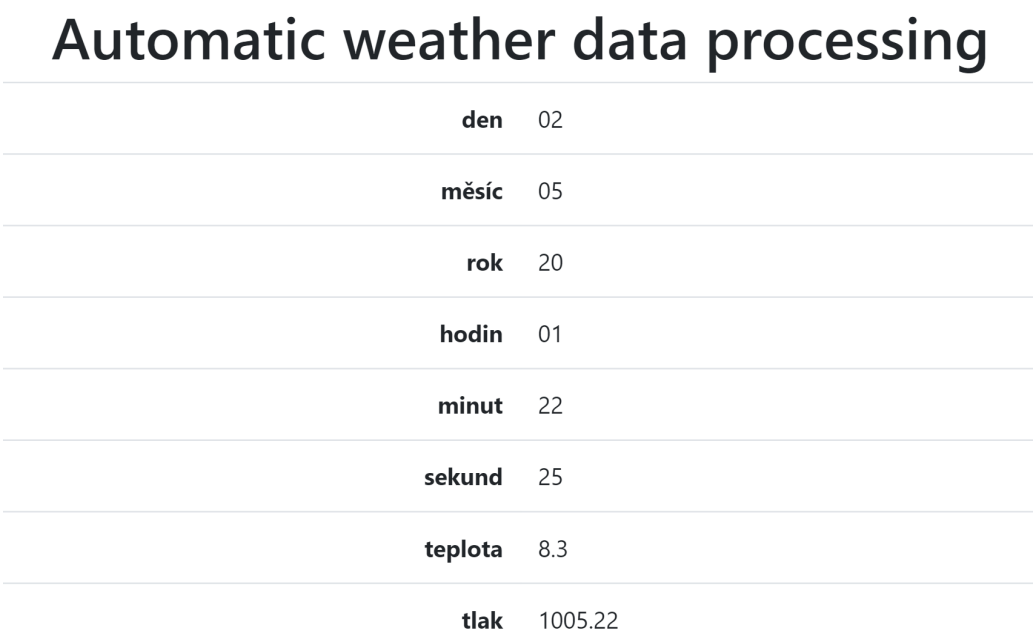
6.2.1 Testování

Správnost generování regulárních výrazů byla testována na větším množství datových souborů s různým obsahem z jedné meteostanice, jejíž aktuální výstup je k dispozici online[11]. Tímto bylo zkoušeno, zda vygenerovaný výraz dokáže parsovat požadovaná data při změně jejich hodnot.

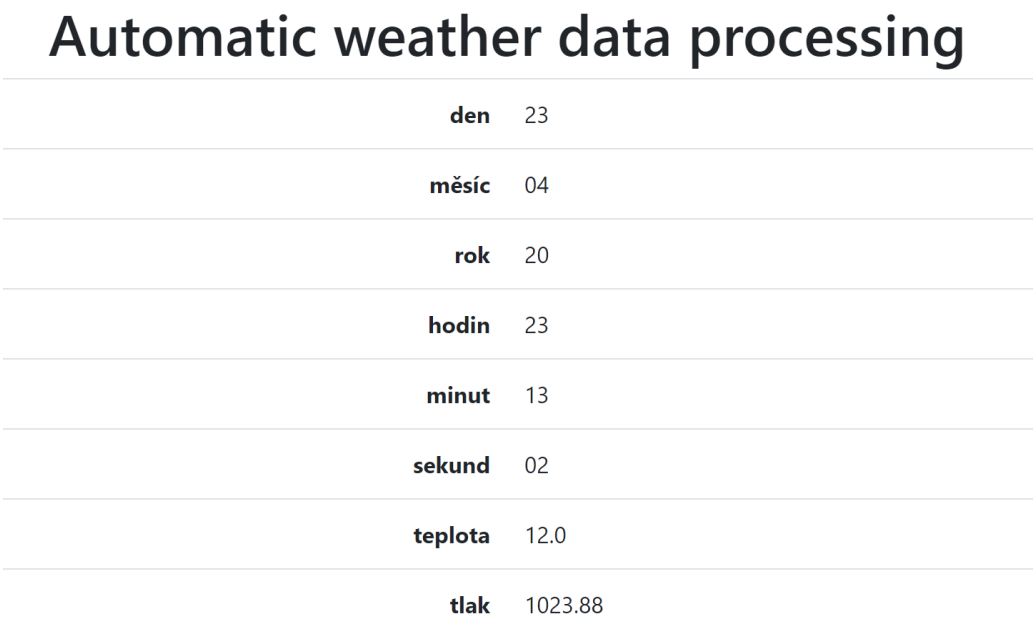
Řetězce ve vstupních datech byly vybírány zejména ze začátku souboru tak, aby byl pomocí výrazu popisován celý soubor a byla největší pravděpodobnost případné chyby. Pomocí vygenerovaného výrazu byly zpracovány 3 různé výstupy z jedné meteostanice, kdy hodnoty zobrazené ve výsledné tabulce odpovídaly jejich přiřazení. Na ukázce 6.1 je k dispozici příklad vstupních dat, která byla zpracovávána. Na obrázku 6.1 je výsledek zpracování některých hodnot z tohoto souboru. Na obrázcích 6.2 a 6.3 jsou vidět výsledky zpracovávání dalších souborů v PHP aplikaci. Na ukázce 6.1 je k dispozici příklad zpracovávaného výstupu. Jak lze z obrázků pozorovat, vygenerovaný regulární výraz je schopen zpracovat libovolné výstupy v rámci jedné meteostanice.

```
02.05.20 01:22:25 8.3 88 6.4 2.7 1.6 326 0.0 0.0 1005.22 SZ 1 km/h C hPa
    mm 3.5 +0.17 1.6 113.8 1.6 24.6 35 8.3 -0.5 9.1 00:00 8.0 00:50 4.8
    00:48 8.0 00:37 1005.42 00:55 1004.95 00:03 1.9.4 1099 3.2 8.3 8.3
    0.0 -0.05 0 325 0.0 3 0 0 SZ 233 m 7.0 0.0 0 0
```

Ukázka 6.1: Vstupní soubor pro testování generování regulárních výrazů

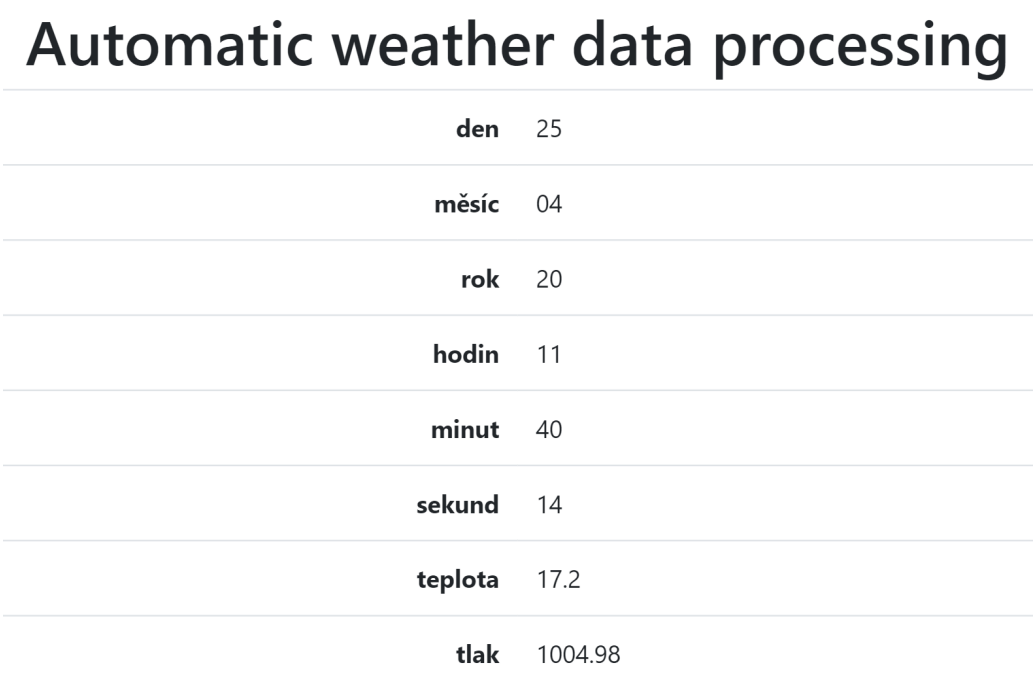


Obrázek 6.1: Testování regulárních výrazů - 1. data



Obrázek 6.2: Testování regulárních výrazů - 2. data

Generování regulárních výrazů se tedy po testování již ukázalo být funkční, nicméně vzhledem k odlišnostem mezi různými meteostanicemi by se mohla najít i nějaká, na kterou by tato metoda nezabírala. S vysokou pravděpodobností by to bylo způsobeno tím, že stanice poskytuje z hlediska generování



Obrázek 6.3: Testování regulárních výrazů - 3. data

výrazů nekonzistentní data, která nějakým zásadním způsobem mění svou podobu. Pro tyto stanice není nicméně generování primárně zamýšlené, jelikož by bylo nutné výstupy z každé takovéto stanice individuálně zanalyzovat a poté generování případně pro ně upravit. Případ, že by například stanice na nějaké místo v datech někdy dávala číselnou hodnotu a jindy slovní, by bylo nutné někdy i vyřešit tak, že by mohlo být negativně ovlivněno generování pro většinu ostatních stanic. Proto se generování omezuje primárně na stanice, které poskytují konzistentní data. Pokud nějaká konkrétní stanice ale například začne poskytovat hodnoty pro více veličin, je nutné pro správné zpracovávání výraz vygenerovat znovu.

6.2.2 Doby běhů

Do doby běhu generování regulárních výrazů se promítá několik faktorů. Nejdůležitější z nich je velikost vstupního souboru, jelikož čím delší vstupní soubor je, tím delší se musí vygenerovat potřebný regulární výraz. Dále také velmi záleží na tom, kde se ve vstupním souboru nacházejí vybrané řetězce. Vzhledem k tomu, že se vždy popisuje vše, co následuje za vybraným řetězcem, tak se rychleji vygenerují výrazy, které se nacházejí blíže konci souboru než ty, které jsou ze začátku.

Provedl jsem několik měření rychlosti generování výrazů, kdy se jednot-

livá měření od sebe lišila jak velikostí vstupního souboru, tak i způsobem výběru vybraných řetězců (někde byly řetězce vybírány pouze v první čtvrtině textu souboru, jinde pouze u konce, někde rovnoměrně). Byl měřen pouze čas samotného generování výrazu, do měření není zahrnut čas pro překreslování okna aplikace, ve které se výsledný výraz zobrazuje.

V tabulce 6.1 jsou k vidění doby běhu při rovnoměrném rozložení vybraných řetězců v celém souboru, tudíž řetězce byly vybírány jak ze začátku, tak i z konce souboru. Pro většinu vstupních souborů je tento případ potenciálního výběru nejčastější, jelikož meteorologické stanice mají ve výstupech požadované hodnoty také rozmístěny po celém souboru. Jak je z měření patrné, doby běhů se v zásadě prodlužují s přibývajícím počtem řetězců, případně se zvyšující se velikostí vstupního datového souboru.

Počet řetězců	Soubor 245 B	Soubor 966 B	Soubor 2230 B
10	2 ms	16 ms	25 ms
25	4 ms	31 ms	80 ms
50	10 ms	56 ms	167 ms

Tabulka 6.1: Doby běhu generování regulárních výrazů – rovnoměrné rozložení

V tabulce 6.2 jsou naměřené hodnoty doby běhu v případě, že veškerý výběr byl proveden pouze v prvních 25% souboru. U souboru s 245 znaky bylo při tomto měření nutné některé výběry kvůli nedostatku možností zopakovat, nicméně byly znovu generovány tak, jako by výběr byl unikátní, tudíž doba běhu není tímto ovlivněna. Jak lze v tabulce vidět, tak doby běhu byly poměrně silně ovlivněny a dosahovaly několikanásobně vyšších hodnot než v případě rovnoměrného výběru. Tento případ výběru v praxi sice není úplně běžný, nicméně mohl by nastat například v případě, že uživatel bude chtít pouze některé základní hodnoty, které se v datovém souboru budou nacházet zrovna ze začátku souboru. Ačkoliv ale tyto hodnoty označují nejhorší možný scénář z hlediska uživatelského výběru, nejsou doby běhu nijak zásadně dlouhé, jelikož vzhledem k tomu, že se jedná o jednorázové generování, tak nevyšší hodnota 460 ms je pro tento účel vcelku přijatelná.

V další tabulce 6.3 jsou naopak hodnoty doby běhu pro případ výběru pouze v posledních 25% souboru. V takovémto případě jsou časy výrazně nižší ve srovnání s rovnoměrným rozložením. Takovýto výběr od uživatele by v zásadě také neměl být příliš běžný, nicméně může se opět stát, že zrovna v této části souboru se nacházejí hodnoty, které uživatel požaduje, tudíž bude vybírat zejména odsud. Zde vzhledem k tomu, že již nezbývá tolik znaků do konce souboru na popis pomocí regulárních výrazů, dochází

k významnému urychlení celého generování.

Počet řetězců	Soubor 245 B	Soubor 966 B	Soubor 2230 B
10	4 ms	19 ms	70 ms
25	18 ms	53 ms	160 ms
50	47 ms	154 ms	460 ms

Tabulka 6.2: Doby běhu generování regulárních výrazů - výběry pouze v prvních 25% souboru

Počet řetězců	Soubor 245 B	Soubor 966 B	Soubor 2230 B
10	1 ms	2 ms	2 ms
25	3 ms	6 ms	13 ms
50	6 ms	13 ms	31 ms

Tabulka 6.3: Doby běhu generování regulárních výrazů - výběry pouze v posledních 25% souboru

Jak lze tedy vidět z předchozích tabulek, doba běhu generování je vcelku uspokojivá. Vzhledem k tomu, že generování je jednorázová záležitost, tak doba generování pro poměrně velký datový soubor je i v horším případě výběru stále poměrně dobrá (460 ms). Pokud navíc nedojde k takovému výběru, tak je navíc doba běhu tak nízká, že uživatel pravděpodobně ani samotné generování nepostřehne. Delší čas než samotné generování může ve výsledku zabrat samotné překreslování okna v javovské aplikaci, kdy je nutné ještě přidat záznam do tabulky a zvýraznit výběr v textovém poli.

6.3 XPath výrazy

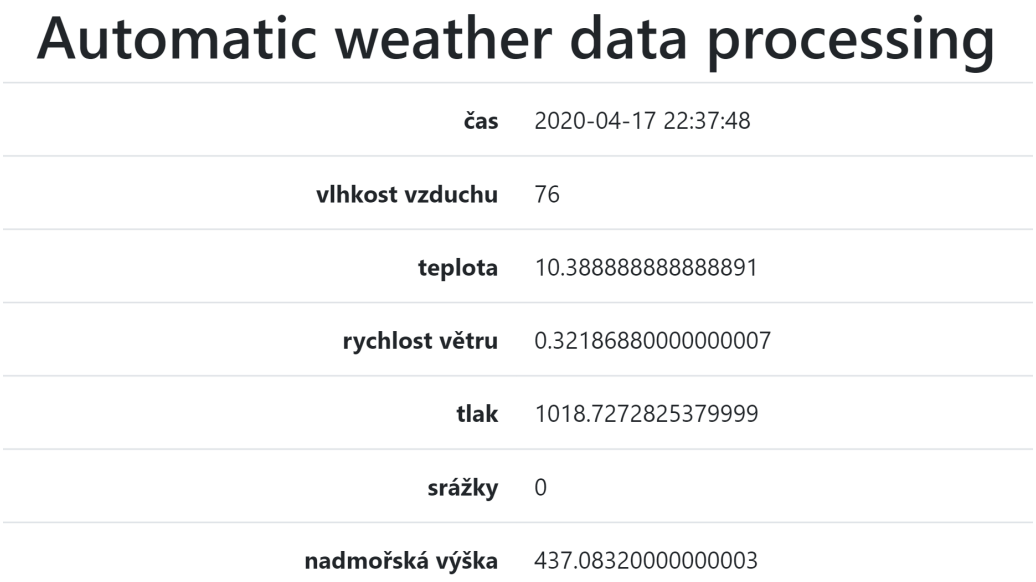
6.3.1 Testování

Správnost vygenerovaných XPath výrazů byla stejně jako u regulárních výrazů testována na větším množství různých výstupů v rámci jedné meteostanice tak, aby se pomocí jednoho vygenerovaného XPath výrazu zpracovalo větší množství dat. Vzhledem k tomu, že XML má poměrně jasně danou datovou strukturu, nemohly zde nastat příliš výjimečné situace. Kvůli chybějícím funkcím pro regulární výrazy v XPath 1.0 se navíc XPath výrazy generují pouze v případě vybrání celého obsahu uzlu od uživatele, tudíž zde není ani velký prostor pro uživatele pro nějaký nestandartní výběr. Díky těmto faktorům nebyla při tomto testování odhalena žádná chyba, která by

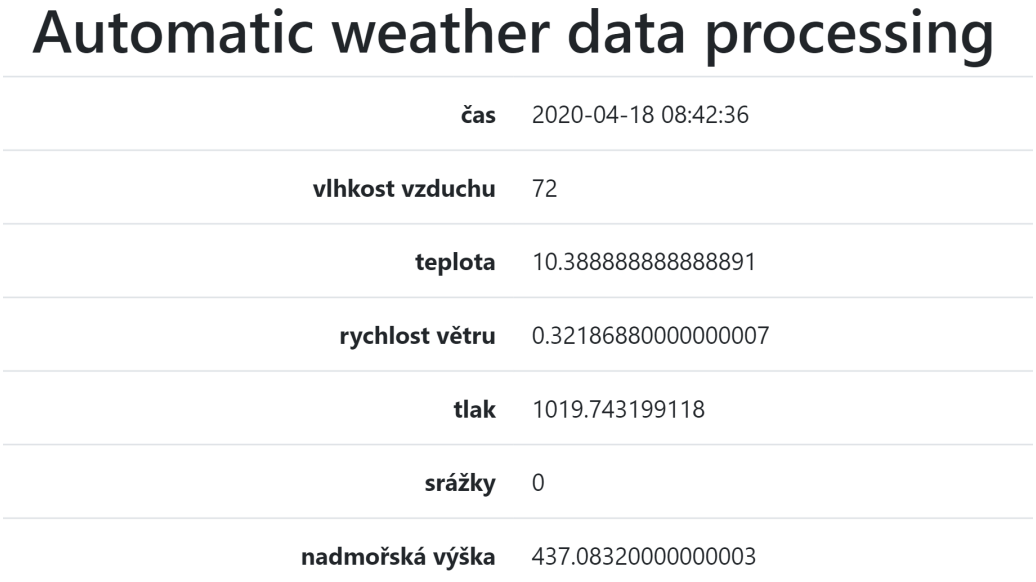
způsobila nefunkčnost výrazu, tudíž výstup v tabulce po zpracování odpovídal vždy předpokladu. V ukázce 6.2 je příklad vstupního XML souboru, jehož zpracování XPath výrazem v PHP aplikaci je k vidění na obrázku 6.4. Na obrázcích 6.5 a 6.6 jsou výsledky zpracovávání dalších výstupních dat zevstejné meteostanice. Jak je z obrázků vidět, hodnoty v tabulce odpovídají svému zařazení.

```
<observations xmlns="">
   <observations>
      <observations>
         <stationID>IPOBOV1</stationID>
         <obsTimeUtc>2020-04-17T20:37:48Z</obsTimeUtc>
         <obsTimeLocal>2020-04-17 22:37:48</obsTimeLocal>
         <neighborhood>Pobovice</neighborhood>
         <softwareType>EasyWeatherV1.4.4</softwareType>
         <country>CZ</country>
         <solarRadiation>0</solarRadiation>
         <lon>12.79992</lon>
         <epoch>1587155868</epoch>
         <lat>49.507984</lat>
         <uv>0</uv>
         <winddir>256</winddir>
         <humidity>76</humidity>
         <qcStatus>-1</qcStatus>
         <metric>
            <temp>10.388888888888891</temp>
            <heatIndex>10.388888888888891</heatIndex>
            <dewpt>6.277777777777776</dewpt>
            <windChill>10.388888888888891</windChill>
            <windSpeed>0.32186880000000007</windSpeed>
            <windGust>1.7702784000000003</windGust>
            <pressure>1018.7272825379999</pressure>
            <precipRate>0</precipRate>
            <precipTotal>0</precipTotal>
            <elev>437.08320000000003</elev>
         </metric>
      </observations>
   </observations>
</observations>
```

Ukázka 6.2: Ukázka XML souboru pro generování XPath výrazů

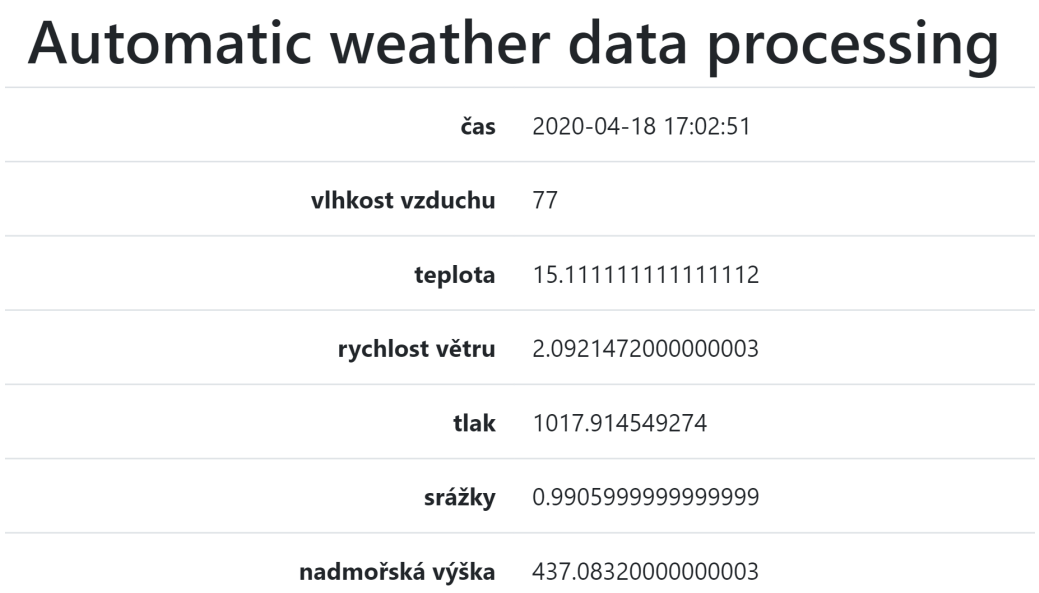


Obrázek 6.4: Testování XPath výrazů - 1. data



Obrázek 6.5: Testování XPath výrazů - 2. data

Vzhledem k tomu, že se ve výrazu používají pouze názvy uzlů a jejich indexy, neměl by se vyskytnout problém v zásadě u žádné jiné meteostanice. Základní podmínkou ovšem je, že meteostanice bude mít konstantní strukturu a bude poskytovat data pro stejné množství veličin. Pokud by stanice přece jenom začala mít data pro více veličin, bylo by nutné, aby se v datech objevily až za posledním vybraným řetězcem od uživatele. Díky tomu by



Obrázek 6.6: Testování XPath výrazů - 3. data

vygenerovaný výraz fungoval i po takovéto změně. Obecně je ale po změně struktury ve vstupních datech vždy nutné vygenerovat nový výraz, který si již s takovýmto datovým XML souborem opět dokáže poradit.

6.3.2 Doby běhů

Do doby potřebné pro vygenerování XPath výrazu se stejně jako u regulárních výrazů promítá několik faktorů. Důležitá je velikost vstupního souboru a také opět pozice výběru ve vstupních datech. Zde ale obecně platí to, že čím blíže k začátku textu v souboru výběr je, tím rychleji aplikace výraz vygeneruje. Bylo provedeno opět několik měření, kdy se od sebe opět lišily jak velikostí vstupního souboru, tak způsobem výběru řetězců v datech.

V tabulce 6.4 jsou zaznamenány doby běhu při rovnoměrném rozložení vybíraných řetězců po celém souboru. Jak je z tabulky patrné, generování probíhá výrazně rychleji než v případě regulárních výrazů. Při většině generování změřený čas nepřesáhl ani 1 ms. Hlavní důvod je ten, že při XPath generování dochází k výrazně menšímu počtu podmínek ve zdrojovém kódu. Zde se v podstatě pouze spočítají výskyty uzlů se stejným názvem, jako má požadovaý řetězec, a to navíc ani ne v celém souboru, ale pouze do indexu, na kterém vyhledávaný řetězec leží. Není také nutné neustále přidávat znaky ke konečnému výrazu, jelikož zde se pouze složí výsledný výraz jako `(//nazev_uzlu)[index_uzlu]`

V tabulce 6.5 jsou zaznamenány doby běhu pro lepší případ výběru, což

Počet řetězců	Soubor 298 B	Soubor 1335 B	Soubor 2257 B
10	<1 ms	<1 ms	<1 ms
25	<1 ms	<1 ms	1 ms
50	<1 ms	<1 ms	2 ms

Tabulka 6.4: Doby běhu generování XPath výrazů - rovnoměrné rozložení

by byla situace, kdy uživatel bude vybírat zkraje souboru. Všechny vybrané řetězce byly tedy vybírány pouze z prvních 25% souboru. V této situaci bylo nutné některé výběry opakovat, aby bylo možné provést měření i pro více vybraných řetězců. Jak je z následující tabulky patrné, doba běhu se ještě o něco vylepšila, a to tak, že momentálně se doba generování již nikdy nedostala nad 1 ms. Je to způsobeno zejména tím, že algoritmus prochází od pozice vybraného řetězce text souboru zpět na začátek, tudíž čím blíže je vybraný řetězec k začátku souboru, tím rychleji dojde k vygenerování výrazu.

Počet řetězců	Soubor 298 B	Soubor 1335 B	Soubor 2257 B
10	<1 ms	<1 ms	<1 ms
25	<1 ms	<1 ms	<1 ms
50	<1 ms	<1 ms	<1 ms

Tabulka 6.5: Doby běhu generování XPath výrazů - výběry pouze v prvních 25% souboru

V tabulce 6.6 jsou doby běhu z měření téměř nejhoršího případu, což je situace, kdy uživatel bude vybírat pouze z konce souboru. V tomto měření bylo nutné opět některé řetězce vybrat vícekrát, nicméně hodnoty to nikterak nezkresluje, jelikož algoritmus probíhá stejně, jako by hodnoty byly unikátní. Jak je z tabulky vidět, pro XML soubor o velikosti 2257 B se doby běhu viditelně zhoršily, nicméně stále se jedná o velmi nízké hodnoty.

Počet řetězců	Soubor 298 B	Soubor 1335 B	Soubor 2257 B
10	<1 ms	<1 ms	1 ms
25	<1 ms	<1 ms	2 ms
50	<1 ms	<1 ms	3 ms

Tabulka 6.6: Doby běhu generování XPath výrazů - výběry pouze v posledních 25% souboru

Generování XPath výrazů se ukázalo z hlediska časové náročnosti velmi efektivní, jelikož i v nejhorších případech jsou doby běhu v řádu několika

málo milisekund. Takovýto čas je pro uživatele nemožné postřehnout, tudíž je naprosto uspokojivý. Dále se ukázalo, že minimálně při generování jsou pro XML soubory XPath výrazy výrazně výhodnější, jelikož doba generování u nich je zhruba 100× kratší než v případě regulárních výrazů. Pokud tedy není zapotřebí vybírat pouze části uzlů, což v XPath 1.0 nejde, tak je lepší generovat výrazy pro XML soubory pomocí jazyka XPath než přes regulární výrazy.

6.4 Zpracování dat

Při zpracovávání řetězců jak s pomocí regulárních výrazů, tak s pomocí jazyka XPath, ovlivňuje dobu běhu zejména velikost vstupního datového souboru a velikost výrazu, který se nad ním použije. S délkou výrazu poté může souviset i počet výsledných řetězců, které vyhovují zadanému výrazu, jelikož jednotlivé vygenerované výrazy jsou složeny z několika kratších výrazů, které jsou spojeny pomocí logického 'OR'.

Vzhledem k tomu, že samotné zpracování není nikterak časově náročné, bylo provedeno měření nad jedním XML souborem o velikosti 2257 B, kdy bylo postupně provedeno 10, 20 a 39 výběrů řetězců. Více řetězců nebylo možné vybrat, jelikož by poté bylo nutné některé výběry opakovat, což by v tomto případě již výslednou dobu měření velmi zkreslovalo, jelikož duplicitní hodnoty by byly ve výsledku ignorovány. Pro tento soubor byly totožné řetězce vybrány jak pro vygenerování regulárního výrazu, tak i pro XPath výraz. Díky tomu je možné zároveň přímo porovnat doby běhů pro oba typy generování.

V tabulce 6.7 jsou zaznamenány doby běhů zpracovávání výrazů v tomto souboru. Měření probíhalo opět na stejném notebooku, jako v předchozích případech. Jak je z tabulky patrné, XPath výraz provede zpracování výrazně rychleji než výraz regulární. Doba zpracování se i v případě s nejvíce vybranými řetězci pohybovala kolem 2 ms, zatímco regulární výraz totožné řetězce zpracovával 15 ms. XPath tedy dokázal zpracovat řetězce zhruba 6× rychleji oproti regulárním výrazům.

Typ výrazu	10 výběrů	20 výběrů	39 výběrů
Regulární	4 ms	7ms	15 ms
XPath	1 ms	1 ms	2 ms

Tabulka 6.7: Doby běhu zpracovávání výrazů

Zajímavá je také velikost vyexportovaných výrazů, kdy regulární výraz

má velikost 64 198 B a XPath výraz má 810 B. Samotný regulární výraz je tedy zhruba 30× větší než vstupní datový soubor a 80× větší než vygenerovaný XPath výraz. Toto je způsobeno zejména tím, že musí popisovat od každého vybraného řetězce všechny znaky až do konce souboru, tudíž při větším počtu výběrů se začne velikost souboru poměrně hodně zvyšovat. Naproti tomu XPath výraz obsahuje pouze jména uzlů, tudíž jeho velikost nepřekročí velikost vstupního souboru.

V takovémto přímém porovnání se jeví pro XML soubory výhodnější XPath výrazy, nicméně je stále nutné počítat s tím, že ty mají řadu omezení z hlediska možností různých výběrů. Zatímco regulární výrazy dokáží zpracovat řetězec kdekoliv v souboru, XPath výrazy dokáží vybrat pouze celý uzel, tudíž není možné s nimi vybrat pouze jeho podčást. Pokud ale uživatel chce vybírat pouze celé uzly, XPath výrazy se jednoznačně vyplatí, jelikož z hlediska výkonosti jsou ve všech ohledech lepší. Byly rychlejší jak při generování, tak i zpracovávání, navíc velikost samotného výrazu je i mnohem menší oproti výrazu regulárnímu.

7 Závěr

Vytvořená aplikace pro generování regulárních a XPath výrazů je funkční a dokáže vygenerovat výrazy, které mohou zpracovávat výstupy z dané meteostanice. Výrazy se generují relativně rychle, tudíž se zde neukazuje nějaká potřeba pro případnou optimalizaci. Prostor pro případné zlepšení by se pravděpodobně dal nalézt v případě velikosti vygenerovaných regulárních výrazů, jelikož jsou většinou několikanásobně větší než samotný datový soubor, který mají zpracovávat. Při zachování současného popisování znaků za vybraným řetězcem by bylo pravděpoobně dobré nalézt nějaký případný kompromis, kdy by se nepopisoval celý soubor až do konce, ale pouze jeho část. S tímto by ale mohl nastat problém v tom, že by vygenerovaný výraz nemusel popisovat pouze jednu hodnotu v datech, čímž by došlo k nežádoucímu konfliktu.

Oproti původnímu plánu bylo nutné značně omezit generování XPath výrazů, jelikož kvůli absenci přímé podpory pro XPath 2.0 v jazyce PHP nebylo možné využít funkcí pro regulární výrazy, které XPath 2.0 nabízí. XPath výrazy jsou tedy generovány pouze v případě, že uživatel vybere celé uzly. Když vybere pouze určitou část uzlu, aplikace vygeneruje pouze regulární výraz tak, aby zpracovával pouze požadovanou část uzlu.

Generování výrazů bylo testováno na nejrůznějších meteorologických datových souborech, nad kterými se ukázalo funkční a výrazy data zpracovávaly. Vzhledem ale k tomu, že stanice poskytují data v nejrůznějších formátech a jednotlivé hodnoty reprezentují různými způsoby, může se pravděpodobně nalézt i stanice, pro jejíž výstup by vygenerovaný výraz nebyl funkční. V případě nefunkčního generování u nějaké stanice by bylo nutné data zanalyzovat a poté případně poupravit popisování znaků pro tento případ. Pokud by ale stanice poskytovala data v nějaké nekonzistentní podobě, nebylo by možné výraz generovat, jelikož konzistence je základní podmínkou pro generování výrazů.

Literatura

[1] *Apache Commons IO* [online]. 2020. [cit. 2020/04/14]. Dostupné z: http://commons.apache.org/proper/commons-io/.

[2] *The DOMXPath class* [online]. 2020. [cit. 2020/04/15]. Dostupné z: https://www.php.net/manual/en/class.domxpath.php.

[3] *Apache Maven Project* [online]. 2020. [cit. 2020/04/12]. Dostupné z: http://maven.apache.org/.

[4] *PHP:Preg Match All* [online]. 2020. [cit. 2020/04/15]. Dostupné z: https://www.php.net/manual/en/function.preg-match-all.php.

[5] *Regex101* [online]. 2020. [cit. 2020/04/04]. Dostupné z: https://regex101.com/.

[6] *fn:replace* [online]. 2018. [cit. 2020/04/04]. Dostupné z: http://www.xqueryfunctions.com/xq/fn_replace.html.

[7] *Saxon - XPath 2.0* [online]. 2020. [cit. 2020/04/05]. Dostupné z: https://www.saxonica.com/documentation9.6/#!conformance/xpath20.

[8] *W3Schools - XPath* [online]. 2020. [cit. 2020/04/05]. Dostupné z: https://www.w3schools.com/xml/xpath_intro.asp.

[9] Satrapa, P. *Regulární výrazy* [online]. 2000. [cit. 2020/04/04]. Dostupné z: https://www.root.cz/clanky/regularni-vyrazy-1/.

[10] Sklenák, V. *Regulární výraz. In: KTD: Česká terminologická databáze knihovnictví a informační vědy (TDKIV)* [online]. Praha: Národní knihovna ČR, 2003. [cit. 2020/04/04]. Dostupné z: https://aleph.nkp.cz/F/?func=direct&doc_number=000000651&local_base=KTD.

[11] *Meteostanice Topoľčany* [online]. [cit. 2020/04/25]. Dostupné z: https://www.meteoto.sk/realtime.txt.

Uživatelská příručka

Aplikace pro generování výrazů

Překlad a spuštění aplikace

Aplikace vyžaduje ke svému spuštění mít nainstalovanou Javu 11. Pro úspěšné přeložení aplikace je dále nutné mít nainstalovaný Apache Maven. Postup jeho instalace je vysvětlen v následující kapitole. V případě operačního systému Windows je možné poté aplikaci pro generování výrazů spustit pomocí skriptu 'run.bat', který se nachází ve složce s aplikací a zajistí přeložení a spuštění aplikace. Pokud používáte systém Linux, je zde připraven skript 'run.sh', pomocí kterého se aplikace také spustí.

Instalace Apache Maven

Apache Maven je možné stáhnout z https://maven.apache.org/download.cgi. Zde doporučuji stáhnout některý z archivů s binární distribucí, kterou stačí po stažení pouze rozbalit a vložit na vámi preferované umístění. Poté je nutné přidat cestu k Mavenu do PATH. To se provádí tak, že na liště necháte vyhledat 'Upravit proměnné prostředí systému', po otevření okna klepnete na proměnné prostředí a zde v systémových proměnných klepnete na tlačítko upravit. Poté stačí klepnout na tlačítko 'Procházet', kde již vyberete cestu do složky 'bin', ve které se nachází soubor 'mvn.cmd'. Nakonec stačí všechna okna potvrzováním na tlačítka 'OK' pozavírat a samotný Maven je poté připravený k použití.

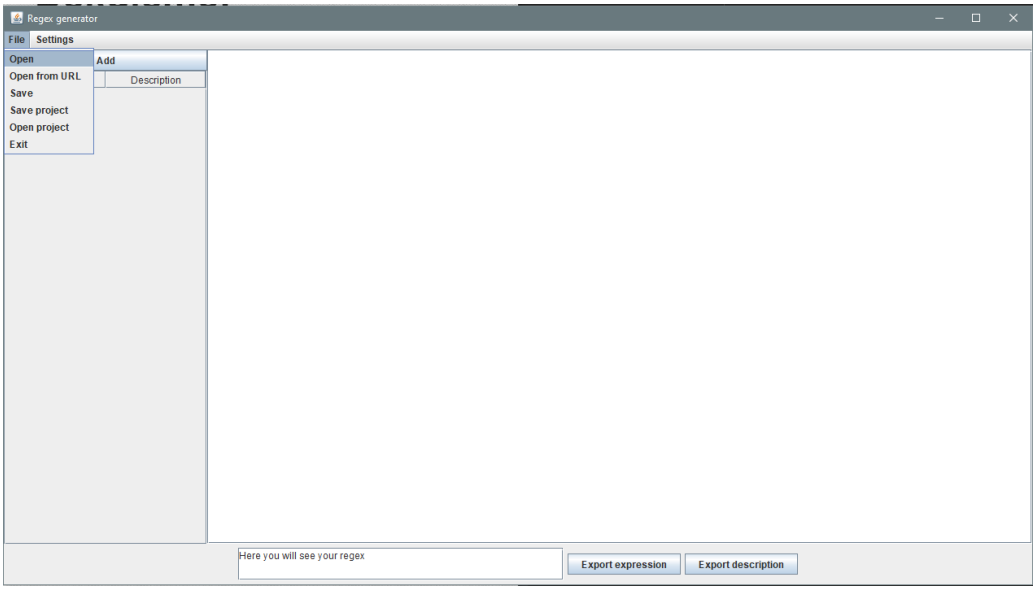
Pro správné fungování Mavenu je také nutné mít nastavenou systémovou proměnnou JAVA_HOME. Ta se nastavuje podobným způsobem jako cesta k Mavenu v PATH, tudíž je nutné na liště nechat vyhledat 'Upravit proměnné prostředí systému', poté klepnout na tlačítko proměnné prostředí a zde u systémových proměnných kliknout na tlačítko 'Nová'. Ve vyskakovacím okně se jako název proměnné zadá JAVA_HOME a poté se klepne na tlačítko 'Procházet adresář', kdy se vybere cesta do složky s Javou. Pozor, nezadává se zde cesta do složky bin v Javě, ale cesta do složky nadřízené, tudíž do složky, která složku 'bin' obsahuje. Poté stačí potvrdit všechna okna tlačítkem 'OK' a Maven by již měl být plně funkční.

Tento postup je platný pro operační systém Windows. Podrobný popis

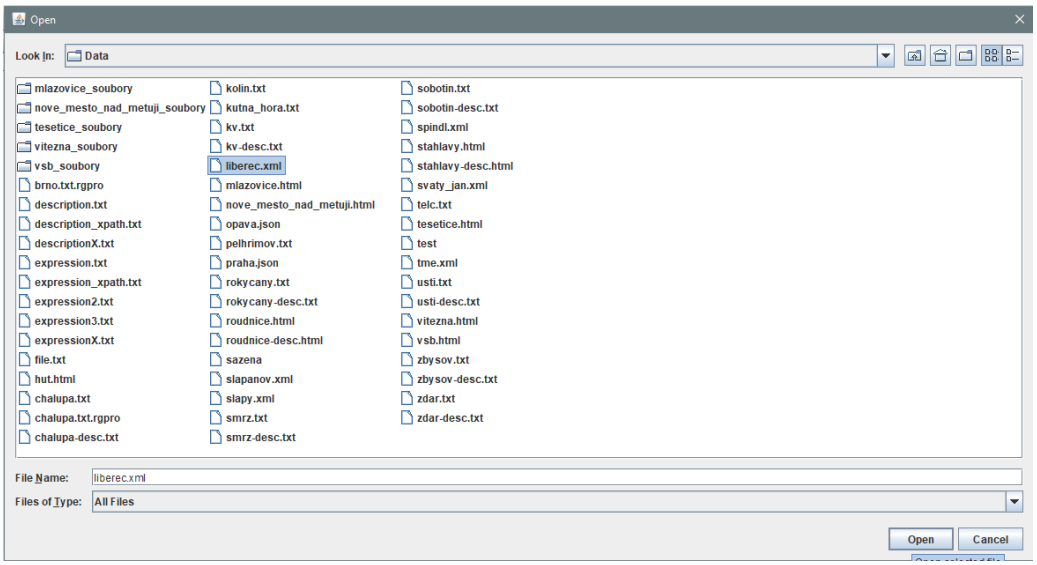
instalace pro operační systém Linux je k dispozici na stránkách Mavenu: http://maven.apache.org/install.html.

Otevření datového souboru

Pro otevření datového souboru slouží v horním menu pod položkou 'File' tlačítko 'Open' (Obrázek 7.1). Po klepnutí na toto tlačítko je možné vybrat soubor, který si uživatel přeje otevřít a po vybrání tohoto souboru se jeho obsah zobrazí v textovém poli v aplikaci (Obrázek 7.2). Pokud zvolený soubor bude mít příponu '.xml', aplikace bude primárně generovat XPath výrazy (pokud nebude vzhledem k výběru řetězce nutné generovat regulární výraz), v opačném případě se generují výrazy regulární. Text datového souboru je možné také přímo nakopírovat do textového pole v aplikaci, nicméně poté bude aplikace nebude vědět v jakém formátu se data nacházejí, tudíž bude generovat automaticky regulární výrazy i v případě, kdy jsou data v XML.

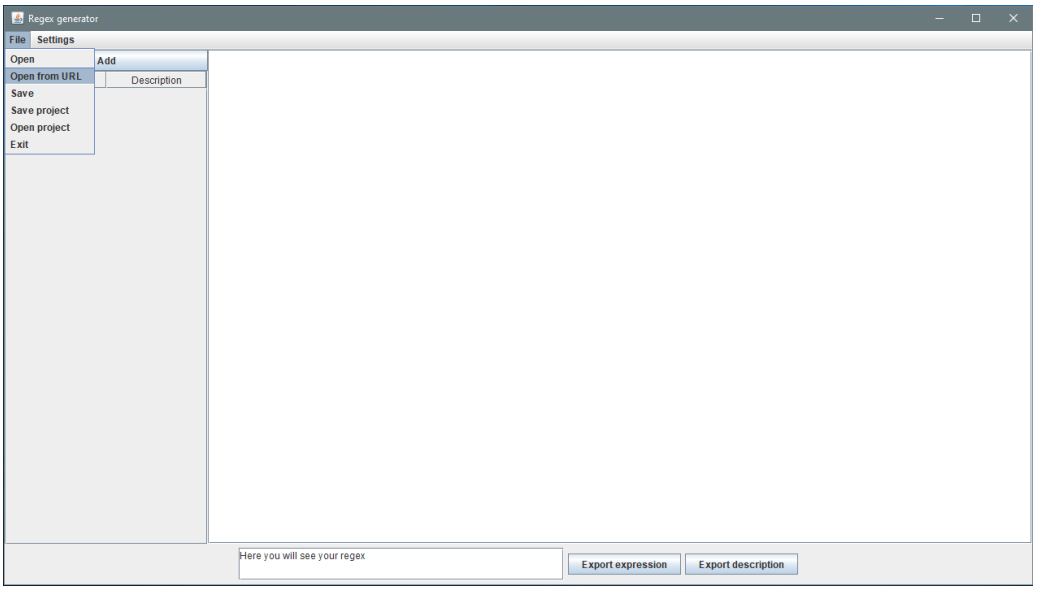


Obrázek 7.1: Otevření dat ze souboru

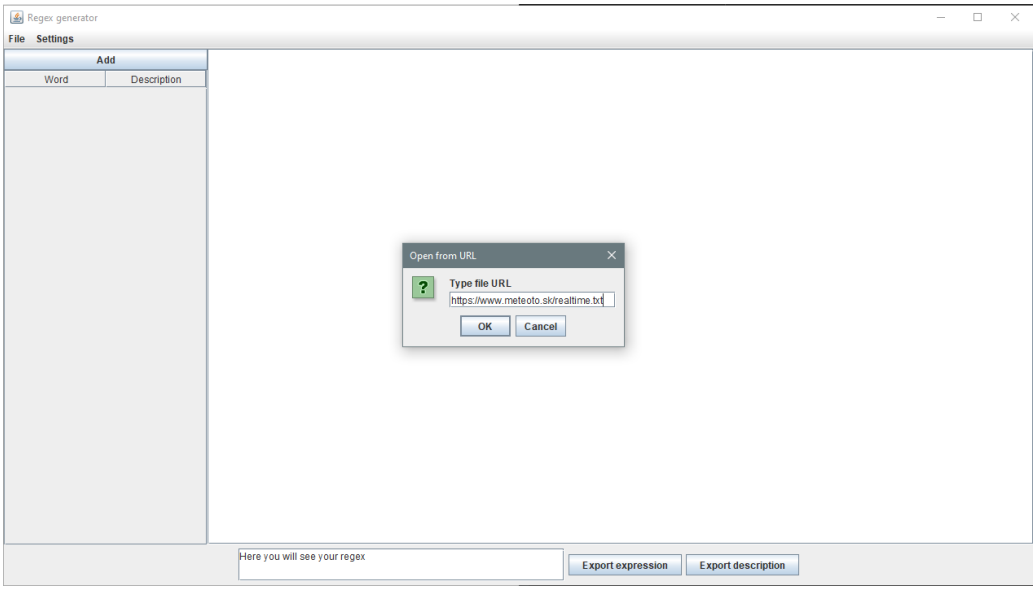


Obrázek 7.2: Vybrání souboru s daty

Datový soubor je možné také otevřít přímo z URL, k tomu slouží pod položkou 'File' tlačítko 'Open from URL' (Obrázek 7.3). Poté bude uživatel vyzván k zadání URL adresy (Obrázek 7.4) a aplikace data z dané adresy stáhne a zobrazí je v textovém poli.



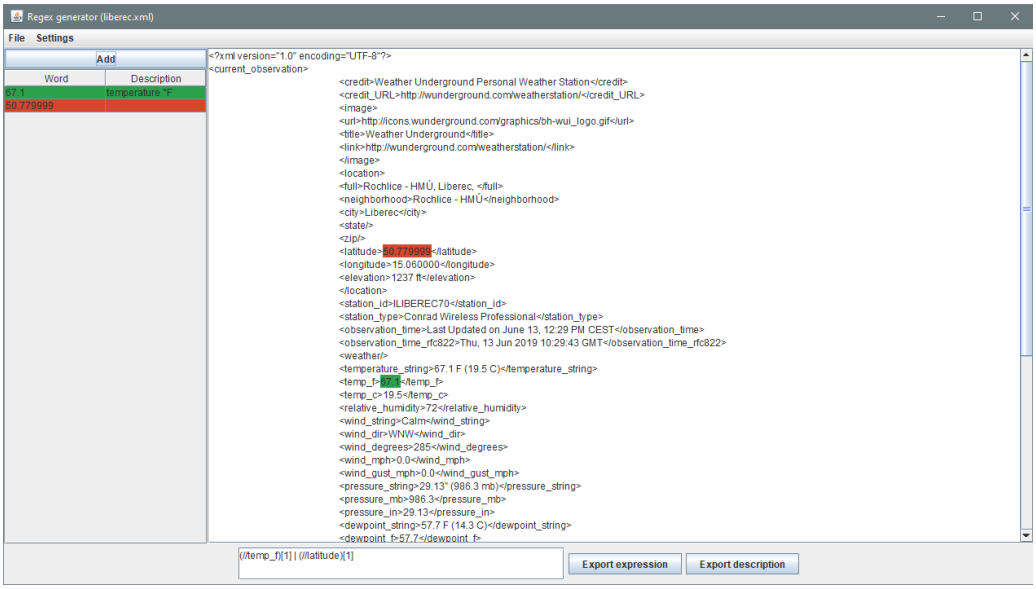
Obrázek 7.3: Otevření dat z URL



Obrázek 7.4: Vložení URL s daty

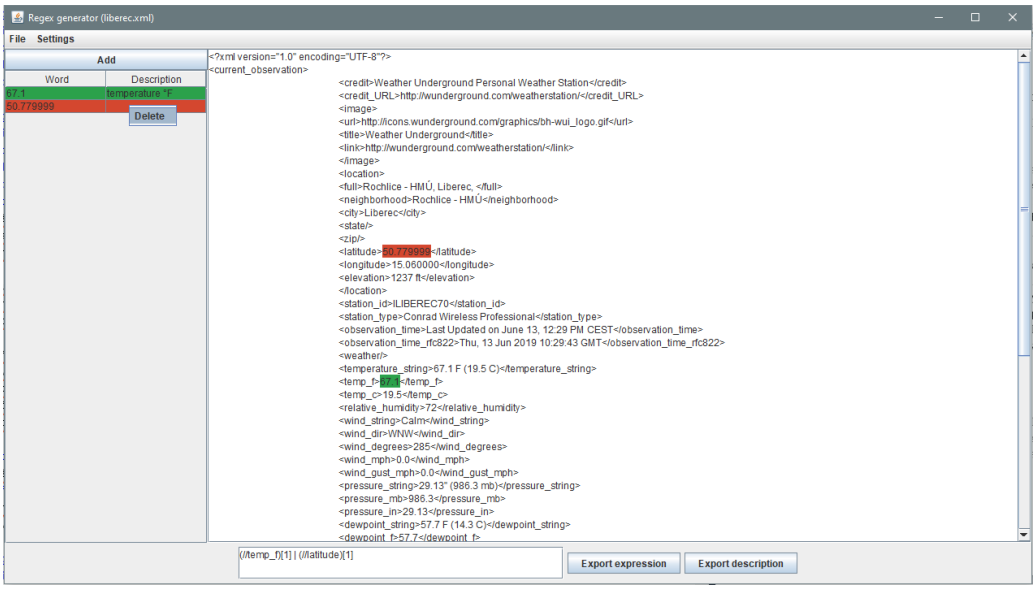
Vybírání řetězců pro generování výrazů

Když si uživatel otevře nějaký datový soubor, pro generování výrazu stačí požadovaný řetězec v textovém poli označit a klepnout na tlačítko 'Add'. Poté dojde k jeho zvýraznění v textovém poli a přidání do tabulky, ve které je možné k němu přidat popisek. Zároveň se přegeneruje výsledný výraz. Okno aplikace po přidání řetězce je na obrázku 7.5



Obrázek 7.5: Okno aplikace po přidání slova

Pokud chce uživatel řetězec z generování odebrat, stačí klepnout pravým tlačítkem myši na záznam v tabulce a vybrat 'Delete' (Obrázek 7.6). Poté dojde k odebrání řetězce z generování a odstraní se i jeho barevné zvýraznění v textu. Tuto činnost je možné vykonat vybráním řádku v tabulce a stiskem klávesy 'Delete' na klávesnici.

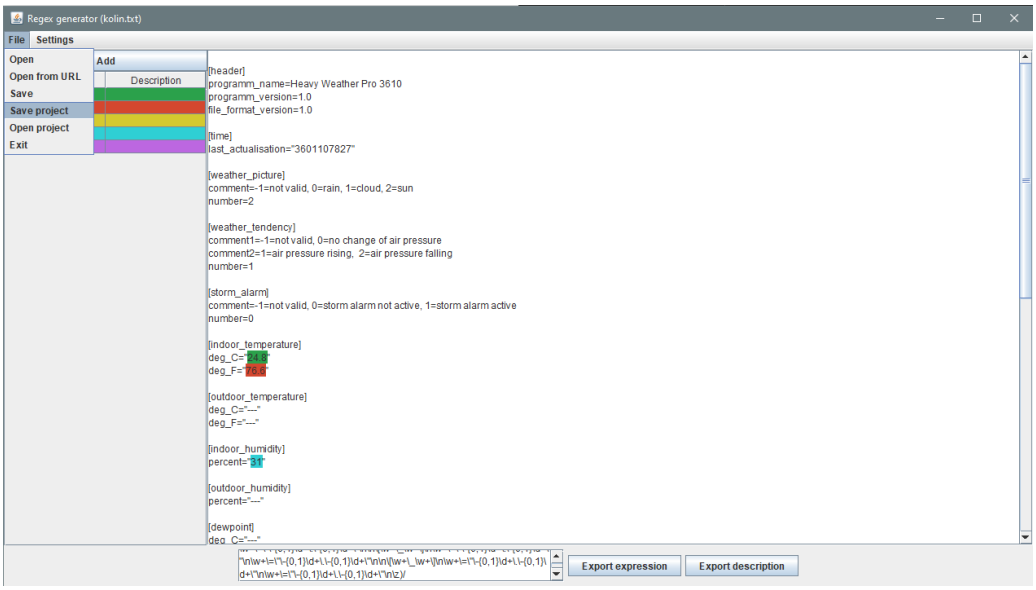


Obrázek 7.6: Odebrání řetězce

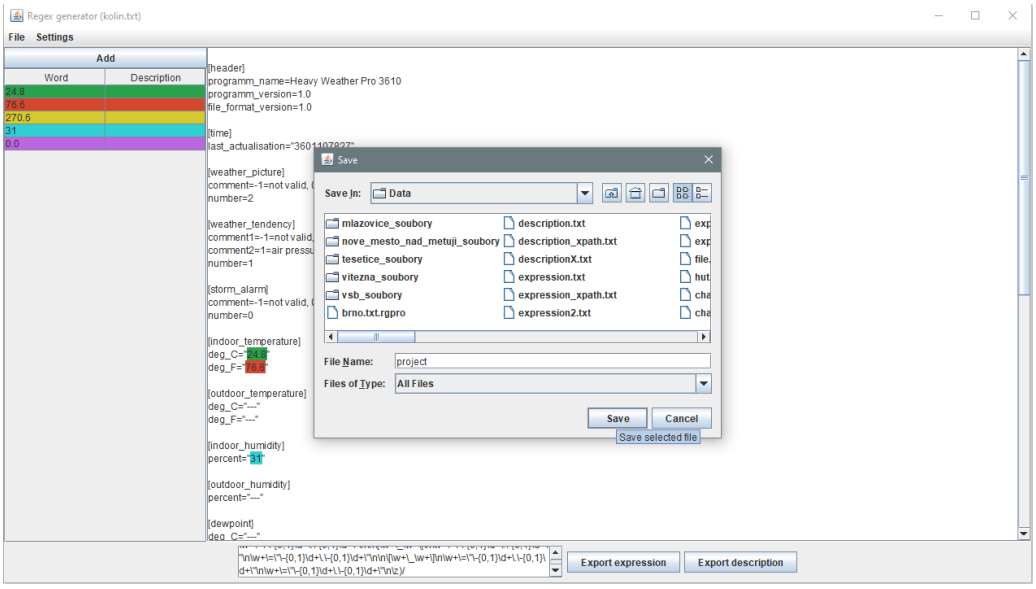
V případě, že má uživatel otevřený XML soubor, tak jsou generovány primárně XPath výrazy. V případě, že uživatel vybere pouze určitou podčást daného uzlu je ovšem nutné vygenerovat výraz regulární. V takovémto případě se regulární výraz musí vygenerovat i pro všechny ostatní řetězce, kterým by za normálních okolností stačil XPath výraz. Pokud dojde k odstranění „problémového" řetězce z generování, začnou se opět automaticky generovat XPath výrazy.

Uložení a načtení projektu

Uložení rozpracovaného projektu se provádí v horním menu pod položkou 'File' tlačítkem 'Save project' (Obrázek 7.7). Zde si v průzkumníkovi uživatel může vybrat, kam chce projekt uložit a zadat jeho název (Obrázek 7.8). Po prvotním uložení projektu je možné pro další ukládání využívat i tlačítka 'Save', kdy dojde k přepsání předchozího uložení daného projektu.

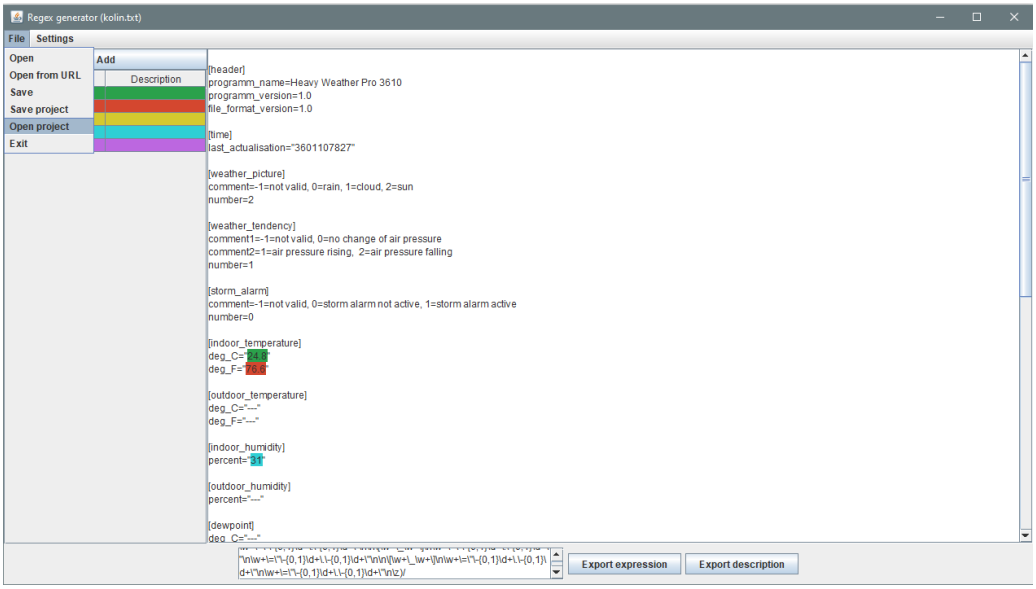


Obrázek 7.7: Uložení projektu

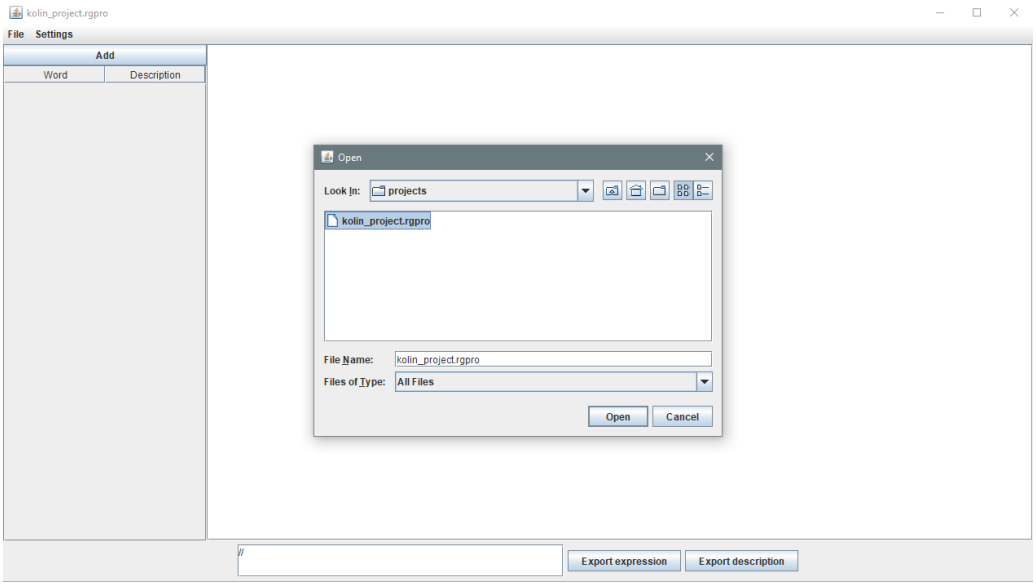


Obrázek 7.8: Výběr umístění pro uložení projektu

K načtení rozpracovaného projektu slouží v položce 'File' tlačítko 'Open Project' (Obrázek 7.9). Po klepnutí na toto tlačítko je uživatel vyzván k vybrání požadovaného projektu (Obrázek 7.10) a poté dojde k jeho otevření.



Obrázek 7.9: Otevření projektu



Obrázek 7.10: Výběr souboru s projektem

Aplikace pro zpracování dat

Spuštění aplikace

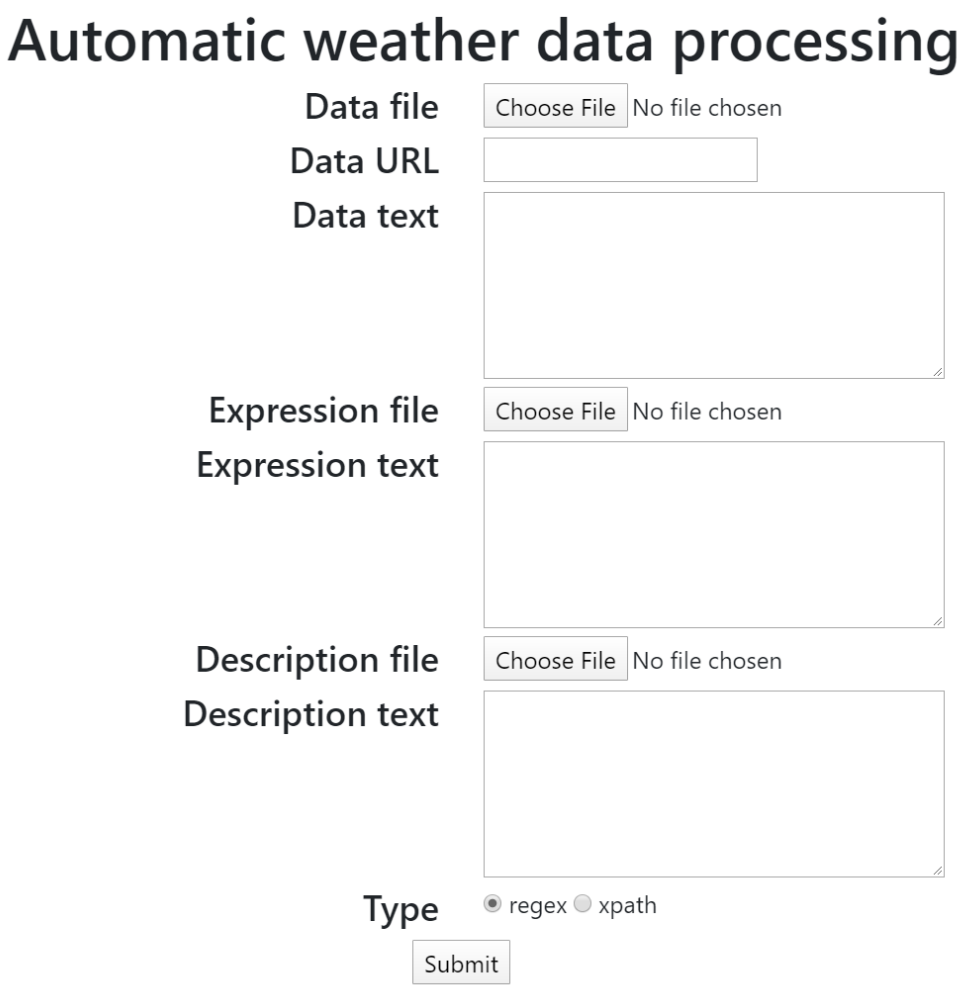
Pro spuštění aplikace je potřeba zapotřebí mít nainstalovaného interpreta jazyka PHP 7 a webový server. Nejsnazší způsob jejich instalace je v rámci nějakého balíčku, např WAMP (pro Windows) nebo XAMPP (multiplat-

formní).
WAMP ke stažení - https://sourceforge.net/projects/wampserver/
XAMPP ke stažení - https://www.apachefriends.org/index.html

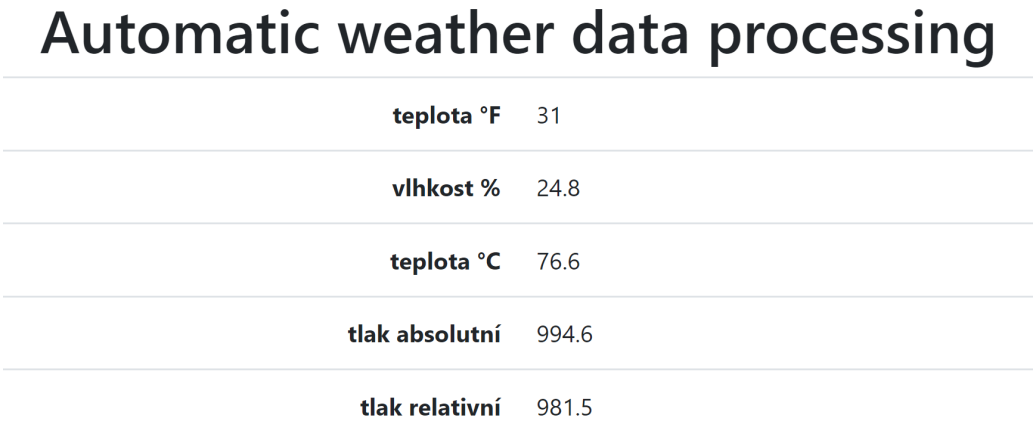
Po nainstalování je nutné vložit složku se soubory PHP aplikace (složka "PHP_aplikace") do speciální složky v této aplikaci, ze které se stránky poté načítají (u WAMP defaultně složka "www", u XAMPP složka "htdocs"). Po spuštění serveru je poté možné jít v prohlížeči na localhost/-PHP_aplikace/index.php, kde již aplikace poběží.

Nahrání dat a odeslání formuláře

Do formuláře (Obrázek 7.11) je nutné vložit data a výraz, který je zpracovává. Pro jejich vložení je zde několik způsobů, u dat je to soubor, URL adresa dat nebo samotný jejich text. Výraz je možné vložit jako soubor nebo text. Nepovinnou položkou jsou popisky, ty je možné vložit jako soubor nebo text. Poté je nutné přepínačem zvolit typ výrazu, jaký bude zpracováván. Když je formulář vyplněný, je možné kliknout na tlačítko 'Submit', čímž se data odešlou ke zpracování. Po zpracování se zobrazí tabulka, ve které jsou zobrazeny získané hodnoty s popisky (pokud byly nahrány). Příklad tabulky je na obrázku 7.12



Obrázek 7.11: Formulář PHP aplikace



Obrázek 7.12: Tabulka s výsledky