VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

PODPORA PRÍKAZOVEHO RIADKU CISCO V HIGHLIGHT.JS

CISCO COMMAND LINE SUPPORT IN HIGHLIGHT.JS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — BRANISLAV MATEÁŠ
AUTHOR

VEDOUCÍ PRÁCE — LIBOR POLČÁK, Ing., Ph.D.
SUPERVISOR

BRNO 2020

Ústav informačních systémů (UIFS) Akademický rok 2019/2020

Zadání bakalářské práce

22303

Student: **Mateáš Branislav**
Program: Informační technologie
Název: **Podpora příkazové řádky Cisco v Highlight.js**
Cisco Command Line Support in Highlight.js
Kategorie: Web
Zadání:

1. Seznamte se s formátem příkazů a režimy příkazové řádky v Cisco IOS.
2. Nastudujte možnosti specifikace gramatiky a submodulů webového frameworku Highlight.js.
3. Navrhněte rozšíření Highlight.js o gramatiku používanou příkazovou řádkou Cisco IOS.
4. Návrh implementujte a otestujte.
5. Demonstrujte použití zvýrazňování syntaxe na vhodných příkladech (např. z přednášek předmětu I1C, konfigurace v oficiální dokumentaci firmy Cisco).
6. Pokuste se o začlenění vašeho kódu do oficiální verze Highlight.js.
7. Zhodnoťte dosažené výsledky a navrhněte možnosti pokračování projektu.

Literatura:

- Introduction to Networks Companion Guide. Cisco Press, 2014. ISBN 978-1-58713-316-4.
- Routing and Switching Essentials Companion Guide, Cisco Press, 2014. ISBN 978-1-58713-318-3.
- Ivan Sagalaev. highlight.js developer documentation. Dostupná online: https://highlightjs.readthedocs.io/en/latest/

Pro udělení zápočtu za první semestr je požadováno:

- Splněné body 1 až 3 zadání včetně vypracované technické zprávy.

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

Vedoucí práce: **Polčák Libor, Ing., Ph.D.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2019
Datum odevzdání: 28. května 2020
Datum schválení: 29. října 2019

Abstrakt

Cieľom práce bolo vytvoriť nástroj na podporu používateľov Cisco zariadení. Podpora spočíva vo farebnom rozlíšení syntaxe príkazov. Výsledkom práce je nástroj na off-line prehliadanie webových stránok obsahujúcich Cisco príkazy a spätné hlásania s farebným rozlíšením kľúčových slov, IP adries, čísiel, reťazcov, domén, sieťových ciest atď. Pri vytvorení pomocného nástroja pre používateľov Cisco boli použité súčasné informačné technológie ako regulárne výrazy, ktoré sú využívané v použitom programovacom jazyku JavaScript a samotné farebné zvýrazňovanie je vyriešené s podporou Highlight.js. Navrhnutá gramatika bola publikovaná na oficiálnych stránkach internetovej komunity Highlight.js, kde sa ku nemu môžu dostať všetci zainteresovaní budúci používatelia. Súčasťou práce je aj testovanie navrhnutého nástroja na počítačoch odlišnej konfigurácie v troch najpoužívanejších internetových prehliadačoch.

Abstract

The aim of the work was to create a tool to support users of Cisco devices. Support is based on color-coded command syntax. The result of the work is a tool for off-line browsing of web pages containing Cisco commands and configuration statements with color resolution of keywords, IP addresses, numbers, strings, domains, network paths, etc. When creating the utility for Cisco users, current information technologies were used as regular expressions, which are used in the JavaScript programming language used, and the color highlighting itself is solved as an extension of the Highlight.js tool. The proposed grammar was published on the official website of the Highlight.js internet community, where it can be accessed by all interested future users. Part of the work is also testing the proposed tool on computers of different configurations in the three most used Internet browsers.

Kľúčové slová

hljs, highlight.js, highlightjs, cisco, cli

Keywords

hljs, highlight.js, highlightjs, cisco, cli

Citácia

MATEÁŠ, Branislav. *Podpora príkazoveho riadku Cisco v Highlight.js.* Brno, 2020. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Libor Polčák, Ing., Ph.D.

Podpora príkazoveho riadku Cisco v Highlight.js

Prehlásenie

Čestne vyhlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána Libora Polčáka. Uviedol som všetky literárne pramene, publikácie a ďalšie zdroje, z ktorých som čerpal.

. . . . . .
Branislav Mateáš
4. júna 2020

Poďakovanie

Ďakujem môjmu školiteľovi Liborovi Polčákovi za užitočné pripomienky, ochotu a usmernenie pri písaní bakalárskej práce. Taktiež ďakujem Jakubovi Gordošovi zo spoločnosti Marlink s.r.o za odborné konzultácie a za poskytnutie údajov potrebných pre vypracovanie praktickej časti bakalárskej práce.A

Obsah

Kapitola 1

Úvod

V tejto práci sa budeme zaoberať príkazmi a konfiguračnými výpismi Cisco sieťových zariadení. Zameriame sa na farebné rozlišovanie statických Cisco príkazov a konfiguračných výpisov. Ide o oblasť, ktorej sa doposiaľ dostávalo málo pozornosti. Samotná práca bude prebiehať na webovej stránke internetového prehliadača. Webové rozhranie sme zvolili pre jeho dostupnosť a jednoduchosť používania.

Výuka v oblasti Cisco zariadení vyžaduje tvorbu študijných materiálov, v ktorých je doposiaľ možné farebne odlišovať syntax príkazov iba manuálne. Nami vytvorený nástroj môže byť integrovaný do systému Reveal.js, ktorý slúži na tvorbu prezentácií a zabezpečiť tak zvýrazňovanie syntaxe v pripravovaných materiáloch automaticky.

Perspektívne využitie tohto nástroja vidíme v jeho rozšírení medzi používateľmi, ktorí už používajú Cisco zariadenia ako aj tých, ktorí Cisco zariadenia ešte len študujú. Existujúci užívatelia môžu tento nástroj využívať na off-line analyzovanie zaznamenaných vykonaných príkazov a konfiguračných výpisov mimo konfiguračného prostredia zariadení. Študujúci užívatelia môžu tento nástroj využívať na výučbu a osvojovanie si syntaxe príkazov.

Predmetom tejto práce je snaha vytvoriť nástroj na farebné zvýrazňovanie týchto príkazov a konfiguračných výpisov Cisco zariadení v prostredí webových prehliadačov formou webovej stránky.

V kapitolách 2 a 3 sa dozvie čitateľ textu o technológiách zaoberajúcich sa touto prácou. Dozvie sa základy nutné pre pochopenie praktickej časti. V kapitole 4 je popísaný návrh praktickej časti práce, od získavania potrebných materiálov pre zvýrazňovanie, cez samotný návrh ako sa samostatné príkazy budú zvýrazňovať a návrh odpovedajúcich regulárnych výrazov. V kapitole 5 si čitateľ môže prečítať popis implementácie gramatiky pre zvýrazňovač týchto príkazov a aké kroky boli nutné pre zaradenie gramatiky do oficiálnej distribúcie Highlight.js. V kapitole 6 sa čitateľ dozvie, akým spôsobom prebiehalo testovanie zvýrazňovania, regulárnych výrazov, webových prehliadačov a samotného zvýrazňovača v praxi. V závere 7 sa čitateľ dozvie zhrnutie dosiahnutých výsledkov práce a aké sú ďalšie plány, vízie a pokračovanie s touto prácou.

Kapitola 2

Príkazový riadok Cisco IOS

Každý počítač na to aby fungoval potrebuje operačný systém, týmto sa myslia aj zariadenia ako sú napríklad prepínače, smerovače, prístupové body, brány firewall, atď. Tieto sieťové zariadenia však potrebujú operačný systém, ktorý je pre prácu so sieťovými prostriedkami prispôsobený. V texte ho budeme ďalej označovať sieťový operačný systém. Definícia sieťového operačného systému podľa firmy Cisco[1]: „*Sieťový operačný systém umožňuje hardvéru zariadenia fungovať a splňať svoj účel a zároveň poskytuje rozhranie pre interakciu s používateľom alebo administrátorom. Cisco IOS(Internetwork Operating system) je všeobecný pojem pre kolekciu sieťových operačných systémov používaných firmou Cisco. Cisco IOS sa používa pre väčšinu sieťových zariadení od firmy Cisco bez ohľadu na typ alebo veľkosť.*“

2.1 Operačný systém

Sieťový operačný systém môžeme v prípade sieťových zariadení rozdeliť na dve časti:

- **Kernel** — časť operačného systému, ktorá priamo interaguje s hardvérom zariadenia.
- **Shell** — časť operačného systému, ktorá slúži na interakciu s aplikáciami a používateľmi.

Používateľ môže interagovať so shell-om dvoma spôsobmi, a to buď použitím rozhrania príkazového riadku, alebo grafického používateľského rozhrania.

Používaním **rozhrania príkazového riadku** interaguje používateľ priamo so systémom v plne textovom prostredí zadávaním príkazov na klávesnici do príkazového riadku. Systém potom zadaný príkaz vykoná a vo väčšine prípadov aj poskytne textovú spätnú väzbu. Výhodou používania rozhrania príkazového riadku je, že vyžaduje veľmi málo režijných nákladov. Ale pritom vyžaduje od používateľa, aby mal vedomosti o základnej štruktúre, ktorá riadi systém.

Grafické používateľské rozhrania ako sú Windows, Apple OS X a iOS, alebo Android umožňujú používateľovi interagovať so systémom pomocou grafického používateľského rozhrania, ktoré je špecifické obsahom grafických elementov ako sú ikonky, menu, okná, atď. Výhodou grafického používateľského rozhrania je jeho prívetivosť a nevyžaduje od používateľa až takú znalosť základnej štruktúry. Väčšinou každé grafické používateľské rozhranie istou formou poskytuje *pomocníka*, ktorý používateľa sprevádza pri konfigurovaní sieťového zariadenia. Vďaka týmto vlastnostiam grafického používateľského rozhrania sa veľa používateľov spolieha práve na toto rozhranie. Nevýhodou však je, že grafické používateľské rozhranie nie je vždy schopné poskytnúť všetky funkcie dostupné v rozhraní príkazového

riadku. Taktiež toto rozhranie môže zlyhať alebo jednoducho nebude fungovať podľa pokynov.

Práve z týchto dôvodov sa k sieťovému zariadeniu používateľ pripojí najčastejšie prostredníctvom rozhrania príkazového riadku. Má oveľa menšie nároky na hardvér zariadenia a je oproti grafickému používateľskému rozhraniu oveľa stabilnejšie. Pri zariadeniach, ktoré sú určené na `domáce` použitie, je najbežnejšou metódou konfigurácie použitie grafického používateľského rozhrania, ktoré je vo väčšine prípadov dostupné cez webový prehliadač. Sieťový operačný systém v takomto zariadení sa nazýva **firmvér**.

2.1.1 Účel operačného systému

Sieťové zariadenia Cisco používajú konkrétne verzie sieťového operačného systému Cisco IOS. Verzia Cisco IOS závisí od typu používaného zariadenia a požadovaných funkcií. Zatiaľ čo všetky zariadenia sú dodávané s predvolenou verziou Cisco IOS a skupinou funkcií, je možné inovovať verziu IOS alebo sadu funkcii pre získanie ďalších možností

Sieťový operačný systém založený na rozhraní príkazového riadku, ako je Cisco IOS na prepínači alebo smerovači, umožňuje sieťovému administrátorovi:

- použiť klávesnicu na spúšťanie sieťový programov založených na rozhraní príkazového riadku
- použiť klávesnicu na zadávanie textu a textových príkazov
- zobrazenie výstupu na monitore

Sieťové operačné systémy sú podobné počítačovým operačným systémom. Prostredníctvom grafického používateľského prostredia umožňuje operačný systém zariadenia používateľovi:

- použitie myši na ovládanie, robenie výberov a spúšťanie programov
- zadávanie textových príkazov
- vyberanie možností v dialógovom okne
- zobrazenie výstupu na monitore

2.1.2 Cisco IOS

Všetky sieťové zariadenia obsahujúce Cisco IOS vykonávajú potrebné funkcie pre to, aby počítačová sieť fungovala bez problémov. Každá táto funkcia má priradenú skupinu konfiguračných príkazov, ktoré umožňujú sieťovému administrátorovi tieto funkcie implementovať a používať. Služby poskytované spoločnosťou Cisco IOS sú všeobecne prístupné pomocou príkazovej riadky. Medzi hlavné funkcie, ktoré tieto zariadenia umožňujú patria:

- zabezpečenie siete,
- adresovanie virtuálnych a fyzických rozhraní IP,
- konfigurácie špecifické pre rozhranie na optimalizáciu prepojiteľnosti príslušného média,
- smerovanie,

- monitorovanie stavu siete,
- použitie technológií kvality služieb (QoS),
- technológie pre správu siete,
- prepínanie rámcov a posielanie paketov.

2.1.3 Prístup k rozhraniu príkazového riadku

Existuje niekoľko rôznych spôsobov ako získať prístup k rozhraniu príkazového riadku. Tie najbežnejšie metódy sú:

- **Konzola** sa používa na priame pripojenie počítača k sieťovému zariadeniu a jeho konfiguráciu. Tento port sa zvyčajne používa na počiatočnú konfiguráciu, pretože na pripojenie cez SSH, HTTP, alebo HTTPS nie spočiatku žiadne zariadenie nakonfigurované.
- **SSH(Secure Shell)** je metóda na vzdialené zabezpečené šifrované pripojenie k rozhraniu príkazového riadku cez virtuálne rozhranie cez počítačovú sieť. Pripojenie SSH narozdiel od konzoly vyžaduje na zariadenach aktívne sieťové služby a aktívne sieťové rozhrania s nakonfigurovanou adresou.
- **Telnet** je metóda na vzdialené **ne**zabezpečené pripojenie k rozhraniu príkazového riadku cez virtuálne rozhranie cez počítačovú sieť. Telnet narozdiel od SSH neposkytuje zabezpečené pripojenie. Všetky informácie ako overovanie používateľov, heslá a príkazy sa odosielajú prostredníctvom siete vo formáte obyčajného textu.

2.2 Cisco IOS režimy

Po pripojení sieťového administrátora k zariadeniu je možné sieťové zariadenie nakonfigurovať. Sieťový operačný systém Cisco IOS je založený na hierarchickej štruktúre režimov, kde každý režim poskytuje administrátorovi určité funkcie špecifické pre daný režim. Sieťový administrátor tak musí počas konfigurácie sieťového zariadenia prechádzať rôznymi režimami Cisco IOS V hierarchickom poradí od najzákladnejších po najšpecializovanejšie sú režimy príkazového riadku:

- užívateľský režim,
- privilegovaný režim,
- globálny konfiguračný režim,
- ďalšie špecifické konfiguračné režimy, napríklad režim konfigurácie rozhrania.

2.2.1 Užívateľský režim

Prvým režimom, s ktorým administrátor príde do kontaktu po pripojení k príkazovému riadku sieťového zariadenia, sa nazýva užívateľský režim. Tento režim má obmedzené možnosti, ale pre niektoré základne operácie je užitočný. Poskytuje iba obmedzený počet základných monitorovacích príkazov, preto sa často označuje ako režim len na zobrazenie. Úroveň tohto režimu neumožňuje vykonávanie príkazov, ktoré by nejakým spôsobom mohli

modifikovať konfiguráciu zariadenia. Pre prístup do užívateľského režimu sa zvyčajne autentifikácia nevyžaduje, je však dobré zabezpečiť aby bola autentifikácia nakonfigurovaná počas počiatočnej konfigurácie. Na ochranu užívateľského režimu môžeme na zariadení vytvoriť kombináciu používateľského mena a hesla. Užívateľský režim je identifikovaný výzvou príkazového riadku, ktorá končí symbolom „>“. Najpoužívanejšími skupinami príkazov, ktoré administrátor môže použiť v tomto režime sú:

- Obsahuje príkazy, ktoré môžeme použiť na testovanie konfigurácie zariadenia / siete, ako sú napríklad `ping` a `traceroute`.
- K dispozícii je skupina príkazov, ktoré zobrazujú konfiguráciu zariadenia, začínajú príkazom `show`.
- K ďalšiemu zariadeniu sa môžeme pripojiť z užívateľského režimu pomocou príkazov `telnet` alebo `ssh`.
- Zadaním príkazu `exit` sa relácia odpojí.

2.2.2 Privilegovaný režim

Aby mohol administrátor vykonávať príkazy na konfiguráciu a správu zariadenia potrebuje používať privilegovaný režim, alebo konkrétnejší režim v hierarchii režimov. To znamená, že administrátor musí najprv vstúpiť do užívateľského režimu a z tohto režimu vstúpiť do privilegovaného režimu. Do privilegovaného režimu sa dostane administrátor príkazom `switch> enable`. Po vykonaní príkazu sa zmení výzva na `switch#` a v tomto stave môže administrátor zadávať príkazy privilegovaného režimu. V predvolenom nastavení tento režim nevyžaduje autentifikáciu. Je však opäť dobré zabezpečiť aby bola autentifikácia pri počiatočnej konfigurácii nakonfigurovaná. Globálny režim a všetky ďalšie špecifickejšie konfiguračné režimy sú dostupné iba z privilegovaného režimu.

- Tu sú k dispozícii všetky príkazy, ktoré sú k dispozícii v užívateľskom režime
- Komplexnejšia skupina príkazov `show`. Napríklad v užívateľskom režime neexistuje príkaz `show running-config`, ale v privilegovanom režime existuje.
- Zadaním príkazu `exit` sa relácia odpojí.

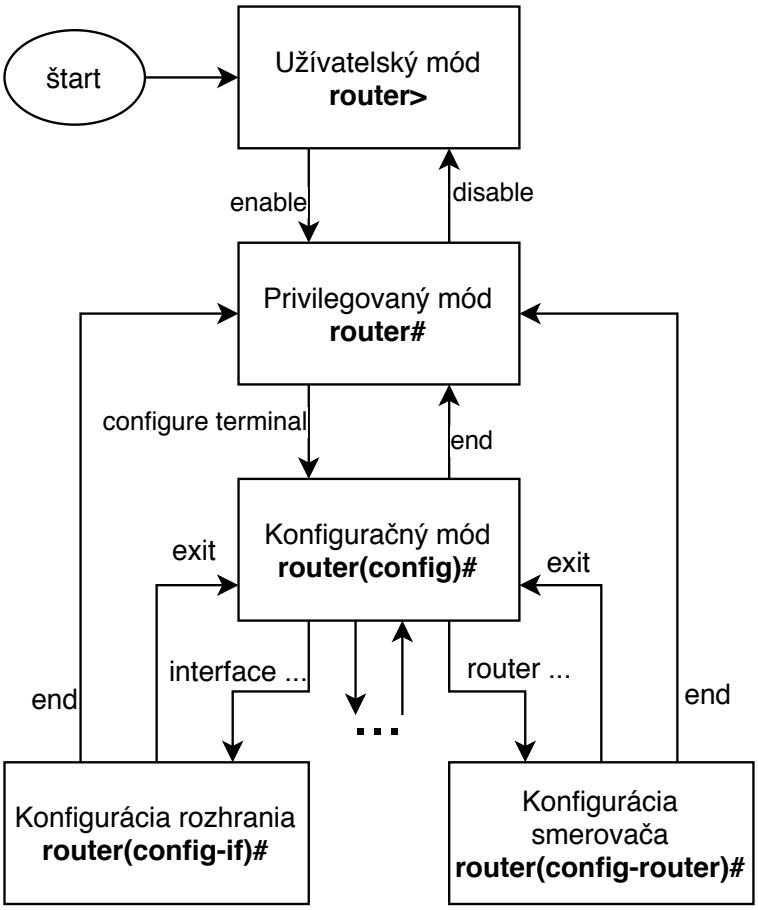
2.2.3 Globálny konfiguračný režim

Primárnym konfiguračným režimom, do ktorého vstúpi administrátor, sa nazýva globálny konfiguračný režim. V tomto režime už môže administrátor vykonávať príkazy, ktoré modifikujú konfiguráciu zariadenia a ovplyvňujú tak činnosť zariadenia ako celku. Nasledujúci príkaz `Switch# configure terminal` príkazového riadku sa používa na prepnutie z privilegovaného režimu do globálneho konfiguračného režimu. Po vykonaní príkazu sa výzva zmení na výzvu reprezentujúcu globálny konfiguračný režim `Switch(config)#`. V tomto stave už môže administrátor zadávať konfiguračné príkazy. Ak chce administrátor použiť príkazy hierarchicky nižších režimov, je nutné použiť na začiatku týchto príkazov kľúčové slovo `do`. Režim globálnej konfigurácie možno chrániť priradením úrovne užívateľských oprávnení k užívateľskému účtu, potom nastaviť povolené príkazy a zablokovať ostatné

2.2.4 Špecifický konfiguračný režim

Z globálneho režimu môže administrátor vstúpiť do rôznych špecifických konfiguračných režimov. Každý z týchto režimov umožňuje administrátorovi konfiguráciu konkrétnej časti alebo funkcie sieťového zariadenia. Pre ľahšiu orientáciu v príkazovom riadku poskytuje výzva príkazového riadku identifikátor, ktorý sa nachádza v oblých zátvorkách. Tento identifikátor informuje administrátora o tom, v ktorom špecifickom konfiguračnom režime sa nachádza. Napríklad, pokiaľ administrátor je v režime konfigurácie fyzického rozhrania sieťového zariadenia, je reprezentovaný tento režim identifikátorom **config-if** a teda výzva príkazového riadku v tomto prípade vyzerá `router(config-if)#`.

Ak chce administrátor ukončiť konkrétny konfiguračný režim a vrátiť sa do globálneho konfiguračného režimu, použije príkaz `exit`. Ak chce administrátor úplne opustiť konfiguračný režim a vrátiť sa do privilegovaného režimu, použije príkaz `end` alebo postupnosť kláves Ctrl-Z. Prehľad hierarchie režimov je znázornený na obrázku 2.1.



Obr. 2.1: Prehľad jednotlivých režimov, prevzaté z [3]

2.3 Formát príkazov

Sieťový administrátor si nemôže zapamätať každý možný príkaz. Pochopenie všeobecnej štruktúry príkazov IOS môže administrátorovi pomôcť pri zadávaní príkazov. Preto si v tejto sekcii predstavíme štruktúru a písomné znázornenie príkazov Cisco IOS.

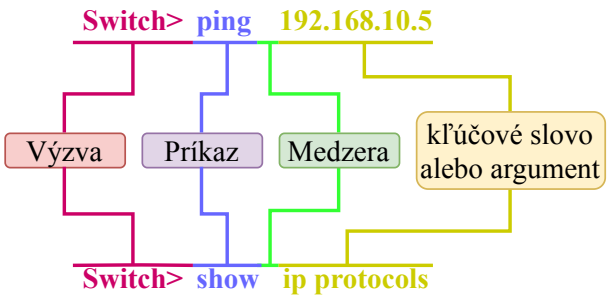
2.3.1 Základný formát príkazov

Sieťový operačný systém Cisco IOS obsahuje veľké množstvo príkazov. Pre každý príkaz platí, že má svoj špecifický formát a syntax a môže sa vykonávať iba v príslušnom režime. Zo všeobecného hľadiska pozostáva syntax príkazu z príkazu (ide o slová, ktoré majú svoj pevný význam a administrátor ich nemôže meniť), za ktorým nasledujú vhodné kľúčové slová a argumenty príkazu. Existujú príkazy, ktoré ďalej obsahujú podmnožinu ďalších príkazov, kľúčových slov a argumentov, ktoré poskytujú ďalšie funkcie. Príkazy sú určené pre vykonanie akcie a kľúčové slova spolu s argumentami slúžia na identifikáciu toho, kde a ako vykonať konkrétny príkaz.

Ako je znázornené na obrázku 2.2, príkaz je počiatočné slovo alebo slová zadané v príkazovom riadku hneď po výzve. Interpret príkazov nerozlišuje veľké a malé písmená príkazov. Za príkazom môže nasledovať jedno alebo viac kľúčových slov a argumentov. Po zadaní kompletného príkazu, musí administrátor stlačiť klávesu „Enter“, aby odoslal zadaný príkaz do interpretu príkazov. Kľúčové slová slúžia na popis jednotlivých parametrov príkazu pre interpret. Napríklad príkaz `show`, ktorý sa používa na zobrazenie informácii o konfigurácii zariadenia. Za príkazom show ide množina kľúčových slov, z ktorých sa musí práve jedno použiť pre definovanie informácii o konfigurácii, ktoré sa majú zobraziť na výstupe. Napríklad:

```
Switch# show running-config
```

Za príkazom `show` nasleduje kľúčové slovo **running-config**. Kľúčové slovo špecifikuje, že aktuálne bežiaca konfigurácia zariadenia sa má zobraziť na výstup príkazového riadku.



Obr. 2.2: Štruktúra Cisco IOS príkazu, prevzaté z [2]

2.4 Ovládanie príkazového riadku

Štandardne administrátor zadáva príkazy z klávesnice v štruktúre uvedenej vyššie. Od administrátora sa neočakáva, že si bude presne pamätať každý príkaz. Alebo sa môže stať, že administrátor omylom zadá chybný argument alebo kľúčové slovo, ktoré je syntakticky správne a vykoná sa teda príkaz s nechceným efektom. Preto rozhranie príkazového riadku poskytuje administrátorovi možnosti dopĺňania príkazov, uľahčuje prácu pri vrátení zmien vykonaných príkazmi, prípadne poskytuje klávesové skratky pre uľahčenie ovládania v prostredí príkazového riadku.

2.4.1 Pohyb kurzoru

Keďže ide o plne textové prostredie, často sa stane, že má administrátor na ovládanie kurzoru v príkazovom riadku k dispozícii len klávesnicu. Preto rozhranie príkazového riadku poskytuje kombinácie kláves alebo ich postupnosti (uvedené v tabuľke 2.1), ktoré umožňujú administrátorovi pohyb s kurzorom v príkazovom riadku. Klávesové skratky nerozlišujú malé a veľké písmená. Kombinácie obsahujúce tlačidlo Ctrl, označujú kombináciu, kedy musia byť obe klávesy stlačené súčasne. Esc označuje kombináciu, kedy je nutné najprv stlačiť Esc a potom priradenú klávesu.

Klávesová skratka	**Popis funkcie**
Šípka doľava alebo Ctrl-B	Posunie kurzor o jeden znak doľava
Šípka doprava alebo Ctrl-F	Posúva kurzor o jeden znak doprava.
Esc, B	Presunie kurzor o jedno slovo späť.
Esc, F	Posúva kurzor o jedno slovo vpred.
Ctrl-A	Presunie kurzor na začiatok riadku.
Ctrl-B	Presunie kurzor na koniec príkazového riadku.

Tabuľka 2.1: Klávsové skratky pre ovládanie kurzoru

2.4.2 Odstránenie znakov okolo kurzora

V prípade, že administrátor urobí chybu alebo jednoducho zmení názor a chce napríklad zmeniť zadaný argument alebo kľúčové slovo, môže administrátor pomocou klávesových skratiek uvedených v tabuľke 2.2 vymazať znaky alebo celé elementy príkazov.

Klávesová skratka	**Popis funkcie**
Delete alebo Backspace	Vymaže znak naľavo od kurzora.
Ctrl-D	Posúva kurzor o jeden znak doprava.
Ctrl-K	Odstráni všetky znaky od kurzora po koniec príkazového riadku.
Ctrl-U alebo Ctrl-X	Odstráni všetky znaky od kurzora po začiatok príkazového riadku.
Ctrl-W	Vymaže slovo naľavo od kurzora.
Esc, D	Odstráni text od kurzora po koniec slova.

Tabuľka 2.2: Klávesové skratky pre mazanie obsahu

2.4.3 Obnova obsahu a presun medzi režimami

Rozhranie príkazového riadku Cisco IOS ukladá príkazy a kľúčové slová, ktoré administrátor odstráni do vyrovnávacej pamäte. Vo vyrovnávacej pamäti sú uložené iba celistvé reťazce. Jednotlivé znaky, ktoré administrátor odstráni použitím kláves Backspace alebo klávesovou skratkout Ctrl-D, sa do vyrovnávacej pamäte neuložia. Vyrovnávacia pamäť si dokáže uložiť maximálne 10 položiek a to iba položky, ktoré boli odstránené klávesovými skratkami Ctrl-K, Ctrl-X a Ctrl-U. Pre vytiahnutie položky z tejto vyrovnávacej pamäte môže administrátor použiť klávesové skratky umiestnené v tabuľke 2.3. V prípade, že administrátor chce vykonať prechod z hierarchicky nadradeného režimu do podradeného, môže namiesto príkazov `end` a `exit` použiť klávesové skratky v tabuľke 2.4.

Klávesová skratka	Popis funkcie
Ctrl-Y	Vyvolá najnovší záznam vo vyrovnávacej pamäti
Esc, Y	Vyvolá predchádzajúcu položku vo vyrovnávacej pamäti histórie

Tabuľka 2.3: Klávesové skratky pre obnovu obsahu

Klávesová skratka	Popis funkcie
Ctrl-C	Keď je v ľubovoľnom konfiguračnom režime, ukončí konfiguračný režim a vráti sa do privilegovaného režimu EXEC. V režime nastavenia sa zruší návrat na príkazový riadok.
Ctrl-Z	Keď je v ľubovoľnom konfiguračnom režime, ukončí konfiguračný režim a vráti sa do privilegovaného režimu EXEC.

Tabuľka 2.4: Klávesové skratky pre ukončenie režimu

2.4.4 IOS funkcia pomocníka

Kontextová pomoc umožňuje administrátorovi rýchlo nájsť, ktoré príkazy sú k dispozícií v každom príkazovom režime. ktoré príkazy začínajú špecifickým znakom alebo skupinou znakov, a ktoré argumenty a kľúčové slová sú dostupné pre jednotlivé príkazy. Ak chce administrátor získať prístup ku kontextovej pomoci, jednoducho do rozhrania príkazového riadku napíše **?**. Spôsob použitia kontextovej pomoci v príkazovom riadku:

- **Router# skrátený-príkaz?** - Zobrazí všetky príkazy v aktuálnom režime, ktoré začínajú konkrétnym reťazcom znakov
- **Router# ?** - zobrazí zoznam všetkých príkazov dostupných v danom príkazovom režime.
- **Router# príkaz ?** - zobrazí dostupné možnosti príkazu (argumenty a kľúčové slová) pre príkaz.
- **Router# príkaz kľúčové_slovo ?** - zobrazí ďalšiu dostupnú možnosť príkazu

Po zadaní príkazu administrátorom prebehne **kontrola syntaxe** príkazu, ktorá overuje či administrátor zadal platný príkaz. Interpret príkazového riadku Cisco IOS vyhodnotí zadaný príkaz zľava doprava. Ak interpret porozumie zadanému príkazu, vykoná sa odpovedajúca akcia a rozhranie príkazového riadku čaká na ďalší príkaz. Ak ale interpret príkazového riadu neporozumie zadanému príkazu, poskytne na výstup spätnú väzbu, čo je v zadanom príkaze zle.

2.4.5 Používanie do a default

Takmer každý konfiguračný príkaz má formu, kde na začiatku príkazu sa nachádza kľúčové slovo **no**. Vo všeobecnosti táto forma slúži na zakázanie nejakých vlastností, alebo funkcii, ale taktiež slúži na vrátenie zmien vykonaných nejakým príkazom. Ak napíšeme nejaký príkaz, ktorý vykoná nejaké zmeny, tak zadaním toho istého príkazu s kľúčovým slovom **no** na začiatku, tieto zmeny na predvolené nastavenie. Väčšina príkazov rozhrania príkazového riadku má taktiež formu, kde na začiatku sa nachádza kľúčové slovo **default**. Vykonaním takéhoto príkazu sa nakonfiguruje príkaz na predvolené nastavenie.

2.4.6 Skratky príkazov a história príkazov

V rozhraní príkazové riadku Cisco IOS nemusí administrátor počas konfigurácie sieťového zariadenia nutne zadávať celý príkaz. Rozhranie totiž poskytuje možnosť vo forme skrátených príkazov, ktoré uľahčujú konfiguráciu, monitorovanie a riešenie problémov. Príkazy a kľúčové slová príkazu môžu byť skrátené na minimálny počet znakov, tak aby stále identifikoval jedinečný príkaz. Cieľom tohto je zjednodušiť písanie dlhších príkazov, ktoré pozostávajú z väčšieho počtu kľúčových slov, príkazov a argumentov. Ak administrátor zadá skrátený príkaz a táto skrátená verzia už reprezentuje jedinečný príkaz, interpret tento príkaz doplní a vyhodnotí tak ako keby bol ten príkaz napísaný celý. Napríklad ak zadá administrátor príkaz conf, priradí interpret tento príkaz k príkazu configure a tento príkaz vykoná, pretože žiadny iný príkaz nezačína na conf. Naopak ak administrátor zadá príkaz con, interpret tento príkaz nevykoná, pretože existuje viacero príkazov, ktoré začínajú s prefixom con.

Rozhranie príkazového riadku Cisco IOS si tiež ukladá do vyrovnávacej pamäti históriu administrátorom zadaných príkazov. Táto funkcia je užitočná najmä pri zadávaní zložitých a dlhých príkazov, v ktorých sa napr. mení len jeden znak atribútu. Pokiaľ chce administrátor použiť príkaz, ktorý sa už nachádza vo vyrovnávajúcej pamäti, môže k tomuto príkazu pristúpiť pomocou klávesy „šípka hore“, alebo klávesovými skratkami Ctrl-P a Ctrl-N.

Kapitola 3

Zvýrazňovač syntaxe Highlight.js

3.1 Syntax programovacieho jazyka

Friedman et al.[4] uvádzajú: „*V počítačovej vede je syntax počítačového jazyka súbor pravidiel, ktoré definujú kombináciu symbolov, ktoré sa v tomto jazyku považujú za správne štruktúrovaný dokument alebo fragment v tomto jazyku*“. Toto platí ako pre programovacie jazyky, kde dokument reprezentuje zdrojový kód daného jazyka, tak aj pre značkovacie jazyky, kde dokument reprezentujú údaje. Počítačové jazyky môžeme rozdeliť do dvoch skupín:

- textové počítačové jazyky - založené na postupnosti znakov,
- vizuálne počítačové jazyky – založené na priestorovom usporiadaní a prepojení medzi symbolmi, ktoré môžu byť grafické alebo textové.

Syntax textových programovacích sa z hľadiska lexikálnej štruktúry definuje pomocou regulárnych výrazov. Z hľadiska gramatickej štruktúry sa používa Backus-Naurová forma, aby sa určili neterminálne a terminálne symboly. Syntaktické kategórie sú definované pravidlami, na základe ktorých sme schopní určiť aká hodnota patrí do akej syntaktickej kategórie. Terminálnymi symbolmi sa rozumejú konkrétne znaky alebo reťazce znakov, z ktorých sa zostavujú syntakticky správne programy. Pri navrhovaní syntaxe počítačového jazyka môže dizajnér začať tvorbou platných a neplatných syntaktických konštrukcii v danom jazyku, pred tým než sa pokúsi tieto pravidlá zovšeobecniť.

3.2 Highlight.js

Highlight.js zvýrazňovač syntaxe, ktorý funguje v prostredí webového prehliadača. Je napísaný v jazyku Javascript a je zverejnený pod BSD licenciou. Dokáže pracovať ako na strane klienta tak na strane servera a nie je závislý na žiadnom framework-u a podporuje automatickú detekciu programovacieho jazyka. Vstupom tohoto skriptu je zdrojový súbor programovacieho jazyka, ktoré syntax chceme zvýrazniť. Výstupom tohoto skriptu je webová stránka, ktorá obsahuje zvýraznenú syntax daného programovacieho jazyka.

3.2.1 Prehľad zvýrazňovania

Kód programovacieho jazyka sa skladá z častí z rôznymi pravidlami analýzy. Každú túto časť je dobré pre lepšiu prehľadnosť a orientáciu v zdrojovom kóde farebne odlíšiť. Vďaka

týmto pravidlám analýzy sme schopní jednotlivé časti odlíšiť. Vieme povedať, že napr. reťazec začína a končí úvodzovkami, že číslo je neprázdna postupnosť čísel, jednoriadkový komentár začína //, a pod. A istým spôsobom vieme tieto pravidlá prepísať formou regulárnych výrazov do režimu, ktorý nám poskytuje skript Highlight.js. Každý režim pozostáva podľa Ivana Sagalaeva[7] z týchto častí:

- podmienka začiatku (regulárny výraz)
- podmienka konca (regulárny výraz)
- zoznam obsiahnutých sub-režimov
- lexikálne pravidlá a kľúčové slová
- ... exotické veci ako iný počítačový jazyk v inom počítačovom jazyku

Práca parseru implementovaného v skripte Highlight.js spočíva v identifikovaní režimov a kľúčových slov v zadanom zdrojovom súbore. Používa k tomu práve regulárne výrazy, ktoré sú špecifikované v podmienke začiatku a podmienke konca. Vďaka tomuto určí v zdrojovom súbore blok kódu, ktorý odpovedá definícii režimu. Z tohto nájdeného bloku vytvorí HTML element `<span class="..." >...</span>` a do class priradí názov režimu, ktorého definícii vyhovuje, alebo názov skupiny kľúčových slov („keyword“, „literal“, „built-in“).

3.2.2 Všeobecná syntax

Definícia programovacieho jazyka je objekt javascript, ktorý popisuje režim a spôsob spracovania daného jazyka. Tento predvolený režim obsahuje ďalšie sub-režimy, ktoré zase môžu obsahovať ďalšie sub-režimy. Týmto spôsobom sa z definície programovacieho jazyka efektívne stáva strom režimov, ktorý sa následne jednoduchšie pretvára do HTML elementov. Príklad štruktúry režimu možno vidieť vo výpise 3.1.

Zvyčajne predvolený spôsob definície režimu funguje pre väčšiu programovacích jazykov a sme schopní týmto spôsobom identifikovať všetky kľúčové vlastnosti daného programovacieho jazyka. Existujú však počítačové jazyky, ktorých vlastnosti nie sme schopní identifikovať predvoleným režimom. Takýmto prípadom je jazyk XML, kde všetok obsah je obsah autora a neobsahuje nejaké kľúčové slová. Preto sa pristupuje mapovať syntax takýchto jazykov ručne, a k tomu sa používajú regulárne výrazy (popísané v sekcii 3.3), ktorými sme schopní vytvoriť vlastný režim.

```
{
  case_insensitive: true, // jazyk nerozlisuje velke a male pismena
  keywords: 'for if while',
  contains: [
    {
      className: 'string',
      begin: '"', end: '"'
    },
    hljs.COMMENT(
      '/\\*', // begin
      '\\*/', // end
      {
        contains: [
          {
            className: 'doc', begin: '@\\w+'
          }
```

```
      ]
    }
  )
]
}
```

Výpis 3.1: Príklad jednoduchej konštrukcie jazyka, prevzaté z [7]

Kľúčové slová

V prípade programovacieho jazyka, kedy vieme jednoznačne určiť kľúčové slová, môžeme tieto slová definovať v reťazci oddelené medzerou, viď výpis 3.2.

```
{
  keywords: 'var let if'
}
```

Výpis 3.2: Definícia kľúčových slov

Existujú však programovacie jazyky, ktoré majú rozdielne druhy kľúčových slov, ktoré nemusia byť nutne špecifikáciou jazyka. Máme kľúčové slová, ktoré definujú štruktúru kódu a určujú ako sa bude blok kódy spracovávať. Potom máme istou formou kľúčové slová ako true, false, null, ktoré majú isté vlastnosti kľúčových slov, ale môžeme tieto hodnoty priraďovať. Pre takúto skupinu je v Highlightjs pripravená skupina literals, viď výpis 3.3.

```
{
  keywords: {
    keyword: 'var let if',
    literal: 'false true null'
  }
}
```

Výpis 3.3: Rozdelenie kľučových slov a literálov

Názov skupiny sa vo vygenerovanom HTML dokumente stane názvom triedy bloku, ktorému definícia skupiny odpovedá, čo nám umožňuje aplikovať rôzne motívy. Pri hľadaní kľúčových slov Highlight.js rozdelí zdrojový kód jazyka na samostatné slová. V predvolenom nastavení režimu sa kľúčové slova identifikujú regulárnym výrazom výrazu \w+ (vysvetlené v sekcii 3.3), čo je všeobecný regulárny výraz pre mnoho počítačových jazykov. Lenže existujú jazyky, kde kľúčovým slovom môžu byť aj výrazy iného tvaru ako definuje regulárny výraz \w+. Preto sa používa atribút $pattern, ktorým možno modifikovať regulárny výraz použitý pre identifikovanie regulárnych výrazov. Znázornené vo výpise 3.4.

```
{
  keywords: {
    $pattern: /#[a-zA-Z]+/, // umoznuje klucovym slovam zacinat s mriezkou
    keyword: '#config #Name'
  }
}
```

Výpis 3.4: Použitie $pattern

Komentáre

Pri definovaní režimu, ktorý identifikuje komentár v programovacom jazyku, je vhodné sa definovaniu klasického režimu vyhnúť. Prostredie Highlight.js poskytuje pre definovanie vlastných komentárov pomocnú funkciu `hljs.COMMENT`. Je vhodné použiť túto funkciu, pretože sama o sebe tiež definuje niekoľko predvolených režimov a hlavne uľahčuje automatickú detekciu jazyka. Definícia komentárov pomocou tejto funkcie je znázornená vo výpise 3.5.

```
hljs.COMMENT(
  begin: /regex/, // pociatocny regularny vyraz
  end: /regex/, // koncovy regularny vyraz
  extra: [...] // dalsie objekty s extra atributmi (napriklad {relevance: 0})
)
```

Výpis 3.5: Konštrukcia komentára

Generovanie značiek

Režimy zvyčajne generujú na základe podmienok HTML bloky. Názov triedy, ktorý sa priradí danému span bloku je definovaný názvom režimu. Pokiaľ ide o predvolený režim, je tento názov zahrnutý v názve atribútu režimu, napr. keywords. Pokiaľ však chce používateľ definovať vlastný režim, je nutné špecifikovať názov triedy použitím atribúty `className` (3.2.3), viď výpis 3.6.

Názvy v atribúte className nemusia byť jedinečné, často sa stáva že sú programovacie jazyky, ktoré majú viacero definícií s rovnakým názvom. Napríklad komentár v programovacom jazyku C môže začínať // alebo /*. Z tohoto dôvodu sa používa atribút `variants` (3.2.3).

V programovacom jazyku existujú situácie kedy v jednom elemente jazyka je vnorený ďalší element jazyka. Napríklad v programovacom jazyku môže element reťazec obsahuje tzv. escape sekvenciu. Môže slúžiť buď na ďalšie odlíšenie tejto escape sekvencie, alebo môže jednoducho slúžiť ako podpora pri syntaktickej analýze. Definíciu vnoreného elementu v inom elemente riešime pomocou atribútu `contains` (3.2.3), viď výpis 3.7.

```
{
  contains: [
    {
      className: 'keyword',
      // ... ostatne atributy
    },
    {
      className: 'string',
      // ... ostatne atributy
    }
  ]
}
```

Výpis 3.6: Definícia dodatočných módov

```
{
  className: 'string',
  begin: /"/,
  end: /"/,
  contains: [{begin: /\\(\.|[xX][\daA-fF]{2}/}],
}
```

Výpis 3.7: Príklad reťazca

3.2.3 Atribúty módu

V tejto sekcii sú popísané dôležité atribúty z hľadiska tejto práce. Typy hodnôt atribútov, ktoré sa vyskytujú v tejto sekcii, sú popísané v tabuľke 3.1.

identifikátor	Reťazec vhodný na použitie ako premenná Javascript-u a názov triedy v CSS (najčastejšie vyhovujúce regulárnemu výrazu / [A-Za-z0-9 _] + /)
regulárny výraz	Reťazec predstavujúci regulárny výraz Javascript-u. Keďže nejde o doslovný regulárny výraz, všetky spätné lomítka by sa mali opakovať dvakrát
boolová hodnota	Boolová hodnota Javascript-u: true alebo false
číslo	štandardné číslo v Javascripte
objekt	objekt v Javascripte: { . . . }
pole	pole v Javascripte: [. . .]

Tabuľka 3.1: Dátové typy atribútov

begin

Dátovým typom hodnoty tohto atribútu je `regulárny výraz`. Tento regulárny výraz určuje začiatok režimu. Pokiaľ tento atribút v definícii režimu chýba, doplní si tento atribút zvýrazňovač automaticky. V tomto automaticky priradenom atribúte je regulárny výraz, ktorý nájde zhodu s čímkoľvek, takže režim začne okamžite.

end

Dátovým typom hodnoty tohto atribútu je `regulárny výraz`. Tento regulárny výraz určuje, kedy režim končí. Môže sa však stať, že regulárny výraz obsiahnutý v atribúte `begin` definuje celý mód a nepotrebuje špeciálne ukončenie. Ak tento atribút chýba v definícii režimu chýba, doplní si tento atribút zvýrazňovač automaticky. V tomto automaticky priradenom atribúte je regulárny výraz, ktorý nájde zhodu s čímkoľvek, takže režim skončí okamžite.

className

Dátovým typom hodnoty tohto atribútu je `identifikátor`. Tento identifikátor definuje názov režimu a používa sa následne ako názov triedy elementu vo výslednom HTML dokumente. Tento identifikátor nemusí byť v rámci viacerých režimov jednoznačný, kľudne môže mať viac režimov rovnaký identifikátor. Je to užitočné, keď má programovací jazyk viacero variant syntaxe pre nejaký element. Príkladom môže byť reťazec v jednoduchých a dvojitých úvodzovkách.

excludeBegin, excludeEnd

Oba tieto atribúty majú dátový typ hodnoty `boolovská hodnota`. Týmto atribútom určujeme, či bude zvýraznená lexéma vyhovujúca regulárnemu výrazu v atribúte `begin`, resp. `end`. Napríklad v programovacom jazyku C končí príkaz znakom bodkočiarka. Vizuálne je lepšie ho však nezvýrazniť. Takže ak je atribút `end: ';'`, tak potom atribút `excludeEnd: true`, zabezpečí, že bodkočiarka nebude zvýraznená.

returnBegin

Dátovým typom hodnoty tohto atribútu je `boolovská hodnota`. Vráti nájdenú lexému, ktorá vyhovuje regulárnemu výrazu v atribúte begin naspäť do syntaktického analyzátora. Používa sa v situáciách kedy atribút begin sub-režimu obsahuje v regulárnom výraze časť regulárneho výrazu, ktorý je použitý v atribúte begin v nadradenom režime.

Pretože analyzátor po spracovaní lexémy vyhovujúcej regulárnemu výrazu v nadradenom režime túto lexému zahodí. V tomto prípade ale časť lexémy je ešte vyžadovaná regulárnym výrazom v sub-režime, ktorý ju ale po zahodený nebude môcť identifikovať, preto sa používa returnBegin. Pretože sa syntaktický analyzátor skutočne vracia späť, je celkom možné vytvoriť nekonečnú slučku, takže je to potrebné používať opatrne.

returnEnd

Dátovým typom hodnoty tohto atribútu je `boolovská hodnota`. Vráti nájdenú lexému, ktorá vyhovuje regulárnemu výrazu v atribúte end naspäť do syntaktického analyzátora. Napríklad blok jazyka Javascript končí v HTML dokumente značkou </script>, ktorú ale nemožno analyzovať pomocou pravidiel jazyka Javascript. Tak je to vrátene späť do rodičovského režimu HTML, ktorý vie čo s tým má robiť. Rovnako ako pri returnBegin je nutné zaobchádzať s týmto opatrne, nakoľko to tiež môže vytvoriť nekonečnú slučku.

contains

Dátovým typom hodnoty tohto atribútu je `pole`. Obsahuje definície ďalších sub-režimov, ktoré sa môžu vyskytnúť v rámci nadradeného režimu.

variants

Dátovým typom hodnoty tohto atribútu je `pole`. Môže nastať situácia, kedy môže pre jeden režim byť definované viacero syntaktických pravidiel, v tom prípade sa použite atribút variants, ktorý nám umožňuje nadefinovať pre daný režim viacero podmienok začiatku a podmienok konca.

3.3 Regulárne výrazy

Regulárne výrazy vznikli z dôvodu potreby práce s textovými reťazcami určitým unifikovaným spôsobom. Samotný výraz je špeciálny reťazec znakov, ktoré predstavujú určitý vzor (masku) pre textové reťazce. Preto sa regulárne výrazy používajú na kontrolu, či reťazec spĺňa určité pravidlá (validácia), alebo na *parsovanie* zdrojového kódu (náš prípad). Regulárny výraz je reťazec, ktorý obsahuje meta-znaky a kvantifikátory. Zápis regulárneho výrazu zvyčajne býva neprehľadný a obsiahly, preto je vhodné tieto výrazy v zdrojovom kóde komentovať. Meta-znaky a kvantifikátory sú vysvetlené v sekciách 3.3.1 – 3.3.4.

3.3.1 Bodka

Bodka zastupuje jeden ľubovoľný znak v reťazci. Napríklad, ak použijeme regulárny výraz (.uto) dostaneme zhodu v slovách textitauto, textitputo, textit_uto. Dokonca aj slovo autoc sa považuje za platné, pretože tento výraz maximálnu dĺžku ignoruje. V danom slove platí regulárny výraz (.uto) a teda platí celé slovo.

3.3.2 Ohraničenie (ukotvenie)

Pokiaľ potrebujeme určiť aby hľadaný reťazec začínal presne na začiatku riadku, alebo za hľadaným reťazcom končil riadok, môžeme použiť ohraničenie (ukotvenie). Pre označenie začiatku riadku používame znak `^` a pre označenie konca riadku znak **$**. Takže ak chceme aby hľadaný reťazec bol na samostatnom riadku, môžeme upraviť predchádzajúci výraz **(.uto)** na `^(.uto)$`.

3.3.3 Vyčíslenie (kvantifikátory)

Vyčíslením určujeme koľko krát sa zvolený znak, alebo skupina znakov môže v danom reťazci opakovať. Jednotlivé typy kvantifikátorov používaných v regulárnych výrazoch sú popísané v tabuľke 3.2.

?	výskyt minimálne 0-krát, maximálne 1-krát
*	výskyt minimálne 0-krát, maximálne neobmedzene
+	výskyt minimálne 1-krát, maximálne neobmedzene
{N}	výskyt práve N-krát
{N,}	výskyt minimálne N-krát
{N, M}	výskyt minimálne N-krát, maximálne M-krát

Tabuľka 3.2: Tabuľka kvantifikátorov v regulárnych výrazoch

Na použitie kvantifikátorov uzavrieme znaky, alebo podreťazec do okrúhlych zátvoriek, za ktoré pridáme zvolený kvantifikátor. Takže pokiaľ upravíme predchádzajúci výraz na `(.uto){2}` dostaneme zhodu v slove *autoauto*.

3.3.4 Skupiny znakov

Skupiny znakov, ktoré sú často používané môžeme zjednotiť do nejakej skupiny. Napríklad, ak chceme akékoľvek číslo tak namiesto písania rozsahu 0-9 stačí napísať \d. Všetky často používané skupiny sú popísané v tabuľke 3.3.

\d	všetky číslice 0-9
\D	ľubovoľný znak okrem číslic 0-9
\w	znak z abecedy a podržítko
\W	ľubovoľný znak okrem znaku abecedy a podržítka
\s	tzv. biele znaky (medzera, tabulátor, zalomenie riadku, atď.)
\S	ľubovoľný znak okrem bielych znakov

Tabuľka 3.3: Preddefinvané skupiny znakov v regulárnych výrazoch

3.3.5 Hranaté zátvorky a interval

Ak v danej situácii nevyhovuje použitie preddefinovaných skupín, ako napríklad chceme povoliť len číslo 0, uzavrieme takúto skupinu znakov do hranatých zátvoriek a vznikne výraz `[0]`. Taktiež pokiaľ chceme povoliť len určitý interval znakov, napríklad hexadecimálna sústava pozostáva z rozsahu znakov abecedy *a* až *f*, tiež uzavrieme tento rozsah do hranatých zátvoriek a vznikne nám výraz `[aA-fF]`.

Veľkou a často používanou výhodou použitia hranatých zátvoriek je, že podporujú obmedzenie výskytu znakov. Pokiaľ nechceme aby sa v danom mieste vo vyhľadávanom reťazci nevyskytoval napríklad znak *a* alebo *b*, zapíšeme tento výraz ako `[^ab]`.

3.4 Lookaround

Jan Goyvaerts[5] tvrdí: „*Lookahead a lookbehind, alebo súhrne nazývané lookaround, sú tvrdenia s nulovou dĺžkou presne ako začiatok riadku a koniec riadku. Rozdiel medzi začiatkom riadku, koncom riadku a lookaround je ten, že lookaround naozaj nájde zhodu znakov, ale potom ju zahodí a vráti iba výsledok či našiel zhodu alebo nenašiel zhodu.*“

Dôvodom prečo sa práve nazývajú *tvrdenia* je, že nespracujú samotné znaky v reťazci, ale iba vyhodnotia či je v skúmanom reťazci zhoda možná alebo nie. Regulárne výrazy, ktoré by boli vytvorené bez použitia techniky lookaround by mohli byť veľmi náročné a neprehľadné. Dokonca môže nastať situácia, kedy by nebolo možné vytvoriť regulárny výraz bez tejto techniky.

3.4.1 Negatívny a pozitívny Lookahead

Negatívny lookahead je potrebný ak potrebujeme nájsť zhodu s reťazcom, ktorý neobsahuje nejaký iný reťazec. Pri klasických regulárnych výrazoch nemožno použiť negáciu výskytu na zložitejšie regulárne výrazy, pretože nemožno použitím klasických tried regulárnych výrazov zostrojiť regulárny výraz, ktorý by našiel zhodu s reťazcom *a* nenasledovaný reťazcom *b*. Riešenie tejto situácie je práve použitie negatívneho lookahead a vznikne nám regulárny výraz `a(?!b)`.

Pozitívny lookahead je zase potrebný, keď chceme nájsť zhodu s reťazcom vyhovujúcim výrazu *a*, ktorý je bezprostredne nasledovaný reťazcom, ktorý vyhovuje regulárnemu výrazu *b*. Výsledný regulárny výraz použitím pozitívneho lookahead má tvar `a(?=b)`.

Pri použití pozitívneho lookahead je potrebné si dať pozor na to, že reťazec, ktorý vyhovuje regulárnemu výrazu b nie je súčasťou nájdenej zhody.

Vo vnútri lookahead je možné použiť ľubovoľný platný regulárny výraz. Samotný lookahead nie je zachytávajúcou skupinou a tiež nie je možné naň spätne odkazovať. Aby bolo možné uložiť zhodu v reťazci s regulárnym výrazom v lookahead, je nutné regulárny výraz vo vnútri lookahead vložiť do samostatnej zachytávajúcej skupiny, napríklad `(?=(regexp))`. Potom bude táto skupina zachytená a môže byť spätne odkazovateľná. Opačným spôsobom to fungovať nemôže, pretože lookahead zahodí zhodu v reťazci ešte pred tým, než zachytávajúca skupina uloží zhodu.

3.4.2 Negatívny a pozitívny Lookbehind

Lookbehind má podobné správanie ako lookahead, len funguje opačným smerom. Pri spracovaní reťazca regulárnym výrazom sa vracia späť v reťazci o nejaké znaky aby skontroloval, či je možné nájsť v reťazci zhodu s regulárnym výrazom, ktorý je obsiahnutý v tele lookbehind.

Negatívny lookbehind má tvar `(?<!a)b` a konkrétne tento výraz nájde zhodu s reťazcom vyhovujúcim výrazu b, pred ktorým sa nenachádza reťazec vyhovujúci výrazu a.

Naopak pozitívny lookbehind, ktorý má tvar `(?<=a)b` a tento regulárny výraz nájde zhodu s reťazcom vyhovujúcim výrazu b, pred ktorým sa nachádza reťazec vyhovujúci výrazu a.

Kapitola 4

Návrh

Zvýrazňovanie syntaxe bude implementované vložením textového súboru s Cisco príkazmi do prostredia webového prehliadača v HTML stránke. Pôjde teda o statické prezeranie zadávaných príkazov a spätných hlásení Cisco zariadenia. Tento štýl predpokladá uloženie zadaných príkazov a spätných hlásení vo forme textového súboru.

Výsledkom bude gramatika pre Cisco príkazový riadok zaradené do Higlihght.js, ktorý bude schopní vďaka tejto gramatike zvýrazňovať syntax vkladaných príkazov. Parametrom programu je textový blok príkazov, ktorý bude zvýraznený. Používateľovi stačí akýkoľvek webový prehliadač s podporou JavaScriptu. Pri zbere príkazov budú použité fyzické zariadenia, prípadne simulačné programy. Implementácia bude v programovacom jazyku Javascript. V prípade úspešnej implementácie bude používateľ vidieť webovú stránku s farebne odlíšenou štruktúrou príkazov príkazového riadku Cisco.

4.1 Dáta použité pre návrh gramatiky

Sady príkazov, s ktorými pracujeme vychádzajú z kurzov CCNA, ktoré sú dostupné pre študentov našej školy. Tieto sady použité pri tvorbe tejto práce sme získali konfigurovaním a sledovovaním stavu Cisco zariadení v prostredí školských laboratórií. Takto sme si otestovali rôzne fyzické zariadenia s rôznymi verziami operačného systému Cisco IOS.

Pre možnosť práce v prostredí mimo laboratória sme si pripravili virtuálne prostredia, kde sme dokázali simulovať rôzne sieťové zariadenia Cisco a prostredníctvom virtuálneho rozhrania konfigurovali a monitorovali nasimulované zariadenia. Tým že sme pracovali na rôznych fyzických a simulovaných zariadeniach, sme boli schopní získať konfiguračné výpisy s rozličným prístupom formátovania.

Sme si vedomí, že prax poskytuje oveľa väčšiu variabilitu problémov, na ktoré môžu existovať neučebnicové postupy, preto sme požiadali o konfiguračné výpisy a príkazy externé zdroje. Analýzou získaných textových súborov sme rozdelili príkazy na príkazy z akademického prostredia, na príkazy z virtuálneho prostredia a na príkazy z prostredia praxe a navzájom ich medzi sebou porovnali aby sme tak zabezpečili komplexnosť pokrytia daných príkazov.

Spracovaním dát navrhneme lexémy, ktoré budú predstavovať jednotlivé časti príkazov pre Cisco zariadenia. Zo štruktúry jednotlivých lexém definujeme požiadavky pre vytvorenie regulárnych výrazov, t.j. akým znakom začína lexéma, čo je pred ňou, čo je za ňou, a čo samotná lexéma bude obsahovať. Internetová komunita umožňuje nezačínať od nuly a získať na niektoré lexémy predpripravené regulárne výrazy. Vzhľadom na variabilitu a komplexnosť

lexém, bude ale potrebné vygenerovať vlastné regulárne výrazy, aby sme pokryli všetky možnosti, ktoré lexému budú vyžadovať.

Highlight.js poskytuje rozhranie pre zvýrazňovanie syntaxe formou farebných tried CSS, kde názvy tried reprezentujú lexémy typického programovacieho jazyka. V typickom programovacom jazyku môžeme lexému definovať ako: reťazec, komentár, kľúčové slovo, číslo, atď. V našom prípade však príkazový riadok Cisco nie je typickým programovacím jazykom a bude potrebné vytvoriť návrh ako mapovať lexémy príkazového riadku na lexémy, ktoré nám poskytuje rozhranie Highlight.js. Napríklad štandardná štruktúra IP adresy sa môže vizuálne reprezentovať ako celé číslo.

Lexéma kľúčové slovo, sú všetky výrazy, ktoré sú pevne definované a používateľ ich nemôže meniť. Pod lexémou číslo rozumieme v prípade príkazového riadku Cisco všetky výrazy, ktoré definovaným spôsobom reprezentujú číslo, ide o lexémy ako IP adresa, číslo, mac-adresa, atď. Pod lexémou reťazca zahrnieme všetky textové argumenty, ktoré sa môžu vyskytnúť v syntaxi daného príkazu, napr. používateľské meno, heslo, atď. Pod lexému komentár sme zahrnuli zvyšok textu, ktorý nepovažujeme za podstatný. Napríklad, môže ísť o konfiguračný režim.

Navrhli sme zoznam lexém príkazov Cisco IOS. Pre každú lexému bude potrebné navrhnúť regulárny výraz, ktorý dokáže túto lexému jednoznačne v texte identifikovať. Rozdelili sme si príkazy na skupiny príkazov s čo najväčším počtom spoločných lexémov. Napríklad oba príkazy hostname a description majú jeden textový argument. Zobrali sme jednu skupinu príkazov a tejto skupine sme priradili podľa návrhu triedu Highlight.js. Napríklad skupine, ktorá obsahuje príkazy hostname a description sme priradili k triede string. Následne sme vyhodnotili výsledok. Proces sme opakovali, pokým sme nedosiahli uspokojivý výsledok.

Pri zvýrazňovaní bola použitá aplikácia s možnosťou formátovania textu, do ktorej sme nakopírovali príkazy príkazového riadku Cisco a formátovaním textu sme modifikovali farebné zvýraznenie jednotlivých častí príkazu. Týmto spôsobom sme si vytvárali farebné schémy štruktúry jednotlivých príkazov, ako vidno na obrázku 4.1.

```
Router(config)#hostname R1
R1(config)#interface Fastethernet0/0
R1(config-if)#ip address 192.168.10.1 255.255.255.0
R1(config-if)#description LAN
R1(config-if)#no shutdown
R1(config-if)#exit
R1(config)#
```

Obr. 4.1: Príklad zvýraznenia

4.2 Návrh regulárnych výrazov

Pri tvorbe regulárneho výrazu je nutné si uvedomiť všeobecné potreby pre reťazec, nad ktorým chceme tento regulárny výraz postaviť. Ak by sa mal reťazec zakotviť na začiatku alebo na konci riadku je potrebné použiť ^ na ukotvenie reťazca na začiatok riadku alebo $ na ukotvenie reťazca na koniec riadku.

Pokiaľ sa v reťazci nachádzajú opakujúce sa znaky je vhodné použiť kvantifikátory *,?,+. Ak ide o určitý počet opakovaní, uzavrieme počet opakovaní do zložených zátvoriek {}.

V prípade, že reťazec obsahuje znaky z už definovanej triedy znakov, je dobré túto triedu použiť ak to je samozrejme vhodné. Môže ísť o číslo, kde namiesto 0-9 je možné napísať \d, alebo \w, ktorý reprezentuje znaky abecedy spolu s podtržítkom. Ak regulárny výraz vyžaduje použitie vlastnej modifikovanej triedy znakov, zaobalíme tieto znaky to hranatých zátvoriek [], takže ak nechceme rozsah celej abecedy, ale iba znaky hexadecimálnej sústavy, použijeme vlastnú triedu znakov [aA-fF].

Pokiaľ reťazec vyžaduje zoskupenie alebo alternatívne zhody, môžeme použiť operátor |. Tento operátor sa často používa so skupinou znakov v regulárnom výraze. Napríklad, aby výraz našiel zhodu s a alebo b ako súčasť reťazca, výraz by obsahoval konštrukciu (a|b). Taktiež tento operátor môžeme použiť s kvantifikátormi alebo na spätné odkazy.

Kapitola 5

Implementácia

Implementácia gramatiky príkazového riadku Cisco je implementovaná programovacím jazykom Javascript. Na identifikáciu jednotlivých komponent príkazov sú použité regulárne výrazy, ktoré tvoria význačnú časť práce. Princípom je nutné určiť hranice bloku. Kde blok začína je určené pomocou atribútu begin a kde končí je určené pomocou atribútu end. Na určenie týchto hraníc sa používajú regulárne výrazy.

Súčasťou implementácie je proces konštrukcie samotných regulárnych výrazov na dostupných reťazcoch a následné zovšeobecnenie pre viacero variant. Keďže každý programovací jazyk pristupuje k regulárnym výrazom inak, na to aby sme postavili správne regulárne výrazy pre jazyk Javascript, použili sme užitočné funkcie pre prácu s regulárnymi výrazmi. Najprv sme pomocou let re = new RegExp(`regex`) vytvorili regulárny výraz v prostredí Javascript. S regulárnym výrazom v re, už sú funkcie v Javascripte schopné pracovať. Potom sme použili užitočné funkcie pre prácu s regulárnymi výrazmi nad reťazcami.

- exec() - Vykoná vyhľadávanie zhody v reťazci. Vracia pole nájdených zhôd alebo NULL pri nezhode.
- test() - Testy na zhodu v reťazci. Vracia true alebo false.
- match() - Vracia pole, ktoré obsahuje všetky zhody v reťazcu, vrátane zachytávacích skupín, alebo NULL, ak sa žiadna zhoda nenašla.

Samotné regulárne výrazy predstavujú určitým spôsobom gramatiku, ktorú je potrebné zvládnuť a následne použiť pri popise jednotlivých častí príkazov Cisco príkazového riadku. Navrhnuté regulárne výrazy sme distribuovali v rámci zdrojového kódu Javascriptu na implementáciu gramatiky príkazového riadku Cisco.

V distribúcii Highlight.js je nutné zavolať funkciu pre registráciu nového jazyka **hljs.registerLanguage();**, aby sme vytvorenú gramatiku mohli používať. Funkcia ako prvý parameter požaduje textový reťazec názvu jazyka, pod ktorým sa daný nový jazyk zaregistruje a druhým parametrom je funkcia, ktorá vráti objekt reprezentujúci samotnú definíciu jazyka. V našom prípade je táto funkcia volaná na začiatku zdrojového súboru v podobe `hljs.registerLanguage("cisco", function(hljs){ ...}`), kde ... reprezentujú definíciu jazyka. Parameter *"cisco"* slúži na identifikáciu blokov v HTML dokumente, ktoré majú byť zvýraznené. Parameter *hljs* vo funkcii function je určený pre to, aby mohli byť prípadne použité pred-pripravené regulárne výrazy, premenné dostupné z distribúcie Highlight.js.

Nasleduje zoznam premenných, ktoré obsahujú jedny z najčastejšie používaných regulárnych výrazov aby bolo možno pre používateľa si tieto regulárne výrazy modifikovať podľa

potreby. V prípade príkazového riadku Cisco je nutné pomocou príkazu `case_insensitive: true` vypnúť rozlišovanie na veľké a malé písmena, pretože príkazový riadok tieto písmena nerozlišuje. Ako už bolo vyššie spomenuté, automatická detekcia jazyka nie je pre implementačné prostredie vhodné, preto je nutné túto vlastnosť vypnúť príkazov `disableAutodetect: true`. Po ošetrení vlastností potrebných pre Highlight.js pokračuje definícia gramatiky príkazového riadku Cisco.

Na začiatok sa celý súbor príkazov urči ako jeden veľký blok, ktorý označíme ako komentár `className: "comment"`. Týmto spôsobom odlíšime automaticky všetky nezaujímavé časti a môžeme sa plne venovať zvýrazňovaniu podstatnejších blokov. Napríklad v našej implementácii považujeme za triedu číslo IP adresy, čísla, a iné útvary. Keďže IP adresa má inú štruktúru ako číslo, vyžaduje si teda aj iný odpovedajúci regulárny výraz ako regulárny výraz dostupný od distribúcie Highlight.js.

Tento veľký blok si ďalej rozdelíme na vnorené bloky, čim si ošetríme vlastnosť, že každý blok môže začínať rôznym spôsobom. Pre to aby sme si ho mohli rozdeliť, je potrebné uviesť kľúčové slovo **`contains:`**, ktorému sa prípade pole (dátový typ jazyka Javascript) všetkých regulárnych výrazov, ktorými tieto bloky môžu začínať, prípadne končiť. Základná štruktúra zdrojového kódu je uvedená nižšie v obrázku 5.1.

```
name: 'cisco',
case_insensitive: true,
disableAutodetect: true,
contains:
[
  {
    className: 'comment',
    variants: [ ···
    ],
  },
],
```

Obr. 5.1: Základná kostra zdrojového kódu

Pre tento prístup sme sa rozhodli hlavne kvôli tomu, že príkazy, ktoré vypisujú plnetextové výstupy sa zvýrazňujú ťažko podľa nejakej štruktúry a je nutné ku každému výpisu pristupovať iným spôsobom. Začali sme príkazmi, ktoré máju podobnú štruktúru. Na základe obr. 2.2 vieme určiť, že príkaz má štruktúru **`názov_zariadenia(konfiguračný režim)#samotný_príkaz`**. Podľa tejto štruktúry sme vytvorili regulárny výraz 5.1, ktorý je obsiahnutý v premennej **command** v zdrojovom súbore cisco.js.

Časť regulárneho výrazu na začiatku **`^\s*`** určuje, že na začiatku riadku môže byť ľubovoľný počet medzier. Aj keď samotný príkaz v príkazovom riadku začína hneď na začiatku riadku, je nutné túto časť regulárneho výrazu uviesť pre elimináciu možných problémov pri vkladaní súboru príkazov do dokumentu HTML, keďže vývojové prostredia môžu automaticky odsadzovať text.

Pokračujúca časť regulárneho výrazu **[a-z0-9A-Z\-]+** reprezentuje názov sieťového zariadenia, kde povolené znaky boli určené podľa dokumentu z Cisco stránky[1].

$$/(?=\hat{}\backslash s*[a\text{-}z0\text{-}9A\text{-}Z\backslash\text{-}]+(?:\backslash(([a\text{-}zA\text{-}Z\backslash\text{-}]*)\backslash)\#|\#)/ \tag{5.1}$$

Ďalšou a zároveň poslednou časťou regulárneho výrazu je **(?:\(([a-zA-Z\-]*)\)#|#)**. Táto časť určuje konfiguračný režim, kde sa muselo zaviesť podporovaný rozsah znakov prípadne pomlčky, pretože keby sa vypisoval každý možný režim Cisco, stal by sa ako regulárny výraz neprehľadný, tak by tam každou operáciu OR narastali výpočetné nároky.

Všetky tieto časti regulárneho výrazu sú zabalené do regulárneho výrazu (?=...), ktorý v rámci našej implementácie hovorí, že takto identifikovaný blok skončí pred nájdenou zhodou s regulárnym výrazom umiestneným v tele (?=...). Takže priradením vyššie uvedeného regulárneho výrazu ku kľúčovému **begin:** a priradením jednoduchého regulárneho výrazu **/$/** ku kľúčovému slovu **end:** sme identifikovali blok, ktorý identifikuje riadok jedného príkazu. Tento identifikovaný blok však môže ešte obsahovať ďalšie bloky, pretože príkaz pozostáva z viacerých prvkov ako sú kľúčové slová a argumenty príkazu, číselné hodnoty a IP adresy, vstup od používateľa, atď..

Aby sme mohli určiť vnorený blok v rámci nadradeného bloku, je nutné v definícii nadradeného bloku zaviesť kľúčové slovo **contains:**, ktorému sa priradí pole (dátový typ jazyka Javascript) definícii ďalších blokov príkazu. V tomto bloku sme určili samotné telo príkazu a to vieme určiť pomocou regulárnych výrazov /#/ a /$/. V tejto situácii je nutné ešte v nadradenom režime zaviesť kľúčové slovo **returnBegin: true**, pretože znak # už bol spracovaný syntaktickým analyzátorom v nadradenom bloku. Aby sme teda tento znak opäť mohli použiť v podradenom bloku, je nutné ho vrátiť späť to syntaktického analyzátora, čo vykonáme spomenutým kľúčovým slovom.

Typickým argumentom príkazov v sieťových zariadeniach bývajú IP adresy. Keďže ide o hodnotu, ktorá má pevnú štruktúru, vieme povedať, že najjednoduchšia štruktúra IP adresy verzie 4 je X.X.X.X, kde na X reprezentuje číslo v rozsahu 0-255. V textových častiach, ktoré reprezentujú výpis konfigurácie sieťového zariadenia sa môže vyskytnúť maska IP adresy, ktorá sa zapisuje v tvare IP adresy alebo v tvare prefixu, ktorý sa zapisuje za IP adresou v tvare /X, kde X opäť reprezentuje číslo v rozsahu 0-32. Na základe tejto štruktúry sme si postavili vlastný regulárny výraz 5.2, ktorý je z tohoto dôvodu priradený k premennej **ip4** v zdrojovom súbore cisco.js.

$$\begin{gathered}/(?:(?:(?:25[0\text{-}5]|2[0\text{-}4][0\text{-}9]|1[0\text{-}9]2|9[0\text{-}9]|[1\text{-}8][0\text{-}9]|[0\text{-}9])\backslash.)\{3\} \\ (?:25[0\text{-}5]|2[0\text{-}4][0\text{-}9]|1[0\text{-}9]2|9[0\text{-}9]|[1\text{-}8][0\text{-}9]|[0\text{-}9])(?:\backslash/(?:3[0\text{-}2]|[1\text{-}2][0\text{-}9]|[0\text{-}9]))?)\backslash b/.\end{gathered} \tag{5.2}$$

Časť regulárneho výrazu **(?:25[0-5]|2[0-4][0-9]|1[0-9]2|9[0-9]|[1-8][0-9]|[0-9])** určuje jeden až troj miestne číslo v rozsahu 0 - 255. Táto časť výrazu sa zabalí to regulárneho výrazu **(?:(?:25[0-5]|2[0-4][0-9]|1[0-9]2|9[0-9]|[1-8][0-9]|[0-9])\.){3}**, ktorý nám hovorí, že toto jeden až troj miestne číslo s bodkou bude zopakovanie 3-krát za sebou, čím nám vznikne formát **X.X.X**. Lenže za posledným číslom bodka nie je a teda uzavrieme regulárny výraz reprezentujúci IP adresu ďalším **(?:(?:25[0-5]|2[0-4][0-9]|1[0-9]2|9[0-9]|[1-8][0-9]|[0-9])\.){3}**. Keďže za IP adresou môže nasledovať prefix ale aj nemusí, priložíme na koniec celého regulárneho výrazu **(?:\/(?:3[0-2]|[1-2][0-9]|-9**

[1] https://www.cisco.com/c/en/us/td/docs/security/ise/2-4/install_guide/b_ise_InstallationGuide24/b_ise_InstallationGuide24_chapter_010.html#id_11096

))?), kde práve otáznik určuje to, že sa prefix môže vyskytovať alebo nemusí. Podobným princípom je vytvorený regulárny výraz pre IP adresu verzie 6

$$/(?![\hat{}\backslash w:])(([0\text{-}9a\text{-}f]\{1,4\}:)\{1,7\}:|:(:[0\text{-}9a\text{-}f]\{1,4\})\{1,7\}|([0\text{-}9a\text{-}f]\{1,4\}:)\{1,7\} \\ [0\text{-}9a\text{-}f]\{0,4\}(:[0\text{-}9a\text{-}f]\{1,4\})\{1,7\})(?![\backslash w:])/, \tag{5.3}$$

ktorý možno nájsť priradený v zdrojovom kóde v premennej ip6. Regulárny výraz bol prevzatý zo stránky regextester[2]. Posledným globálne využívaným regulárnym výrazom je regulárny výraz pre fyzickú adresu rozhrania, tzv. MAC-adresa. V príkazovom riadku Cisco má MAC-adresa štruktúru X.X.X, kde X reprezentuje 4 miestny text v hexadecimálnej sústave(znaky 0-9 a a-f). Podľa tejto štruktúry bolo postavený regulárny výraz

$$/(?:[0aA\text{-}9fF]\{4\}\backslash.)\{2\}(?:[0aA\text{-}9fF]\{4\})/, \tag{5.4}$$

ktorý je priradený v zdrojovom kóde k premennej **mac**. Časť regulárneho výrazu (?:[0aA-9fF]{4}\.){2} určuje, že sa 2-krát zopakuje štvormiestny text vo formáte hexadecimálnej sústavy s bodkou. Posledná časť (?:[0aA-9fF]{4}) nám opäť určuje štvormiestny text vo formáte hexadecimálnej sústavy tentokrát, ale bez bodky. Aj keď sa dá povedať, že všeobecne číslo má rovnakú štruktúru, nedá sa v rámci gramatiky Cisco postaviť nejaký všeobecný regulárny výraz, pretože záleží na kontexte použitých čísel.

Existujú príkazy, po ktorých zadaní do príkazového riadku dostane používateľ plnetextový výstup zvyčajne reprezentujúci aktuálnu konfiguráciu daného zariadenia. Pretože každý tento výstup má iný formát a štruktúr, nemožno takéto výstupy zvýrazňovať globálne, ale je nutné ku každému výpisu pristupovať samostatne a originálne. Všeobecne sa každý blok výpisu určí počiatočným regulárnym výrazom priradeným kľúčovému slovu **begin:** (?<=show\s_príkaz_), napríklad (?<=show\sip\sprotocols).

Ďalej na týchto výstupoch nie je možné určiť koniec takéhoto výstupu, preto sme sa koniec takéhoto bloku rozhodli riešiť tým, že na konci takéhoto výstupu je výžadovaný ďalší príkaz kľudne aj prázdny. Takže v kľúčovom slove **end:** je priradená premenná **command**, ktorá obsahuje regulárny výraz spomenutý a rozobratý vyššie.

5.1 Demonštrácia zvýrazňovania

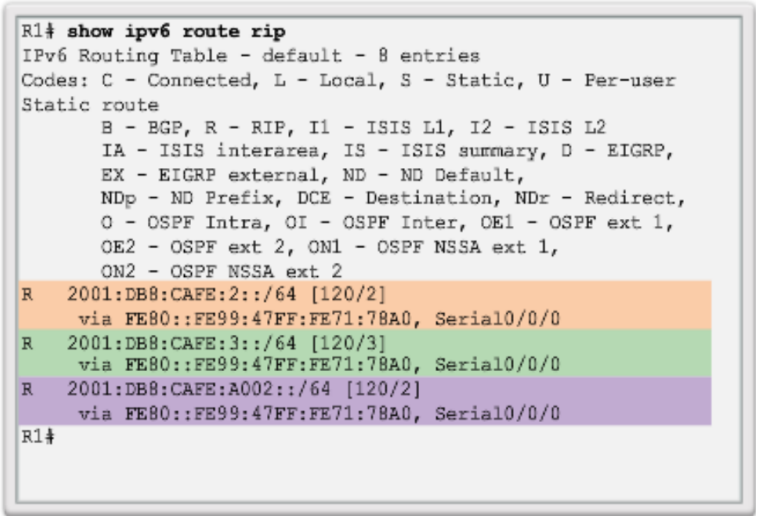
Na demonštráciu výsledkov našej práce sme použili príklady príkazov a konfiguračných výpisov z prednášok z predmetu I1C. V nasledujúcej časti bude vždy dvojica obrázkov, kde prvý obrázok reprezentuje pôvodný formát z prezentácie, a druhý obrázok reprezentuje zvýraznenie syntaxe našou prácou.

[2] https://www.regextester.com/104187

```
R1(config)#router rip
R1(config-router)#passive-interface Fa0/1
```


```
R1(config)#router rip
R1(config-router)#passive-interface Fa0/1
```


Obr. 5.2: Porovnanie zvýraznenia nastavenia pasívneho rozhrania



```
R1# show ipv6 route rip
IPv6 Routing Table - default - 8 entries
Codes: C - Connected, L - Local, S - Static, U - Per-user
Static route
       B - BGP, R - RIP, I1 - ISIS L1, I2 - ISIS L2
       IA - ISIS interarea, IS - ISIS summary, D - EIGRP,
       EX - EIGRP external, ND - ND Default,
       NDp - ND Prefix, DCE - Destination, NDr - Redirect,
       O - OSPF Intra, OI - OSPF Inter, OE1 - OSPF ext 1,
       OE2 - OSPF ext 2, ON1 - OSPF NSSA ext 1,
       ON2 - OSPF NSSA ext 2
R   2001:DB8:CAFE:2::/64 [120/2]
     via FE80::FE99:47FF:FE71:78A0, Serial0/0/0
R   2001:DB8:CAFE:3::/64 [120/3]
     via FE80::FE99:47FF:FE71:78A0, Serial0/0/0
R   2001:DB8:CAFE:A002::/64 [120/2]
     via FE80::FE99:47FF:FE71:78A0, Serial0/0/0
R1#
```


```
R1# show ipv6 route rip
IPv6 Routing Table - default - 8 entries
Codes:  C - Connected, L - Local, S - Static, U - Per-user
Static route
        B - BGP, R - RIP, I1- ISIS L1, I2 ISIS L2,
        IA - ISIS interarea, IS - ISIS summary, D - EIGRP,
        EX - EIGRP external, ND - ND Default,
        NDp - ND Prefix, DCE - Destination, NDr - Redirect.
        O - OSPF Intra, OI - OSPF Inter, OE1 - OSPF ext 1,
        OE2 - OSPF ext 2,  ON1 - OSPF NSSA ext 1,
        ON2 - OSPF NSSA ext 2
R      2001:DB8:CAFE:2::/64 [120/2]
        via FE80::FE99:47FF:FE71:78A0, Serial0/0/0
R      2001:DB8:CAFE:3::/64 [120/2]
        via FE80::FE99:47FF:FE71:78A0, Serial0/0/0
R      2001:DB8:CAFE:A002::/64 [120/2]
        via FE80::FE99:47FF:FE71:78A0, Serial0/0/0
R1#
```


Obr. 5.3: Porovnanie zvýraznenia IPv6 smerovacej tabuľky

5.2 Zaradenie do distribúcie Highlight.js

Na základe diskusie s vývojárom distribúcie Highlight.js, už nie je možné začleniť novú gramatiku do oficiálnej distribúcie, pretože nemajú dostatok ľudí, ktorí by sa starali udržiavanie týchto gramatík.

Riešením je vytvoriť si na stránke github vlastný repozitár. Do tohto repozitár bude obsahovať všetky zdrojové kódy a testovacie príklady v požadovanej adresárovej štruktúre danej komunitou Highlight.js.

```
SwitchX# configure terminal
SwitchX(config)# vlan 2
SwitchX(config-vlan)# name switchlab99
```

```
SwitchX# configure terminal
SwitchX(config)# vlan 2
SwitchX(config-vlan)# name switchlab99
```

Obr. 5.4: Porovnanie zvýraznenia vytvorenia a pomenovania vlan

```
R2#show ip route rip
R     192.168.10.0/24 [120/1] via 192.168.12.1, 00:00:07, FastEthernet0/0
```

```
R2#show ip route rip
R 192.168.10.0/24 [120/1] via 192.168.12.1, 00:00:07, FastEthernet0/0
```

Obr. 5.5: Porovnanie zvýraznenia IPv4 smerovacej tabuľky

```
SwitchX# configure terminal
SwitchX(config)# interface range fastethernet 0/2 - 4
SwitchX(config-if)# switchport access vlan 2

SwitchX# show vlan

VLAN Name                             Status    Ports
---- -------------------------------- --------- -------------------------
1    default                           active    Fa0/1
2    switchlab99                       active    Fa0/2, Fa0/3, Fa0/4
```

```
SwitchX# configure terminal
SwitchX(config)# interface range fastethernet 0/2 - 4
SwitchX(config-if)# switchport access vlan 2
SwitchX# show vlan
VLAN Name                             Status    Ports
---- -------------------------------- --------- -------------------------
1    default                           active    Fa0/1
2    switchlab99                       active    Fa0/2, Fa0/3, Fa0/4
```

Obr. 5.6: Porovnanie zvýraznenia vlan tabuľky

Následne sme po vyriešení tohto problému požiadali o začlenenie gramatiky do zoznamu podporovaných jazykov v oficiálnej distribúcii Highlight.js, v ktorom sme špecifikovali názov gramatiky spolu s názvami tried, pod ktorými bude gramatika vo zvýrazňovači dostupná a s odkazom na repozitár, kde sa daná implementácia nachádza. Podmienkami pre začlenenie vytvorenej gramatiky Cisco do oficiálnej distribúcie je nutné otestovať gramatiku v distribúcii Highlight.js pomocou testovacieho skriptu. V tomto prípade je nutné vytvoriť súbor Cisco.js, ktorý namiesto funkcie registerLanguage() používa export default function(hljs), pretože to tak testovací skript požaduje. Je nutné vytvoriť adresár s testami. Týmito testami testovací skript otestuje automatickú detekciu jazyka a schopnosť zvýraznenia syntaxe. Ide o štandardný textový súbor, v ktorom sa nachádzajú príkazy a konfiguračné výpisy, na ktorých prebehne testovanie skriptom. Na záver je ešte nutné priložiť do koreňového adresára súbory:

- Package.json – určený pre Node.js, obsahuje popis vytvoreného skriptu,
- README – obsahuje informácie pre používateľa, ktorý ten skript chce použiť,
- LICENSE – obsahuje licenciu vytvoreného skriptu

5.3 Popis použitia

S touto implementáciu môžu prísť do kontaktu traja aktéri:

- programátor – tvorba a modifikácia implementácie gramatiky,
- tvorca webovej stránky – vytvára stránku, v ktorej chce použiť vytvorenú gramatiku,
- návštevník – prezerá si webovú stránku obsahujúcu vytvorenú gramatiku.

Všeobecne platí, že ktokoľvek chce pracovať s vytvorenou gramatikou, si potrebuje stiahnuť oficiálnu distribúciu Highlight.js. Môže to urobiť dvoma spôsobmi. Prvým spôsobom je stiahnuť si skript z oficiálnej stránky a uložiť do nejakého priečinka, a vložiť cestu k tomuto priečinku a skriptu do HTML dokumentu. Nevýhodou tohto spôsobu je, že vyžaduje prenos skriptu spolu webovou stránkou, a je nutné si dávať pozor na cestu k súborom. Výhodou tohto spôsobu je, že nevyžaduje internetové pripojenie.

Druhým spôsobom je vloženie skriptu do HTML dokumentu z tzv. CDN úložiska. Výhodou je, že používateľ sa nemusí starať o to, kam a ako uložil stiahnutý Highlight.js skript, a upravovať cesty v HTML dokumente. Nevýhodou je, že tento spôsob vyžaduje pripojenie k internetu.

Aby si používateľ mohol prezerať takto vytvorenú webovú stránku, tak musí byť špecifikované tvorcom webovej stránky akým spôsobom webová stránka požaduje aby bol k dispozícii skript Highlight.js.

Pre programátora a tvorcu webovej stránky je spoločným prvým bodom vytvorenie HTML dokumentu so základnými elementami potrebnými pre webovú stránku. Do tohto dokumentu vložia skript Highlight.js pomocou HTML značiek <script> . V ďalšom bloku <script> vložia vytvorenú gramatiku Cisco.js a zavolajú funkciu hljs.initHighlightningOnLoad();. Touto funkciou sa inicializuje zvýrazňovač syntaxe hneď počas načítavania stránky. Funkcia registerLanguage je zahrnutá v rámci skriptu Cisco.js a nie je ju teda nutné vkladať do HTML dokumentu.

Ďalej je nutné špecifikovať blok HTML elementov <pre> <code class="cisco">, do ktorých sa vkladajú príkazy príkazového riadku Cisco. Hodnota parametru class v elemente code odpovedá názvu jazyka, pod ktorého gramatikou chceme zvýrazniť syntax.

V prípade, že chce programátor upravovať gramatiku príkazového riadku Cisco, nájde ju v skripte Cisco.js.

Kapitola 6

Testovanie

6.1 Použité skupiny dát

Pre otestovanie tejto práce boli použité testovacie sady dát z rôznych oblasti aby sa čo najlepšie otestovala komplexnosť zvýrazňovania. Konfiguračný výpis každého príkazu sa môže zobraziť s inou dĺžkou, iným obsahom, a podobne. Preto je nutné testovacie dáta rozdeliť do niekoľkých skupín.

Prvou skupinou sú súbory príkazov a výpisov konfigurácie zariadení z akademickej oblasti. Zdrojmi tejto skupiny príkazov sú súbory, ktoré boli získané buď účasťou a prácou na laboratórnych cvičeniach v predmetoch I1C a I2C, alebo získaním príkazov použitých v školských prezentáciách a učebniciach. Charakterom tejto skupiny dát je, že nejde často o veľmi rozsiahlu konfiguráciu a tým hlavne konfiguračné výpisy neobsahujú toľko informácii.

Druhou skupinou boli príkazy a výpisy zo zariadení nasadených vo firmách v praxi. Takéto zariadenia už sú používané v praxi a často obsahujú rozsiahlejšie konfigurácie, ktoré slúžia na nájdenie nedostatkov pri zvýrazňovaní podľa príkazov z akademickej oblasti. Pri získavaní takýchto dát boli požiadané niektoré firmy o sprístupnenie príkazov a výpisov konfigurácii na testovacie účely.

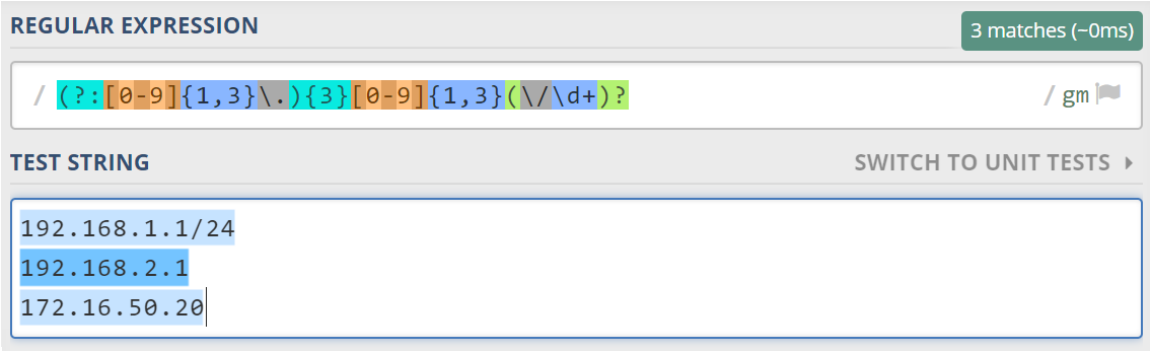
Treťou skupinou dát boli súbory príkazov získané z virtuálnych prostredí, ktorými sú simulačné prostredia alebo virtuálne laboratória. Pre získanie týchto dát boli použité virtuálne prostredia PacketTracer (oficiálna distribúcia od firmy Cisco), GNS3 a Ciscolab(dostupný iba v rámci akademickej pôdy). Charakteristikou týchto dát je, že tým že ide o simulačné prostredie, môžu sa niektoré výpisy líšiť buď kvôli komprimovaným výpisom, alebo tým, že nie všetky technológie je možné v takomto prostredí otestovať.

6.2 Testovanie regulárnych výrazov

Pri používaní a testovaní regulárnych výrazov by sme si mali byť vedomí niektorých všeobecných ustanovení [8]. V prvom rade je nutné si uvedomiť kontext použitých regulárneho výrazu. Ak sa regulárny výraz používa na určenie súvislého bloku textu podľa statického textu (?<=show\sip\sroute), takýto regulárny výraz také podrobné testovanie nevyžaduje. Naopak ak sa bude regulárny výraz používať na identifikovanie a validáciu údajov dynamického rozsahu, ako napríklad názov sieťového zariadenia, ktoré je obmedzené na určite znaky, treba venovať testovaniu takýchto regulárnych výrazov viac času a úsilia. V ideálnom

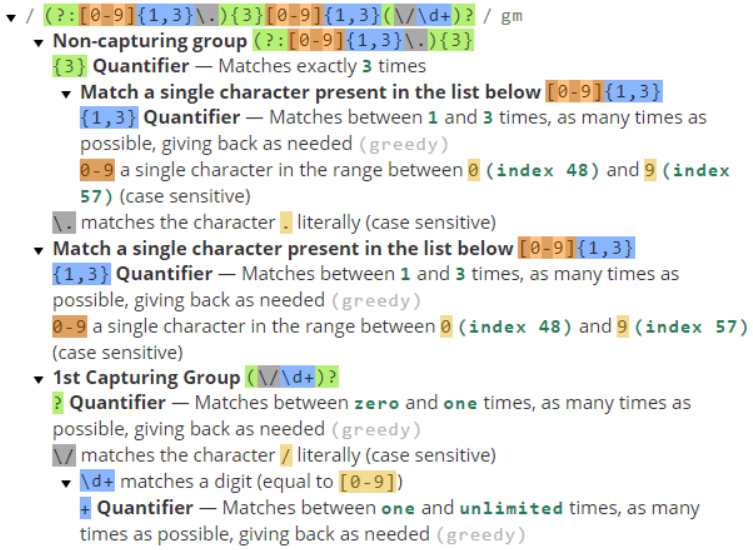
prípade by sa mala použiť testovacia súprava, aby sa zabezpečilo, že sa nevhodné reťazce nezvýraznia.

Pri testovaní regulárnych výrazov som použil webovú stránku regex101[1]. Výhodou tejto stránky je, že poskytuje všetko potrebné pre testovanie regulárnych výrazov. Umožňuje vytváranie a modifikáciu regulárnych výrazov. Navyše akákoľvek modifikácia, resp. zmena v regulárnom výraze sa v reálnom čase objaví na zmene zhody v testovacej súprave a poskytne nám časový údaj (zelený obdĺžnik v pravom hornom rohu na obr. 6.1), koľko času zabralo daným regulárnym výrazom nájsť všetky zhody v testovacej súprave, ako môžeme vidieť na obrázku 6.1.



Obr. 6.1: Použité prostredie pre testovanie regulárnych výrazov

Ďalšou silnou vlastnosťou tejto webovej stránky je, že pri testovaní regulárnych výrazov v tomto prostredí poskytuje programátorovi podrobný popis jednotlivých častí regulárneho výrazu, znázornené na obr. 6.2. Tento popis je viac než dostačujúci pre nájdenie chyby počas testovania, kedy regulárny výraz, buď nájde zhodu s niečím s čím nemá, alebo naopak nenájde zhodu s čím by tú zhodu nájsť mal.



Obr. 6.2: Podrobný popis regulárneho výrazu, vytvorený stránkou regex101[1].

[1] https://regex101.com

6.3 Testovanie webových prehliadačov

Výsledkom aplikovania nástroja Highlight.js a vytvorenej gramatiky pre príkazový riadok Cisco vznikne webová stránka. V tomto dôsledku prebehlo testovanie na najčastejšie používaných webových prehliadačoch, ktoré sú dostupné. Išlo o webové prehliadače s najnovšou verziou Google Chrome, Mozila Firefox, Opera.

6.3.1 Testovanie schopnosti zvýraznenia

Pri webových prehliadačoch Google Chrome a Opera prebehlo testovanie zvýrazňovania úspešne bez chyby(uvedené na obrázku 6.3).

```
Router(config)#hostname R1
R1(config)#interface Fastethernet0/0
R1(config-if)#ip address 192.168.10.1 255.255.255.0
R1(config-if)#description LAN
R1(config-if)#no shutdown
R1(config-if)#exit
```

Obr. 6.3: Úspešné zvýraznenie syntaxe

Pri webovom prehliadači Mozilla Firefox môžu vzniknúť dve možnosti problémov, s ktorými sa používateľ môže stretnúť. Prvým problémom je, že výsledný súbor príkazov nemusí byť vôbec zvýraznený. V tomto prípade sa text zobrazí čierno-bielo(uvedené v obrázku 6.5) a používateľ je informovaný o chybe v konzole webového prehliadača. Druhým problémom, ktorý na prvý pohľad nemusí byť jasný je, že nástroj nebude zvýrazňovať podľa špecifikácie v zdrojovom kóde. Oba tieto problémy vznikli z dôvodu nedostatku podpory pokročilých metód regulárnych výrazov ako sú metódy lookaround. Pri webovom prehliadači Chromium vznikne druhý spomenutý problém, čiže môže nastať situácia, kedy nebude zvýrazňovať podľa špecifikácie. Táto situácia nastať môže a nemusí, závisí to od použitých príkazov v súbore, ktorý má nástroj zvýrazniť, viď obrázok 6.4.

```
Router(config)#hostname R1
R1(config)#interface Fastethernet0/0
R1(config-if)#ip address 192.168.10.1 255.255.255.0
R1(config-if)#description LAN
R1(config-if)#no shutdown
R1(config-if)#exit
```

Obr. 6.4: Zvýraznenie bez rešpektovania implementácie

Ojedinele sa môže vyskytnúť chyba, ktorá nie je závislá na použitom webovom prehliadači. Jej pôvod sa mi nepodarilo určiť, pretože sa nevyskytuje tak často. A teda sa ako tvorca domnievam, že ide o chybu na strane skriptu Highlight.js. Chyba sa prejavuje tak, že skript je síce schopní určiť blok príkazov, dokáže priradiť farebné pozadie, ale nie je však schopní zvýrazniť samotný blok príkazov. Znázornené na obrázku 6.6.

6.3.2 Hardvérová a časová náročnosť

Testovanie sme vykonali podľa postupu uvedeného v literatúre [6]. Priebeh testovania sme si rozdelili do niekoľkých krokov.

```
Router(config)#hostname R1
R1(config)#interface Fastethernet0/0
R1(config-if)#ip address 192.168.10.1 255.255.255.0
R1(config-if)#description LAN
R1(config-if)#no shutdown
R1(config-if)#exit
```

Obr. 6.5: Neúspešné zvýraznenie syntaxe

```
Router(config)#hostname R1
R1(config)#interface Fastethernet0/0
R1(config-if)#ip address 192.168.10.1 255.255.255.0
R1(config-if)#description LAN
R1(config-if)#no shutdown
R1(config-if)#exit
```

Obr. 6.6: Čiastočné zvýraznenie

V prvom kroku sme si postavili testovacie prostredie. Testovanie sme vykonávali na troch rôznych zariadeniach s rôznymi hardvérovými parametrami, aby sme si otestovali ako sa správa webová stránka na rôznych hardvérových konfiguráciách. Použité zariadenia sú popísané v tabuľke 6.1. Na každom zariadení sme testovali webovú stránku na troch najpopulárnejších (podľa stránky asdata[2]) webových prehliadačoch, viď tabuľku 6.2.

Zariadenie	**Procesor**	**Pamäť**
Lenovo	Intel Core i5, 2-jadrá, 2.2GHz	8GB DDR3
MSI	Intel Core i7, 4-jadrá, 2.8GHz	16GB DDR4
Acer	Intel Atom N2600, 2-jadrá, 1.6GHz	2GB DDR2

Tabuľka 6.1: Tabuľka použitých zariadení

V druhom kroku sme si museli zabezpečiť testovacie skupiny príkazov, na ktorých sa testovala webová stránka. Rozhodli sme sa vykonať testovanie na súboroch príkazov o dvoch rôznych dĺžkach. Prvou skupinou je kratší súbor príkazov (veľkosť na disku: 208 bajtov), ktorý obsahuje príkazy, ktoré môžete vidieť na vyššie uvedených obrázkoch 6.3, 6.4, 6.5. Druhou skupinou príkazov je dlhší súbor (veľkosť na disku: 24 576 bajtov), ktorý obsahuje približne 700 riadkov príkazov a konfiguračných výpisov určených pre zvýraznenie. Aby sme vedeli pri testovaní porovnať správanie stránky pri zvýrazňovaní syntaxe kratšieho súboru príkazov a dlhšieho súboru príkazov.

V treťom kroku sme vykonali samotné testovanie webovej stránky na jednotlivých zariadeniach a webových prehliadačoch. Každý webový prehliadač poskytuje prostredie pre vývojára, ktorého súčasťou sú aj nástroje pre testovanie webových stránok, ktoré nám poskytnú spätnú väzbu o aktuálne načítanej webovej stránke a o priebehu jej spracovania. Výsledne údaje sme získali z tohoto testovacieho prostredia použitím sekcií sieť, výkon a pamäť. Sekcia sieť nám poskytla údaje o vyťaženosti siete počas načítavania a spracovania webovej stránky a celkovej veľkosti stránky. Sekcia výkon nám poskytla súhrnný prehľad o využití procesoru a pamäte RAM spolu s časovým grafom jednotlivých udalostí, z ktorých sme boli schopní určiť konkrétny čas spracovania stránky. V každom webovom prehliadači sa test vykonal 4-krát a získané hodnoty sme spriemerovali.

[2] https://www.asdata.sk/prehliadace-2019/

Zariadenie	Webový prehliadač	Verzia prehliadača
Lenovo	Mozilla Firefox	75.0 (64-bit)
	Chrome	81.0.4044.122 (64-bit)
	Opera	68.0.3618.125 (64-bit)
MSI	Mozilla Firefox	76.0.1 (64-bit)
	Chrome	83.0.4103.61 (64-bit)
	Opera	68.0.3618.125 (64-bit)
Acer	Mozilla Firefox	76.0.1 (32-bit)
	Chrome	83.0.4103.61 (32-bit)
	Opera	68.0.3618.125 (32-bit)

Tabuľka 6.2: Tabuľka testovaných webových prehliadačov

Výsledky testovania

V tabuľke 6.3 sú uvedené výsledky testovania krátkeho súboru príkazov. Pre každý webový prehliadač platilo, že urobil v priemere 4 požiadavky a preniesol priemerne cez sieť blok o veľkosti 96.3 kB. Čo sa týka času, najlepšie si viedol webový prehliadač Opera, pri práci s pamäťou si najlepšie viedol prehliadač Chrome. V tabuľke 6.4 sú uvedené výsledky testovania dlhého súboru príkazov. Pre každý webový prehliadač platilo, že urobil v priemere 4 požiadavky a preniesol priemerne cez sieť blok o veľkosti 120 kB. Časovo aj pamäťovo si najlepšie viedol prehliadač Chrome. Na záver možno zhodnotiť, že z pohľadu zariadenia MSI je najlepšia voľba prehliadač Chrome.

Webový prehliadač	Čas [ms]	Pamäť [kB]
Mozilla Firefox	175	5 057
Chrome	189.5	1 496
Opera	167.5	1 764

Tabuľka 6.3: Štatistiky zariadenia MSI (krátky súbor)

Webový prehliadač	Čas [ms]	Pamäť [kB]
Mozilla Firefox	508.5	10 145
Chrome	250.75	1 695
Opera	329	1 757

Tabuľka 6.4: Štatistiky zariadenia MSI (Dlhý súbor)

V tabuľke 6.5 sú uvedené výsledky testovania krátkeho súboru príkazov. Rovnako ako pri zariadení MSI najlepšie z hľadiska času dopadol prehliadač Opera, z hľadiska pamäte prehliadač Chrome. Jediným rozdielom je, že webové prehliadače pri zariadení Lenovo priemerne preniesli cez sieť 96.2 kB. V tabuľke 6.6 sú uvedené výsledky testovania dlhého súboru príkazov. Podľa času aj práce s pamäťou dopadol najlepšie prehliadač Opera. Sieťový prenos bol rovnaký pre všetky prehliadače a to 120 kB. Na základe výsledkov možno zhodnotiť, že pre zariadenie Lenovo je najlepšie a voľba prehliadač Opera.

V tabuľke 6.7 sú uvedené výsledky testovania krátkeho súboru príkazov. Veľkosť pamäťového bloku a počet požiadavok je rovnaký ako pri predchádzajúcich zariadeniach. Ako z hľadiska času potrebného na načítanie stránky, tak z hľadiska práce s pamäťou najlepšie

Webový prehliadač	Čas [ms]	Pamäť [kB]
Mozilla Firefox	236.25	5 152
Chrome	141.5	1484
Opera	133.75	1 521

Tabuľka 6.5: Štatistiky zariadenia Lenovo (krátky súbor)

Webový prehliadač	Čas [ms]	Pamäť [kB]
Mozilla Firefox	772.5	10 122
Chrome	235.25	1 805
Opera	231	1 660

Tabuľka 6.6: Štatistiky zariadenia Lenovo (Dlhý súbor)

skončil webový prehliadač Chrome. Sieťový prenos bol 96.2kB a počet požiadavok nezmenený. V tabuľke 6.8 sú uvedené výsledky testovania dlhého súboru príkazov. Na základe potrebného času je najlepšou voľbou prehliadač Chrome. Podľa práce s pamäťou je to prehliadač Opera. Sieťový prenos bol rovnaký pre všetky prehliadače a to 120 kB. Na základe výsledkov možno zhodnotiť, že pre zariadenie Acer je na základe pomerov medzi hodnotami prehliadačov Opera a Chrome najlepšia voľba prehliadač Opera.

Webový prehliadač	Čas [ms]	Pamäť [kB]
Mozilla Firefox	1 280	4 410
Chrome	1 220	1 428
Opera	1 339	1 963

Tabuľka 6.7: Štatistiky zariadenia Acer (krátky súbor)

Webový prehliadač	Čas [ms]	Pamäť [kB]
Mozilla Firefox	4 000	9 060
Chrome	2 571	1 614
Opera	2 581	1 583

Tabuľka 6.8: Štatistiky zariadenia Acer (Dlhý súbor)

Celkovo z výsledkov nezávisle na zariadení vyplýva, že pokiaľ chce používateľ používať daný Highlight.js skript s gramatikou Cisco, je najlepšie pri nízkej a strednej triede zariadení použiť webový prehliadač Opera a naopak pri vyššej triede zariadení použiť prehliadač Chrome. Webovému prehliadaču Firefox sa snažiť vyhnúť.

6.4 Užívateľské testovanie

Zásadnou časťou testovania tejto práce je užívateľské testovanie. Tohoto testovania sa zúčastnili hlavne štyria skúsení používatelia, ktorí so sieťovými zariadeniami pracujú a venujú sa ich konfigurácii. Testovací skript bol poskytnutý cez e-mail a spätné väzby zamestnancov boli predávané pomocou e-mailu a pohovorov cez online aplikácie na to určené. Plánované testovanie bolo rozdelené do dvoch prostredí.

Prvým prostredím bolo prostredie z praxe, kedy boli požiadaní zamestnanci firiem aby otestovali program pri vykonávaní pracovnej činnosti. V praktickom prostredí bolo testované zvýrazňovanie hlavne z praktickej sady príkazov, kedy zamestnanci testovali schopnosť odlišovania prvkov konfiguračných výpisov a príkazov. A či bolo naozaj zvýrazňovaním zachytené podstatné informácie z hľadiska zamestnancov. Či naozaj nejaké to farebné odlíšenie k niečomu prispelo, či je to rýchlejšia orientácia v príkazoch a výpisoch, alebo len jednoduché pre oči spríjemniteľné prostredie.

V akademickom prostredí sa testovalo použiteľnosť tejto práce hlavne pri akademických činnostiach ako je tvorba prezentácii pre prednášky, pomocné zvýrazňovanie syntaxe pri demonštrovaní nejakých príkladov a podobne. V tomto prostredí boli testované hlavne sady príkazov získaných z akademického a simulačného prostredia.

Z praktického hľadiska bola spätná väzba zamestnancov vyhovujúca samozrejme s drobnými výhradami, ktoré boli na základe spätnej väzby buď prerobené alebo sa prišlo k nejakému kompromisu. Išlo viac menej o detaily zvýraznenia, kedy z hľadiska zamestnancov program dostatočne nezvýraznil podstatné informácie z niektorých *show* konfiguračných výpisov. Príkladom bolo napríklad, že nebola venovaná dostatočná pozornosť sekcii rozhraní v príkaze show run, alebo smerovacích konfiguračných výpisoch boli IP adresy zvýraznene formou čísla, kdežto napríklad použitím vlastností kľúčového slova sa tieto IP adresy stali výraznejšími. So zvýrazňovaním syntaxe príkazov boli zamestnanci spokojní.

Kapitola 7

Záver

Cieľom tejto práce bolo vytvoriť zvýrazňovač syntaxe pre príkazový riadok Cisco navrhnutím gramatiky pre už existujúci nástroj Highlight.js. Tento zvýrazňovač syntaxe by umožnil administrátorom prehľadnejšie orientovanie sa v zmesi rôznych príkazov a výpisov konfigurácie sieťových zariadení Cisco. Všetky body boli docielené a vznikla z toho aplikácia pre úspešné zvýrazňovanie syntaxe, ktorú možno nájsť na stránke github.

Splnenie prvého a druhého bodu zadania bolo dosiahnuté naštudovaním si odpovedajúcej literatúry spracovanej v kapitolách 2 a 3.

Tretí bod bol splnený návrhom zvýraznenia príkazov v aplikácii, ktorá umožňuje formátovanie textu, viď kapitolu 4.

Štvrtý bod zadania bol splnený vytvorením regulárnych výrazov na základe návrhu zvýraznenia z kapitoly 4 a tieto regulárne výrazy boli priradené kľúčovým slovám nástroja Highlight.js. Popis implementácie je uvedený v kapitole 5.

Piaty bod zadania bol splnený demonštráciou zvýrazňovania syntaxe príkazov použitých v prezentáciach predmetu I1C, viď sekciu 5.1.

Šiesty bod zadania bol splnený nahraním implementovanej gramatiky do vlastného repozitára na stránke github.com[1] a úpravou požadovaných dokumentov v oficiálnej distribúcii Highlight.js. Popísané v kapitole 5.

Najväčším prínosom tejto práce pre mňa, ako riešiteľa bolo zoznámenie sa s nástrojom Highlight.js, a tým sa naučiť možnosť zvýrazniť v podstate akýkoľvek jazyk napísaním si vlastnej gramatiky jazyka. Veľmi poučná bola hlavne práca s regulárnymi výrazmi, ktoré sú v dnešných technológiách používané veľmi často.

Výsledkom tejto práce je gramatika príkazové riadku Cisco, ktorá má mnoho možností rozšírenia. Najdôležitejším aspektom rozšírenia je doplnenie gramatiky o ďalšie príkazy a rozšíriť tak gramatiku, keďže príkazy buď stále pribúdajú alebo sa syntakticky menia. Ďalšou možnosťou je zaradiť zvýrazňovač syntaxe do nejakého terminálového okna a spraviť tak zvýrazňovanie v *reálnom čase*, prípadne ešte doplniť o nejakú interaktívnu nápovedu, ktorá by informovala používateľa o chybne zadanom príkaze, alebo chybne zadanom argumente, prípadne neočakávanom argumente príkazu.

[1] https://github.com/BMatheas/highlightjs-cisco-cli

Literatúra

[1] *Introduction to Networks* [online]. [cit. 2020-01-10]. Dostupné z: https://static-course-assets.s3.amazonaws.com/ITN6/en/index.html.

[2] *Introduction to networks companion guide*. Indianapolis: Cisco press, 2014. ISBN 978-1-58713-316-9.

[3] DABLER. *Cs.wikibooks.org*. Feb 2007. Dostupné z: https://cs.wikibooks.org/wiki/Příkazy_Cisco_IOS#/media/Soubor:Cisco-router-1.svg.

[4] FRIEDMAN, D. P., WAND, M. a HAYNES, C. T. *Essentials of Programming Languages*. The MIT Press, January 1992. ISBN 978-0262560672.

[5] GOYVAERTS, J. *Lookaround* [online]. [cit. 2020-04-20]. Dostupné z: https://www.regular-expressions.info/lookaround.html.

[6] GURU99. *Performance Testing Tutorial: What is, Types, Metrics & Example* [online]. 2020 [cit. 2020-04-29]. Dostupné z: https://www.guru99.com/performance-testing.html.

[7] SAGALAEV, I. *Highlight.js developer documentation* [online]. [cit. 2020-01-13]. Dostupné z: https://highlightjs.readthedocs.io/en/latest/index.html.

[8] SATERNOS, C. *Develop and Test Regular Expressions with a Unit Testing Package*. Január 2006 [cit. 2020-05-25]. Dostupné z: https://www.oracle.com/technical-resources/articles/saternos-regexp.html.