Sem vložte zadání Vaší práce.

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra softwarového inženýrství

Bakalářská práce

Webové uživatelské rozhraní NETCONF klienta s využitím modelů YANG

David Alexa

Vedoucí práce: Ing. Tomáš Čejka

5. května 2013

Poděkování

Chtěl bych poděkovat organizaci CESNET za možnost být součástí tohoto projektu a hlavně vedoucímu práce za pomoc při řešení mnohdy nelehkých problémů při vývoji a psaní této práce.

Prohlášení

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

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

V Praze dne 5. května 2013 .....

České vysoké učení technické v Praze
Fakulta informačních technologií

Odkaz na tuto práci

Alexa, David. *Webové uživatelské rozhraní NETCONF klienta s využitím modelů YANG*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2013.

Abstract

This thesis deals with the development of *NetopeerWebGUI* — web user interface for network devices management and configuration. *NetopeerWebGUI* can communicate with any device which supports NETCONF protocol. It uses web server *Apache* with module *mod_netconf*.

NetopeerWebGUI is implemented in PHP and based on Symfony2 framework. Technologies such as XML, jQuery, SASS, OOCSS, SQLite and templating system TWIG were used for its development.

NetopeerWebGUI is regularly tested using automated tests within Selenium framework. Testing of GUI behavior — usability testing and heuristic evaluation, were also carried out by selected experts.

Keywords webGUI, NETCONF, YANG, PHP, Symfony2, XML, jQuery, TWIG, SASS, CSS, SeleniumHQ

Abstrakt

Tato práce se zabývá vývojem systému *NetopeerWebGUI* — webového uživatelského rozhraní pro správu a konfiguraci síťových zařízení. *NetopeerWebGUI* umožňuje komunikovat s libovolným zařízením podporujícím protokol NETCONF. Systém je postaven na webovém serveru *Apache* s modulem *mod_netconf*.

Systém *NetopeerWebGUI* je implementován v jazyce PHP ve frameworku Symfony2. Při vývoji byly využity technologie XML, jQuery, SASS, OOCSS, SQLite a šablonovací systém TWIG.

Testování *NetopeerWebGUI* probíhá pomocí automatizovaných testů frameworku Selenium. Chování aplikace bylo dále podrobeno uživatelskému testování použitelnosti a heuristickému vyhodnocení zvolenými experty.

Klíčová slova webGUI, NETCONF, YANG, PHP, Symfony2, XML, jQuery, TWIG, SASS, CSS, SeleniumHQ

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Správa síťových zařízení může být pro uživatele mnohdy složitá a komplikovaná. Zařízení jsou často konfigurovatelná z prostředí příkazové řádky. Ne všichni uživatelé ale chtějí příkazovou řádku používat, protože se jim to zdá příliš složité a namáhavé. Takovým uživatelům pak může konfiguraci zpříjemnit a usnadnit grafické uživatelské rozhraní (GUI).

Primární cílovou skupinou, pro kterou je GUI určeno, jsou běžní uživatelé. V dnešní době prakticky všechna zařízení, která tito uživatelé kupují, podporují ovládání přes GUI. Typickým příkladem může být GUI některého z domácích směrovačů/modemů. Díky němu dokáže i méně znalý uživatel nastavit základní parametry své domácí sítě.

Usnadněním práce pro uživatele je jejich odstínění od konkrétních, mnohdy složitých příkazů nutných k nastavení základních parametrů. GUI dokáže vyřešit veškerou logiku i technickou stránku věci za ně. Uživatelé se tak pohybují v prostředí, které využívá důvěrně známé ovládací prvky (např. formulářové prvky).

Cíle bakalářské práce

Cílem mé bakalářské práce je vytvoření grafického webového uživatelského rozhraní klienta pro správu a konfiguraci síťových zařízení zvané *NetopeerWebGUI* (dále *webGUI*). *WebGUI* komunikuje se síťovými zařízeními přes protokol NETCONF (více v sekci Protokol NETCONF). *WebGUI* uživatelům umožní oprostit se od konfigurace z prostředí příkazové řádky, kde je nastavování mnohdy nekomfortní, nepřehledné a zdlouhavé.

Síťová zařízení mívají výrobcem vytvořené GUI, které si ale každý výrobce řeší po svém. Tím pádem neexistuje univerzální rozhraní dostupné

pro všechny výrobce zařízení. Hlavní výhodou mého *webGUI* bude nezávislost na daném zařízení a tím pádem i výrobci, stačí pouze, aby zařízení umělo komunikovat přes protokol NETCONF.

Výsledkem práce bude funkční webová aplikace, pomocí které již bude možné spravovat jakékoliv zařízení používající protokol NETCONF. GUI by mělo být přehledné, uživatelsky přívětivé a hlavně by mělo usnadnit práci operátorům sítě.

Přehled požadavků na GUI

Podle požadavků operátorů z praxe by GUI mělo umět splnit následující požadavky:

- základní operace nad konfiguračními daty ve formě XML stromu (úprava, vkládání, mazání) bez nutnosti ručního použití protokolu NETCONF,
- možnost připojit se k více zařízením najednou,
- zobrazit konfigurační a stavové informace a validovat uživatelský vstup podle libovolných datových modelů v jazyce YANG,
- rozdělit přijaté stavové a konfigurační informace do sekcí s využitím modelů YANG,
- zobrazit historii zařízení, ke kterým se uživatel připojil.

Tyto požadavky se staly také zadáním mé bakalářské práce. Zadavatelem této práce je organizace CESNET, z. s. p. o. (dále jen CESNET).

Struktura textu bakalářské práce

Má bakalářská práce je rozdělena do tří základních částí: analýza 1, realizace 2 a testování 3. V úvodu Vám nejprve představím základní pojmy, které patří k problematice hromadné správy a konfigurace síťových zařízení.

V kapitole 1 se budu zabývat analýzou existujících řešení, možných způsobů realizace *webGUI* a představím si také strukturu samotného NETCONF klienta, či stav vývoje a způsob práce s NETCONF klientem před vytvořením *webGUI*. Sekce bude završena popisem architektury *webGUI*, včetně ukázky wireframů, na jejichž základě jsem postupně vytvořil celou grafickou podobu.

V kapitole 2 již budu popisovat způsob použití technik a postupů zvolených při analýze, čili popíši konkrétní a zajímavé části implementace požadavků. Jako zajímavost uvedu nestandardní využití a zápis kaskádových stylů (CSS) v kompilovatelné formě — jazyce SASS[1]. V neposlední řadě uvedu využití JavaScriptu pro asynchronní načítání informací o zařízení (*AJAX*) či použití JavaScriptu pro operace nad zobrazenými daty (transformovaným XML stromem) na klientské straně aplikace.

Závěrečná sekce bude věnovaná testování *webGUI*, kde uvedu poznatky a výsledky z testování pomocí frameworku Selenium a z reálného testování s uživateli.

Protokol NETCONF

Protokol NETCONF dle RFC6241 [10] poskytuje prostředky k získávání stavových informací, nastavení, manipulaci a mazání konfigurace síťových zařízení. Využívá k tomu kódování založené na *XML* (Extensible Markup Language), které je použito také pro zprávy protokolu. Operace protokolu jsou pak realizovány jako vzdálené volání procedur (*remote procedure calls - RPCs*).

Další zdroj [14] popisuje protokol takto: „Protokol v sobě skrývá celou řadu otevřených standardů zaměřených na vzdálenou konfiguraci síťových prvků, serverů a služeb všeho druhu. Hlavní výhodou protokolu je nezávislost na výrobci zařízení a široká možnost automatizace.“

Protokol NETCONF je již nyní využíván např. oddělením nástrojů pro monitoring a konfiguraci v organizaci CESNET nebo u produktů komerčních firem YumaWorks [2], Juniper [3], MGSoft [4]...

Datové modely YANG/YIN

Datovým modelem se rozumí technický popis zařízení, který definuje jeho vlastnosti a vazby mezi jednotlivými komponentami zařízení, vytvořený výrobcem. Operátorovi zařízení tím dokáže určit, co všechno může editovat nebo jaké informace se mu mají zobrazit.

K zápisu datových modelů používaných pro komunikaci v rámci NETCONF protokolu se používá jazyk YANG (více informací v RFC 6020 [2]).

[1] `http://sass-lang.com`
[2] `http://www.yumaworks.com`
[3] `http://www.juniper.net`
[4] `http://www.mg-soft.si`

Jedná se o logické členění zápisu v podobě stromové struktury. Existuje také alternativa, která používá syntaxi jazyka XML — formát YIN. Formát YIN nám umožňuje jednodušší strojové zpracování, protože nástroje pro práci s XML jsou dostupné téměř v každém programovacím jazyce.

Kapitola 1

Analýza a návrh

1.1 Přehled existujících řešení

Jelikož protokol NETCONF i jazyk YANG jsou poměrně mladými technologiemi (první zmínka je v RFC4741 [9] z prosince 2006), není na trhu nástrojů pracujících s těmito technologiemi mnoho. Příklady aplikací, které s protokolem pracují, jsou YUMA[5] [35], NETCONF Browser Pro[6] [15] či Netopeer[7] [4]. Jako příklad zařízení uvedu produkty firem Juniper[8], CISCO[9] či CESNET — COMBO karta[10].

1.1.1 YumaPro

Dle oficiálního popisu [33] je YumaPro [11] množinou nástrojů pro automatizaci vývoje a nastavení distribuovaných rozhraní jednotlivých zařízení. Využívá standardu YANG, který umožňuje vytvořit „univerzální rozhraní“. Dle tohoto oficiálního popisu se jedná o službu stavějící na shodné technologii, kterou se budeme zabývat (tedy kombinace NETCONF a YANG).

Po komplexnějším prozkoumání informací uvedených na oficiálních stránkách se ovšem YUMA jeví jako nástroj velmi rozsáhlý. Hlavní část aplikace, o kterou jsem se zajímal, protože se týká mé bakalářské práce, je webové uživatelské rozhraní. Dle vyjádření autorů YUMY, které jsem kontaktoval

[5] http://www.yumaworks.com
[6] http://www.mg-soft.si/mgNetConfBrowser.html
[7] https://www.liberouter.org/technologies/netconf/
[8] http://www.juniper.net
[9] http://www.cisco.com
[10] https://www.liberouter.org/technologies/card-test/
[11] http://www.yumaworks.com/products/yumapro/

e-mailem (komunikace proběhla ke dni 2. 11. 2012), jejich *webGUI* ještě není připraveno ani zveřejněno. Zmínka je uvedena na webu [34], kde je jako jedna z částí komunikující s YumaPro serverem popsáno *WEBui*. Více informací bohužel v době psaní tohoto textu nebylo k dispozici.

1.1.2 NETCONF Browser Pro

Oficiální popis aplikace [15] uvádí, že se jedná o NETCONF klienta, který umožňuje získat, upravovat, instalovat a mazat konfiguraci libovolného NETCONF zařízení na síti. Poskytuje intuitivní GUI, které umožňuje načíst jakýkoliv validní YANG či YIN modul a zobrazit ho v podobě hierarchického stromu. Umožňuje samozřejmě použití základní příkazů jako je *get*, *get-config*, *edit-config*... Dle dalších uvedených informací se jedná o univerzální aplikaci napsanou v jazyce JAVA.

Srovnání s *webGUI* V porovnání s mým *webGUI* by měl tento nástroj obsahovat v podstatě vše, ale v jiné formě. Rozdíl je ve způsobu zobrazení dat uživateli. *WebGUI* transformuje a upravuje data z XML do uživatelsky přívětivé podoby, nezobrazuje tedy přímo daný XML soubor jako např. textový editor. Navíc obsahuje tato usnadnění:

- předvyplněné výchozí hodnoty
- nápověda a popis k jednotlivým prvkům (na základě dostupných datových modelů)
- omezení konfigurovatelných položek (výrobce může v datovém modelu omezit editovatelné položky)
- transformace a seskupení hodnot do formulářových prvků (např. výběrové boxy pro výběr z výčtu povolených hodnot)

Výhodou je také odstínění operátora od příkazů NETCONF protokolu, které řeší aplikace za něj. Další vylepšení, které zatím nejsou implementované, ale počítá se s jejich nasazením, jsou uvedeny v sekci 3.5.2.2.

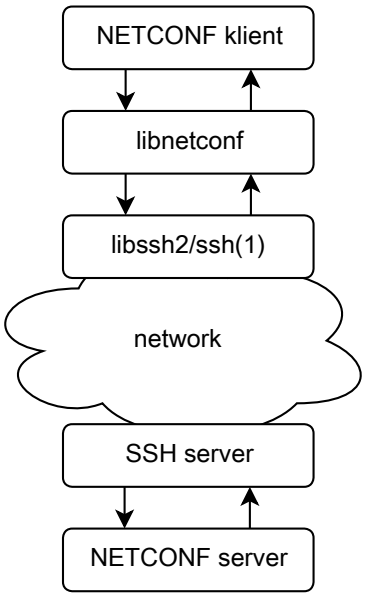
1.1.3 Netopeer

Netopeer je podle [4] projekt zaměřený na vzdálenou konfiguraci za použití NETCONF protokolu. Jedná se o projekt vyvíjený organizací CESNET. Obsahuje dílčí části Netopeer server a Netopeer klient postavené nad knihovnou *libnetconf*. *Libnetconf* je určena pro snadný vývoj NETCONF klientů a serverů.

Mé *webGUI* využívá také knihovnu *libnetconf*. Více informací je uvedeno v následujících sekcích.

1.2 Architektura klienta s knihovnou libnetconf

Architektura NETCONF klienta je vícevrstvá. Podle [5] komunikuje NETCONF klient s NETCONF serverem pomocí knihovny *libnetconf* napsané v jazyce C. Tato knihovna implementuje protokol NETCONF a poskytuje základní funkce pro vytvoření spojení, přijímání a odesílání zpráv a práci s konfiguračními daty. Spojení využívá protokol SSH. Názorně je komunikace zobrazena na obrázku 1.1.

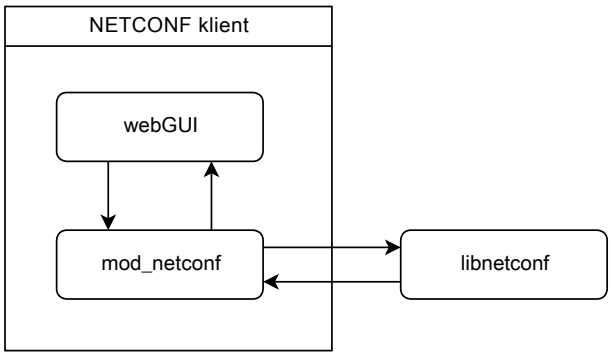


Obrázek 1.1: Diagram vrstev aplikací mezi NETCONF klientem a NETCONF serverem, které spolu komunikují pomocí knihovny libnetconf [5].

V rámci vývoje knihovny *libnetconf* byla kromě Netopeer serveru vytvořena konzolová aplikace *netconf-client*, která je alternativou pro *webGUI*. Její využití v rámci *webGUI* je uvedeno v sekci 3.1.

1.3 Architektura *webGUI*

WebGUI rozšiřuje schéma uvedené na obrázku 1.1 o *mod_netconf*, jak je uvedeno na obrázku 1.2. *Mod_netconf* je modul pro server Apache[12]. S tímto modulem probíhá komunikace z webové aplikace přes *UNIX socket*. Modul využívá *libnetconf*, jak je uvedeno v hierarchii na obrázku 1.2.



Obrázek 1.2: Zjednodušená architektura pro komunikaci *webGUI* (NETCONF klienta) s *libnetconf* pomocí *mod_netconf*.

Veškerou obsluhu komunikace s *mod_netconf* provádí třída napsaná v jazyce PHP (*Data.php* na obrázku 2.3, podrobnější popis je uveden v sekci 2.2). Tato třída zajišťuje např. provedení základních příkazů, které můžeme v rámci protokolu NETCONF použít. Kromě toho umožňuje také zpracování vstupu a výstupu komunikace, tedy načtení a následné parsování odpovědi ve formátu YIN.

1.4 Analýza možných způsobů realizace

Generování grafického webového uživatelského rozhraní pro konfiguraci NETCONF zařízení je v našem případě založeno na datových modelech YANG, resp. YIN.

Cílem je transformovat datový model do formátu zobrazitelného ve webovém prohlížeči — tzn. transformace do šablon. K tomu budou využity standardní prostředky pro tvorbu webových stránek, tedy i našeho *webGUI*. Konkrétně se jedná o značkovací jazyk HTML pro zápis šablon a programovací jazyk PHP pro jejich obsluhu. Tato kombinace je velmi rozšířená v rámci vývoje webových aplikací. Další technologie, jako např. JAVA2EE

[12] `http://httpd.apache.org`

či ASP.NET, byly po dohodě se zadavatelem bakalářské práce zamítnuty z důvodu kompatibility s prostředím zařízení zadavatele.

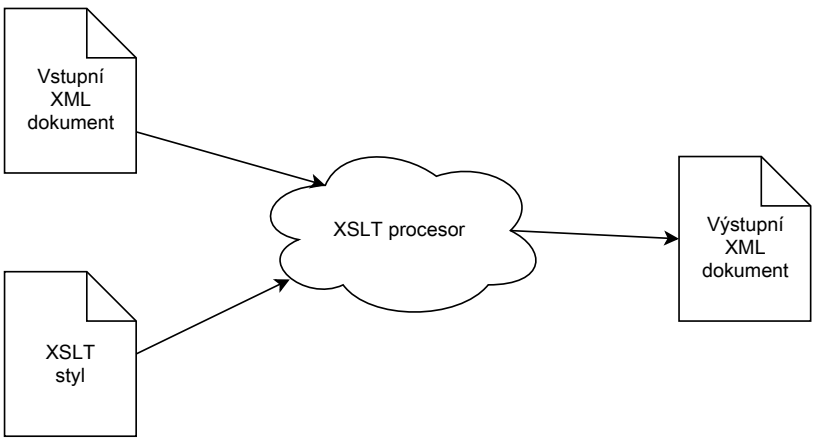
1.4.1 Možnosti generování šablon pro *webGUI*

HTML soubor (šablonu) je možné získat buď jako čistou podobu zápisu kódu, nebo také z jiných formátů zápisu transformací. V rámci této práce připadají v úvahu tyto možnosti:

1.4.1.1 Využití XSLT transformace

XSLT je podle specifikace uvedené na stránkách W3C [29] jazyk, který umožňuje transformaci XML dokumentů do jiných XML dokumentů. Pro přiblížení této obecné definice uvedu také alternativní definici, zveřejněnou v článku „Kompletní průvodce XSLT - úvod do problematiky“ [3].

„Programátor ve svém kódu zavolá XSLT procesor a předá mu vstupní XML data a XSLT styl, což je vlastně předpis, podle kterého se XML data budou zpracovávat. XSLT procesor pak vygeneruje výstupní data. Jak takový procesor funguje uvnitř, to nás vůbec nemusí zajímat.“ Tento proces je názorně zobrazen na obrázku 1.3. V našem případě by jako zdrojový XML soubor sloužil soubor ve formátu YIN, výstupem by byl HTML soubor.



Obrázek 1.3: Princip využití XSLT transformace dle [3]

Výhody:

- jedná se o relativně jednoduchý způsob generování, kde je XML kód na základě pevně daných pravidel transformován do HTML.

Nevýhody:

- složitá manipulace se samotným kódem — hlavně nepřehlednost a složitost samotné syntaxe,
- minimální možnost dynamického vkládání hodnot za běhu — bylo by nutné zdrojový XML soubor na začátku načíst a zpracovat pomocí programovacího jazyka,
- tento způsob je vhodný pro zobrazení dat, nikoliv však pro editaci a manipulaci s nimi — pokud bychom chtěli výsledek transformace nějakým způsobem upravit, bylo by nutné provést opět zpětnou transformaci.

1.4.1.2 Využití HTML šablon (či šablonovacích systémů)

HTML šablonou chápeme dokument obsahující zápis HTML značek dle standardu W3C [31]. Pomocí HTML popisujeme základní kostru dokumentu — rozdělení obsahu dokumentu do jednotlivých částí, jako je hlavička, obsahová část, patička, nadpisy, tabulky a další možné elementy. Tohoto způsobu zápisu bychom dosáhli také po aplikování XSLT transformace. Velkou výhodou oproti XSLT transformace je ale možnost dynamického generování obsahu ze strany programovacího jazyka — jednotlivé bloky kódu mohou být např. vypsány v cyklech na základě dynamických dat. Oddělíme tím logiku a přípravu dat od samotné reprezentace v HTML kódu.

Alternativou ke klasickému zápisu pomocí HTML mohou být šablonovací systémy, které přinášejí mnohá zjednodušení či vylepšení. Zpravidla obsahují klasické HTML značky, liší se ale v syntaxi dynamických částí. Navíc nabízejí například automatické *escapování* znaků, vkládání dalších šablon, rozšíření rodičovských šablon a tím možnost výpisu rekurzivních struktur, nebo funkce pro formátování vstupu dle zvolených kritérií...

Na stránkách frameworku Symfony2 [23] můžeme nalézt srovnání zápisu kódu pomocí HTML + PHP a šablonovacího sytému TWIG. Jak je vidět z následujících ukázek, šablonovací systém má mnohdy také jednodušší syntaxi.

Výhody:

- HTML šablony umožní vypsat dynamická data, která jsou získána programovacím jazykem (v našem případě ze zdrojového XML či databáze),

```
<!DOCTYPE html>
<html>
  <head>
    <title>Welcome to Symfony!</title>
  </head>
  <body>
    <h1><?php echo $page_title ?></h1>

    <ul id="navigation">
      <?php foreach ($navigation as $item): ?>
        <li>
          <a href="<?php␣echo␣$item->getHref()␣?>">
            <?php echo $item->getCaption() ?>
          </a>
        </li>
      <?php endforeach; ?>
    </ul>
  </body>
</html>
```

Výpis 1.4: HTML v kombinaci s PHP.

- zápis kódu je přehlednější a lépe čitelný — oddělili jsme totiž přípravu a zpracování dat od samotného výpisu kódu a díky tomu získáváme naprostou kontrolu nad výsledkem.

Nevýhody:

- v případě zvolení šablonovacího systému je nutné naučit se novou syntaxi pro výpis dynamických částí

1.4.2 Implementace *webGUI* v jazyce PHP

Aplikaci napsanou v jazyce PHP je možné implementovat dvěma způsoby. Buď použijeme pouze základní sadu příkazů a funkcí, které jazyk PHP obsahuje nebo využijeme některého z dostupných frameworků jako nadstavbu nad základními knihovnami.

Nyní si ukážeme rozdíly mezi způsoby tvorby aplikace v PHP.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Welcome to Symfony!</title>
  </head>
  <body>
    <h1>{{ page_title }}</h1>

    <ul id="navigation">
      {% for item in navigation %}
        <li>
          <a href="{{ item.href }}">
            {{ item.caption }}
          </a>
        </li>
      {% endfor %}
    </ul>
  </body>
</html>
```

Výpis 1.5: Ukázka zápisu pomocí šablonovací systému TWIG.

1.4.2.1 Aplikace bez využití PHP frameworku

Výhody:

- díky tomu, že využíváme pouze základní knihovny, **klesá velikost zdrojových kódů** potřebných k běhu aplikace,
- nemáme dodatečné nové **závislosti a požadavky** na systém.

Nevýhody:

- nutnost programovat aplikaci pouze se základními knihovnami, které jsou k dispozici přímo v jazyce PHP — jedná se o zdlouhavý proces, ve kterém se musíme jakoukoliv funkčnost navíc vyřešit (naprogramovat) sami (potenciální zdroj chyb),
- pro lepší kontrolu nad projektem a **udržitelnost či přehlednost** vývoje bude nutná intenzivnější analýza a návrh — je nutné vytvořit a dodržovat určitý styl a konzistenci vývoje,

- pro **znovupoužitelnost**, přehlednost a srozumitelnost pro další vývojáře je nutné vytvořit podrobnou dokumentaci i pro základní funkčnost, která se může zdát pro nás jasná,
- vyšší **časová náročnost** příprav i realizace.

1.4.2.2 Využití PHP frameworku

Výhody:

- rozšiřuje základní možnosti jazyka PHP o množinu **nových funkcí**,
- často opakované a požadované vlastnosti aplikace jsou již **vyřešeny za nás** (např. přihlašování uživatelů),
- vedou programátora k dodržování určitých, autory frameworku zvolených, **standardů vývoje**, tedy i k nutnosti dodržování zvolených návrhových vzorů, logického členění struktury adresářů, zdrojových kódů apod.,
- frameworky bývají (až na některé výjimky) dostatečně **zdokumentovány**, často včetně *best practices*, jak samotnou aplikaci vyvíjet,
- lepší **čitelnost a znovupoužitelnost** kódu, při dodržení návrhových vzorů i přehlednost a srozumitelnost pro další vývojáře.

Nevýhody:

- u jednodušších projektů může být framework **zbytečně složitým nástrojem**, který kvůli několika málo stránkám potřebuje mnohdy desítky MB kódu, který aplikace stejně nevyužije,
- často se zvyšuje množství **závislostí na samotný systém**,
- nutná delší doba pro **naučení a seznámení** se s daným frameworkem a pro zjištění, jakým způsobem máme implementovat konkrétní požadavek, jaký je životní cyklus uvnitř frameworku...

1.4.3 Výběr perzistentní vrstvy

V průběhu realizace jsem byl postaven před nutnost zvolit způsob ukládání relačních dat. Důvodem jsou informace, které si chceme ukládat u jednotlivých uživatelů:

- přihlašovací údaje — buď získané pomocí eduID či jiné autority, nebo vlastní autentizace (v rámci aplikace),
- historie připojených zařízení,
- nastavení uživatelských preferencí — nastavení aplikace.

Jako možné varianty uložení těchto dat vyplynuly pouze dvě — vlastní struktura ukládání dat do souborů nebo využití databáze. Důležitým kritériem ve výběru vhodné technologie je eliminace závislostí aplikace na dalších službách systému. Z tohoto hlediska je vhodné ukládání informací do souborů. Nevýhodou ale může být složitější reprezentace relačních informací a nutnost vytvořit si vlastní strukturu a logiku pro čtení/zápis dat.

Na druhou stranu použití databáze by tyto komplikace odstranilo. Framework Symfony2 umožňuje použití frameworku Doctrine[13] pro ORM (objektově – relační modelování). Díky tomu můžu v aplikaci použít jakoukoliv dostupnou databázovou vrstvu bez nutnosti změny zápisu dotazů na databázi (až na některé složitější konstrukty). Doctrine umí nativně podle [22] komunikovat s těmito databázemi:

- MySQL — při použití PDO Mysql extension
- PgSQL — při použití PDO PostgreSQL extension
- Oracle — při použití OCI extension
- MSSQL — při použití PDO SQLSRV extension
- SQLite — při použití PDO SQLite extension

Vzhledem k tomu, že jsme se po dohodě se zadavatelem rozhodli vybrat *Open-source* databázi, vyřadili jsme tím z výběru Oracle a MSSQL. Pro snížení závislosti na systému jsme vyřadili také PqSQL a MySQL — tyto databáze jsou příliš složité a komplexní pro naše účely, navíc vyžadují instalaci — zbytečná závislost. Díky jednoduchosti a využití pouze jednoho souboru pro uložení celé databáze a bez nutnosti instalace jsem po dohodě zvolil poslední zbývající, SQLite.

Do budoucna jistě najdu další využití relačního způsobu ukládání dat do databáze, které ještě více podpoří volbu databáze namísto vlastní struktury souborů.

[13] `http://www.doctrine-project.org`

1.4.4 Zpracování XML souborů

Většina programovacích jazyků obsahuje nástroje pro práci s XML a PHP není výjimkou. Jako hlavní možné přístupy uvedu porovnání SimpleXML a DOM. Podle srovnání uvedeného v příspěvcích [17] a [7] pokrývá SimpleXML jednoduché případy použití a obsahuje malou množinu funkcí — je zaměřeno hlavně na čtení a zápis dat. Oproti tomu implementuje DOM libovolný případ použití a je přímou implementací W3C DOM API[14] [30].

Pro jednoduchost a snadnou manipulaci jsem zvolil třídu SimpleXML. Podle podrobnější definice [27] poskytuje SimpleXML jednoduchou a snadno použitelnou sadu příkazů pro konvertování XML dokumentu do formy SimpleXML objektu, nad kterým umožňuje dotazování (XPath selektory), iterování, modifikaci hodnot a zpětný export do XML. Má ovšem také jeden, pro nás nepříjemný, nedostatek. U jednotlivých elementů si neudržuje informaci o XPath cestě k tomuto elementu. Tím pádem je nutné si při iterování nad XML stromem vytvářet XPath cestu od kořenového elementu ručně. XPath je pro nás důležitý kvůli jednoznačné identifikaci elementu, který chceme editovat. Tento nedostatek v rozhodování převážila jednoduchost práce se SimpleXML a fakt, že vlastní implementace XPath cesty není pro nás nikterak složitá.

1.4.5 JavaScript

Ve *webGUI* budeme potřebovat použít pro snížení komunikace a datové náročnosti skriptovací jazyk, který bude spuštěn na straně klienta, tedy v prohlížeči uživatele. Jako příklad uvedu možnost editace vypsaného XML stromu či asynchronní požadavky pro aktualizaci obsahu bez nutnosti znovu načtení celé stránky. Skriptování je podle [28] „program, který nepotřebuje kompilaci před jeho spuštěním. V kontextu webového prohlížeče se jedná většinou o program napsaný v JavaScriptu, který je vykonán prohlížečem, jakmile je stránka stažena nebo v průběhu práce uživatele se stránkou jako odpověď na událost vyvolanou uživatelem. Skriptování umožňuje vytvářet dynamičtější stránky, například bez nutnosti obnovení stránky může změnit její obsah.“ AJAXem rozumíme podle shrnutí uvedeném v článku [12] asynchronní načítání XML dokumentů pomocí JavaScriptu. Nejedná se o žádnou novou technologii, ale pouze o spojení stávajících technologií do jedné nové (např. využití standardního HTML a CSS, DOMu (vysvětleno níže), XML a XSLT, XMLHttpRequest a JavaScriptu).

[14] `http://www.w3.org/DOM/`

Základním skriptovacím rozhraním vyvinutým v organizaci W3C[15] je DOM (Document Object Model). DOM podle [28] umožňuje programům a skriptům dynamicky přistupovat a aktualizovat obsah, strukturu a styly dokumentu. Tento popis naprosto vystihuje požadavky, které si na skriptování na straně klienta klademe.

V dnešní době stále bohužel neinterpretují všechny prohlížeče JavaScript stejným způsobem. Aby programátor nemusel řešit nekompatibilitu mezi prohlížeči, vznikly různé knihovny, které tyto problémy odstiňují. Mezi nejznámější patří jQuery[16] a MooTools[17]. Vzhledem k tomu, že jQuery využívám již několik let na jiných webových projektech, byla pro mě díky této zkušenosti volba JavaScriptového frameworku jednodušší. Dalším důvodem je také podrobná dokumentace nebo množství volně dostupných hotových zásuvných modulů.

1.4.6 Kaskádové styly — SASS

Kaskádové styly (CSS) jsou podle [32] jednoduchý mechanismus pro přidávání stylů (např. písma, barev, pozadí) webovým dokumentům. Jedná se o oddělení vzhledu stránky od samotného HTML dokumentu. CSS prošlo dlouhým vývojem a ustálilo se dnes ve verzi CSS3.

Po nástupu CSS3 došlo k podobné situaci, která nastala i u JavaScriptu — nekompatibilita zápisu mezi jednotlivými prohlížeči. Jednotlivé prohlížeče řeší zápis nových CSS3 vlastností většinou přidáním vlastních prefixů (např. -webkit-, -o-, -ms-, -moz-). Složitější zápis pomohl většímu rozšíření CSS preprocesorů.

Preprocesory obsahují základní programovací konstrukty jako jsou proměnné, cykly a funkce. Tím se stává přístup k zápisu CSS zcela odlišný, než dříve. Usnadnění zápisu je také umocněno tím, že se upravila syntaxe. Jako nejdůležitější považuji možnost zanořování jednotlivých elementů do sebe (*nesting*). Mezi základními funkcemi oceňuji zejména práci s barvami (např. ztmavení) či matematické operace. Tyto preprocesory nám obecně umožňují odstínit se od složitých zápisů prefixů a zbytečného duplikování kódu, podobně jako frameworky. Nutné je ovšem vytvořit si tyto funkce pro nové CSS3 vlastnosti — opět i zde ale existují frameworky, které tyto funkce řeší za nás.

Jako příklad preprocesorů bych uvedl dva nejrozšířenější — SASS[18]

[15] `http://www.w3.org`
[16] `http://www.jQuery.com`
[17] `http://www.mootools.net`
[18] `http://sass-lang.com`

a LESS[19]. Oba dva zmíněné preprocesory nabízí v podstatě totožné možnosti. Rozdílný je ovšem způsob kompilace, kdy SASS využívá Ruby a LESS node.js. Při volbě budu opět vycházet ze svých zkušeností, kdy jsem se na začátku rozšiřování těchto preprocesorů rozhodl právě pro SASS, který od té doby také pravidelně používám.

Zmínil jsem se také, že existují frameworky pracující např. nad jazykem SASS. Příkladem je COMPASS — framework využívající SASS. Nabízí velké množství funkcí pro zápis nových CSS3 vlastností i jiné usnadnění. Jelikož CSS3 je navíc podporováno pouze v moderních prohlížečích, nabízí COMPASS také *fallback* pro starší prohlížeče. Jedná se zatím o nejkomplexnější framework nad jazykem SASS, a opět díky dobré zkušenosti z minulosti jsem se rozhodl tento framework využít.

1.5 Výběr vhodného způsobu realizace

Po konzultaci možných řešení s vedoucím práce jsme zvolili kombinaci HTML šablonovacího systému s využitím PHP frameworku, jako databázovou vrstvu jsme zvolili SQLite, jak již bylo zmíněno v sekci 1.4.3. Důvodem této volby byla hlavně jednoduchá a efektivní správa šablon, znovupoužitelnost zdrojových kódů, přehlednost zápisu kódu či jasná pravidla (*best practices*) frameworku, která při jejich dodržování umožní i dříve nezasvěcenému programátorovi rychle pochopit celou strukturu aplikace.

PHP frameworků existuje celá řada, mezi nejpoužívanější patří podle [6] a [18] tato čtveřice:

- CakePHP[20],
- CodeIgniter[21],
- Symfony[22],
- Zend[23].

V České republice je také velmi rozšířený framework Nette[24]. Vzhledem k pozitivním zkušenostem s frameworkem Symfony2 v rámci předmětu BI-WT2 jsem zvolil právě tento framework. Důvodem byl také rozsah možností, které Symfony2 nabízí, nebo např. moderní způsob zápisu veškerého

[19] `http://lesscss.org`
[20] `http://cakephp.org`
[21] `http://ellislab.com/codeigniter`
[22] `http://www.symfony.com`
[23] `http://framework.zend.com`
[24] `http://www.nette.org`

nastavení pomocí anotací či kvalitní dokumentace a uživatelská základna pro řešení problémů.

Symfony2 používá k zápisu HTML šablon šablonovací systém Twig[25], který nabízí programátorům sadu funkcí pro modifikaci výpisu kódu. Patří mezi ně hlavně možnost rekurzivního výpisu kódu, což nám umožňuje efektivní generování celého XML stromu ve formě HTML. Oproti transformace pomocí XSLT (1.4.1.1) je generování prováděno v rámci frameworku, díky čemuž máme v šablonách plnou kontrolu nad formou výstupu. Jako další výhodu bych vyzdvihl také např. automatické cachování a tím snížení výpočetní náročnosti aplikace.

1.6 Framework Symfony2

1.6.1 Představení frameworku

Jak bylo zmíněno již v sekci 1.5, pro tvorbu tohoto projektu jsem zvolil framework Symfony2 [25]. Framework je postaven na architektuře MVC (Model-View-Controller).

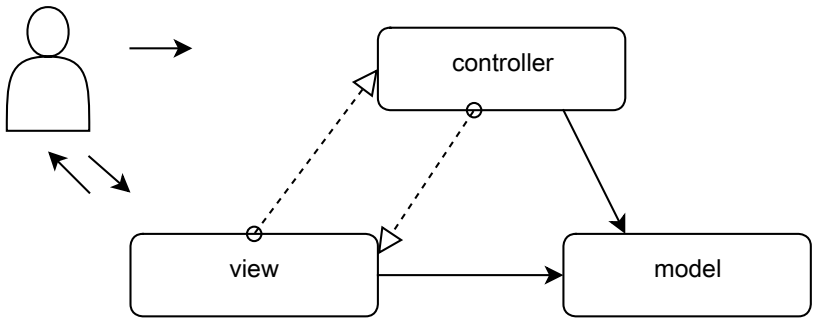
1.6.2 Aplikování a práce s MVC

V článku „Úvod do architektury MVC“ [1] je návrhový vzor MVC popsán takto: „Architektura MVC dělí aplikaci na 3 logické části tak, aby je šlo upravovat samostatně a dopad změn byl na ostatní části co nejmenší. Tyto tři části jsou Model, View a Controller. Model reprezentuje data a business logiku aplikace, View zobrazuje uživatelské rozhraní a Controller má na starosti tok událostí v aplikaci a obecně aplikační logiku.“ Názorná komunikace je uvedena na obrázku 1.6.

Při používání a snaze o dodržování *best practices* pro vytváření aplikací v rámci Symfony2 ale přijdeme na to, že se striktně dle definice o třívrstvou MVC architekturu nejedná. Architekturu bych nazval podle zkušeností získaných v předmětu BI-ZSI spíše „dvou a půl vrstvou“, protože Controller je často nucen řešit také business logiku aplikace.

Celý framework je silně závislý na používání *jmenných prostorů (namespaces)*, které poskytuje PHP od verze 5.3 [26]. S tím souvisí také logické členění zdrojových souborů aplikace (viz. 1.6.3).

[25] `http://twig.sensiolabs.org`



Obrázek 1.6: Názorná ukázka komunikace mezi jednotlivými vrstvami v aplikace založené na návrhovém vzoru MVC.

1.6.3 Stromová struktura zdrojových souborů

V rámci frameworku jsou veškeré spolu související části aplikace shlukovány do balíčků, tzv. *bundles*. Aplikace může obsahovat neomezené množství těchto balíčků. Ty jsou samostatnými jednotkami, které jsou závislé na samostatném jádře frameworku, tzv. *vendors*.

Každý balík má doporučenou strukturu adresářů, která je logicky členěná dle potřeb MVC. V manuálových stránkách [24] je doporučená struktura popsána jako u výpisu 1.7:

```
FIT/...
└── NetopeerBundle/
    ├── FITNetopeerBundle.php
    ├── Controller/
    ├── Resources/
    │   ├── meta/
    │   │   └── LICENSE
    │   ├── config/
    │   ├── doc/
    │   │   └── index.rst
    │   ├── translations/
    │   ├── views/
    │   └── public/
    └── Tests/
```

Výpis 1.7: Doporučená struktura adresářů v balíku.

Oproti základní struktuře potřebuji do balíčku přidat také informace

o modelech (datová vrstva, příkladem je třída *Data.php* zmíněná v sekci 1.3). Stromovou strukturu jsem rozšířil o další datovou vrstvu — *Models*, ve které jsou tyto třídy seskupeny.

Pro ukládání datových modelů získaných ze zařízení jsem doplnil stromovou strukturu také o *Data*, která obsahuje vlastní strukturu ukládání dat. Celá tato upravená struktura, včetně ukázky uložení předgenerovaných modelů, je patrná z následujícího výpisu stromové struktury balíku *NetopeerBundle* 1.8.

Pro třídy, které jsou pomocí *ORM* mapovány na zvolenou databázi, jsem přidal složku *Entity*. Jako příklad uvedu entitu „Připojené zařízení (*MyConnection.php*)“, která uchovává informace o připojeném zařízení nutné pro komunikaci s nižšími vrstvami aplikace, nebo také mapování na uživatele pro zobrazení historie připojených zařízení.

```
FIT/...
└── NetopeerBundle/
    ├── FITNetopeerBundle.php
    ├── Command
    ├── Controller
    ├── Data
    │   └── models
    │       ├── hashed namespace da39a3ee5e...
    │       │   └── module name ietf-netconf-monitoring
    │       │       └── revision 2011-03-08
    │       │           ├── section capabilities
    │       │           ├── section datastores
    │       │           ├── section schemas
    │       │           ├── section sessions
    │       │           └── section statistics
    │       └── hashed namespace 038f52ba9e...
    │           └── module name nc-notifications
    │               └── revision 2008-07-14
    ├── DependencyInjection
    ├── Entity
    ├── Models
    ├── Resources
    │   ├── config
    │   ├── doc
    │   ├── public
    │   │   ├── dokumentace aplikace docu
    │   │   ├── images
    │   │   ├── js
    │   │   ├── sass
    │   │   └── style
    │   ├── translations
    │   └── views
    ├── Tests
    └── Twig
```

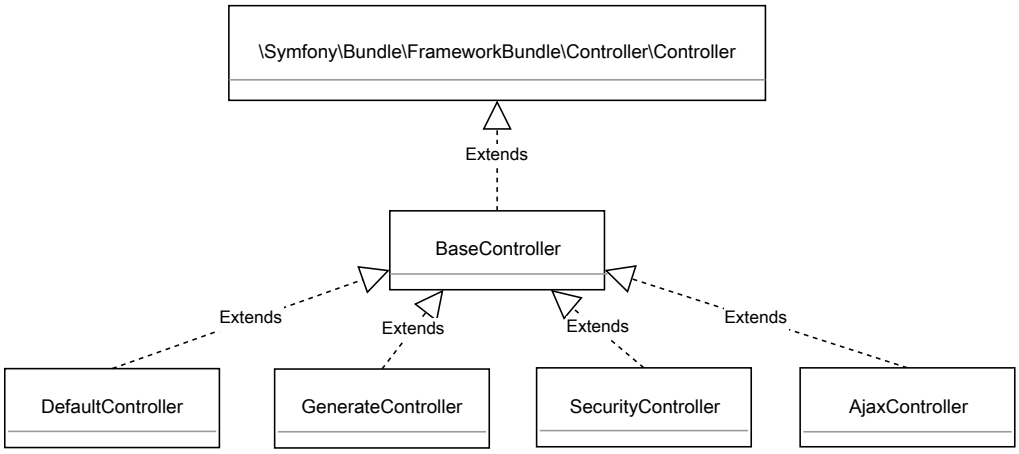
Výpis 1.8: Modifikovaná struktura adresářů v balíku. *Kurzívou* je zobrazen popis jednotlivých složek (pokud je užitečný), **tučně** jsou označeny složky, které jsem vytvořil navíc oproti základní struktuře.

Kapitola 2

Realizace

2.1 Členění kontrolerů

Pro lepší orientaci v rámci stromové struktury souborů frameworku jsem rozdělil akce (zobrazení jednotlivých typů stránek) do několika kontrolerů. Tento způsob rozdělení je aplikován také na soubory se šablonami. Zvolená hierarchická struktura kontrolerů je uvedena na obrázku 2.1.



Obrázek 2.1: Hierarchická struktura kontrolerů použitých ve *webGUI*. *BaseController* je v tomto případě rodičem všech kontrolerů a obsahuje společné metody pro všechny kontrolery.

2.1.1 BaseController

Základním kontrolerem, který je rodičem pro všechny další kontrolery, je *BaseController*. Jeho úkolem je umožnit dalším kontrolerům zpřístupnit metody, které jsou pro všechny z nich společné.

Jako typický příklad uvedu metodu *assign()*. Pro předávání proměnných do šablon je nutné v každé akci (metoda připravující vykreslení dané stránky) kontroleru vrátit pole proměnných a hodnot. Abych zjednodušil práci s tímto polem a nemusel jsem řešit logiku v každé akci zvlášť, vytvořil jsem si tuto společnou metodu *assign()*. Díky ní můžu plnit pole hodnot z kteréhokoliv místa kontroleru a mám jistotu, že vždy šabloně předám požadované hodnoty.

BaseController se také stará o přípravu dat pro výpis horního menu (seznam modelů) a levého menu (výpis sekcí). Další funkcí je třídění chybových (úspěšných hlášek) pro jejich korektní vykreslení na požadovaných místech layoutu. Protože se jedná o činnosti, které je nutné provést před každým vykreslením stránky, je tento kontroler ideálním místem, kde tuto logiku řešit.

2.1.2 DefaultController

Nejdůležitějším kontrolerem v celém *webGUI* je *DefaultController*. Obstarává totiž veškerý přímý výpis šablon a jednotlivých částí aplikace, mezi které patří:

- rozcestník pro připojení k novému zařízení včetně výpisu všech připojených zařízení,
- výpis konfigurační a stavové části — výpis XML stromu,
- zpracování formulářů po úpravě hodnot XML stromu.

DefaultController komunikuje s třídou *Data.php* pro jakýkoliv výpis či editaci XML stromu, ale i pro získání jednotlivých modelů a sekcí pro vytvoření horního a levého menu (implementováno v rámci *BaseControlleru)*. Třída *Data.php* obstarává komunikaci s nižšími vrstvami aplikace (viz. sekce 2.2). Pro každou akci (komunikaci) s touto třídou je nutné posílat požadavky ve správném formátu, včetně identifikátoru zařízení, o které se jedná, filtru pro výpis a vlastní zprávy (XML řetězec) v požadovaném formátu.

Po odeslání formuláře s úpravami XML stromu je nutné tyto změny nějakým způsobem zpracovat — typicky nahradit některé hodnoty v odpovědi ze serveru, přidat atributy odpovídající požadované akci, omezit XML

strom pouze na editované informace apod. Celá tato příprava odpovědi pro další zpracování probíhá právě v tomto kontroleru.

Kontroler se kromě zpracování dat stará také o správné zobrazení odpovědi ze serveru. Nutné je aplikování filtru pro zobrazení správného modulu (odkaz v horním menu), rozdělení odpovědi do jednotlivých sekcí, aplikování filtrů pro omezení výpisu či oddělení výpisu stavových a konfiguračních informací (rozdělení do dvou sloupců).

Před výpisem XML stromu dojde ke spojení odpovědi s datovým modelem (pokud je k dispozici), který obohatí odpověď o další sémantické atributy a informace. Výpis XML stromu probíhá iterováním nad *SimpleXML* objektem a jeho rekurzivním výpisem pomocí dvou šablon rodiče a dítěte (unikátní dvojice jak pro výpis stavových, tak konfiguračních dat). V těchto šablonách již vypisuji jednotlivé XML elementy ve formě požadovaného HTML elementu, včetně zobrazení různých typů formulářových prvků, nápovědy apod.

Více informací o zpracování XML pro vytvoření požadované odpovědi a o nutných úpravách XML odpovědi získané ze serveru pro výpis je uvedeno v sekci 2.3.

2.1.3 AjaxController

Pro všechny asynchronní akce spuštěné Ajaxovým voláním je určen *AjaxController*. Prozatím se jedná o tyto akce:

- zpracování <get-schema> po úspěšném připojení k zařízení,
- výpis historie a profilů zařízení,
- uložení zařízení z historie připojených zařízení do profilů zařízení,
- smazání zařízení z historie nebo profilů zařízení.

Odpovědi jednotlivých akcí jsou vypisovány jako JSON. Tím je umožněno jednoduché zpracování výsledku JavaScriptem (více v sekci 2.4).

2.1.4 SecurityController

SecurityController má pouze dvě základní funkce — přihlášení a odhlášení uživatele. Jeho úkolem je vykreslení přihlašovacího formuláře, pokud se nepřihlášený uživatel pokouší přistoupit do aplikace, a jeho následná autentizace, případně odhlášení. Vstup pod entitou přihlášeného uživatele je vyžadován pro všechny stránky ve *webGUI*.

Entita uživatele je uchovávána v instanci třídy *User.php*. Ta je pomocí *ORM* přímo mapována na zvolenou databázi. Autentizace uživatele probíhá oproti údajům uloženým v této databázi. V rámci frameworku se jedná o tzv. způsob autentizace *user_db*.

Hlavním důvodem, proč je vyžadována autentizace uživatele, je potřeba jednoznačné identifikace uživatele pro ukládání historie připojených zařízení a profilů zařízení. Do budoucna je např. plánováno upravitelné nastavení *webGUI* pro každého uživatele.

Kromě autentizace oproti údajům uloženým v databázi existuje ještě jedna výjimka — uživatel, který je pevně určen v konfiguračních souborech balíčku (bundle *NetopeerBundle*), tzv. uložení *in_memory*. Tento uživatel nemá žádný záznam v databázi, tudíž není možné ukládat jeho informace o připojených zařízeních.

Slouží k ověření funkčnosti aplikace i bez použití databáze, protože do budoucna je v plánu umožnit vstup do *webGUI* i bez přihlášení — nazvěme to *demo* účtem. Vše je za předpokladu omezení některých funkčností *webGUI*, které nejsou nezbytně nutné pro běh aplikace a jsou spojeny s konkrétním uživatelem — příkladem může být historie připojených zařízení.

2.1.5 GenerateController

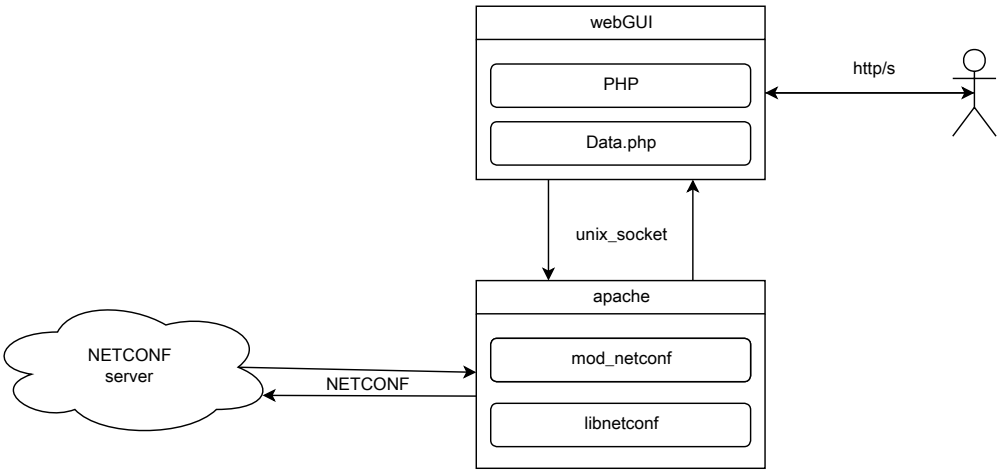
GenerateController slouží prozatím pouze k jedné činnosti. Vypisuje požadovaný XML strom (závislý na zvoleném modelu, případně sekci) buď ve formátu XML nebo HTML (tedy zpracovaný šablonami pro výpis stromu). Tento výstup ovšem není zasazen přímo do *layoutu* stránky, jedná se o formu výpisu vhodnou pro další zpracování. Pro určení, o který formát výpisu se má jednat, jsem si pro zajímavost nastavil trochu složitější *routu* (definice cesty v URL adrese) pro tuto akci. *Routa* je uvedena na výpisu 2.2.

```
@Route("/generate/{level}/{xPath}/{key}/{module}/{↙
    ↳ subsection}/model.{_format}", defaults={"↙
    ↳ module" = null, "subsection" = null, "_format↙
    ↳ " = "html"}, requirements={"_format" = "html|↙
    ↳ xml"}, name="generateFromModel")
```

Výpis 2.2: Definice *routy* pro správné rozpoznání požadovaného formátu výpisu souboru. Jak je vidět z nastavení, povolené formáty jsou XML a HTML, kde je jako výchozí nastaven HTML formát.

2.2 Spolupráce s nižšími vrstvami aplikace

WebGUI spolupracuje s nižšími vrstvami aplikace pomocí třídy *Data.php*, jak je znázorněno na obrázku 2.3. Výchozí metodou pro veškerou práci a komunikaci s *mod_netconf* je metoda *handle()*. Umožňuje provést operace, které API *mod_netconf* poskytuje a dále zpracovává.



Obrázek 2.3: Podrobná architektura pro komunikaci *webGUI* (NETCONF klienta) s NETCONF serverem.

Seznam možných operací, které metoda *handle()* poskytuje, vychází z RFC 6241 [10] a RFC 6022 [20], kde je uveden také jejich popis.

- <lock> — uzamyká celé datové úložiště,
- <unlock> — odemyká datové úložiště,
- <get> — získává konfigurační a stavové informace,
- <get-config> — získává konfigurační informace,
- <edit-config> — upravuje data v konfiguraci úložiště.
- <get-schema> — získává schéma z NETCONF serveru.

Knihovna *mod_netconf* obohacuje základní operace navíc o interní příkazy pro komunikaci s NETCONF serverem.

- <connect> — připojuje se k *UNIX socket*u,

- <disconnect> — odpojuje se od *UNIX socket*u,
- <info> — získává informace o navázaném spojení.

Třída *Data.php* je v rámci *webGUI* velmi důležitá – provádí veškerou komunikaci s *mod_netconf*. Z tohoto důvodu jsem tuto třídu vytvořil jako *službu* (*Service*) v rámci návrhového vzoru *Dependency Injection* (DI). DI podle [11] [19] „slouží pro snížení závislostí mezi jednotlivými částmi systému. Jeho provedení vychází z obecnějšího návrhového vzoru *Inversion of Control* (IoC).“

Framework Symfony2 má pro definování závislostí připravenou strukturu v souboru *Resources/config/services.xml*. Díky tomu je nyní možné používat třídu z jakéhokoliv Controlleru (či ji vložit do jiné třídy (nutné přidat závislost)) bez nutnosti vytváření konkrétních instancí přímo v kódu jednotlivých metod. Pro ukázku, definice třídy *Data.php* jako *služby* v rámci frameworku Symfony2 je uvedena na výpisu 2.4. Jak můžeme vidět i na výpisu, třída *Data.php* je také závislá na dalších *službách*. V průběhu provádění příkazů a komunikace s *mod_netconf* využívám *službu* pro logování nebo cachování určitých odpovědí ze serveru.

2.3 Práce s XML

Na práci s XML v rámci *webGUI* bychom se mohli dívat dvěma pohledy — zpracování odpovědi ze serveru pro výpis v šabloně a přípravu XML řetězce s úpravami nastavení zařízení pro odeslání zpět na server.

2.3.1 Zpracování odpovědi ze serveru

V prvním kroku se pokusím načíst odpověď serveru v podobě XML řetězce pomocí *SimpleXML* objektu. Pokud načtení proběhne v pořádku, znamená to, že je XML řetězec je validní a lze jej dále zpracovávat. V opačném případě se často stává, že XML řetězec neobsahuje kořenový uzel. Pokusím se proto obalit XML řetězec do chybějícího kořenového uzlu a zkusím načíst XML znovu pomocí *SimpleXML*. Pokud ani tento pokus není úspěšný, oznámím uživateli chybu při zpracování pomocí chybové hlášky a zpracování odpovědi zde končí — nemáme validní XML řetězec a nemůžeme jej ani vypsat.

Pokud je načtení úspěšné, podle hodnoty vstupního argumentu metody provedu spojení odpovědi s předgenerovaným modelem (získaného pomocí operace <get-schema> po navázání spojení se serverem) nebo vrátím rovnou původní odpověď. Spojení odpovědi s předgenerovaným modelem do-

```
parameters:
    netopeer.data.class: FIT\NetopeerBundle\Models\↙
        ↳ Data

services:
    data_logger:
        class: Symfony\Bridge\Monolog\Logger
        arguments: [app]
        calls:
            - [pushHandler, [@data_handler]]

    data_handler:
        class: Monolog\Handler\StreamHandler
        arguments: [%kernel.logs_dir%/%kernel.↙
            ↳ environment%.data.log]

    DataModel:
        class: %netopeer.data.class%
        arguments: [@service_container, @data_logger↙
            ↳ ]

    winzou_cache:
        factory_service: winzou_cache.factory
        factory_method:  get
        class:           %winzou_cache.driver.↙
            ↳ abstract%
        arguments:
            - file
```

Výpis 2.4: Definice třídy *Data.php* jako *služby* v souboru *Resources/-config/services.xml*. Tato třída je navíc také závislá, konkrétně na *service_container* a *data_logger*, které jsou do třídy vkládány v konstruktoru jako argumenty.

plňuje k odpovědi atributy se sémantickým významem pro výpis v šabloně. Příkladem takové sémantické informace jsou datové typy pro správnou transformaci do formulářových prvků (inputů), text nápovědy, informace, zda je hodnota povinná pro vyplnění apod.

V obou případech následují buď některé z dalších operací s XML od-

povědí, nebo následuje rekurzivní výpis do šablony. Při rekurzivním výpisu si u jednotlivých elementů navíc sám uchovávám informaci o XPath cestě — *SimpleXML* objekt tuto informaci nenabízí, jak jsem zmínil v analýze. XPath cestu potřebuji pro následné zpracování upravených hodnot stromu pro jednoznačnou identifikaci elementu, kterého se změna týká — díky tomu jsem schopen změnit správnou hodnotu v XML odpovědi. Tu pak pošlu zpět na server pomocí operace <edit-config>, která tyto změny na serveru aplikuje.

2.3.2 Příprava XML řetězce obsahující konfigurační informace

Po změně hodnot jednotlivých XML elementů je nutné odeslat serveru zprávu pomocí operace <edit-config> ve správném formátu. XML řetězec musí obsahovat požadované elementy a atributy a musí být v požadované struktuře. Proto je nutné po úpravě hodnot XML odpovědi provést další modifikace.

Jako příklad můžu uvést doplňování XML stromu u elementu, který chceme smazat. Nejprve z formuláře zjistím, který element v XML stromu chci smazat. Díky tomu, že si u elementu udržuji XPath cestu, můžu pomocí ní jednoduše najít konkrétní element. K němu stačí doplnit atribut *operation="remove"*, abych jasně určil, který element má být smazán.

Abych připravil validní zprávu operace <edit-config>, musí XML strom odpovídat datovému modelu. To znamená, že musím doplnit všechny nadřazené elementy k upravovanému elementu a tím vytvořit kompletní cestu od kořenového elementu. Navíc musím dodat některé další povinné elementy, které jsou zpravidla nutné pro unikátní identifikaci modifikovaného elementu. Doplnění elementů do stromu probíhá v metodě *findAndComplete()*.

Duplikování a vkládání nových konfiguračních informací v datovém úložišti serveru se provádí podobným způsobem. I v těchto případech je potřeba vytvořit validní zprávu podle datového modelu.

2.3.3 Validace XML výstupů

Před odesláním zprávy pomocí operace <edit-config> provádím sémantickou validaci odesílaného XML řetězce pomocí Relax NG [26] a W3C XML Schema [27].

[26] `http://relaxng.org`

[27] `http://www.w3.org/XML/Schema`

XML řetězec je validní po syntaktické stránce již před touto validací, protože by jej jinak nebylo možné ani načíst. Výsledek těchto sémantických validací ukládám do logu aplikace, aby jej bylo možné zpětně zkontrolovat a analyzovat. Prozatím není důvod, pokud validace selže, neprovést operaci <edit-config>. Server si totiž se syntakticky validní zprávu zpravidla poradí.

Soubory s validačními schématy jsou vygenerovány zároveň v rámci generování datových modelů pro připojené zařízení. Vygenerované soubory jsou uloženy do stejné stromové struktury, jako datové modely. (Více informací je uvedeno v sekci 1.6.3).

2.4 Využití JavaScriptu jako hlavního nástroje pro editaci

JavaScript využívám ve *webGUI* pro dva základní druhy operací — asynchronní požadavky a úpravy objektového modelu dokumentu (*DOM*).

2.4.1 Asynchronní (Ajaxové) požadavky

Bez použití asynchronního načítání se stránka zobrazená v prohlížeči může změnit pouze znovunačtením nebo přechodem na stránku novou. Každé načtení stránky znamená načtení a zpracování dat, která obdržíme ze serveru. Použitím asynchronního načítání můžu načítat některé informace na pozadí a uživatel již může se stránkou pracovat. Jakmile je asynchronní požadavek dokončen, můžu obnovit pouze požadované části webové stránky, aniž bych ji musel načítat a překreslovat celou znovu.

Asynchronním požadavkem provedu akci, kterou mi umožňuje provést *AjaxController* (více informací v sekci 2.1.3). Jako příklad uvedu uložení zařízení z historie zařízení do profilu zařízení. Po provedení této akce získám v JavaScriptu odpověď zpravidla ve formě JSON řetězce, který dekóduji a dále zpracovávám. V tomto konkrétním případě, pokud proběhlo přidání úspěšně, následuje znovunačtení sekce profily připojených zařízení. Abych nemusel v JS zbytečně řešit logiku řazení výsledků, provedu jeden další dotaz, kterým získám HTML kód celé sekce profilů připojených zařízení. Tímto kusem HTML kódu nahradím stávající kód a jednotlivým odkazům přiřadím akce jako zobrazení ikonek pro editaci při přejetí myší nebo vyplnění formuláře pro připojení k zařízení po kliknutí na odkaz.

Dalším asynchronním požadavkem je operace <get-schema>, která se volá po úspěšném připojení k zařízení. Uživatel může dále pracovat, indikátor mu totiž oznamuje, že zpracování modelů běží na pozadí. Po dokončení operace podle návratového typu zobrazím chybovou/úspěšnou hlášku

a přidám odkaz pro konfiguraci připojeného zařízení místo indikátoru zpracování.

Do budoucna se počítá s intenzivnějším využitím této technologie pro nové, doposud neimplementované, případy použití — více v sekci 3.5.2.2.

2.4.2 Úpravy a práce s objektovým modelem dokumentu (dále DOM)

2.4.2.1 Přeskupení odkazů v horním menu

Prvním příkladem práce s DOMem je automatické přeskupení odkazů v horním menu, pokud pro ně není dostatečný prostor. Jako příklad funkčnosti uvedu např. známé chování webového prohlížeče, který má otevřené velké množství stránek, které se nevejdou do lišty s kartami (otevřená stránka v rámci okna prohlížeče, anglicky *tab*). Prohlížeč v této chvíli automaticky přesune *taby*, které již není schopen zobrazit, do vysouvacího seznamu na konci lišty. Pokud se místo pro *tab* uvolní, opět je přesune zpět. Stejnou funkčnost pomocí manipulace s DOMem umí i *webGUI*. Manipulace probíhá tímto způsobem:

1. kontrola, zda se v horním menu zobrazují odkazy, pro které není dostatečný prostor,
2. pokud ano, zobrazení ikonky pro vysouvací seznam odkazů,
3. zduplikování přebývajících odkazů do vysouvacího seznamu,
4. odstranění odkazu z lišty odkazů.

Ukázka výsledného vzhledu horního menu s vysouvacím seznamem odkazů si můžete prohlédnout na obrázku 2.5.

Pro odstranění odkazů z vysouvacího seznamu odkazů je nutné přesunout všechny odkazy z vysouvacího seznamu zpět do lišty. Poté se provede stejný postup, jaký je uveden výše. Přeskupení odkazů v horním menu probíhá automaticky po každé změně velikosti okna.

2.4.2.2 Duplikování, vytváření a mazání uzlů v XML stromě

Poněkud složitějšími operacemi jsou operace s XML stromem — duplikování, vytváření a mazání uzlů. Pro ně je společná základní funkcionalita — překrytí daného sloupce (obsahující daný konkrétní XML strom) tmavým poloprůhledným obalem a zobrazení formuláře pro zvolený způsob editace. Tento krok s sebou přináší nutnost vyjmutí celého stromu z formuláře,

Obrázek 2.5: Prostor pro zobrazení všech odkazů v horním menu není dostatečný. Proto byly odkazy, které se do prostoru nevešly, přesunuty do vysouvacího seznamu na konci prostoru pro zobrazení odkazů.

ve kterém je umístěn, protože v HTML není povoleno vnořovat do sebe více formulářů. Originální formulář si uchovám v DOMu pro zpětné převedení po ukončení editace pomocí jQuery funkcí *wrap()* a *unwrap()*. Náhled formuláře si můžete prohlédnou v příloze na obrázku E.10.

Na obrazovce máme nyní zobrazený poloprůhledný obal s prázdným formulářem. Pokud máme v plánu smazat daný uzel, zobrazíme potvrzovací tlačítko na smazání daného uzlu — tak, jak bychom to očekávali. Zajímavější situace nastává v případě duplikování (podobně i vytváření) uzlu. Zde potřebuje zduplikovat (funkce *clone()*) uzel a provést na něm několik transformací.

Prvním krokem u naklonovaného uzlu je odstranění řádků (elementů), které mají nastaveno, že se nejedná o editovatelné položky (buď stavové položky, nebo ty, které jsou vypnuté (*disabled*)). Zůstaly nám pouze editovatelné elementy i s originálními formulářovými prvky (dále inputy). Všechny inputy ale mají špatné jméno (atribut *name*) pro jeho správnou identifikaci po odeslání formuláře. Toto jméno je nutné změnit, abychom odlišili původní inputy od nových — díky tomu také zjistíme, že se nejedná o pouhou editaci hodnot, ale vytváření nových. S tím souvisí také vložení informace o XPath cestě (případně úprava cesty u jednotlivých inputů), aby bylo možné v kontroleru jednoznačně určit místo, do kterého máme nový uzel vložit. Více o důležitosti XPath cest a jejich využití je uvedeno v sekci 2.3.

Druhým krokem je nastavení výchozích hodnot pro jednotlivé inputy, pokud mají tuto hodnotu nastavenou. Duplikování v tomto případě není 1:1, ale zobrazujeme jakýsi výchozí stav jednotlivých elementů.

Posledním krokem je přidání tlačítek pro odeslání formuláře a zrušení zadávání, včetně přiřazení (*bind()*) akce po kliknutí na tlačítko pro odeslání či zrušení zadávání formuláře.

2.5 Grafické ovládací prvky

2.5.1 Layout stránky (rozvržení prvků na stránce)

Jako základ pro vytvoření designu aplikace jsem si vytvořil „drátové modely“ (dále wireframy) jednotlivých stránek, které se v aplikaci budou vyskytovat. Nástrojem, který jsem pro tvorbu wireframů použil, je online služba MockFlow[28]. Výsledné náhledy si můžete, včetně popisu, prohlédnout v příloze E.1.

Pro vytvoření grafického návrhu se na základě wireframů většinou přechází do některého z bitmapových grafických editorů, jakým je např. Adobe Photoshop. Zde grafik nakreslí design jednotlivých stránek, vzhled hlavičky, menu a základních prvků stránky. Poté grafický návrh dostane kodér, který návrh rozřeže a přepíše do HTML kódu, vzhled definuje za použití CSS. V tomto případě jsem ale tento „zavedený postup“ nedodržel. Vynechal jsem fázi kreslení grafického návrhu a design aplikace jsem vytvořil rovnou v rámci kódování celé aplikace. Rozvržení stránky, které bylo vytvořeno v jednotlivých wireframech, bylo dodrženo. Vynechat mezikrok v bitmapovém editoru jsem si mohl dovolit z důvodu, že pro vzhled *webGUI* nevyužívám (kromě ikonek) žádné obrázky, veškerá grafika je vytvořena v kódu. Nepotřeboval jsem si tudíž připravovat žádné obrázky ani layout stránky, který byl již jasně určen wireframy.

Layout stránky byl ve wireframech navržen tak, aby byla využita maximální plocha pro zobrazení informací. Nejdůležitější horní lišta (menu) je zobrazena ve fixní poloze tak, aby mohl mít uživatel stále zobrazeny všechny potřebné nástroje pro manipulaci a orientaci na stránce. Jednotlivé sekce (sloupce) jsou nezávisle *scrollovatelné* (pokud je zvoleno zobrazení ve dvou sloupcích), aby uživatel mohl mezi sebou co možná nejlépe porovnat výsledný obsah sloupců.

2.5.2 Výběr barev

Jednou z nejdůležitějších součástí jakéhokoliv *designu* je barevná paleta. Na zvolených barvách závisí výsledný dojem stránky na uživatele. Pomáhají také uživateli rychleji pochopit různé konotace nebo mohou zvýraznit důležité prvky na stránce. Ve článku „A Look Into Color Theory In Web Design“[29] [16] je zdůrazněno, že výběr barev by neměl probíhat bez jasného zdůvodnění, proč jsme se pro danou barvu rozhodli. Vhodný výběr barev je

[28] `http://www.mockflow.com`

[29] `http://sixrevisions.com/web_design/a-look-into-color-theory-in-web-design/`

tedy důležitý pro navození patřičného a hlavně zamýšleného dojmu na uživatele.

Existuje několik způsobů dělení barev. Ve výběru barev pro design webových stránek budu uvažovat dělení na teplé a studené barvy. Teplé barvy přinášejí uživateli emoce spojené s letními dny, dny dovolených a odpočinku — často jsou proto použity na stránkách poskytující služby, nabízející zážitky či cestování, nebo jen obsahující informace pro zlepšení nálady. Naopak studené barvy jsou využívány nejčastěji pro profesionální, seriózní, důvěryhodné a čistě vypadající stránky velkých korporací, bank, ale také často pro administrační rozhraní či systémového nastavení...

Každá skupina barev je vhodná pro jiný typ stránek a má jiný význam pro uživatele. Např. červená barva evokuje nebezpečí, chybu, zelená pak úspěch — uživatelé zde mohou nalézt souvislost např. se semaforem na silnici, kde červená znamená STÁT a zelená VOLNO. Toto je hlavní důvod, proč jsem zvolil tyto barvy, spolu s neutrální šedou, pro barvu textu v chybových, úspěšných či oznamovacích hláškách. Je možné, že nám tato volba přijde samozřejmá a naprosto jasná, ale někdy je potřeba se i nad tak banální volbou zamyslet a najít důvod PROČ danou kombinaci barev volíme.

Po zvážení možných barevných kombinací jsem se na základě poznatků ze zmíněného článku rozhodl pro světlé barevné schéma s primární studenou — modrou barvou. Finální výběr barev jsem provedl v online nástroji Kuler[30]. Seznam zvolených barev je uveden v příloze E.12.

2.5.3 Aplikace barev a výsledný vzhled

Po výběru barev přichází na řadu také volba písma. Jelikož bude *web-GUI* zobrazeno hlavně na monitorech počítačů, zvolil jsem základní bezpatkové písmo Arial. Oproti patkovému písmu je bezpatkové lépe čitelné právě na monitorech.

Mezi uživatelské prvky, které na stránce využívám, patří také různé ikonky. Jako základní sadu jsem si zvolil ikony, které jsou volně dostupné na adrese Entypo[31]. Tato sada je dostačující pro běžné akce jako odkaz domů, ikonka uživatele apod. Pro vlastní akce nad vypsaným XML stromem jsem si ale musel vytvořit ikonky vlastní — konkrétně se jedná o přidání, duplikování a mazání jednotlivých elementů stromu.

Barvy zvolené v nástroji Kuler jsem aplikoval tímto způsobem: Modrá barva slouží pro důležité orientační prvky stránky jako jsou nadpisy či navigace. Pro rozlišení jednotlivých sekcí stránky jsem vybral odstíny šedé. Pro

[30] `http://kuler.adobe.com`

[31] `http://entypo.com`

podpoření čistoty a realističnosti grafického návrhu jsem na některé prvky stránky aplikoval také základní efekty jako je vržený stín (odstíny šedi) či barevný přechod. Pro dostatečně kontrastní zobrazení ikonek v horním menu jsem zvolil bílou barvu.

Jako zpětná vazba pro uživatele je u každého prvku stránky, se kterým může nějakým způsobem pracovat, zobrazena grafická změna daného prvku po najetí kurzorem myši (tzv. *hover effect*). Zde využívám např. změnu průhlednosti daného prvku či změnu barvy pozadí apod.

Výsledný grafický návrh aplikace je přiložený v příloze E.2.

2.6 Kaskádové styly — nový a nestandardní přístup k zápisu

Jak jsem již zmínil v analýze, kompilované CSS (v mém případě SASS) umožňuje použít základní konstrukty programovacích jazyků také pro zápis CSS. Díky tomu mám možnost využívat, pro mě hlavně uživatelsky definované proměnné a funkce (v SASSu zvané *mixiny*), či některé základní knihovní matematické funkce a funkce pro práci s barvami.

Na začátku jsem si nadefinoval jednotlivé vybrané barvy do proměnných, se kterými jsem schopen dále pracovat. Jako další užitečné proměnné mohu uvést definici písma, jeho velikosti či různé velikosti (např. výška horního menu). Výhodou proměnných je možnost jednoduché konfigurace celého stylu — tedy změnou jedné proměnné dokážu změnit barvy na všech místech. Navíc proměnné mohou vstupovat jako argumenty mixinům či knihovním funkcím pro práci s barvami. S oblibou proto využívám např. mixiny pro vytvoření lineárních přechodů na pozadí prvků tím, že ztmavím (či zesvětlím) požadovanou barvu na jednom konci přechodu.

Co se týče syntaxe zápisu, dodržování základních pravidel nutí autora logicky slučovat a shlukovat kusy kódu, které se týkají stejných částí HTML kódu. Tím se stává zápis mnohem přehlednější pro čtení.

Zjednodušení zápisu a strukturalizace obdobně jako v HTML stromu mi pomohla pochopit a aplikovat také jeden z nových přístupů pro zápis CSS, tzv. OOCSS (Object-Oriented CSS). Jako příklad mohu uvést tlačítko — „*.button*“. To mohu pomocí dalších tříd různě modifikovat (např. zvětšit velikost písma), ale základ zůstává stále stejný — objektový přístup je zde takovýto: „*.button*“ je základní instance třídy, pomocí dalších tříd nastavuji této instanci jiné instanční vlastnosti (proměnné), což způsobí změnu konečného výpisu. Pro představu, dalším příkladem může být rozdělení sloupců (není implementováno v této práci). Zde jsem schopen pomocí

relativních jednotek šířky nastavit např. „*.column-30*“, „*.column-70*“ pro sloupce šířky 30%, resp. 70%. Takto nadefinované sloupce pro vytvoření layoutu můžu aplikovat dále kdekoliv na stránce. V rámci *webGUI* jsem si zvolil např. třídy „*.left*“ a „*.right*“ pro určení, zda má být prvek obtékán (*float*) zleva či zprava. Tyto třídy opět používám nezávisle na umístění daného HTML elementu v rámci stromové struktury.

2.7 Dokumentace kódu

Pro lepší orientaci ve stromové struktuře balíku *NetopeerBundle* jsem vytvořil pro všechny vlastní třídy a kontrolery programátorskou dokumentaci. Zdokumentovány jsou všechny soubory, metody a funkce, včetně vstupních a výstupních parametrů.

Pro vygenerování dokumentace jsem použil nástroj phpDocumentator 2[32]. Výsledná dokumentace je zobrazitelná v rámci *webGUI* na adrese */bundles/netopeer/docu/*.

[32] `http://www.phpdoc.org`

Kapitola 3

Testování grafického rozhraní

3.1 Manuální testování

V rámci vývoje bylo nutné provádět průběžné testovaní aktuálně implementovaných funkcí, hlavně pak práci s XML stromem pomocí JavaScriptu. Pro kontrolu správnosti výstupu PHP skriptů, které vytváří obsah zpráv posílaných protokolem NETCONF, jsem použil referenční výstup konzolové aplikace *netconf-client* (zmíněnou v sekci 1.2).

Veškeré důležité akce, které jsou v rámci *webGUI* vykonány (např. komunikace se serverem, načítání modelů...), jsou logovány pomocí *služby Monolog*, zmíněné v sekci 2.2. Jakmile nastane po provedení určité akce chyba, je možné získat z chybového logu sekvenci příkazů, které v chybu vyústily, včetně odeslaných dat. V konzolové aplikaci provedu stejnou sekvenci příkazů a zkontroluji shodu s odpovědí serveru. Tím dokážu zjistit, jestli se jedná o chybu způsobenou ve *webGUI* či o chybu nižších vrstev aplikace.

Manuální testování GUI není vždy jednoduché a je časově náročné testovat po jakékoliv změně celé chování aplikace. Je proto vhodné pouze pro testování určité, často právě implementované, části aplikace. Zjednodušení testování nabízí automatizované testy, které jsou popsány v následujících sekcích.

3.2 Automatizované testy pomocí frameworku Selenium

Zjednodušení testování správné funkčnosti *webGUI* nabízí automatizované testy. V přehledech vhodných nástrojů pro testování webových aplikací

[13] [8] je uveden jako vhodný nástroj pro testování GUI nebo jakékoliv webové aplikace framework SeleniumHQ[33]. Dle oficiálního popisu [21] se jedná o „nástroj pro automatizované testování webových aplikací, ale není limitován pouze pro ně — jednoduché úkony v administračním systému mohou (a měly by) být také otestovány“. Selenium nabízí podporu testování ve všech majoritních webových prohlížečích a je dostupné v mnoha programovacích jazycích, včetně PHP. Pro testování jsem zvolil základní verzi Selenia 1.0.

Pro automatizované testování pomocí Selenia je nutné mít spuštěný *Selenium server*. Jedná se o Java aplikaci, která je volně dostupná na adrese [34]. Samotné testy napsané v mém případě v programovacím jazyce PHP jsou spuštěny pomocí nástroje PHPUnit [35] (na adrese [35] je uveden také návod na instalaci a ukázky spuštění).

Testování jsem zpočátku prováděl lokálně na vlastním počítači. Časem jsem se se zadavatelem dohodl na vytvoření virtuálního stroje, na kterém budu moci tyto testy spouštět. Testy jsou díky tomu dostupné i ostatním programátorům pracujících na nižších vrstvách aplikace. Instalace a nastavení testovacího prostředí není nijak složité. Po nainstalování nezbytných nástrojů jako je PHP, PHPUnit, Java, prohlížeč Firefox a Google Chrome jsem pro zobrazení průběhů testů nainstaloval také VNCserver, na jehož obrazovce průběh testu (tedy otevřený prohlížeč) zobrazuji. Můžu tak kontrolovat samotný průběh testu (hlavně při chybě) také vizuálně.

Výstupem ze spuštěného testu je log průběhu testu a informace o úspěchu či neúspěchu testu do souboru. V případě neúspěchu je vypsána chybová hláška s informací, ve které části (na kterém řádku) testu byla chyba objevena. Díky tomu můžu při vizuální kontrole očekávat chybu na určitém místě a jednoduše ji také ve zdrojovém kódu testu najít. Podle druhu chyby mohu odhadnout, ve které části aplikace k chybě došlo a mohu z toho vyvodit další postupy opravy chyby — pokud jsem schopen chybu opravit sám v krátkém čase, chybu opravím ihned. Pokud se jedná o problém složitější, nahlásím chybu do interního nástroje pro hlášení chyb (*issue tracker*) a domluvím se s ostatními programátory na řešení.

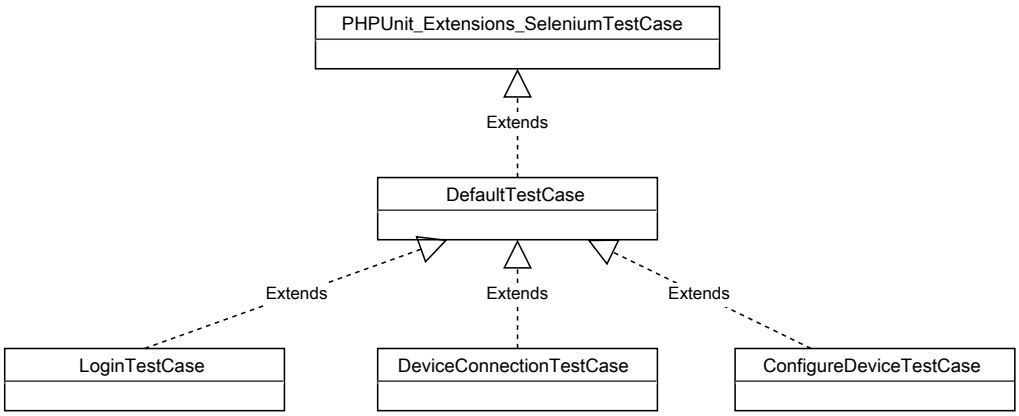
Spouštění testů probíhá manuálně po každé změně kódu (typicky v rámci atomické jednotky — úkolu či commitu). Díky tomu zjistím, zda má aktuálně naimplementovaná úprava neovlivnila jinou funkčnost v aplikaci. Po nasazení aplikace do provozu budou testy spuštěny automaticky pomocí úlohy naplánované pomocí Cronu či jiné služby tak, aby byla správná

[33] `http://docs.seleniumhq.org`
[34] `http://docs.seleniumhq.org/download/`
[35] `http://www.phpunit.de/manual/3.8/en/selenium.html`

funkčnost aplikace testována neustále.

3.3 Okruhy pokrytí automatizovanými testy

Pomocí Selenium testů jsem pokryl veškeré případy použití, které jsou v aplikaci implementovány. Testy jsem rozdělil do několika částí (samostatné třídy), aby bylo možné testy pouštět jak cíleně na určitý případ použití, tak hromadně v rámci testování celé aplikace. Výsledná struktura je zobrazena na obrázku 3.1.



Obrázek 3.1: Hierarchie tříd testů. *DefaultTestCase* je v tomto případě rodičovská třída pro vlastní testy, ve které nastavuji volbu prohlížeče a obecné metody, které používám v potomcích této třídy. Potomci jsou již samostatně funkční jednotky — samostatné testy.

LoginTestCase testuje přihlášení do aplikace a správnou autentizaci uživatele. V testu zkouším různé kombinace neexistujícího jména a hesla a očekávám zamítnutí přístupu. Jako poslední se zkusím přihlásit s existujícími a správnými údaji a otestuji následné odhlášení z aplikace.

DeviceConnectionTestCase testuje připojení se k zařízení včetně načítání modelů, práci s historií a profily zařízení, přípojení se k více zařízením najednou a postupné odhlášení se od zařízení.

ConfigureDeviceTestCase testuje správné rozdělení horního menu do modulů, levého sloupce do sekcí, zobrazení jedno- a dvou-sloupcového layoutu, provedení změn ve vypsaném XML stromě — editace hodnot, duplikování uzlů, mazání uzlů či zamykání a odemykání úložiště.

Kontrola zamykání a odemykání úložiště probíhá tak, že se připojím dvakrát ke stejnému zařízení. V prvním připojení zamknu zařízení a pokusím se jej editovat z druhého připojení. Očekávám chybu, protože tato operace by neměla být povolena. Vyzkouším také, jestli je možné zamknout druhé připojení, opět by mělo vyústit v chybu. Poté odemknu první připojení a zkusím konfigurovat zařízení — nyní by mělo proběhnout bez problémů.

Kontrola testovaných kroků probíhá nejčastěji pomocí kontroly shody zobrazených chybových/úspěšných hlášek nebo elementů s předpisem definovaným v testu.

3.4 Nasazení aplikace a průběh testování

Aplikace je prozatím nasazena na dvou serverech CESNETu. Jeden je používán jako vývojový a druhý se stabilní verzí *webGUI*. Celé *webGUI* pravidelně vydáváno v podobě RPM balíku.

Z obou serverů je možné komunikovat s libovolným NETCONF serverem a testovat tak správné chování aplikace. Automatické testy jsou nyní spouštěny oproti vývojové verzi z důvodu stálé kontroly správné funkčnosti celé aplikace. Po aktualizování nového balíku na stabilním serveru je tento server otestován také.

Celá práce na *webGUI* je od začátku vývoje verzovaná pomocí verzovacího nástroje GIT[36]. Jednotlivé úkoly jsou propojeny s interním „bug tracking“ systémem, ve kterém může vedoucí i zadavatel práce lépe kontrolovat průběh vývoje.

3.5 Testování GUI

Testování grafického rozhraní probíhalo ve dvou fázích — průběžný test použitelnosti a heuristické vyhodnocení po implementaci požadavků této práce. Pro testování grafického rozhraní jsem definoval personu jako uživatele, který zná princip fungování protokolu NETCONF a již někdy konfiguroval NETCONF zařízení z prostředí příkazové řádky — zná tedy jazyk YANG a jeho možnosti.

Jelikož těchto expertů vhodných pro testování není mnoho, testování probíhalo a v rámci vývoje stále probíhá na vzorku čtyř osob (včetně mě). Do okruhu testovaných lidí patří zaměstnanci CESNETu, kteří spadají do kritérií určené persony. Do testování se v průběhu vývoje na chvíli za-

[36] `http://git-scm.com`

pojili ještě cca 2 lidé, kteří měli možnost s *webGUI* pracovat — ty jsem pozoroval při ukázce funkčnosti aplikace.

3.5.1 Testování použitelnosti

Testování použitelnosti *webGUI* bylo poněkud rozsáhlejší a probíhalo od začátku vývoje aplikace průběžně. Nebylo tedy celou dobu cílené, ale vyplynulo z reálného používání aplikace. Zpětnou vazbu a připomínky jsem obdržel buď v rámci pozorování činností experta, typicky při prezentaci nové funkčnosti aplikace nebo jako zpětnou vazbu při práci experta s *webGUI*. V průběhu se objevily některé nedostatky, které popíši včetně řešení níže.

3.5.1.1 Chybové a úspěšné hlášky

Původní rozmístění úspěšných/chybových hlášek bylo pro uživatele při používání aplikace nepříjemné. Hláška se totiž objevila vždy v horní části sloupce jako reakce po provedení akce uživatelem. Hláška posunula po zobrazení celý obsah sloupce dolů. Pokud se jednalo o hlášku úspěšnou, po pěti sekundách zmizela. To způsobilo opět posun celého obsahu zpět nahoru, což bylo pro uživatele nepříjemné. Negativní odezvu na toto chování jsem obdržel u více testovaných osob.

Řešení: Navrhl jsem jiný způsob zobrazování hlášek. Hlášky se zobrazují (vyjíždějí) v pravém horním rohu okna prohlížeče. Pozice hlášky je fixní. Jakmile se zobrazí více hlášek, seskupí se pod sebe. Úspěšná hláška, oproti chybové, automaticky po několika sekundách zmizí. Pokud se jedná o hlášku chybovou, je v okně zobrazena do té doby, dokud uživatel neklikne na křížek pro skrytí hlášky. Výsledek si můžete prohlédnou na obrázku E.11, nebo jako porovnání na obrázcích D.1 a D.2.

V určité situaci při editaci hodnot XML stromu došlo ke zmatení uživatele tím, že se zobrazila jak úspěšná, tak chybová hláška. První hláška byla s informací, že byl formulář úspěšně odeslán a zpracován, ale přitom se hned vzápětí zobrazila hláška chybová. Ta obsahovala informaci, že došlo k chybě při zpracování. Uživatel byl zmaten tím, že si nemohl být jistý, která informace je pravdivá a zda úprava byla aplikována či nikoliv.

Řešení: Na základě tohoto podnětu jsem objevil tři místa v aplikaci, kde k takovýmto chybám může dojít a opravil jsem logiku zobrazování hlášek. K situaci, že se objeví dvě sobě odporující hlášky, by již nemělo dojít.

3.5.1.2 Nevýraznost aktivní části navigace

Některé testované osoby uvedly, že není vždy jasné, ve kterém modulu se nachází.

Řešení: Zvolil jsem lepší grafické zvýraznění aktuálně vybraného modulu v horním menu aplikace — kromě změny barvy jsem přidal také vystupující šipku směrem dolů. Výsledek si můžete prohlédnou na obrázcích D.3 a D.4.

3.5.1.3 Grafický výpis stromu

Výpis stromu byl zpočátku nepřehledný — jednotlivé úrovně byly pouze horizontálně odsazeny zleva, při zobrazení delšího stromu nebylo jasné, ve které úrovní se konkrétní uzel nachází a jakého má rodiče.

Řešení: Pro zvýšení přehlednosti jsem kromě horizontálního odsazení zleva pro každou úroveň přidal také vertikální čáru znázorňující jednu úroveň zanoření. Při spočítání zobrazených vertikálních čar od levé hrany výpisu tak dokáže uživatel snadněji určit hloubku zanoření a vizuálně kontrolovat umístění jednotlivých uzlů při posouvání obsahu stromu (*scrollování*). Výsledek si můžete prohlédnout na obrázcích D.5 a D.6.

3.5.2 Heuristické vyhodnocení

V rámci předmětu BI-TUR (Tvorba uživatelského rozhraní) jsem postupně *webGUI* testoval metodou heuristického vyhodnocení. Jako experty pro vyhodnocení tohoto průchodu jsem zvolil vedoucího práce a dva další programátory pracující na nižších vrstvách aplikace.

Oproti plnému výčtu otázek Nielsenovy heuristiky [37] jsem seznam upravil vzhledem k použitelnosti na tomto konkrétním projektu (seznam je uveden v příloze D.1.1). Testované experty jsem požádal, aby prošli všechny typové stránky *webGUI* (jejich seznam je uveden v příloze D.1.2) a zodpověděli mi definované otázky.

3.5.2.1 Vyhodnocení testu experty

Testovaní experti odpovídali na výše uvedené otázky samostatně a bez mé kontroly. Do emailu jsem popsal metodu testování, na co se mají zaměřit (otázky), čeho mají docílit a jaký má být jejich výstup. Musím poznamenat, že výsledky tohoto testu pocházejí z doby po opravách chyb, které

[37] `http://www.usability.gov/methods/test_refine/heuristic.html`

vyšly z průběžného testování použitelnosti. Jejich výsledky jsou uvedeny v příloze D.1.3 (v tabulkách D.1, D.2 a D.3).

3.5.2.2 Zhodnocení výsledků testu

Heuristické vyhodnocení jednotlivých částí *webGUI* třemi experty bych označil za úspěšné. Testování proběhlo po opravách funkčnosti a vylepšeních z průběžného testování použitelnosti. Výsledky ukázaly u každého problému shodu minimálně dvou expertů, expert 3 dokonce objevil nedostatků více. Problémy, které experti objevili naštěstí nejsou kritické, jejich opravení ale určitě zpříjemní uživatelům práci s *webGUI*.

V reakci na objevené problémy jsem navrhl tyto kroky:

- upravit animaci zobrazení hlášek, aby nezasahovaly do tabulky s připojenými zařízeními,
- rozdělit záznam v historii připojených zařízení do dvou řádků a přidat výpis všech parametrů spojení jako titulek odkazu,
- opravit špatné chování tlačítka ZPĚT po úspěšném přihlášení — nyní není možné se dostat zpět na přihlašovací formulář,
- přidat popisek pro povinné položky formuláře pro připojení se k zařízení,
- přidat tlačítko „Zobrazit všechny sekce“ do levého menu.

Problém je ale se stálým zobrazováním posuvníků (*scrollbarů*), i když je obsah nižší a tedy není co posouvat. Prohlížeč totiž tyto posuvníky zobrazuje vždy, kdy má HTML element nastavenou vlastnost *overflow: scroll;*. Řešením je proto jedině kontrola výšky obsahu pomocí JS. Pokud bude obsah vyšší, nastavíme vlastnost *overflow*, jinak ponechám element nezměněný.

Závěr

Vývoj webového grafického uživatelského rozhraní mě posunul opět dále a přinesl mi jednu velkou zkušenost — práci na projektu, který komunikuje s několika dalšími nižšími vrstvami a je pomyslnou špičkou tohoto ledovce, tedy úrovní nejvyšší. Doposud jsem měl zkušenosti s tvorbou webových prezentací či elektronických obchodů (*e-shop*), které až na některé výjimky nebyly závislé na jiných službách, ani na komunikaci s dalšími oddělenými systémy.

Tato práce mi umožnila v praxi uplatnit převážnou část znalostí, které jsem v průběhu studia nabyl. Měl jsem možnost zdokonalit se v práci s frameworkem Symfony, aktivně využívat Ajaxové požadavky, skriptovací jazyk a knihovnu jQuery, vzdálenou práci na serveru a jeho konfiguraci, intenzivní využití příkazové řádky či verzování souborů. Důležitou zkušeností byla možnost být součástí takto rozsáhlého projektu, který byl realizován ve spolupráci s CESNETem. Mohl jsem sledovat jeho průběh a řízení, účastnit se pravidelně schůzek, porad a konzultací vývoje v rámci týmu vývojového oddělení CESNETu.

Shrnutí průběhu a výsledků práce, vlastní přínos bakalářské práce

Prvním nutným krokem pro začátek práce na tvorbě *webGUI* bylo prostudování RFC týkajících se protokolu NETCONF, datovým modelů YANG a pochopení hierarchie a funkčnosti jednotlivých vrstev nutných pro komunikaci s NETCONF serverem.

Před samotnou implementací práce bylo nutné zjistit požadavky na výslednou aplikaci od uživatelů z praxe. Byla provedena analýza možných

způsobů řešení a zvolena optimální cesta.

V samotné implementaci jsem měl volnost v návrhu funkčnosti, takže jsem si vyzkoušel např. použití návrhového vzoru Dependency Injection nebo logické rozdělení jednotlivých částí podle návrhového vzoru MVC. Zajímavá byla analýza výběru barev a jejich vliv na výsledný dojem.

Testování aplikace přineslo zajímavé výsledky a objevilo mnohé nedostatky. Poprvé jsem se zabýval testováním s reálnými uživateli, tedy testováním použitelnosti a heuristickým vyhodnocením. Zpětná vazba od uživatelů byla velmi přínosná. Odstranění nalezených chyb zpříjemnilo celkový zážitek uživatelů při používání *webGUI*.

Má práce přinesla doposud chybějící webové grafické uživatelské rozhraní pro správu a konfiguraci zařízení komunikujících pomocí protokolu NETCONF. Veškeré požadavky, které byly z řad uživatelů (a v zadání) definovány, byly splněny. Implementovaná aplikace je již hotová a funkční. Splnění ostatních cílů práce, hlavně týkajících se spokojenosti uživatelů, bude jasné až rozsáhlejším nasazení a používání aplikace.

Budoucí práce

V této sekci je shrnuto plánované pokračování vývoje *webGUI* nad rámec původních cílů, protože odevzdáním bakalářské práce vývoj *webGUI* nekončí. Pokračování vývoje ve spolupráci s CESNETem a další práce na *webGUI* je naplánována v blízké době v tomto rozsahu:

1. zobrazení notifikací odeslaných NETOPEER serverem, komunikace pomocí WebSocketů,
2. uživatelsky volitelné řazení položek v elementech *leaf-list* a *list* (RFC 6020),
3. vytvoření nového uzlu (včetně výběru možných hodnot definovaných v modelu),
4. zálohování/obnova konfigurace zařízení,
5. změna aktuálního *datastore* (running/startup/candidate),
6. implementace akce <copy-config> (kopírování mezi jednotlivými *datastore*),
7. uživatelsky definované konfigurační datové modely (nahrávání modelů v případě, že zařízením není podporováno <get-schema>),

8. autentizace za použití IdP - přihlášení pomocí EduId a zobrazení demo verze pro nepřihlášeného uživatele,
9. uživatelská konfigurace vlastností *webGUI* (nastavení *webGUI*),
10. ukládání historie příkazů, které uživatel provedl,
11. textové pole pro zadávání hromadného seznamu příkazů, které *webGUI* vykoná.

Další náměty a požadavky na vylepšení by měly přicházet s implementací dalších funkčností na nižších vrstvách aplikace či jako reakce na připomínky uživatelů.

Do budoucna by se *webGUI* mělo stát plnohodnotným nástrojem pro správu síťových zařízení v rámci práce síťových operátorů CESNETu. Časem by mělo být dostupné i dalším uživatelům.

Literatura

[1] Bernard, B.: Úvod do architektury MVC - Zdroják. Duben 2009, [Online; přístup dne 29. 1. 2013]. Dostupné z: http://www.zdrojak.cz/clanky/uvod-do-architektury-mvc/

[2] Bjorklund, M.: YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF). RFC 6020 (Proposed Standard), Říjen 2010, [Online; přístup dne 30. 10. 2012]. Dostupné z: http://www.ietf.org/rfc/rfc6020.txt

[3] Bříza, P.: Kompletní průvodce XSLT - úvod do problematiky. Březen 2004, [Online; přístup dne 13. 2. 2013]. Dostupné z: http://interval.cz/clanky/kompletni-pruvodce-xslt-uvod-do-problematiky/

[4] Cesnet TMC group: Netopeer. [Online; přístup dne 11. 3. 2013]. Dostupné z: https://www.liberouter.org/technologies/netconf/

[5] Cesnet TMC Group: libnetconf - the NETCONF library in C. 2012, [Online; přístup dne 23. 1. 2013]. Dostupné z: https://code.google.com/p/libnetconf/

[6] Chan, K.: Choose the right PHP framework. Prosinec 2012, [Online; přístup dne 8. 3. 2013]. Dostupné z: http://www.netmagazine.com/features/choose-right-php-framework

[7] Davis, J.: What the difference between PHP's DOM and simpleXML extensions? Leden 2011, [Online; přístup dne 11. 3. 2013]. Dostupné z: http://stackoverflow.com/a/4817550

[8] DreamCSS.com: 22 best GUI testing tools for developer. [Online; přístup dne 16. 4. 2013]. Dostupné z: http://blog.dreamcss.com/tools/gui-testing-tools/

[9] Enns, R.: NETCONF Configuration Protocol. RFC 4741 (Proposed Standard), Prosinec 2006, [Online; přístup dne 26. 4. 2013]. Dostupné z: http://www.ietf.org/rfc/rfc4741.txt

[10] Enns, R.; Bjorklund, M.; Schoenwaelder, J.; aj.: Network Configuration Protocol (NETCONF). RFC 6241 (Proposed Standard), Červen 2011, [Online; přístup dne 30. 10. 2012]. Dostupné z: http://www.ietf.org/rfc/rfc6241.txt

[11] Fowler, M.: Inversion of Control Containers and the Dependency Injection pattern. Leden 2004, [Online; přístup dne 27. 2. 2013]. Dostupné z: http://martinfowler.com/articles/injection.html

[12] Garrett, J. J.: Ajax: A New Approach to Web Applications. Únor 2005, [Online; přístup dne 11. 3. 2013]. Dostupné z: http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications

[13] Jain, S.: 10 Best Tools for Test Automation. Červen 2011, [Online; přístup dne 16. 4. 2013]. Dostupné z: http://www.toolsjournal.com/articles/item/195-10-best-tools-for-test-automation

[14] Lhotka, L.: Konfigurace síťových zařízení pomocí protokolu NETCONF. Duben 2012, [Online; přístup dne 30. 10. 2012]. Dostupné z: http://www.root.cz/clanky/konfigurace-sitovych-zarizeni-pomoci-protokolu-netconf/

[15] MG-SOFT NETCONF Browser Professional Edition. 2012, [Online; přístup dne 30. 10. 2012]. Dostupné z: http://www.mg-soft.si/mgNetConfBrowser.html

[16] Noack, S.: A Look into Color Theory in Web Design. Březen 2010, [Online; přístup dne 2. 4. 2013]. Dostupné z: http://sixrevisions.com/web_design/a-look-into-color-theory-in-web-design/

[17] Oheim, G.: What the difference between PHP's DOM and simpleXML extensions? Leden 2011, [Online; přístup dne 11. 3. 2013]. Dostupné z: http://stackoverflow.com/a/4803264

[18] PHPFrameworks.com: Top 10 Ranking PHP Frameworks. [Online; přístup dne 8. 3. 2013]. Dostupné z: http://www.phpframeworks.com/top-10-php-frameworks/

[19] Purchart, V.: Seriál: Jak na Dependency Injection. Červen 2011, [Online; přístup dne 27. 2. 2013]. Dostupné z: http://www.zdrojak.cz/serialy/jak-na-dependency-injection/

[20] Scott, M.; Bjorklund, M.: YANG Module for NETCONF Monitoring. RFC 6022 (Proposed Standard), Říjen 2010, [Online; přístup dne 27. 2. 2013]. Dostupné z: http://www.ietf.org/rfc/rfc6022.txt

[21] SeleniumHQ.org: SeleniumHQ Browser Automation. [Online; přístup dne 16. 4. 2013]. Dostupné z: http://docs.seleniumhq.org

[22] 3. Configuration — Doctrine DBAL 2.1.0 documentation. [Online; přístup dne 11. 3. 2013]. Dostupné z: http://docs.doctrine-project.org/projects/doctrine-dbal/en/latest/reference/configuration.html

[23] Creating and using Templates. [Online; přístup dne 13. 2. 2013]. Dostupné z: http://symfony.com/doc/current/book/templating.html

[24] How to use Best Practices for Structuring Bundles (2.0) - Symfony. [Online; přístup dne 26. 1. 2013]. Dostupné z: http://symfony.com/doc/2.0/cookbook/bundles/best_practices.html

[25] Hight Performance PHP Framework for Web Development - Symfony. 2012, [Online; přístup dne 23. 1. 2013]. Dostupné z: http://symfony.com

[26] PHP: Namespaces overview - Manual. Leden 2013, [Online; přístup dne 29. 1. 2013]. Dostupné z: http://www.php.net/manual/en/language.namespaces.rationale.php

[27] PHP: SimpleXML Manual - Introduction. Leden 2013, [Online; přístup dne 25. 1. 2013]. Dostupné z: http://www.php.net/manual/en/intro.simplexml.php

[28] JavaScript Web APIs. [Online; přístup dne 24. 2. 2013]. Dostupné z: http://www.w3.org/standards/webdesign/script.html

[29] XSL Transformations (XSLT). Listopad 1999, [Online; přístup dne 13. 2. 2013]. Dostupné z: http://www.w3.org/TR/xslt

[30] Document Object Model (DOM). Leden 2009, [Online; přístup dne 24. 2. 2013]. Dostupné z: http://www.w3.org/DOM/

[31] HTML: The Markup Language (an HTML language reference). Říjen 2012, [Online; přístup dne 13. 2. 2013]. Dostupné z: http://www.w3.org/TR/html-markup/Overview.html

[32] Cascading Style Sheets. Únor 2013, [Online; přístup dne 24. 2. 2013]. Dostupné z: http://www.w3.org/Style/CSS/

[33] YANG-Based Unified Modular Automation. 2012, [Online; přístup dne 30. 10. 2012]. Dostupné z: http://www.yumaworks.com/products/overview/

[34] YumaPro. 2012, [Online; přístup dne 20. 2. 2013]. Dostupné z: http://www.yumaworks.com/products/yumapro/

[35] YumaWorks. 2012, [Online; přístup dne 30. 10. 2012]. Dostupné z: http://www.yumaworks.com

Příloha A

Seznam použitých zkratek

API Application Programming Interface

CSS Cascading Style Sheets

DI Dependency Injection

GUI Graphical user interface

JS JavaScript

JSON JavaScript Object Notation

HTML HyperText Markup Language

MVC Model-View-Controller

ORM Object-relational mapping

OOCSS Object-Oriented CSS

PHP Hypertext Preprocessor

RPM Red Hat Package Manager

SASS Syntactically Awesome Stylesheets

UI User interface

UX User experience

XML Extensible markup language

XSLT eXtensible Stylesheet Language Transformations

Příloha **B**

Obsah přiloženého CD

- `docu` ..... dokumentace zdrojového kódu
- `preview` ..... náhledy aplikace použité v příloze
- `readme.txt` ..... stručný popis obsahu CD
- `RPM` ..... adresář s RPM balíkem pro instalaci
- `src` ..... zdrojové kódy implementace
- `text` ..... text práce
  - `BP_Alexa_David_2013.tex` zdrojová forma práce ve formátu LaTeX
  - `BP_Alexa_David_2013.pdf` ..... text práce ve formátu PDF

Příloha C

Instalační manuál

Instalační manuál byl sepsán a otestován na Scientific Linux [38] distribuci, která se používá na vývojových serverech CESNETu a na které je také *web-GUI* aktuálně nasazeno. Podobnou instalaci je možné provést na některé jiné distribuci, která umí pracovat s RPM balíky (např. Fedora). Před samotnou instalací je nutné do YUM konfigurace přidat repositář CESNETu. Jeho obsah je uveden na výpisu C.1.

```
[Liberouter-devel]
name=Liberouter Devel - Tools for Monitoring and ↙
    ↳ Configuration
baseurl=http://homeproj.cesnet.cz/rpm/liberouter/↙
    ↳ devel/$basearch
enabled=1
gpgcheck=1
gpgkey=http://homeproj.cesnet.cz/rpm/liberouter/RPM-↙
    ↳ GPG-KEY-liberouter
```

Výpis C.1: Konfigurace repozitáře CESNETu v souboru */etc/yum.repos.d/liberouter.repo*.

C.1 Seznam závislostí

WebGUI je instalováno z RPM balíku, ve kterém jsou definovány potřebné závislosti. Níže je uveden jejich seznam:

[38] `http://ftp1.scientificlinux.org/linux/scientific/livecd/63/x86_64/`

- httpd
- php
- php-xml
- mod_netconf
- python
- php-pdo
- php-process

V balíku není definována jedna externí závislost, program *PYANG*[39]. RPM balík *PYANGu* neexistuje, proto jej není možné nastavit jako závislost pro naše RPM a bude jej nutné nainstalovat ručně před vlastní instalací RPM.

C.2 Instalace

Před instalací RPM balíku je nutné nainstalovat program *PYANG*:

```
$ wget http://pyang.googlecode.com/files/pyang-1.2.↙
  ↳ tar.gz
$ tar -xf pyang-1.2.tar.gz
$ cd pyang-1.2
$ sudo python setup.py install
```

Pro instalaci *webGUI* ze souborů na přiloženém CD proveďte následující kroky:

- zkopírujte soubory ze složky *RPM* na CD do lokálního adresáře
- jako *root* spusťte:

  ```
  $ yum install ./*.rpm
  ```

Nastavte správnou časovou zónu v souboru */etc/php.ini*:

```
date.timezone "timezone"
```

„*timezone*“ je jedna z podporovaných *časových zón* uvedených na stránce `http://php.net/manual/en/timezones.php`.

[39] `http://code.google.com/p/pyang/`

C.3 Spuštění aplikace

WebGUI se nainstalovalo do adresáře */var/www/netconfwebgui*. Spuštění *webGUI* provedete otevřením vašeho webového prohlížeče a zadáním adresy `http://localhost/netconfwebgui/` (případně místo *localhost* zvolte název vašeho serveru).

C.4 Možné problémy

Pokud po pokusu o připojení se k zařízení obdržíte tuto formu chybové hlášky „Connection could not be established / Unable to connect to the server (Permission denied) / Connecting NETCONF server failed.“, zkontrolujte nastavení SELinux spuštěním jako *root*:

```
$ /usr/sbin/setsebool httpd_can_network_connect=1
```

Příloha D

Testování GUI

D.1 Testování GUI — heuristické vyhodnocení

D.1.1 Seznam otázek definovaných pro heuristické vyhodnocení

1. Ví vždy uživatel, na kterém místě aplikace se zrovna nachází?
2. Jsou zvolené popisky, nadpisy a texty srozumitelné?
3. Následuje po každé akci adekvátní reakce? (např. zobrazení informace o úspěchu/neúspěchu akce)
4. Fungují tlačítka zpět a vpřed v prohlížeči správně? Nedochází k opětovnému odeslání formuláře? Dokáže se uživatel jednoduše dostat na úvodní stránku (rozcestník)?
5. Jsou všechny ovládací prvky vidět vždy, kdy jsou potřeba?
6. Odpovídá vzhled předpokladům uživatele? Neobsahuje aplikace některé nečekané či rušivé grafické prvky?

D.1.2 Seznam typových stránek pro heuristické vyhodnocení

1. přihlášení
2. připojení se k zařízení

3. práce s historií a profily zařízení
4. konfigurace zařízení
5. provedení editace, duplikace a smazání uzlu
6. průchod jednotlivými moduly a sekcemi
7. odhlášení se od zařízení a z aplikace

D.1.3 Výsledky heuristického vyhodnocení expertů

Výsledky jednotlivých expertů jsou uvedeny v tabulkách D.1, D.2 a D.3.

Stránka/akce	**Výsledek**
1. přihlášení	Vše v pořádku
2. připojení k zařízení	Vše v pořádku
3. práce s historií	*Otázka 5:* Levý sloupec je příliš úzký na to, aby bylo možné zobrazit více informací o zařízení.
4. konfigurace zařízení	Vše v pořádku
5. editace, duplikace…	Vše v pořádku
6. průchod moduly	Vše v pořádku
7. odhlášení	Vše v pořádku

Tabulka D.1: **Expert 1:** *Ing. Tomáš Čejka*, vedoucí práce, programátor nižších vrstev

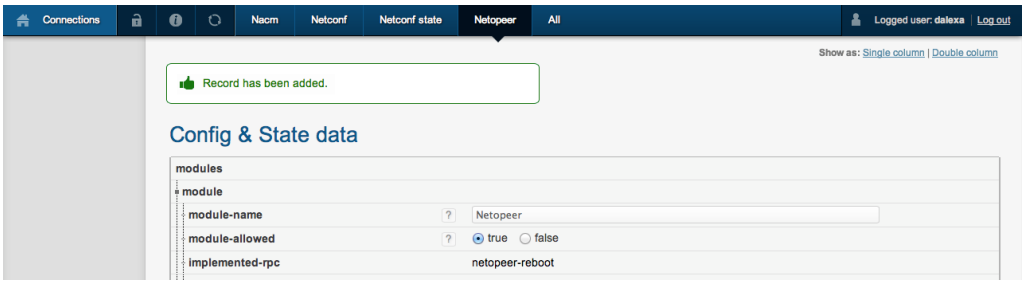
Stránka/akce	**Výsledek**
1. přihlášení	Vše v pořádku
2. připojení k zařízení	*Otázka 5:* Oznámení o (ne)úspěchu překrývá na chvíli prvky, které chci použít.
3. práce s historií	*Otázka 5:* Uvítal bych zobrazení, ve které by byly vidět všechny parametry spojení.
4. konfigurace zařízení	Vše v pořádku
5. editace, duplikace...	Vše v pořádku
6. průchod moduly	Vše v pořádku
7. odhlášení	Vše v pořádku

Tabulka D.2: **Expert 2:** *David Kupka*, programátor nižších vrstev

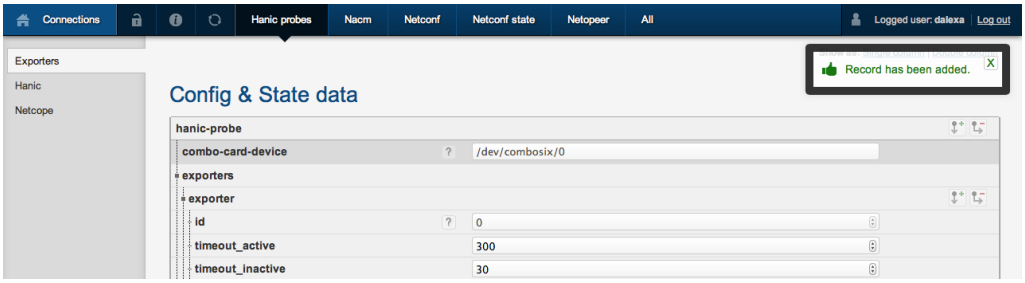
Stránka/akce	**Výsledek**
1. přihlášení	*Otázka 4:* Po úspěšném přihlášení a použití tlačítka ZPĚT se objeví prázdná stránka.
2. připojení k zařízení	*Otázka 2:* U polí formuláře pro připojení k serveru je u každého popisku hvězdička, ale nikde není uvedeno vysvětlení. *Otázka 5:* Okénko se zprávou překrývá odkaz na konfiguraci zařízení. *Otázka 6:* Posuvníky (*scrollbary*) jsou zobrazeny, i když není co posouvat.
3. práce s historií	Chyby jsou shodné s předchozí otázkou.
4. konfigurace zařízení	Vše v pořádku
5. editace, duplikace...	Vše v pořádku
6. průchod moduly	*Otázka 5:* Sekce v levém sloupci neobsahují tlačítko „Zobrazit všechny sekce“.
7. odhlášení	Vše v pořádku

Tabulka D.3: **Expert 3:** *RNDr. Radek Krejčí*, programátor nižších vrstev

D.2 Testování GUI — grafické úpravy



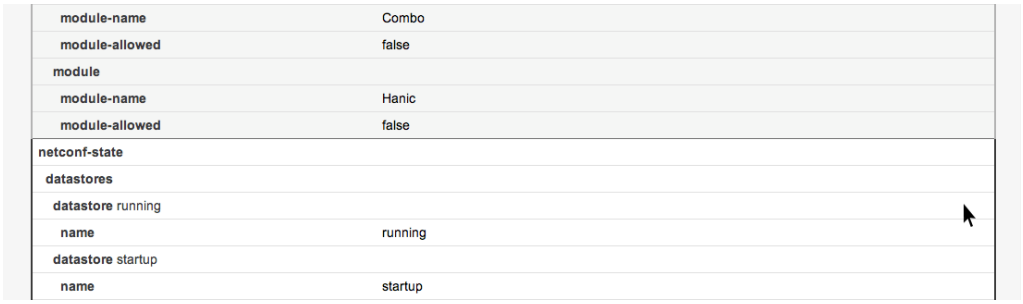
Obrázek D.1: Původní zobrazení chybových a úspěšných hlášek.



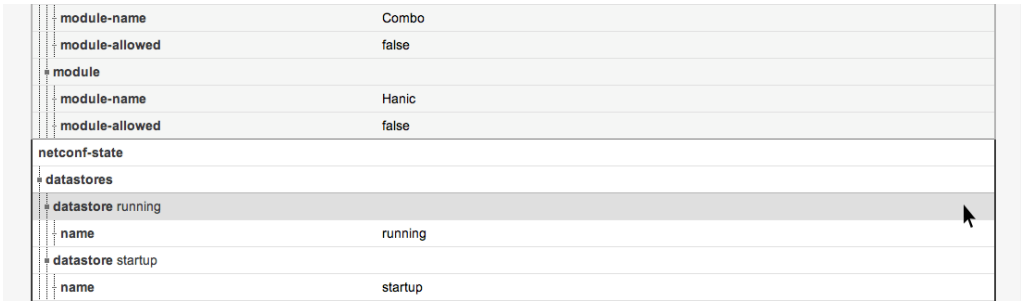
Obrázek D.2: Nové zobrazení chybových a úspěšných hlášek. Hlášky jsou zobrazeny ve fixní poloze a nedochází tedy k posunu obsahu.

Obrázek D.3: Původní vzhled aktivní položky horního menu.

Obrázek D.4: Nový vzhled aktivní položky horního menu. Ztmavil jsem pozadí aktivní položky a přidal navíc šipku směřující na obsah.



Obrázek D.5: Původní zobrazení výpisu XML stromu, pouze s horizontálním odsazením zleva.

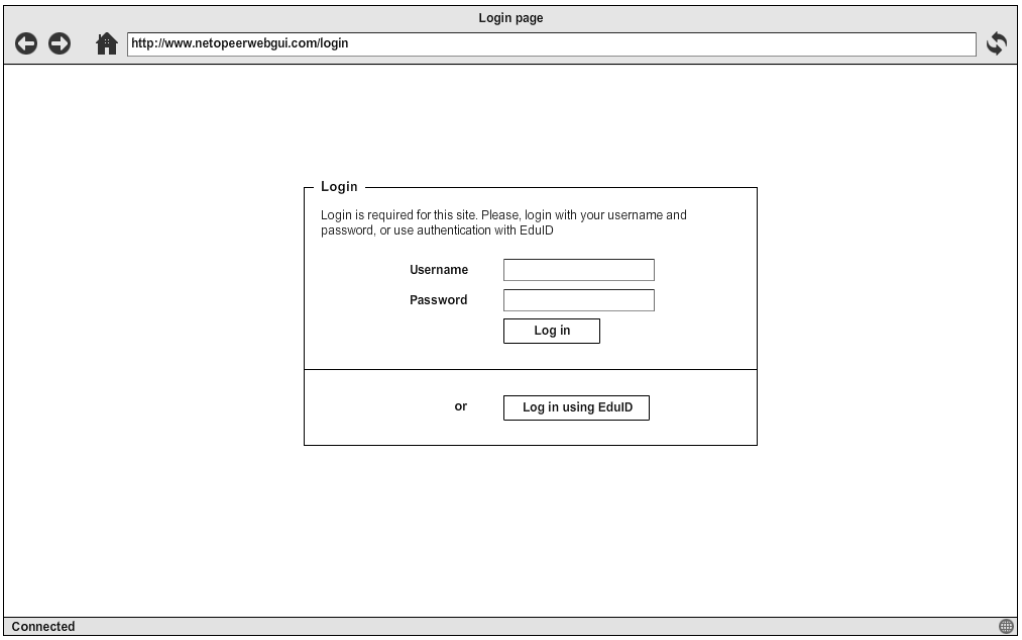


Obrázek D.6: Nové zobrazení výpisu XML stromu, kde jsou k horizontálnímu odsazení zleva přidány další grafické prvky pro lepší přehlednost. Ke každému řádku tabulky jsem přidal také podbarvení jako reakci na přejetí myší (*hover*) nad řádkem.

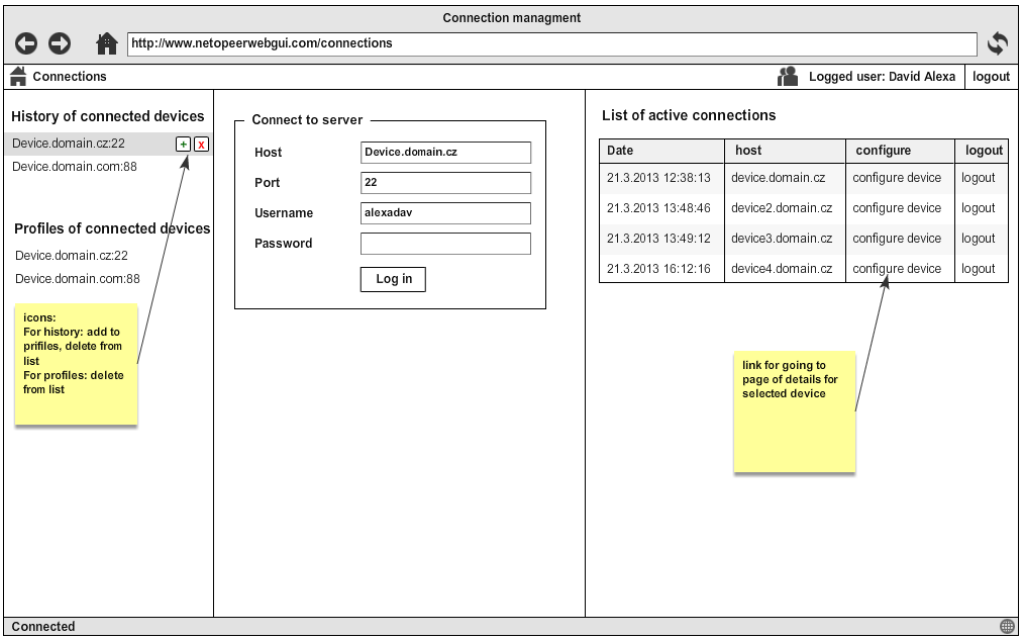
Příloha E

Obrázkové přílohy

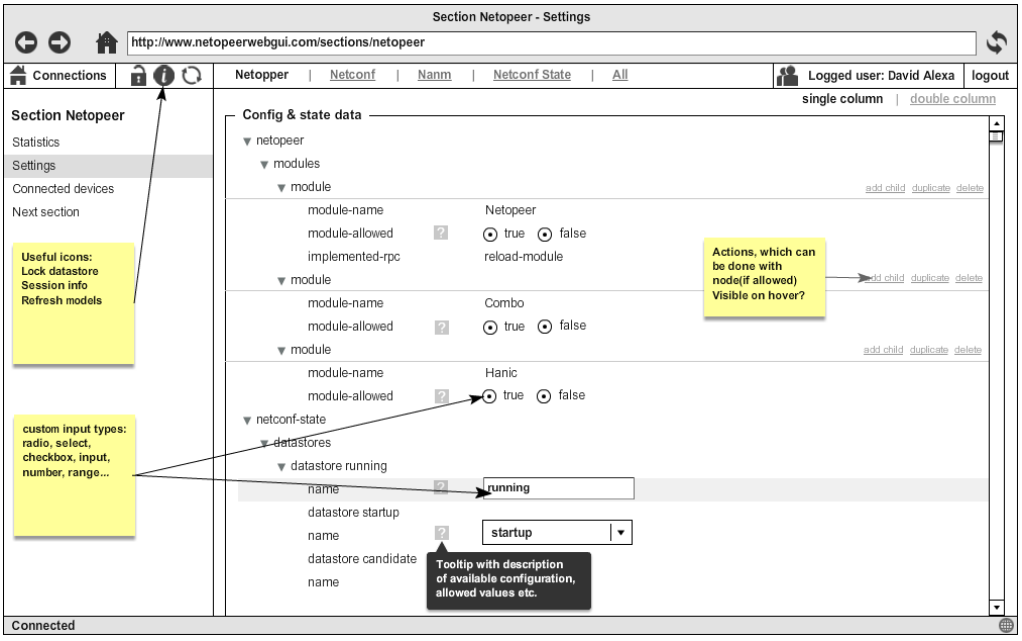
E.1 Wireframy použité jako základ návrhu designu



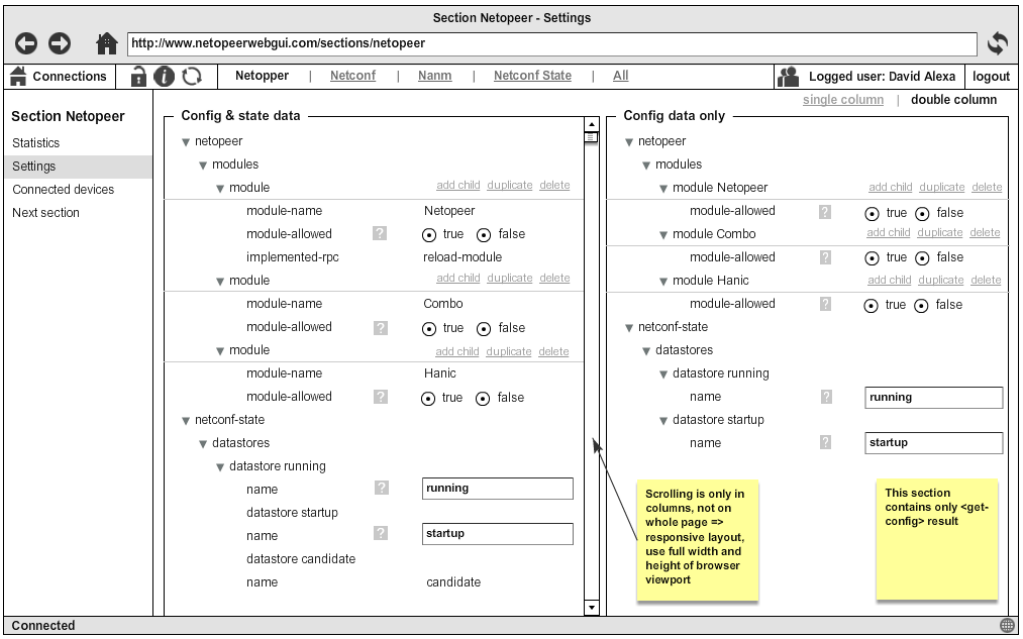
Obrázek E.1: Přihlašovací stránka pro vstup do aplikace. Nepřihlášený uživatel nemá právo zobrazit jakoukoliv jinou stránku před tím, než se autentifikuje.



Obrázek E.2: Pokud proběhne přihlášení uživatele úspěšně, zobrazíme rozcestník pro práci s aplikací. Ten obsahuje historii připojených zařízení a na stálo uložené zařízení (nezávisle na historii). Následuje formulář pro připojení k zařízení a seznam aktuálně připojených zařízení v pravém sloupci. U každého řádku pravé tabulky je možné konfigurovat zařízení nebo se od něj odpojit.

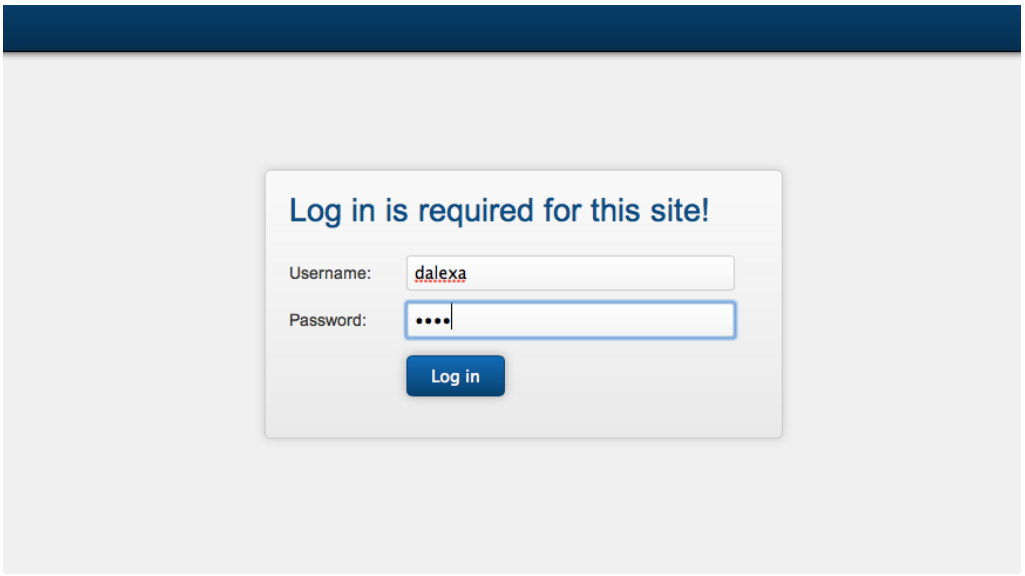


Obrázek E.3: Zobrazení jednotlivých modulů a sekcí daného zařízeni. Horní menu bylo doplněno o seznam modulů, které jsou u zařízení k dispozici. Levý sloupec obsahuje seznam sekcí zvoleného modulu. V obsahu je zobrazen XML strom po zavolání požadavku *<get>*. XML strom je obohacen informacemi získanými z modelu — např. popis některých elementů nebo vhodné formulářové prvky.

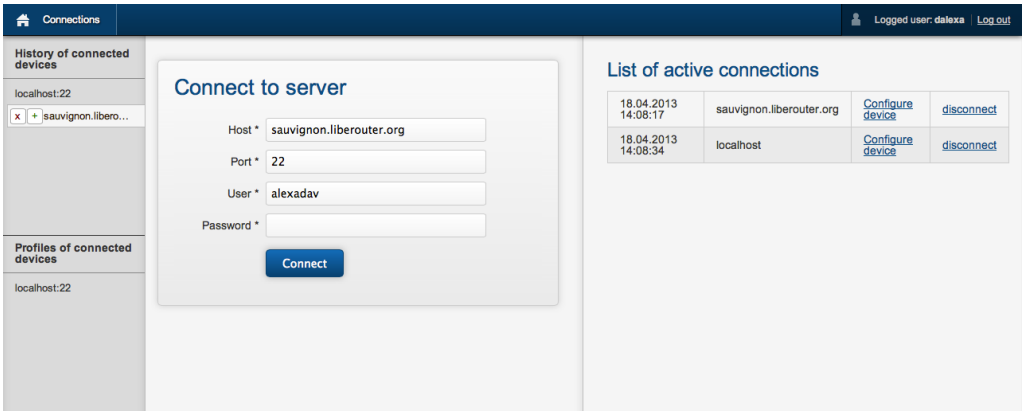


Obrázek E.4: Obdobný výpis, jaký je uveden na obrázku E.3. Rozdíl je ve způsobu výpisu odpovědi serveru, kdy je v levém sloupci zobrazena modifikovaná odpověď *<get>*, v pravém pak *<get-config>*.

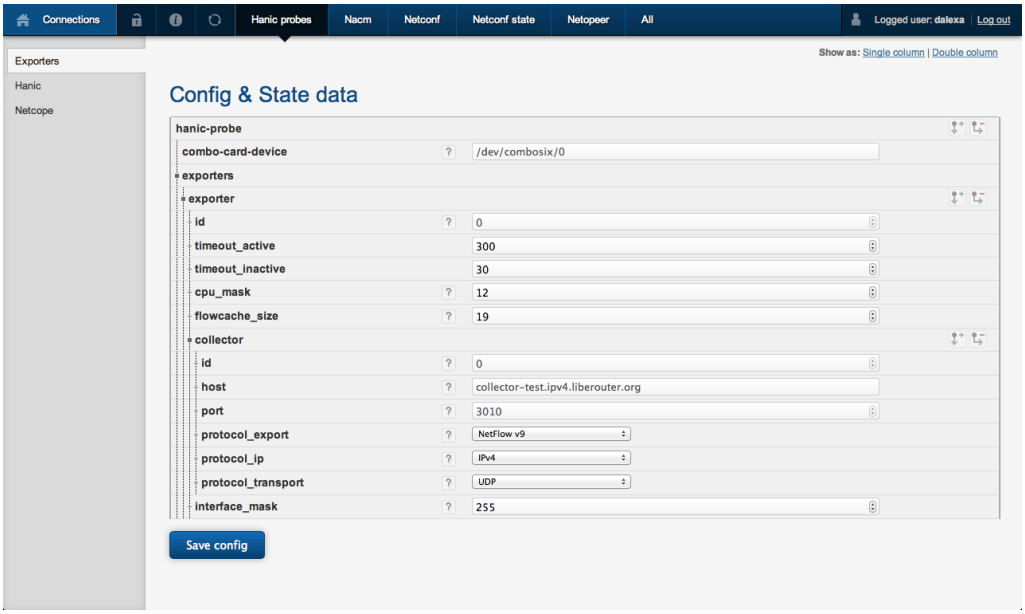
E.2 Výsledné náhledy designu *webGUI*



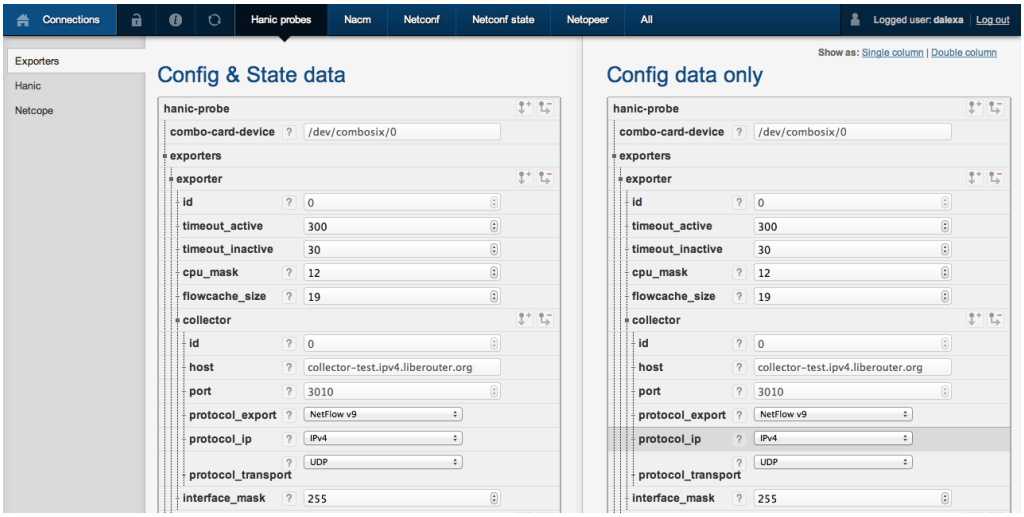
Obrázek E.5: Přihlašovací stránka pro vstup do aplikace. Vychází z wireframu uvedeného na obrázku E.1.



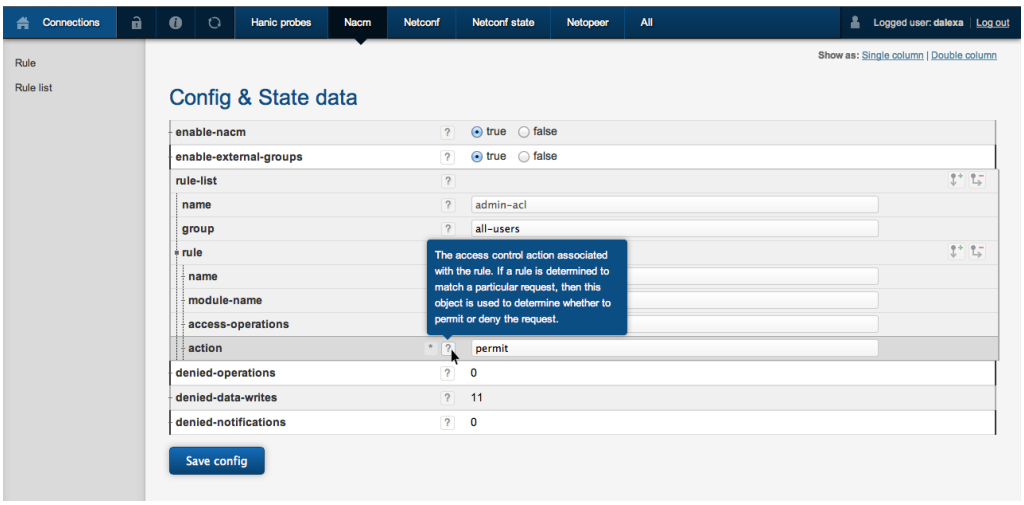
Obrázek E.6: Rozcestník pro práci s aplikací, včetně historie připojených zařízení, formuláře pro připojení k zařízení a seznam aktuálně připojených zařízení. Odpovídá wireframu uvedeného na obrázku E.2.



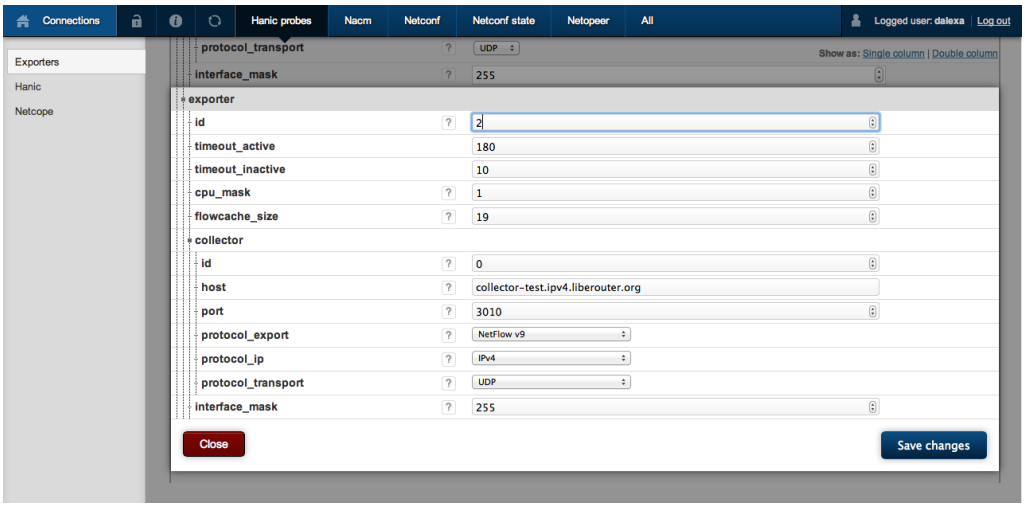
Obrázek E.7: Zobrazení modulů a sekcí v jedno-sloupcovém layoutu. Odpovídá wireframu uvedenému na obrázku E.3.



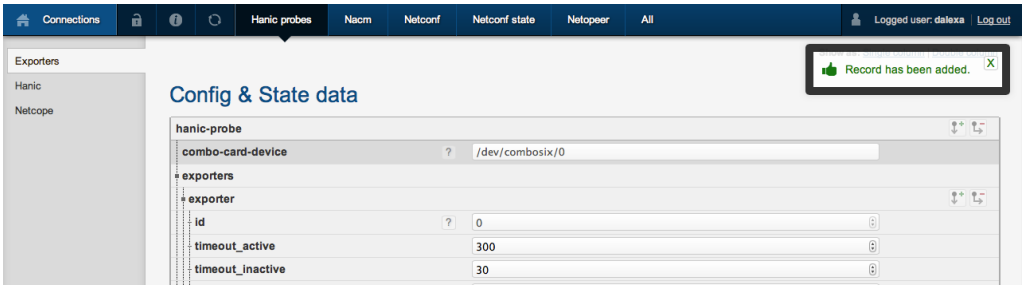
Obrázek E.8: Na obrázku je zobrazen dvousloupcový layout, kdy je v levém sloupci zobrazena modifikovaná odpověď *<get>*, v pravém pak *<get-config>*. Odpovídá wireframu z obrázku E.4.



Obrázek E.9: Zobrazení nápovědy k danému uzlu. Nápověda obsahuje popis vlastností daného uzlu a jeho význam.

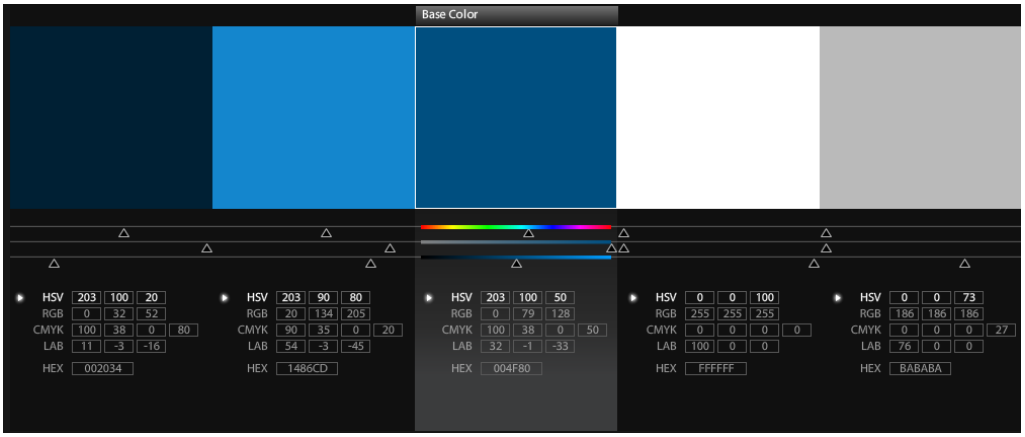


Obrázek E.10: Formulář pro vytváření nových uzlů. Konkrétně je zde zobrazen formulář pro duplikování uzlu, který má předvyplněné výchozí hodnoty. Formulář je zobrazen jako modální okno. Více informací o vytváření nových uzlů je uvedeno v sekci 2.4.2.2.



Obrázek E.11: Umístění úspěšné hlášky v pravém horním rohu. Pokud má být zobrazeno více hlášek najednou, seskupí se postupně pod sebe. Úspěšné hlášky se po několika sekundách automaticky schovají, chybové zůstanou, dokud je uživatel sám nezavře kliknutím na křížek.

E.3 Výběr barev



Obrázek E.12: Vytvoření barevné palety pomocí on-line nástroje Kuler, který je dostupný také jako plugin do programu Adobe Photoshop. Výsledné barvy jsou (v pořadí zleva doprava): tmavší barva — **#002034**, světlejší barva — **#1486CD**, základní barva — **#004F80**, světlá barva pozadí — **#FFFFFF**, barva pozadí — **#BABABA**.