BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF COMPUTER SYSTEMS
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

AUTOMATED INFERENCE OF USER INTERFACE FOR NETCONF PROTOCOL

AUTOMATIZOVANÉ ODVOZENÍ UŽIVATELSKÉHO ROZHRANÍ PRO PROTOKOL NETCONF

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR — **JAKUB MAN**
AUTOR PRÁCE

SUPERVISOR — **Ing. PETER TISOVČÍK**
VEDOUCÍ PRÁCE

BRNO 2020

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačových systémů (UPSY) Akademický rok 2019/2020

Zadání bakalářské práce

Student: **Man Jakub**
Program: Informační technologie
Název: **Automatizované odvození uživatelského rozhraní pro protokol NETCONF**
Automated Inference of User Interface for NETCONF Protocol
Kategorie: Počítačové sítě
Zadání:

1. Seznamte se s vývojovými nástroji relevantními pro konfigurační protokol NETCONF.
2. Nastudujte potřebné nástroje pro tvorbu moderní webové aplikace (např. Angular 2 a vhodné knihovny).
3. Navrhněte automatizované zobrazení dat, které administrátor vyžaduje během konfigurace síťových zařízení.
4. Navržené řešení implementujte.
5. Implementaci ověřte na vzorových příkladech konfigurace, které navrhne pedagogický vedoucí.
6. Diskutujte dosažené výsledky a navrhněte další rozšíření.

Literatura:

- Dle pokynů vedoucího.

Pro udělení zápočtu za první semestr je požadováno:

- Splnění bodů 1 až 3 zadání.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/
Vedoucí práce: **Tisovčík Peter, Ing.**
Vedoucí ústavu: Sekanina Lukáš, prof. Ing., Ph.D.
Datum zadání: 1. listopadu 2019
Datum odevzdání: 28. května 2020
Datum schválení: 25. října 2019

Abstract

This thesis focuses on creating a user-friendly user interface for devices that use the NETCONF protocol for configuration. A web user interface was selected to make using the user interface simple and for it to not require any additional installation. In this thesis, a user interface with a plugin architecture and with a NETCONF library was created. The NETCONF library allows users to connect devices, save them, group devices in profiles and to modify the device's configuration. The plugin architecture allows device manufacturers to provide a graphical user interface that does not require additional user training, without having to develop a whole new application.

Abstrakt

Tato práce se zaměřuje na vytvoření uživatelsky přívětivého grafického rozhraní pro zařízení komunikující protokolem NETCONF. Bylo zvoleno řešení pomocí webového rozhraní, aby bylo použití co nejjednodušší a z pohledu uživatele nevyžadovalo další instalaci. V práci bylo vytvořeno uživatelské rozhraní obsahující systém pro rozšíření a knihovnu pro operace nad protokolem NETCONF. Knihovna pro NETCONF umožňuje uživatelům připojení k zařízením, uložení informací o zařízení do databáze, připojování ke skupinám zařízení pomocí profilů a modifikace konfiugrace zařízení. Systém rozšíření umožňuje výrobcům zařízení poskytnout uživatelům grafické rozhraní, které nevyžaduje další zaškolování uživatelů, aniž by museli vyvýjet celou aplikaci.

Keywords

NETCONF, network devices, configuration, liberouter, YANG, user interface

Klíčová slova

NETCONF, síťová zařízení, konfigurace síťových zařízení, liberouter, YANG, uživatelské rozhraní

Reference

MAN, Jakub. *Automated Inference of User Interface for NETCONF Protocol.* Brno, 2020. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Peter Tisovčík

Rozšířený abstrakt

Počítačové sítě se stále rozrůstají a obsahují stále více zařízení, které je třeba konfigurovat a monitorovat. Kvůli tomu je náročnější nastavovat nové sítě a monitorovat stávající sítě ručně.

Protokol SNMP byl vyvinut pro monitorování a správu sítí. Tento protokol je stále používán pro monitorování sítí, ale pro konfiguraci sítí byl ve většině případů opuštěn. Protokol SNMP je složitý a neobsahuje standardní proces, jenž automaticky vyhledá správné informační moduly používané konkrétním zařízením. Další nevýhodou protokolu SNMP je jeho použití protokolu UDP pro přenos informací, čímž je možná ztráta dat při přenosu. Programování skriptů pomocí rozhraní příkazové řádky jendotlivých zařízení je možným řešením automatizace procesu konfigurace. Psaní skriptů pro konfiguraci je relativně jednoduché, ale velmi náročné na údržbu a úpravy.

Nutnost řešení snadné konfigurace rozsáhlých sítí vedla ke vzniku modelem řízené správy sítí. Důležitou výhodou modelem řízené správy sítí je oddělení modelů, protokolů a kódování, což umožňuje snadné přidání nových protokolů a kódování ke stávajícím modelům. Pro přenos dat v modelem řízené správě sítí lze použít protokol NETCONF. Modelovací jazyk YANG definuje data přenášená tímto protokolem.

Cílem této práce je vytvořit uživatelské rozhraní zjednodušující práci se zařízeními, které podporují protokol NETCONF. Uživatelské rozhraní bude poskytovat uživatelsky přívětivý způsob konfigurace zařízení, bez nutnosti znalosti protokolu NETCONF nebo modelovacího jazyku YANG. Uživatelské rozhraní bude dále poskytovat programové rozhraní pro vývojáře, čímž umožní vývoj dalších nástrojů pro práci s konfigurací zařízení. Existující řešení pro správu zařízení pomocí protokolu NETCONF jsou často proprietární, nebo nejsou udržované. Uživatelské rozhraní popsané v této práci bude vytvořeno jako volně dostupný software s otevřeným zdrojovým kódem (open-source). Proto všechny použité knihovny musejí být volně šiřitelné.

Implementace protokolu NETCONF, *libnetconf2*, byla zvolena pro konfiguraci zařízení. Pro správu YANG schémat je použita knihovna *libyang*. Obě tyto knihovny jsou volně šiřitelné. Pro vývoj uživatelského rozhraní bylo použito prostředí *liberouter GUI*, které využívá technologii Angular pro grafické uživatelské rozhraní a programovací jazyk Python pro zpracování informací na serveru.

Vyvinuté uživatelské rozhraní umožňuje uživatelům vytvářet profily obsahující skupiny zařízení, pro jednoduché připojení několika zařízení najednou. Poskytuje jednoduché připojení nových zařízení a uložení informací o zařízení do databáze. Dále uživatelské rozhraní umožňuje načítání a úpravu konfigurace zařízení pomocí protokolu NETCONF a nahrávání a procházení YANG schémat. Uživatelské rozhraní poskytuje programové rozhraní umožňující vývojářům přidávat rozšíření pro zjednodušení práce s konfigurací. Implementace systému pro načítání rozšíření byla inspirována článkem, jehož autorem je Alexey Zuev [18]. Rozšíření jsou načítána za běhu. To umožňuje přidávat další rozšíření bez nutnosti překládat celou aplikaci. Všechny knihovny sdílené mezi rozšířeními a základní aplikací jsou v rozšířeních pouze odkazovány, nedochází tedy k duplicitě kódu knihoven.

Automated Inference of User Interface for NETCONF Protocol

Declaration

I hereby declare that this thesis is my original authorial work, which I have worked out on my own under the leadership of Ing. Peter Tisovčík. All sources, references, and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

.....
Jakub Man
May 27, 2020

Acknowledgements

I would like to thank my supervisor Ing. Peter Tisovčík and RNDr. Radek Krejčí from the CESNET company for a lot of valuable help and their feedback.

Contents

List of Figures

Chapter 1

Introduction

There is a visible trend, that networks are getting much larger, with more and more devices that need to be configured and monitored. This means, that networks are much harder to monitor and configure manually. There is a need for automating the process of setting up new networks and monitoring them.

The Simple Network Management Protocol (SNMP) was developed to be used for both monitoring and management. While it is still being used for monitoring, it has been mostly abandoned for management purposes. It is complex and lacks a standard automatic discovery process, that finds the correct Management information base (MIB) modules, the device is using. Another disadvantage of the SNMP protocol is the use of the UDP protocol where the messages can be lost.

Device's Command-line interface (CLI) scripting can be used to automate the configuration process. Writing scripts to configure networks is relatively easy to do, but very hard to maintain. Because of that, data model-driven management replaced scripting in large network monitoring and configuration. An important advantage of data model-driven management is the separation of models, protocols, and encoding, which means it is easier to add protocols and encodings [10]. Data model-driven management was initially built on the NETCONF protocol, described in chapter 2.1. The YANG data modeling language is used to define the data sent over this protocol. This modeling language is described in chapter 2.2.

The goal of this thesis is to provide a simple to use and extensible User Interface (UI) to configure devices using the NETCONF protocol. The UI should provide a user-friendly way to configure devices without having to know how to use YANG or NETCONF. The UI should also provide a way for developers to extend its functionality with plugins so that developers can easily extend this UI with a device-specific configuration interface. Currently, most implementations of the NETCONF protocol are distributed as proprietary software or are not maintained. The implementation described in this thesis will be developed as free and open-source software, therefore all libraries utilized to handle the NETCONF protocol operations must be free and open-source as well. NETCONF implementation *libnetconf2* will be used to handle the configuration itself. *Libnetconf2* is an open-source software developed by CESNET. *Libnetconf2* uses the open-source library *libyang* to handle YANG data models.

The implementation described in this thesis will build on and extend the existing UI for NETCONF, also developed by CESNET, called Netopeer2GUI. Netopeer2GUI can connect to devices using *libnetconf* and provides a way to display and edit configuration and display data models. However, it is not beginner-friendly as it requires the user to know how to use

YANG. In this thesis, Netopeer2GUI will be improved to display configuration in a user-friendly format and to support external plugins to make the configuration of any device easier. A notification system will also be implemented to display messages from a server to the user in real-time. A way to display notification logs will also be implemented.

Chapter 2 describes technologies and libraries that will be used to implement the new UI. This chapter also explains, why certain libraries was selected. Chapter 3 describes how the UI was designed and explains the requirements for the UI. In chapter 3.4, Graphical User Interface (GUI) design mockup is introduced. Chapter 4 mentions the most important parts of the UI and describes their implementation. This chapter also describes the differences between the design from chapter 3 and the actual implementation. Chapter 4.2 describes, how the plugins to manipulate device's configuration are implemented and what can the tools do. Chapter 5 focuses on testing. Testing methods are described, and some test cases are introduced. In the final chapter, the whole thesis is concluded. The differences between the design and the implementation are discussed. This chapter also mentions plans for future work for the project described in this thesis.

Chapter 2

Used Technologies

The project described in this thesis will be implemented as a part of the CESNET toolkit for network device management. Because of that, it will use an existing implementation of libraries made by and/or maintained by CESNET. Specifically, the implementation of the NETCONF protocol, *libnetconf2*, and implementation of the YANG data modelling language, *libyang*.

The Liberouter GUI framework will be used for the UI as it implements user authorization and authentication and can be extended with new functionality using modules. Liberouter GUI is built on Angular framework and uses Python with Flask as its backend. Liberouter GUI is a web application framework, meaning that the UI will be available for anyone on any device, without having to install anything. The only requirement of using the UI is a device with an internet connection and a web browser. This provides users with the ability to configure and monitor their network from anywhere, even from their smartphone.

In this chapter, both Netconf and YANG will be described. Liberouter GUI will also be introduced.

2.1 NETCONF

The Network Configuration Protocol (NETCONF) is a protocol standardized as RFC 4741 [17] and later updated to a new version as RFC 6241 [11].

The NETCONF protocol defines a simple mechanism through which a network device can be managed, configuration data information can be retrieved, and new configuration data can be uploaded and manipulated. The protocol allows the device to expose a full, formal application programming interface (API). Applications can use this API to send and receive full or partial configuration data sets. A key aspect of NETCONF is that it allows the functionality of the management protocol to closely mirror the native functionality of the device.

NETCONF sends messages over the SSH protocol, so the communication between the client and the server is always encrypted.

The NETCONF protocol uses a Remote Procedure Call (RPC) paradigm to facilitate communication between a client and a server. The client can be a script or an application, typically running as part of a network manager. The server is usually a network device. RPC is similar to a function call. A client can pass arguments to remote procedures. A client encodes RPC in XML and sends it to a server using a secure session. The caller

then waits for a response from the remote procedure. A server responds with a reply encoded in XML[11].

Operations

The NETCONF protocol provides a small set of low-level operations to manage device configurations and retrieve device state information. The base protocol provides operations to retrieve, configure, copy, and delete configuration datastores. Additional operations are provided, based on the capabilities advertised by the device. Additional RPC operations can be defined and implemented [11]. The basic operations are:

- **get**: Retrieve running confiruration and device state information.
- **get-config**: Retrieve part or all of specified confiruration datastore.
- **edit-config**: Edit a configuration datastore by creating, deleting, merging or replacing content. If the target configuration datastore does not exist, it will be created.
- **copy-config**: Create or replace an entire configuration datastore with the contents of another complete configuration datastore. If the target datastore exists, it is overwritten. Otherwise, a new one is created, if allowed.
- **delete-config**: Delete a configuration datastore. The <running> configuration datastore cannot be deleted.
- **lock**: Allows the client to lock the entire configuration datastore system of a device. Locks are intended to be short-lived and allow client to make a change without fear of interaction with other clients or human users. Only one lock can exist on a device at a time.
- **unlock**: Releases configuration lock obtained with previous lock operation.
- **close-session**: Request graceful termination of NETCONF session.
- **kill-session**: Force the termination of NETCONF session. Aborts any operations that are currently in process.

A protocol operation can fail for various reasons, including `operation not supported`. An initiator should not assume that any operation will always succeed. The return values in any RPC reply should be checked for error responses [11].

Datastores

A datastore is a conceptual place to store and access information. A datastore might be implemented, for example, using files, a database, flash memory locations or their combinations. A datastore maps to a YANG data tree.

Data that can be retrieved from a device are divided into two categories: configuration and state data. Configuration data are writeable and can be modified, while the state data are read-only and usually contain information about the device, like CPU usage, disk usage, number of running processes et cetera.

NETCONF specifies three conventional datastores as of RFC 6241: running, candidate and startup [11]. The conventional datastores are a set of datastores, that share the same

schema, allowing data to be copied between them. Future standards may define other conventional configuration datastores. The original model of the conventional datastores as it is currently used by NETCONF is in figure 2.1. The operational state is not considered a datastore, although there were proposals in the past to introduce an operational state datastore [5]. The conventional datastores serve the following purposes:

- **Startup**: Data representing device configuration set after device system start.
- **Running**: A configuration datastore holding the complete configuration currently active on a device.
- **Candidate**: Data representing device configuration, that the device supports and therefore can be set. A `<commit>` operation will cause the device's running configuration to be set to the value of candidate configuration.

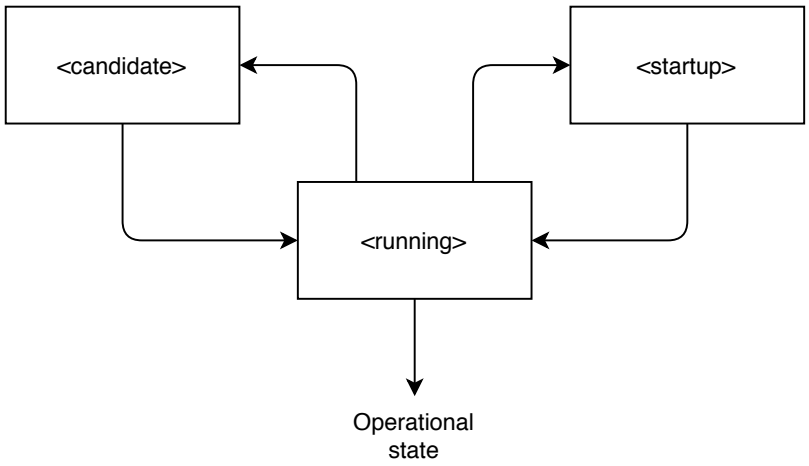

Figure 2.1: The original model of datastores as it is currently used by NETCONF [5]

Datastores are an optional feature, so they might not be supported on all devices. Schema extensions that modify datastores can be implemented, for example, the extensions might allow writing directly into the running datastore, which is not permitted by default.

An example of a `get-config` RPC call to get the running datastore and server response with the data element is in the listing 2.1. The `message-id` attribute of the RPC response must match the message-id of the request. An error is returned if an `<rpc>` element is received without a `message-id` attribute. The data element in the response contains the requested configuration data.

Capabilities

NETCONF allows a client to discover the set of protocol extensions supported by a server. These capabilities allow the client to adjust its behavior to take advantage of the features exposed by the device. A NETCONF capability is a set of functionality that extends the base NETCONF specification. A capability is identified by a uniform resource identifier (URI).

Capabilities augment the base functionality of the device, describing both additional operations and the content allowed inside operations. The client can discover the server's capabilities and use any additional operations, parameters, and content defined by those

```
Request:
  <rpc message-id="101"
      xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
     <get-config>
       <source>
         <running/>
       </source>
     </get-config>
   </rpc>

Response:
  <rpc-reply message-id="101"
      xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
     <data>
     ...
     </data>
  </rpc-reply>
```

Listing 2.1: Example of the `get-config` RPC call [11]

capabilities. Additional capabilities can be defined at any time in external documents, allowing the set of capabilities to expand over time.

After a NETCONF session is established, a client and a server immediately exchange Hello messages to each other. A Hello message from the client contains a set of capabilities, that are supported locally. If both ends support a capability, they can implement special management functions based on this capability. The capability negotiation result depends on the capability set on the server-side for standard capabilities while it depends on the capabilities that both ends support for extended capabilities [13]. A NETCONF server can send a `<hello>` element to advertise the capabilities that it supports. This exchange is shown in figure 2.2.

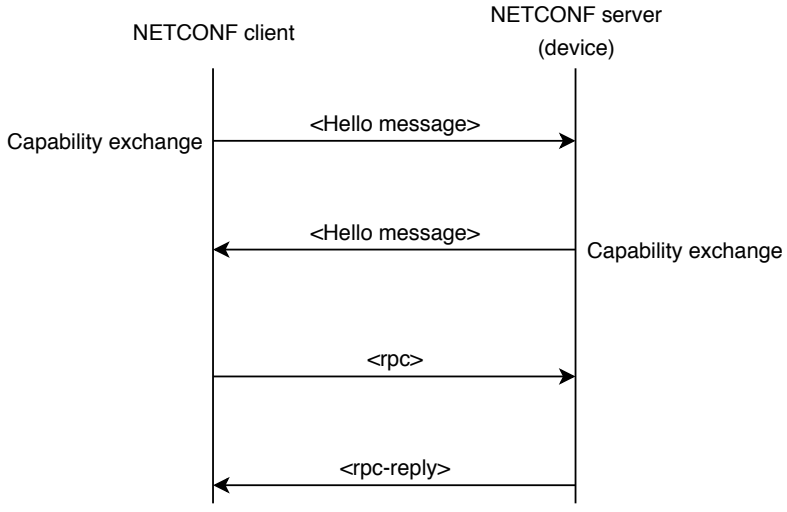

Figure 2.2: Example of `<hello>` element sent by the NETCONF server and clinet

The `<hello>` message also announces YANG modules, that the server implements. YANG modules from YANG version 1.0 are announced as a capability, directly in the `<hello>` message. Format of the NETCONF capability of a YANG 1.0 module is in listing 2.2. The `revision-date` is the revision of the module that the NETCONF server implements, `module-name` is the name of a module as it appears in the "module" statement, `namespace-uri` is the namespace URI for the module as it appears in the `namespace` statement, `feature` is the name of an optional feature implemented by the device, and `deviation` is the name of a module defining device deviations.

```
capability-string = namespace-uri [ parameter-list ]
parameter-list = "?" parameter *( "&" parameter )
parameter = revision-parameter /
                    module-parameter /
                    feature-parameter /
                    deviation-parameter
revision-parameter = "revision=" revision-date
module-parameter = "module=" module-name
feature-parameter = "features=" feature *( "," feature )
deviation-parameter = "deviations=" deviation *( "," deviation )
```

Listing 2.2: Format of a YANG 1.0 module in the NETCONF capability format [3]

Since YANG 1.1, the server may only advertise its yang version in the `<hello>` message. A NETCONF server must announce the modules it implements by implementing the YANG module `ietf-yang-library` and listing all implemented modules in the `/modules-state/module` list.

The server also must advertise the capability from the listing 2.3 in the `<hello>` message.

```
urn:ietf:params:netconf:capability:yang-library:1.0?
    revision=<date>&module-set-id=<id>
```

Listing 2.3: The NETCONF capability of a YANG 1.1 module [4]

The parameter `revision` has the same value as the revision date of the `ietf-yang-library` module implemented by the server. The parameter `module-set-id` has the same value as the leaf `/modules-state/module-set-id` from `ietf-yang-library`. With this mechanism, a client can cache the supported modules for a server and only update the cache if the `module-set-id` value in the `<hello>` message changes [4].

Libnetconf2

The *libnetconf2* is an open-source library for building NETCONF clients and servers developed by CESNET in C programming language. It provides a Python wrapper, therefore it can be used in a Liberouter GUI module. It handles NETCONF authentication and all NETCONF RPC communication, both client-side and server-side.

Libnetconf2 uses the *libyang* library for handling YANG models. The *libyang* library is described later in this thesis, in chapter 2.2. *Libnetconf2* is a continuation of the *libnetconf* library. *Libnetconf2* was reworked, using experiences gained from developing *libnetconf*. The older version, *libnetconf*, uses *libxml2* to represent schema and data tress. *Libxml2* is intended for different purposes, therefore it is less efficient at working with YANG schemas and data. *libnetconf* also implements configuration datastores besides NETCONF transport. Datastores are not a part of *libnetconf2* [6].

```
module a {
   yang-version 1.1;
   namespace "urn:example:a";
   prefix "a";

   container cont {
       leaf-list domain-search {
           type string;
           description
           "List of domain names to search.";
       }
   }
}
```

Listing 2.4: YANG leaf-list example [4]

Other implementations

Many other implementations for various programming languages are available, for example, *Net-netconf* for Ruby, *NetconfX* for Java, or *ncclient* for Python. *Libnetconf2* will be used in this thesis as it provides a Python wrapper and runs fast because it is implemented in the C language. *Libnetconf2* is also a free and open-source library, but most importantly, it is still being maintained and updated by CESNET. Many alternatives to *libnetconf2* do not provide a Python wrapper, therefore they can not be used in a Liberouter GUI module.

2.2 YANG

YANG is a data modeling language used to model configuration data, state data, RPCs, and notifications for network management protocols. It was created by IETF and standardized as RFC 6020 [3] and later updated to version 1.1 as RFC 7950 [4]. YANG is a language originally designed to model data for the NETCONF protocol, but it is a standalone modeling language. Although it is not explicitly specified in RFCs, YANG can also be used with protocols other than NETCONF. One of the advantages of the YANG language is, that it is human-readable, and therefore it is relatively easy to learn.

YANG models the hierarchical organization of data as a tree in which each node has a name, and either a value or a set of child nodes. YANG models provide short and clear descriptions of the nodes, as well as the interaction between those nodes.

Data in YANG are structured into modules and submodules. A module can import definitions from other external modules and can include definitions from submodules. Example of the YANG format is in listing 2.4 and example of XML encoded yang is in listing 2.5. The example shows a definition of a leaf-list, which defines a sequence of values of a particular type.

Data models can describe constraints to be enforced on the data, restricting the presence or value of nodes based on other nodes in the hierarchy.

YANG allows the definition of operations. The operations' names, input parameters, and output parameters are modeled using YANG data definition statements. Operations can also be tied to a container or list data node.

```
<cont xmlns="urn:example:a">
    <domain-search>high.example.com</domain-search>
    <domain-search>low.example.com</domain-search>
    <domain-search>everywhere.example.com</domain-search>
</cont>
```

Listing 2.5: YANG leaf-list value in XML encoding[4]

YANG defines a set of built-in types and can define derived types, that can restrict their base type's set of valid values, like range or pattern restrictions [4].

YIN

YANG modules can be translated into an equivalent XML syntax called YANG Independent Notation (YIN). The conversion between YIN and YANG is lossless. There is a one-to-one correspondence between YANG keywords and YIN elements. YIN elements corresponding to the YANG keywords belong to the namespace whose associated URI is `urn:ietf:params:xml:ns:yang:yin:1`. The argument of a YANG statement is represented in YIN as either an XML attribute or a subelement of the keyword element. The table defining these conversions is available in the RFC 7950 and will not be mentioned here, as converting between YANG and YIN formats is not a focus of this thesis. Comments in YANG may be mapped to XML comments, but it is possible that comments are lost when converting from YANG to YIN [4]. An example of conversion between YIN and YANG is in Listing 2.6.

XPath Node Selection

Nodes of the schema can be addressed using XPath addressing. The XPath addressing can be used when checking if a node exists in a schema or when getting a node value from the configuration [8]. The XPath evaluation is a part of the *libyang* library described later in this chapter.

```
/module-name:*
```

Listing 2.7: XPath for getting all top-level nodes of the `module-name` [8]

Libyang

Libyang is a library developed by CESNET. It implements the processing of the YANG schemas and data modelled by the YANG language. It can parse and process schemas in both YANG and YIN format, as well as parse, process, and validate instance data encoded in XML or JSON format. In the current version, it covers both YANG version 1.0 and version 1.1 [7]. Since the *libnetconf2* library was chosen to handle the NETCONF communication, *libyang* will be used as well, as it handles all YANG operations whithin *libnetconf2*.

```
The following YANG module:

    module example-foo {
       yang-version 1.1;
       namespace "urn:example:foo";
       prefix "foo";
       import example-extensions {
            prefix "myext";
       }
       list interface {
            key "name";
            leaf name {
                type string;
            }
            ...
        }
    }

is translated into the following YIN:
    <module name="example-foo"
            xmlns="urn:ietf:params:xml:ns:yang:yin:1"
            xmlns:foo="urn:example:foo"
            xmlns:myext="urn:example:extensions">
       <namespace uri="urn:example:foo"/>
       <prefix value="foo"/>
       <import module="example-extensions">
         <prefix value="myext"/>
       </import>
       <list name="interface">
         <key value="name"/>
         <leaf name="name">
           <type name="string"/>
         </leaf>
         ...
       </list>
     </module>
```

Listing 2.6: YANG and its equivalent YIN schema [4]

2.3 CMake

CMake is an open-source tool designed to build, test and package software. Simple configuration files placed in each source directory called CMakeLists files are used to generate standard build files, like Makefile. CMakeLists files can import other files based on configuration, so the user can toggle some features that do not have to be included by default. While generating the standard build files, CMake checks if all required libraries are installed on the user's computer. If a required library is not installed, the user is notified before building the project [14]. Many libraries, including the `libnetconf2` and `libyang` libraries, use CMake to compile them, as it became a common practice to install Linux libraries using three commands: `cmake ..`, `make`, and `make install`.

2.4 Node.js and npm

Node.js is a JavaScript runtime designed to build network applications. The project described in this thesis does not directly use Node.js, but it does use the node package manager (npm), which comes with Node.js by default. The Angular framework uses Node.js to run its development server. The development server then allows programmers to test their applications faster than if they had to recompile the whole application and upload it to a web server. Chapter 2.5 describes the Angular framework in more detail. Since the GUI of the application described in this thesis will be built on the Angular framework, npm will be used to handle all the packages required by the GUI.

Npm is a command-line utility for interacting with the node open-source package repository that aids in package installation, version management, and dependency management. The author of a project has to specify all dependencies in the `package.json` file. A user can then install all project's dependencies just by running a single command, `npm install`, in the project's directory [15]. The `package.json` file can also include scripts. Scripts in npm are usually a shortcut for a long command or a chain of few commands to interact with the application. The `npm run` command followed by the script name invokes the npm script. The script named "start" has a specific use and should only be used to launch the whole application. The shortened `npm start` command invokes the start script.

Thanks to the `package.json` file, a user can start an application from the source code by just running two commands, `npm install` followed by `npm start`. The simplicity of starting applications with npm is especially helpful when running applications downloaded from a git repository since npm handles both the dependency installation and the process of building the source code.

2.5 Angular

Angular is a cross-platform framework for making interactive UIs using web technologies developed by Google. Application structure is defined in HTML, styling is done using cascade styles (CSS) and application logic is written in the TypeScript language. The whole application then compiles into JavaScript. The advantage of using Angular is that when a user loads the page, the whole app is downloaded and when using the app, only the changed parts of the page are redrawn, without sending a new HTTP request.

An Angular application consists mainly of components and their templates, that can not be understood by the browser directly. Therefore, Angular applications require a com-

pilation process before they can run in a browser. Angular offers two ways to compile an application: Just-in-Time (JIT) and Ahead-of-Time (AOT).

The JIT compilation compiles the application in the browser at runtime [2]. The main benefit of the JIT compilation is easier debugging since the developer can see the original typescript file names while the application is running. For example, when the app outputs an error, the developer can see which file caused the error in the browser's console. The liberouter GUI's frontend uses JIT compilation.

The AOT compilation compiles the application at build time. The main benefits of the AOT compilation are faster rendering and smaller application file size. The AOT compilation can also help the developer to find HTML template errors earlier because the AOT compiler detects and reports template binding errors during the build step before users can see them [2]. Plugins to manipulate the device's configuration will use the AOT compilation. Chapter 4.1 describes how plugins are implemented.

2.6 Flask

Flask is a lightweight web application framework. The Flask documentation describes Flask as a microframework. The "micro" in microframework means Flask aims to keep the core simple but extensible. By default, Flask does not include a database abstraction layer, form validation or anything else where different libraries already exist that can handle that. Instead, Flask depends on extensions to implement those features, allowing programmers to choose which library they want to use [16]. The project described in this thesis will only use the core functions of the Flask framework.

Flask provides an HTML template framework that will not be used since the Angular framework is used to handle the HTML templating instead. The reason for using Angular instead of Flask for HTML templating is that Angular downloads all the required files for the application GUI to the user's browser and the pages are rendered on the client-side. If the Flask templating was chosen, the pages would have to be rendered on the server-side, increasing the requirements for server resources.

2.7 Liberouter GUI

Liberouter GUI is a UI framework developed by CESNET to provide a simple way to develop UIs for various projects. It is built on the Angular framework for the frontend and uses Python 3 programming language for the backend with the Flask library to handle the HTTP requests. Liberouter GUI is a modular system. It can be extended using modules, that provide requested functionality. By default, Liberouter GUI provides only user authentication and authorization, UI design, configuration management, and session handling. The default design of the Liberouter GUI is in Figure 2.3.

Each module in Liberouter GUI is divided into two parts: frontend and backend. Backend is processed on a webserver and usually handles communication with other libraries and programs running on the same server. In this thesis, the backend will be used to provide communication between the frontend and *libnetconf* and *libyang*. Backend is communicating with the frontend using the HTTP protocol and data are sent in a JSON format.

The frontend is processed on the user's side, in the browser, as Angular compiles into JavaScript, which runs natively in most web browsers. The frontend can also handle some

of the calculations that do not need to be calculated on the backend to lower the system requirements of a server, that the backend is hosted on.

Liberouter GUI uses Angular framework to implement its frontend. Therefore, all frontend modules must be built on Angular too.

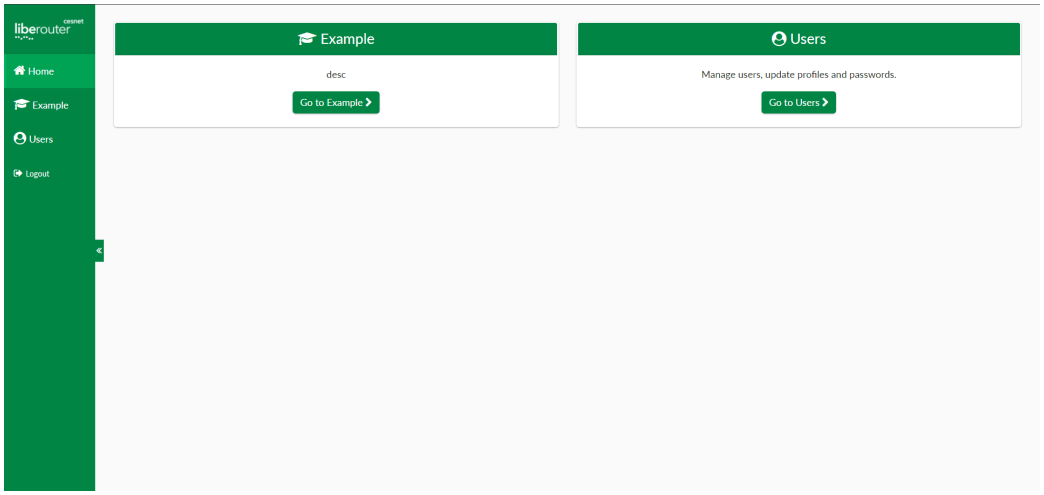

Figure 2.3: Home page of a Liberouter GUI application

2.8 WebSockets

WebSocket is a communication protocol, providing a two-way communication channel over a single TCP connection between a client and a server. The IETF standardized the WebSocket protocol as RFC 6455 in 2011 [12]. The WebSocket protocol can operate over HTTP ports 80 and 443 as well as support HTTP proxies, thus making it compatible with the HTTP protocol. However, the design does not limit the WebSocket protocol to HTTP, and future implementations could use a simpler handshake over a dedicated port without reinventing the entire protocol. The WebSocket protocol enables interaction between a client and a server with lower overhead than its alternatives, such as HTTP pooling, facilitating real-time data transfer from and to the server. The WebSockets provide a standardized way for the server to send content to the client without being first requested by the client.

The Angular framework already implements a library for WebSockets. On the backend, the Socket.IO library will handle the WebSocket requests.

2.9 Netopeer2

Netopeer2 is a set of tools implementing network configuration tools based on the NETCONF protocol. One of the tools included in Netopeer2 is `netopeer2-server`, which can act as a NETCONF capable device. In this thesis, `netopeer2-server` will be used to test functions that connect to devices and change their configuration using the NETCONF protocol. Another tool included in the Netopeer2 toolset is `netopeer2-cli`, which can configure the `netopeer2-server` and check the server configuration[1].

The Netopeer2 toolset was chosen because it is simple to use and open-source. In the most basic use case, no further configuration is required after installation. After the

[1] https://github.com/CESNET/netopeer2

netopeer2-server is started, any program implementing the NETCONF protocol can connect to the netopeer2 server using the SSH protocol. The Netopeer2 toolset is also still maintained and developed, therefore if there are any problems with its functionality, it is easy to get support from the developers. The Netopeer2 toolset uses the *libnetconf2* library to handle NETCONF requests, simplifying the installation process, since the *libnetconf2* library will be used to handle NETCONF requests in the project described in this thesis as well. The Netopeer2 toolset can be used for more advanced use cases that will not be described in this thesis, as the toolset will only be used as a simple NETCONF capable device.

2.10 Netopeer2GUI

The Netopeer2GUI is a web-based NETCONF management software built on the liberouter GUI framework. It allows users to connect to a device using the NETCONF protocol, load and modify the device's configuration and browse YANG schemas in text format. The Netopeer2GUI will serve as a base for developing the project described in this thesis. Better support for connecting to multiple devices at once will be added to the backend. The frontend will be rewritten to support more features. Chapter 3.4 describes all of the new features. The most important new feature missing from the Netopeer2GUI is a way to modify the device's configuration without having experience with the NETCONF protocol and the YANG modelling language. As shown in figure 2.4, editing the device's configuration using the Netopeer2GUI can be complicated for an inexperienced user.

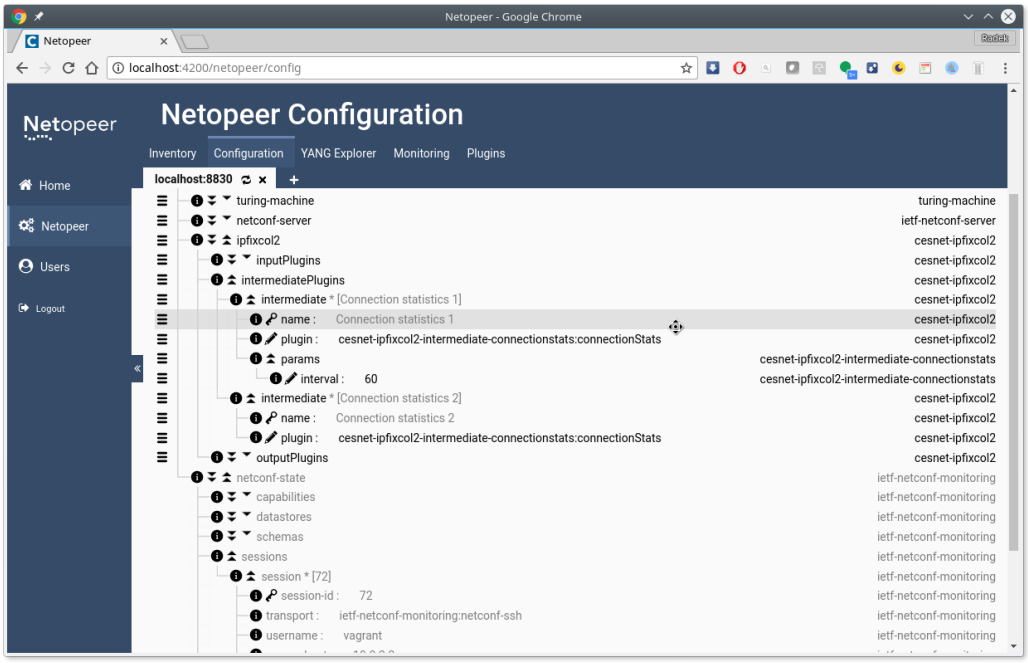

Figure 2.4: The Netopeer2GUI user interface [9]

2.11 Adobe XD

Adobe XD is a UI/UX design tool developed by Adobe. In the starter plan, it is free to use [1]. Adobe XD is a rapid prototyping tool. It is simple to use, and therefore great even for beginners while providing advanced functionality for professional users. Prototypes designed in Adobe XD can be shared with anyone, allowing quick feedback from the collaborators and potential users of the final product, consequently speeding up the design process. There are three main views of the Adobe XD application.

Designing a prototype in Adobe XD starts on the Design tab with a screen size selection. Many pre-configured screen sizes are available for various devices, including popular smartphone models, tablets or computer screens. After a screen size is selected, the prototype design consists mainly of placing rectangles, text and images on the page. Grouping of these components simplifies reusing designed complex components. The Adobe XD community provides pre-made design kits and plugins, many of them for free, speeding up the design process even further.

Each Adobe XD project can contain multiple pages. In the Prototype tab, the designer can set triggers for the preview view. When a user is trying the design in the preview view, these triggers switch between the designed pages. Adobe XD provides triggers for clicking/tapping, keyboard key presses, dragging and even for saying voice commands. With the click/tap trigger, any component placed on the page can serve as a trigger allowing the designer to create working buttons to switch screens.

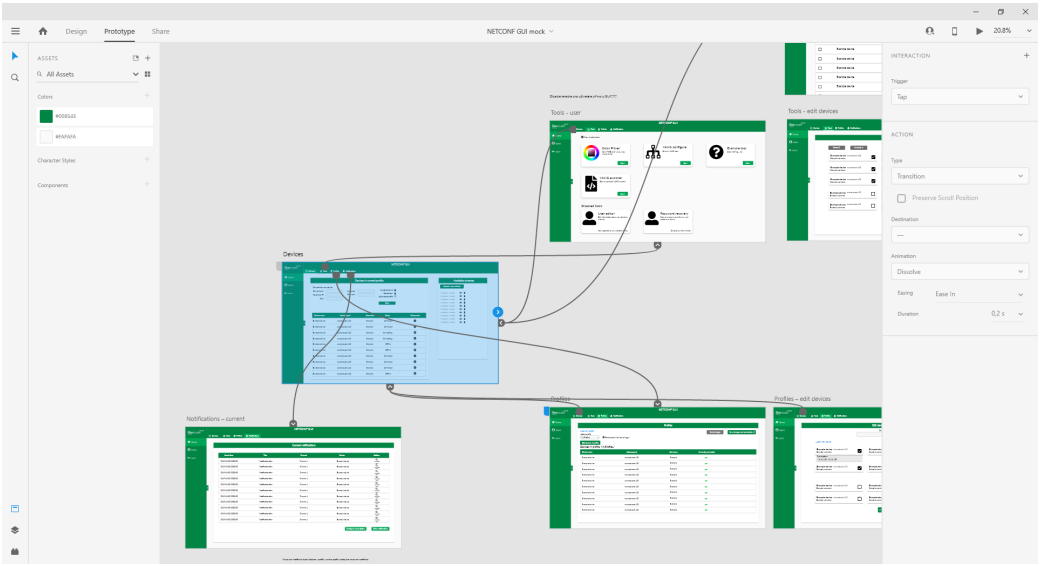

Figure 2.5: The Adobe XD user interface

In the preview view, users can test the design. There are two ways to access the preview view. There is a button for opening the preview view in the Adobe XD application, which will open a new application window with the preview. The second way to view the preview is sharing the design from the Adobe XD application. Sharing the design means uploading it into the Adobe cloud. With the starter plan, the designer can only share one design. After sharing the design, the designer will get a URL address to view the design in a web browser. The designer can share this URL address with collaborators and potential users.

Anyone with the URL address of a project can view and interact with the prototype. Users can also leave comments for the designer directly in the browser preview.

The Adobe XD was chosen to design the prototype because it is free while providing great features. It is easy to learn and use, and the design sharing feature is helpful when a design needs to be iterated over quickly with the users.

Chapter 3

Design

To design this application, first, the use-case scenarios had to be determined. The UI prototype was designed based on those scenarios and discussed with potential users and edited based on their feedback. When the UI design was finalized, the application interface between the module's frontend and backend was designed based on data needed by the UI.

3.1 Application Overview

Connections to all devices are handled by the server, that the backend is running on. Because there is no implementation of the *libnetconf2* library, that supports the TypeScript language, the *libnetconf2* library can not be used directly in frontend. The devices that need to be configured are often on the same network as the backend. Since the backend will handle all the NETCONF connections, there is an added benefit of being able to connect to devices, that are behind a firewall, if they are on the same network. Some devices may even not require a password to connect to them, because the server that the backend is running on may have private keys replacing the need for passwords.

Libnetconf2 and *libyang* send NETCONF messages to target devices and parse their responses from the module backend. Those libraries are wrapped in a Python wrapper, so their functionality can be used directly from the backend code. When a user connects to a device from the UI, the UI module sends an HTTP request to the Liberouter GUI's backend. The backend then parses the HTTP request and forwards the data from the request to the NETCONF backend module. The module then calls a *libnetconf2* function to try to connect to the device using the credentials provided by the user. If the connection was successful, *libyang* and *libnetconf2* parse the device's answer. The Liberouter GUI's backend then sends the answer to the Liberouter GUI's frontend using the HTTP protocol. The Liberouter GUI forwards this data to the NETCONF module, which displays the response to the user. A schema of this communication cycle is shown in Figure 3.1.

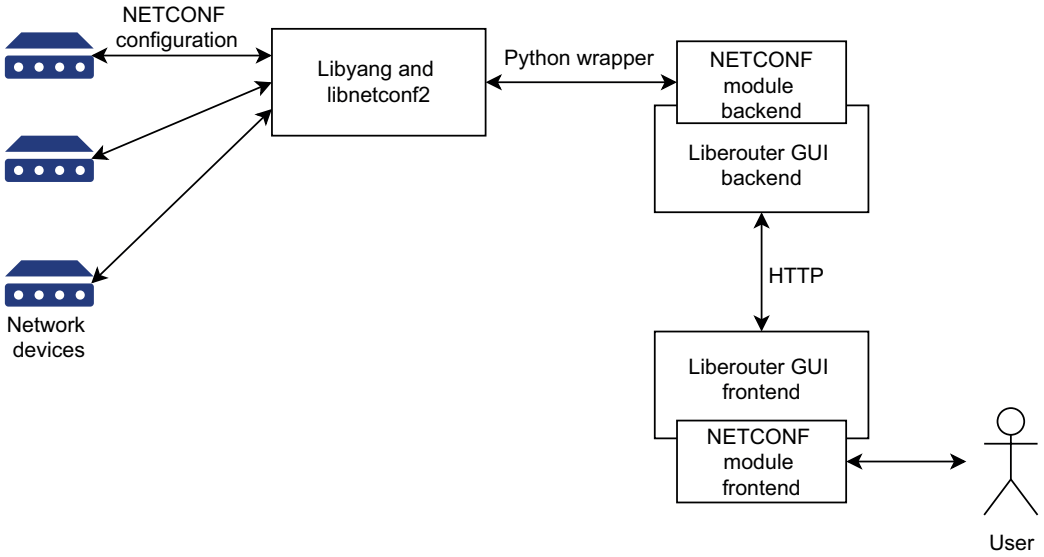

Figure 3.1: All aplication modules. Blocks marked "NETCONF module" will be implemented in this thesis

Additional information may be required when connecting to a device or making changes to a configuration. For example, the device may require a user to sign in. There has to be a way to ask the user to fill in the required data and send the data to the device without interfering with or canceling the operation that is currently running. The user should also be able to subscribe to notifications from a specific device. Notifications may occur at any time during device operation.

WebSockets, described in chapter 2.4, solve both of these issues. A WebSocket connection will be used instead of using HTTP requests when communicating with devices. If the user input is required while processing a request on a device, the backend sends a WebSocket message to the frontend. The frontend then reacts to these messages by displaying a message and a form that the user can use to input the required data. The data from the form are then sent back to the backend as a response to the previous message.

An example of the communication between modules when connecting to a device with a wrong password can be seen in figure 3.2. There is another layer when communicating with *libnetconf2* from the Liberouter GUI's backend, the *libnetconf2* Python wrapper. The Liberouter GUI's backend registers callbacks to the *libnetconf2* Python wrapper. The Python wrapper then registers those callbacks to the *libnetconf2* C library. When an event that should trigger one of these callbacks occur, the *libnetconf2* calls the specified Python wrapper function, which then calls the specified Liberouter GUI's backend function. The *libnetconf2* library communicates directly with the device using the NETCONF protocol. The schema shows an attempt of connecting to a device with a wrong password and then the request from the *libnetconf2* for a new password. After the request, the user enters the correct password, and the connection is established.

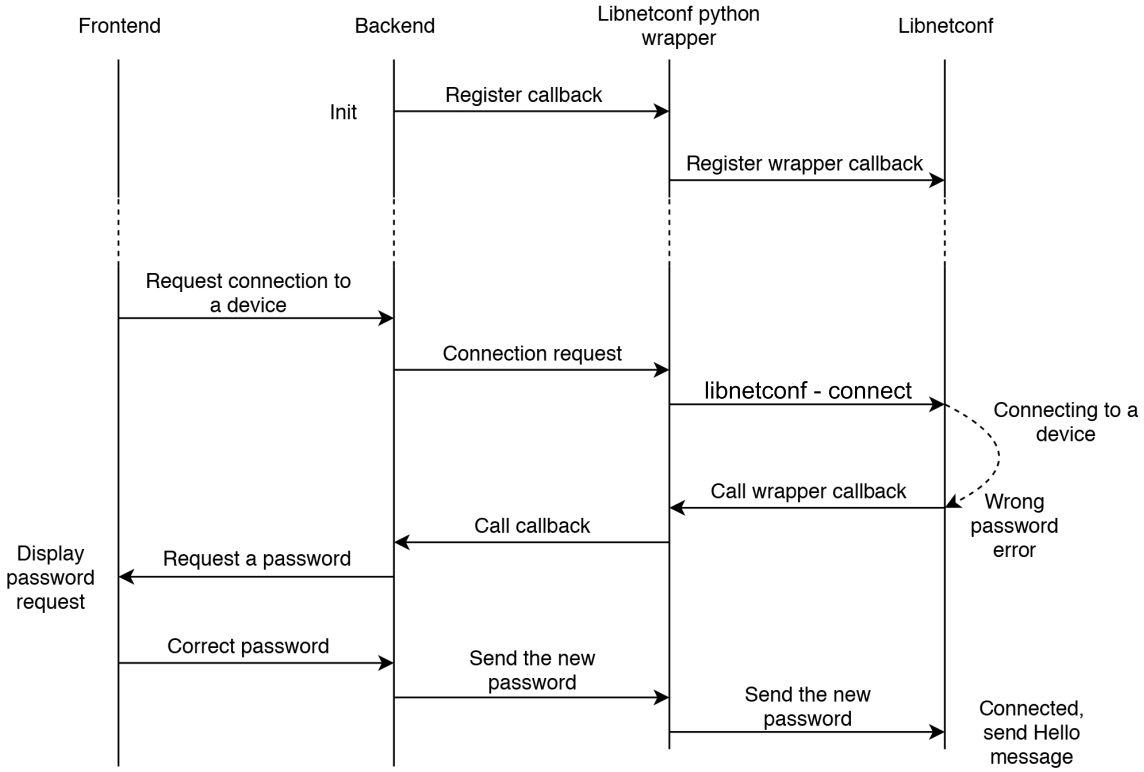

Figure 3.2: Communication between modules while connecting to a device and entering a wrong password

3.2 Use Cases

Use-cases were made after discussion with potential users. It was determined that two personas need to be considered: user and administrator. A user can only use the UI to modify or display device configuration. On top of that, an administrator can specify rules for connecting to devices and limit connection to specific device addresses. The use-case diagram is in Figure 3.3.

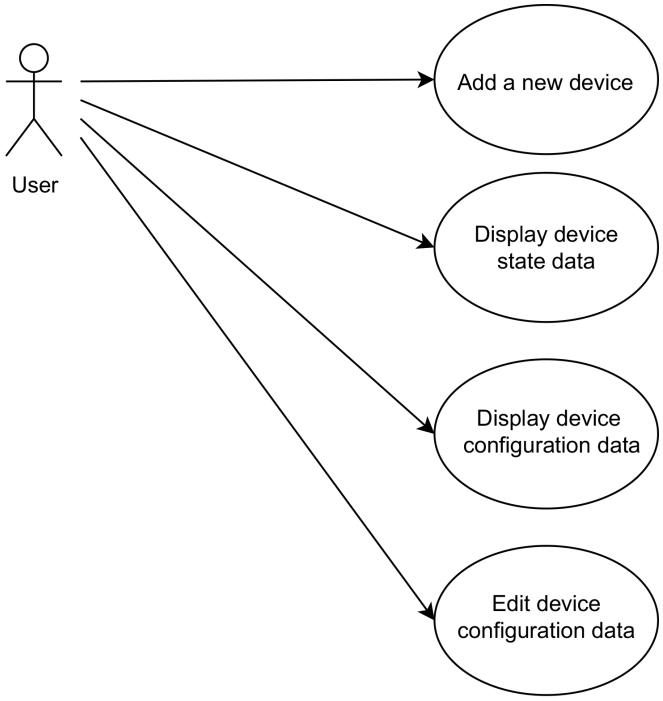

Figure 3.3: Application use-case diagram

Use Case 1 – Add a New Device

When a new device is added to the network, a user adds the new device to the system. They input the required information into the GUI. Then they choose, whether they want to connect to the new device right away and whether they want to save it for later use.

If the user chose to connect to the new device right away, a connection is established to the device.

If the user chose to save the device for later use, they can add it to a profile.

Use Case 2 – Display Device State Data

A user wants to do a routine check on devices or diagnose a problem with a device. They select the device whose state they want to check. If the device is not connected, they connect to it. Then they look for the specific information they want to check. They find it in the state data and decide the next action based on its value.

Use Case 3 – Display Device Configuration Data

A user wants to check if the device is configured correctly, diagnose a problem or copy a value from the device's configuration to another device. They select the device whose configuration they want to check. If the device is not connected, they connect to it. The device's full configuration is then displayed. User searches in the configuration for a specific value. After finding the value, they check it and decide their next action based on it.

Use Case 4 – Edit Device Configuration Data

A user wants to change devices configuration or configure a new device. They select the device whose configuration they want to change. If the device is not connected, they connect to it. They look for the data that they want to change. After that, they change the value and send the changes to the device. If the new values were correct, the changes are applied in the configuration of the device. If the values were incorrect, the user changes the values to be correct and sends them again.

3.3 Behaviour Design

When adding a new device, users must enter an IP address or a hostname of the device and its port. Username is also required. A password may be required, but some devices use a certificate to authenticate connections instead of using a password. Users can choose a name for any device they choose to save to make them easier to recognize. An error may occur when trying to connect to a device. The most common error would probably be an incorrect password. In case of an incorrect password, an error will be displayed to inform users, that they entered the wrong password and prompting them to enter the password again.

Working with device data will be done using "tools". Tools will serve as an easy to use interface for working with YANG configuration data. An example of a tool can be a color picker that targets a specific value in the data model that represents a color in an RGB model and displays it to the user as a color instead of three numerical values. There will be several built-in tools, but more importantly, anyone can develop new tools and use them to view and edit a device's configuration. Users will interact with device configurations using these tools, so they do not need to be knowledgable about NETCONF or YANG.

Administrators can disable certain tools to prohibit users from accessing them. Since users with different roles will have access to different tools, both system administrators and regular users can use a single application.

Configuring allowed devices is meant for disabling connection to untrusted devices by administrators. It will only be available as a configuration file. Administrators can set allowed or prohibited addresses in a configuration file that will be checked every time a user adds a new device. If a user tries to connect to an address that is not allowed, an error message will display. When an administrator adds a new address to the prohibited list or removes an address from the allowed list, all devices in the database are checked and if they do not comply with the new policies, they are removed from the database. Having this configuration only available as a file acts as another layer of security because this file can only be accessed by administrators that have access to the file system of the server that the backend is running on.

3.4 User Interface

When designing the GUI, the Liberouter GUI's design was the most influential. The goal was to make the module described in this thesis fit in with the Liberouter GUI. First, a rough sketch was drawn on paper with a pencil. After figuring out the layout on a paper, a prototype was created using Adobe XD. There are four major parts of the UI.

Devices

The "Devices" tab allows users to add new devices and check which ones are currently connected. Users can also browse available YANG schemas, and they can upload additional schemas if schemas are not downloaded automatically from a device. As shown in figure 3.3, each device displays a current state of connection, that can be either connected, connecting or offline. There is a button to disconnect a device in each row of the device table. When adding a new device, users can select whether they want to save the device for further use or only connect to it. There is also an option to connect to a device without saving it, intended for connecting to a device only once. After connecting to a device, new schemas are downloaded if the device supports automatic schema downloading. If not, a user is prompted to upload a schema. If the user does not have the schema, they can skip this step. If a user chose to save the device, all information about the device is saved to the database. The password is only saved if the administrator allows password saving in the application configuration. Because passwords can only be saved in plain text, it is unsafe to save passwords on servers, that are not well protected against unauthorized connections.

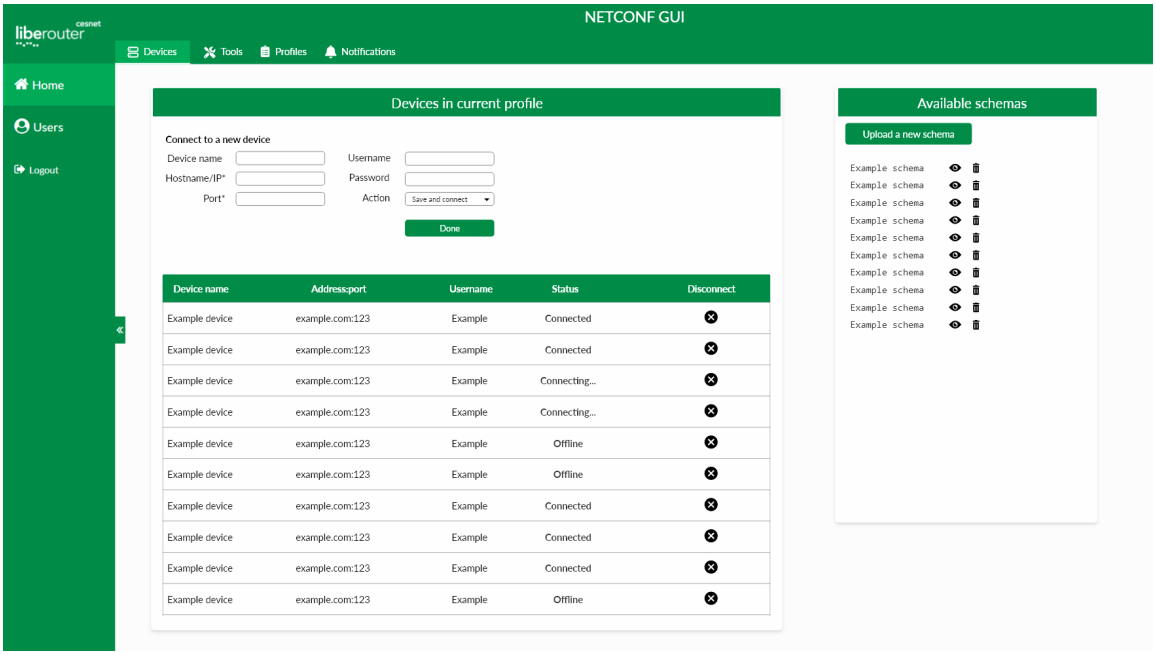

Figure 3.4: Mockup of the connected device list.

Tools

The "Tools" tab allows users to access all available tools. As described in chapter 3.3, tools allow users to view and modify a device's configuration data. As a result of this thesis, an application programming interface (API) will be implemented to allow anyone to develop more tools for configuring specific devices based on their needs. After a user clicks on a tool, a dialog will appear, prompting the user to choose devices that they want to configure using the selected tool. Tools may have limitations and may require certain parts of a device's schema to match the tool's settings. For example, the color picker tool will require a schema

containing an RGB color value. The devices that will show up in the device selection list must contain the required schema parts and must be connected, otherwise, user can not edit their configuration using the selected tool and the devices will not show up in the tool selection list.

In this thesis, four tools will be implemented.

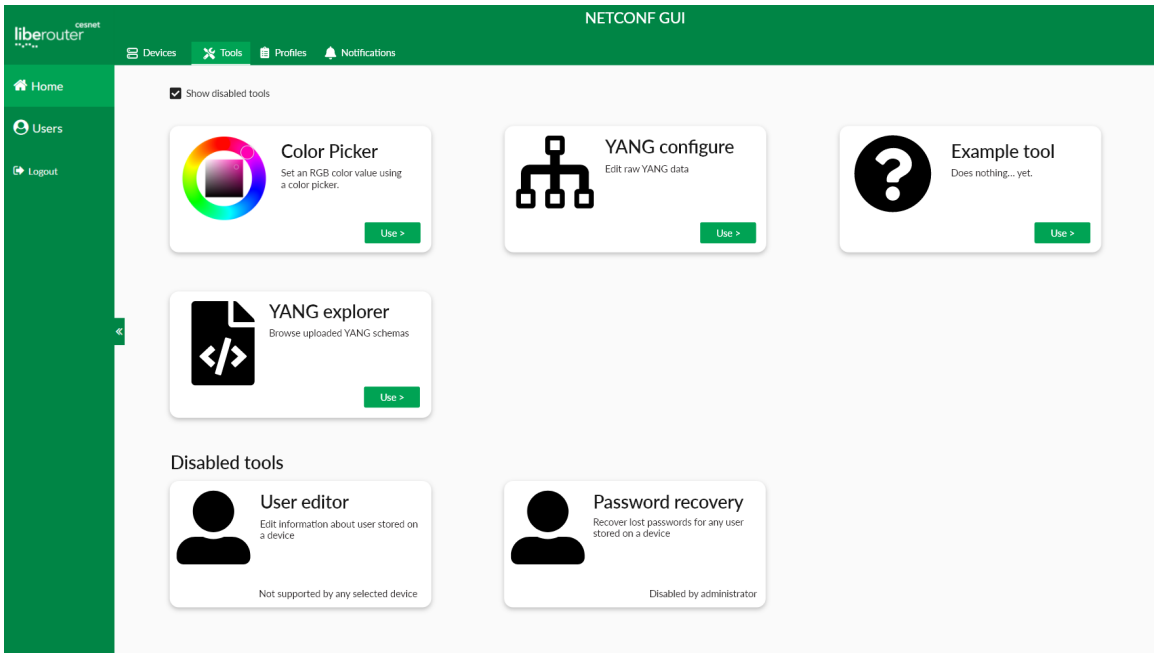

Figure 3.5: Mockup of the tool selection

Example Tool

The "Example tool" will have no functionality and will serve as a base for developing other tools. It will contain all the required settings and some example behavior to make creating a new tool easier.

YANG Explorer

The "YANG explorer" will allow users to browse uploaded YANG schemas and view them. From the YANG explorer users can also remove schemas they do not need anymore.

Color Picker

The "Color picker" will serve as an example of a fully functioning tool. The tool will require an RGB color value somewhere in the device's YANG schema. When a supported device is selected, a color wheel will be displayed and the device's current color settings will be loaded and displayed as a color. Users can then set a new color value by moving a cursor in the color wheel and clicking the "save" button. If a user selects more devices at a time, the default color will only be loaded if all devices share the same color. If not, "multiple values" will be displayed instead of a default color setting. With multiple devices selected, a quick switch bar will display next to the color wheel. The quick switch bar allows the user

to quickly switch between selected devices and set a color on them individually, instead of setting them all to the same color.

YANG Configure

The "YANG configure" tool will use most of the functionality of the API. It will display the full configuration file as an editable tree. It will be based on the "Configurations" tab in the Netopeer2GUI. It will allow users to configure all values of the device's configuration. It will also allow users to copy configurations from other connected devices for easier configuration of new devices. Users will be able to hide parts of the configuration to make orientation in the values easier. The YANG configure tool will also provide a way to configure multiple devices at once, simplifying the configuration process for multiple devices of the same type.

Profiles

The "Profiles" tab will allow users to create device profiles. These profiles will serve to make configuration easier by grouping devices. The user can then connect to all devices in selected groups at once. The user will also have an option to connect to all devices from the selected profile on login. The profile tab mockup is in figure 3.6. Profiles and device settings are not shared between users.

When a user clicks on the "Edit devices in profile" button, a device selection screen appears, shown in figure 3.6. From there, a user can search in devices and add or remove them from the profile by checking or unchecking the checkbox next to their information. From the same screen, a user can also add a new device.

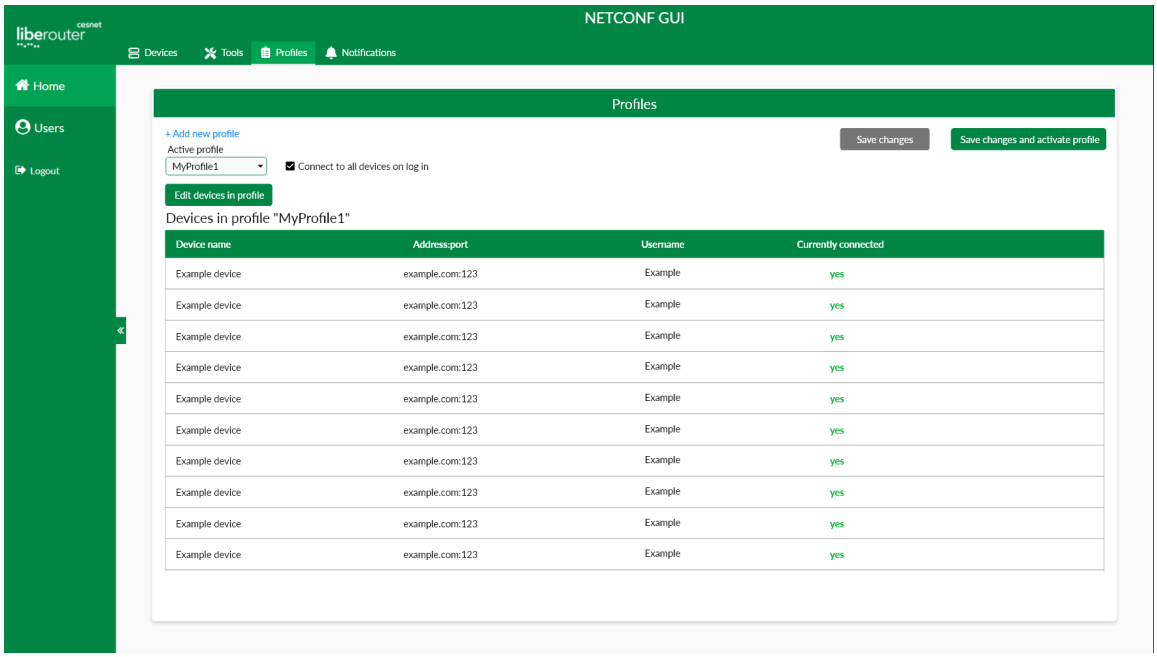

Figure 3.6: Mockup of the profile selection and editing

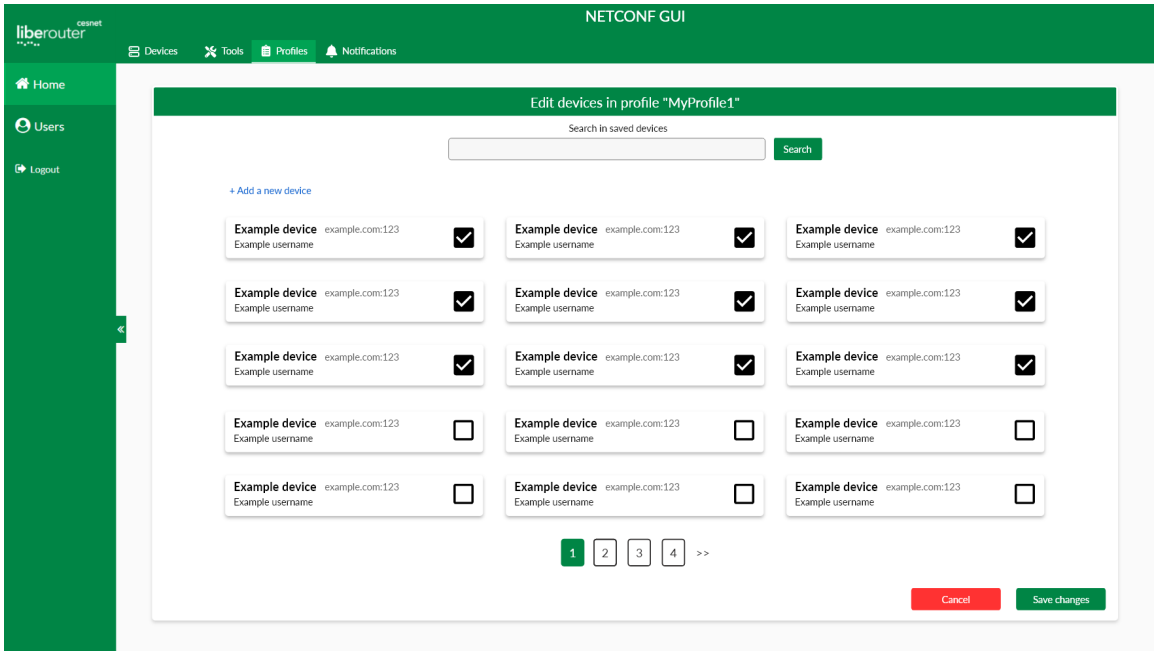

Figure 3.7: Mockup for selecting devices, that will be included in a profile

Notifications

The "Notifications" tab allows the user to subscribe to notifications generated by devices. Devices can generate notifications during their operation, but not all notifications are relevant to every user. Notification can contain any number of information of various severity, from status updates to critical errors. Notifications can be separated into different channels for easier management, each device specifies its channels. When the user subscribes to a notification channel, they will be notified immediately after a new notification is generated in that channel by a device. The user can also browse notification history.

From the UI shown in figure 3.8, the user can see the count of new notifications of that device on the "Browse notifications" button. New notifications are only added from subscribed channels.

By clicking on the "Browse notifications" button, the user is taken to a table of notifications from the device shown in figure 3.9, where they can filter by channel or search in notifications by other parameters. The user can also load notification history from that device. The user can quickly filter the notifications by the date of the creation of the notification. From the table of notifications, the user can load the detail of the notification or filter notification by the title of the notification, to quickly look for similar notifications in the past.

By clicking on the "Manage subscriptions" button from the notifications tab, the user can see all notification channels of the selected device. They can see, which channels are they subscribed to and which channels are no longer available, shown in figure 3.10. Subscribed channel names are saved to the device profile and the channels are subscribed to when the user logs in. The channel names may change over time. Because of that, some channels may no longer be available. When the application detects, that the channel is no longer available, the user is prompted to remove the channel from the profile. Until the user

removes the missing channel from the active profile, a yellow triangle with an exclamation mark is shown in the menu next to the “Notifications” tab button.

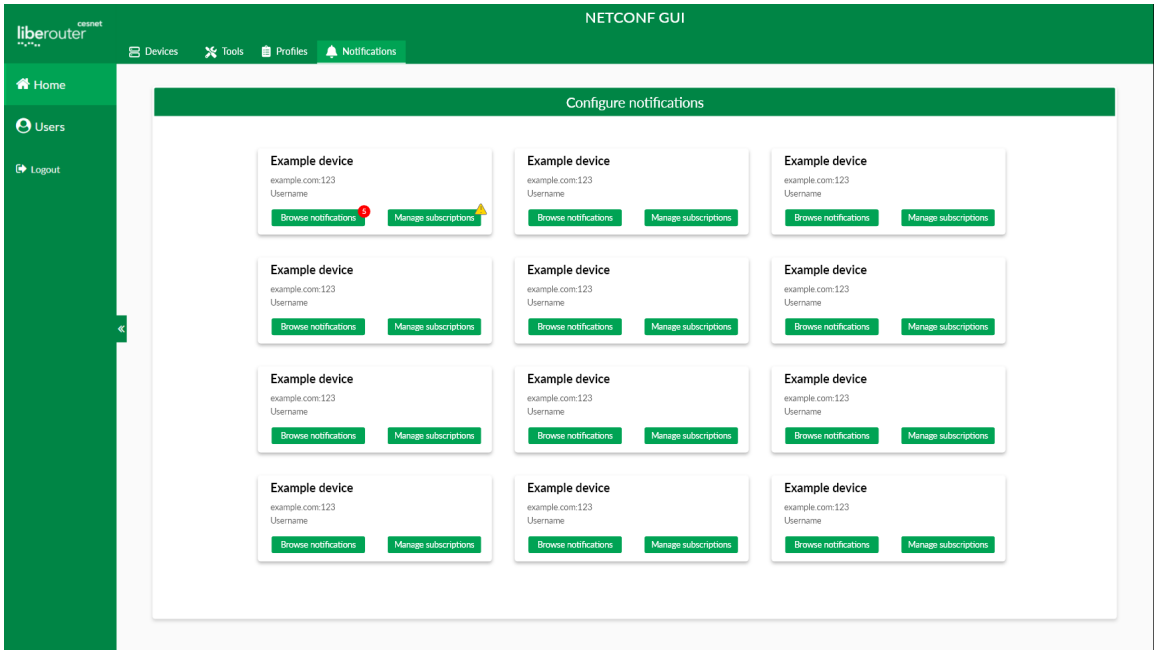

Figure 3.8: Mockup for selecting devices before loading notifications

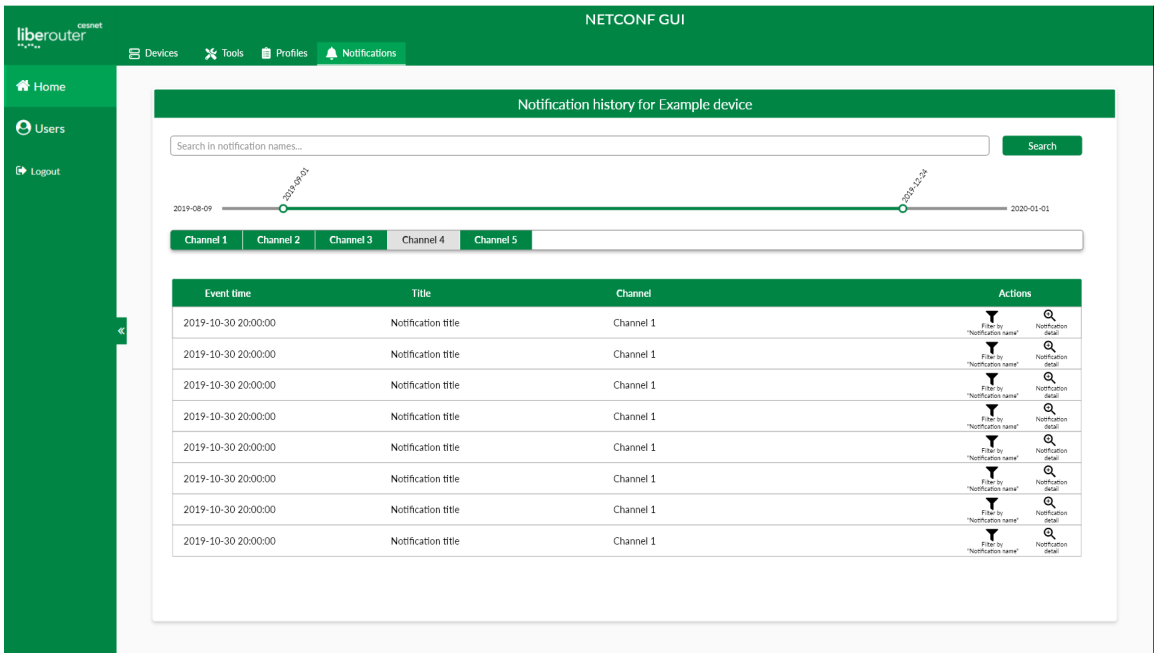

Figure 3.9: Mockup for a table of notification history

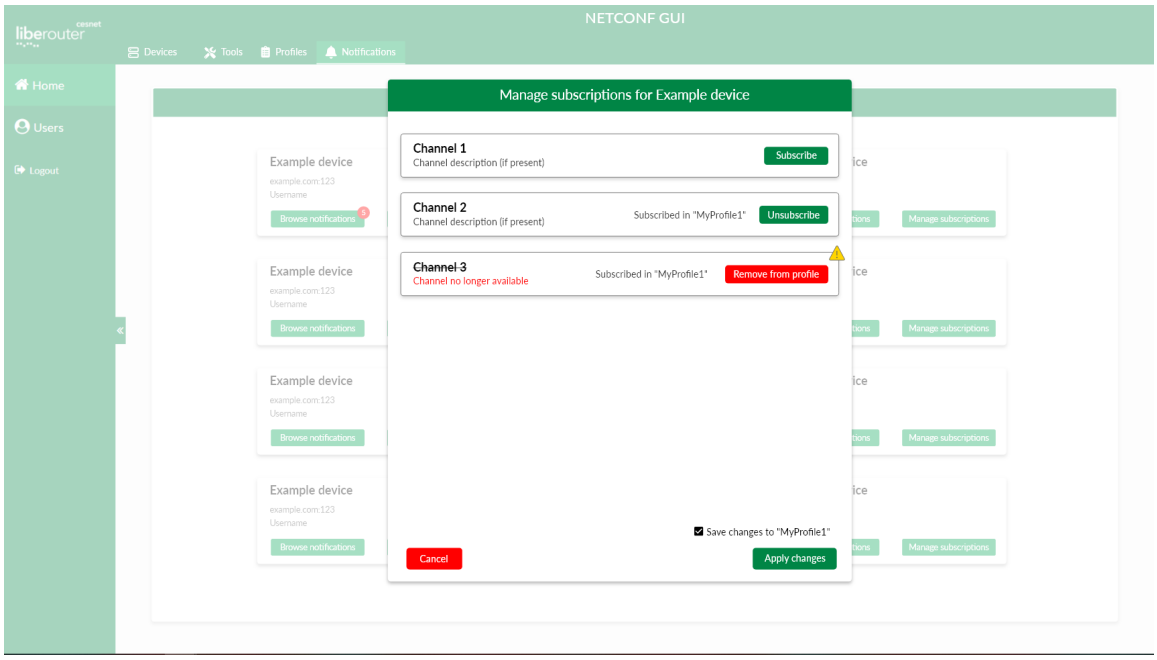

Figure 3.10: Mockup for subscribing to notification channels

Chapter 4

Implementation

This chapter describes the main parts of the system that were implemented. The implementation was divided into four parts; frontend, tools, a tool library, and backend. The frontend handles rendering the user interface in a web browser, sending some of the HTTP requests to the backend and loading tools and the tool library. Tools are an extension of the user interface, allowing developers to design new ways to configure devices without developing a whole new application. The tool library contains components and services that can be used when developing new tools to speed up the development process. The backend handles communication with devices, the device database, and the profile configuration.

4.1 Frontend of the Liberouter GUI Module

Most of the frontend is implemented in a liberouter GUI module. Frontend uses some of the the tool library components and services. The tool library is described in chapter 4.3.

Tool Loader

The most important feature of the frontend module is the tool loader. The tool loader and the whole plugin architecture for tools are based on an article on Medium by Alexey Zuev [18]. The tool loader allows the frontend module to dynamically load AOT compiled plugins without duplicating the Angular library code. The plugins are called "tools" in this project. The tools can share any library that is included in the frontend module. If a library is to be shared with tools, it needs to be added to the "externals" list in the frontend module and configured in the custom tool builder. All libraries defined in the "externals" list are then defined as global libraries, and the tools can easily import them. That is why a custom builder is needed. The custom builder prevents tools to include the library code in the AOT compiled code. The principle of sharing libraries is indicated in figure 4.1.

When a user loads a tool, the AOT compiled code is loaded from the server, and the tool's content is rendered to the page. The tool loader implements an error prevention system. If there is a critical error in the tool, the tool is not loaded, but the whole app can continue to operate normally. The user can see an error message explaining why the tool was not loaded.

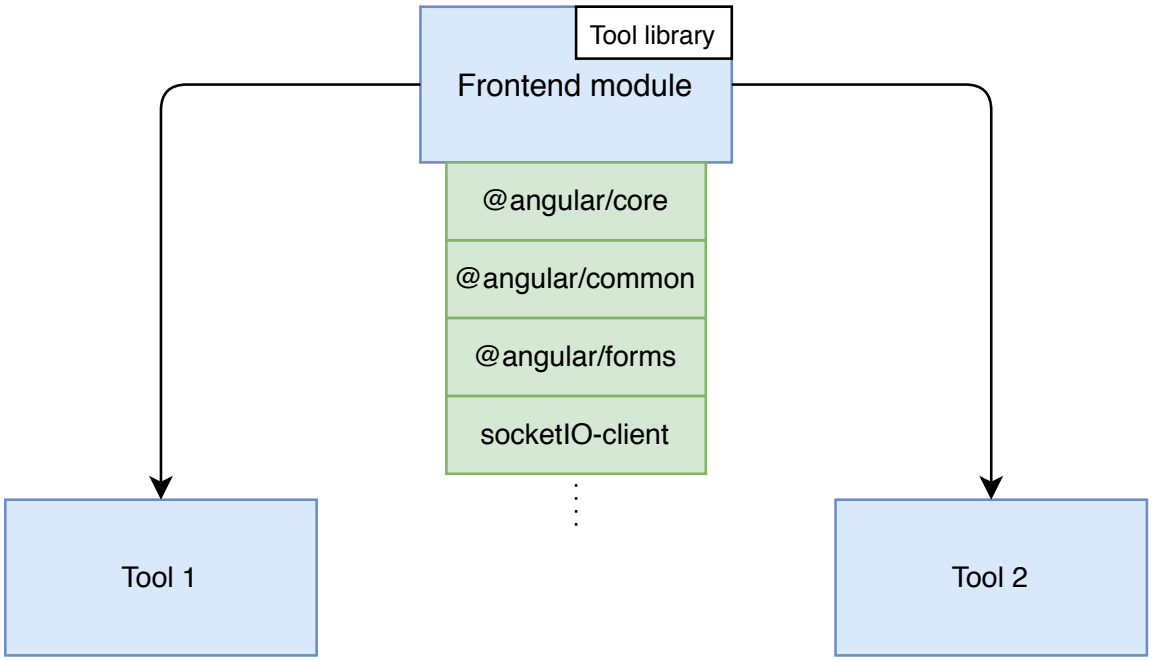

Figure 4.1: Tools share all common code

Profiles

Profiles allow the user to group devices. The user can connect to all devices in a profile with one click or automatically after logging in. Users can create many profiles and switch between them when using the GUI. When connecting to a profile, the user decides if the current device connections should be kept or discarded. If the user chooses to discard the previous connections, a request is sent to the backend to close all currently open connections before connecting to devices in the other profile.

When a user opens the NETCONF module from the liberouter GUI, an "active profile" is loaded. Any profile can be set as active from the "profiles" tab. If the active profile has the "Connect on login" option set to true, a connection is created to all devices in the active profile. Devices being connected immediately after opening the module makes the work with the GUI faster, since users can start to configure devices without having to manually re-connect them each time.

The finished "Profiles" page looks different from the mockup. In the mockup from figure 3.6, all the components are in one box. The original layout from the mockup turned out to be quite confusing to use. In the final layout in figure 4.2, the profile controls and the device list are in separate boxes. The separation of the profile control elements and the device list makes the GUI easier to use and less complicated. The page that is displayed after clicking on the "Edit devices in profile" button was implemented exactly as in figure 3.7.

4.2 Tools

Tools are AOT compiled plugins that are generally used to manipulate the device's configuration, but can serve other purposes as well.

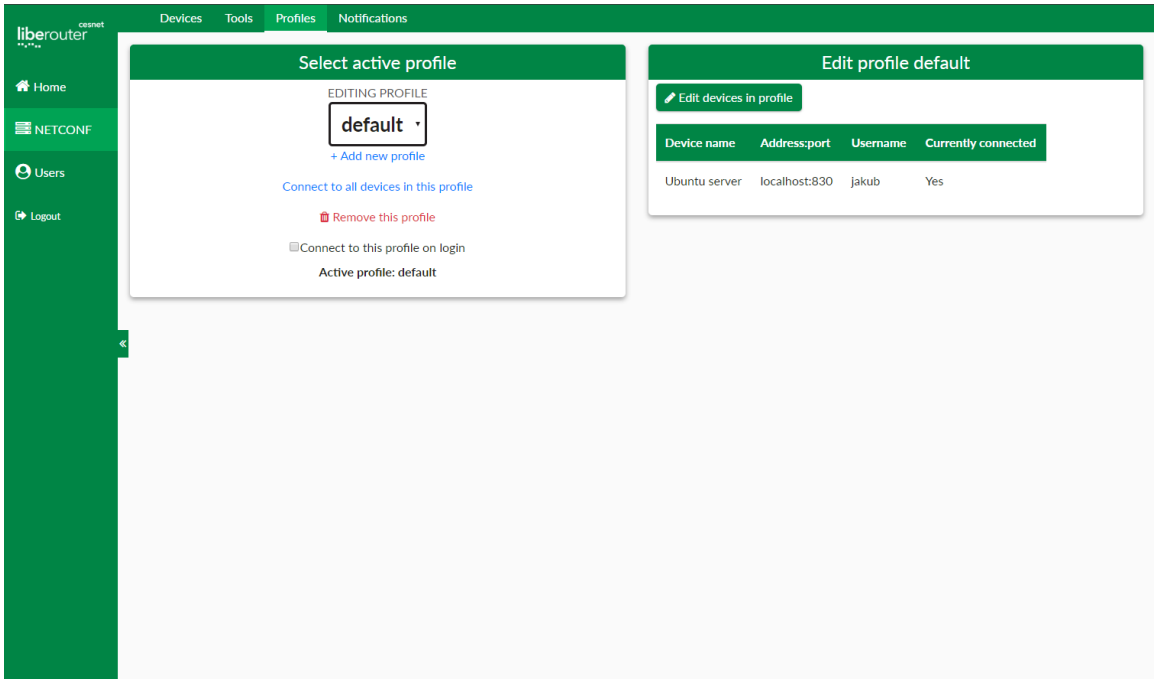

Figure 4.2: Finished profiles page differs from the mockup.

Yang Explorer

The Yang Explorer tool allows users to load all YANG schemas they uploaded and schemas that were imported from connected devices.

With no schema selected, the Yang Explorer tool displays a list of schemas that are available for the user. When a user clicks on a schema name, the schema is loaded from the server and displayed. A list of all available schemas is shown next to the schema, so the user can quickly load a different schema.

The Yang Explorer tool supports a basic syntax highlighting to make the reading of YANG schemas easier. The syntax highlighting does not support the full YANG language specification, as it is not needed to read the schema and would be very difficult to implement. First, the whole schema content is divided into characters using the triple-dot operator. The triple-dot operator assures that if there are any complex Unicode characters in the schema, they will not be separated. After dividing the schema content into an array of characters, the array is iterated over. The finite state machine controlling the schema highlighting is in Figure 4.3. The finished tool is in Figure 4.4.

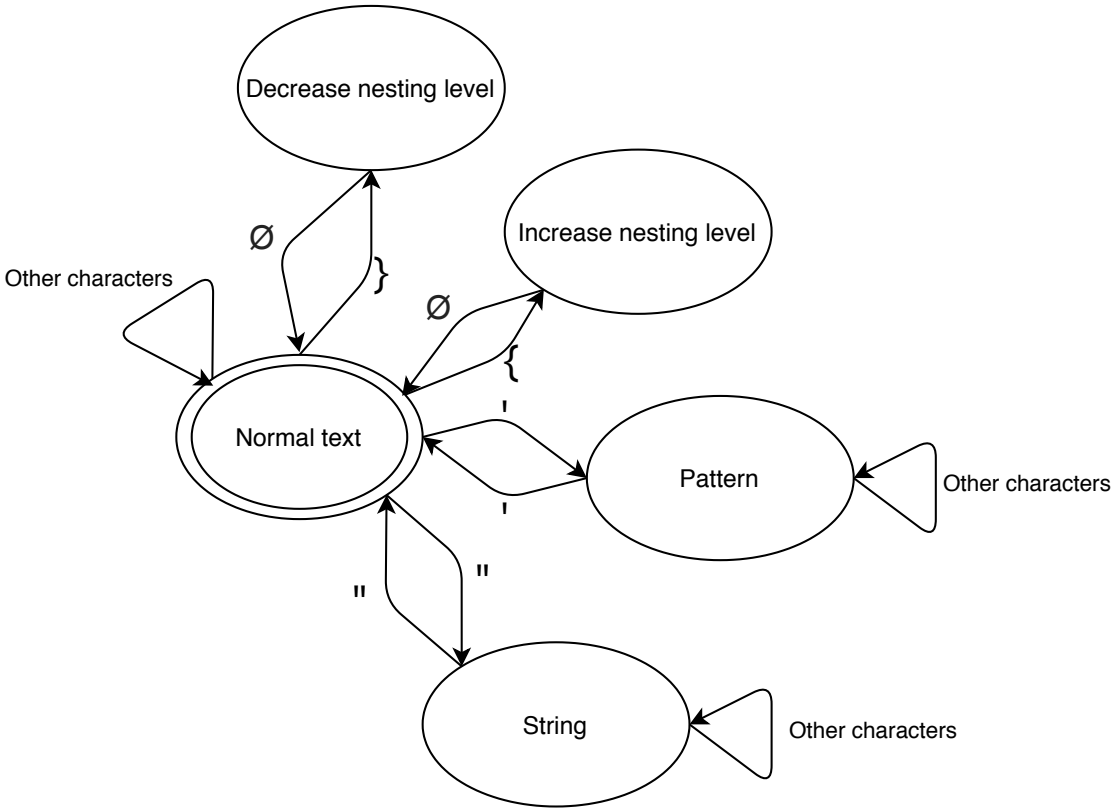

Figure 4.3: Finite state machine for the simplified YANG syntax highlighting

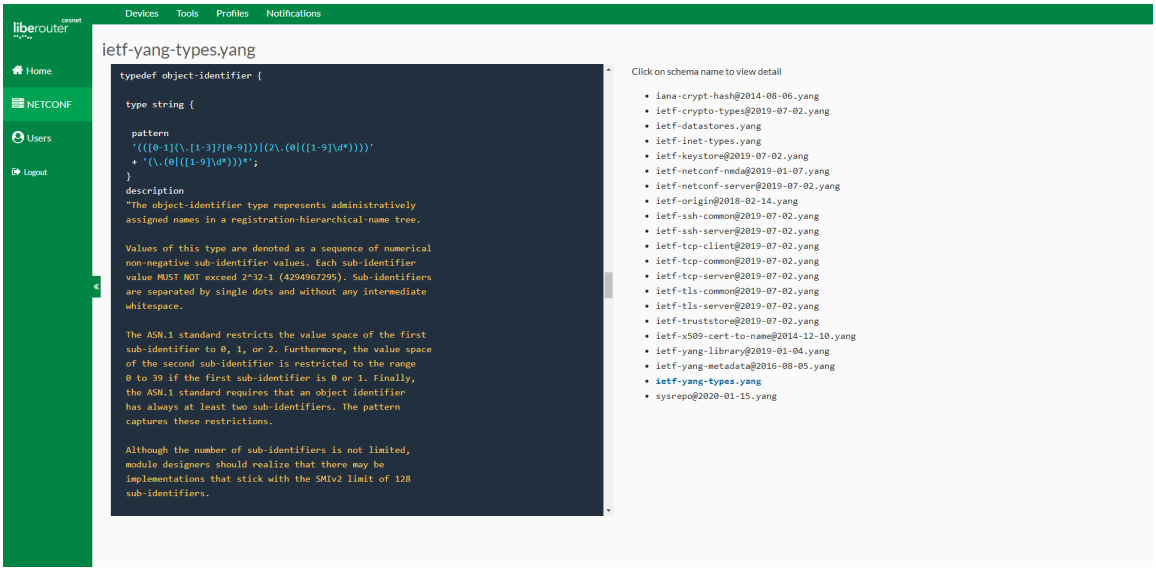

Figure 4.4: Finished Yang Explorer tool

YANG Configure

The *Netopeer2GUI* is an inspiration for the YANG Configure tool. It allows users to browse the device's configuration data as a data tree. Users can also edit the device's configuration

using this tool. The functionality of this tool is currently very similar to the configuration tab in the *Netopeer2GUI*. Currently, only one device can be configured at a time, but it is planned, that in future versions, the YANG Configure tool will be capable of configuring multiple devices at once, making bulk changes faster. Figure 4.5 shows the current state of the YANG Configure tool. It lacks many of the planned features, as some of those features were too complex and not in the focus of this thesis.

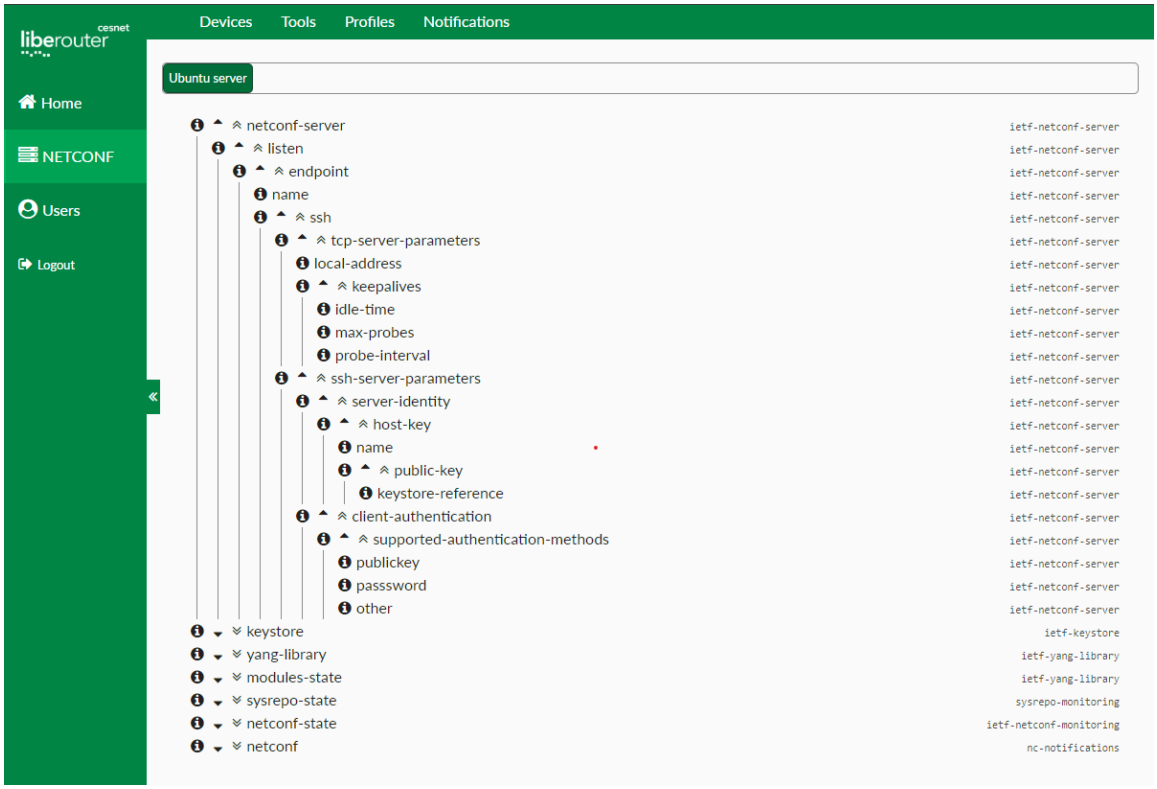

Figure 4.5: The current state of the YANG configure tool

Example Tool

The Example tool serves as a minimal tool implementation. Developers can copy files from the Example tool to create a new tool. Developers can start working with the device's configuration without having to deal with device connections, just by copying and renaming the Example tool. The example tool implements the device picker from the tool library, and after a user selects devices to configure, the example tool displays the number of selected devices.

4.3 Tool Library

The tool library contains components and services, that can be used in the frontend and all tools. Developers can use library services and components when developing a new tool to speed up development and for better integration into the whole system.

Components

The tool library provides several components. Most of the components are functional, while some are just for decoration purposes. All of the components have the "lib" prefix in the component selector. The components provided by the tool library are:

- The **schema list** component: Loads a list of schemas uploaded by the logged-in user and displays them. From this component, users can click on a schema's name to view them in the Yang Explorer tool or remove them from the server by clicking the trash can icon next to the schema's name.

- The **content box** and the **popup** components: Purely decorative components. They both use the Angular's `ng-content` element. The `ng-content` element allows the components to serve as a container for other HTML elements or components. The content box component allows developers to include the liberouter GUI design in new tools. It contains a header with a configurable text and a space for content under the heading. The popup component is intended for modal windows. It contains a wrapper for content, that is always on top of other content. It also provides developers with two buttons, both of which can be configured to execute a custom function when clicked. The text on the buttons can be changed as well as the title of the modal window.

- The **device selection** component: Loads a list of connected devices and displays them to a user with a checkbox next to each device. Users can select devices from connected devices by clicking the checkbox next to the device's name. After a user selects devices and clicks the "Select" button, the device selection component sends the selected device connections as an output using an event emitter. Any component that uses the device selection component can listen to the event emitter and react to selection changes.

- The **device quick switch** component: Loads devices selected with the device selection component. It allows users to switch between devices to configure. The `allowMultiple` component input sets, if users can select multiple devices at once and is set to true by default. When a user changes the device selection, the device quick switch component emits an event to the `selectionChanged` output. The event contains all selected sessions. A tool using this component can listen to the `selectionChanged` event and react to the changes. If a tool supports configuring multiple devices at once, the device quick switch component is useful for making bulk changes to the configuration of multiple devices, since users can quickly switch between device combinations. If a tool only supports configuring one device at a time, the device quick switch component makes switching between selected devices much faster, because the user only has to click once to switch a device.

- The **now connecting form** component: Displayed every time a request to connect a device is created. The component uses the device service as well as the session service. Each time a new device should be connected, the device service creates an event. The now connecting form component listens to these events and reacts with showing itself and starting the connection process. During connecting a new device, four common situations preventing a successful connection may happen: A host key check may be required, authentication may be required, a schema may be missing, or a general

connection error can happen. The now connecting form component accounts for all of these situations. If a host key check is required, this component displays two buttons to the users, allowing them to choose if they want to continue connecting or abort the connection. If authentication is required, a text box is shown to users, prompting them to enter a password for the device. If a schema is required, the schema's name and revision are displayed to the user. Users can then click on the "browse" button next to the schema's name and upload the schema from their computer, or click the "Skip schema upload" button to continue without uploading the schema. If some other error happens during the connecting of a device, the error message is shown to the user with a reconnect button. When a user clicks the reconnect button, a new request to connect to the same device is created. If all connections are successful, the now connecting form component is hidden. The finished now connecting form is in figure 4.6. The component in the screenshot is connecting to a single device, that is currently offline. Since the device is in an error state, the retry button is visible.

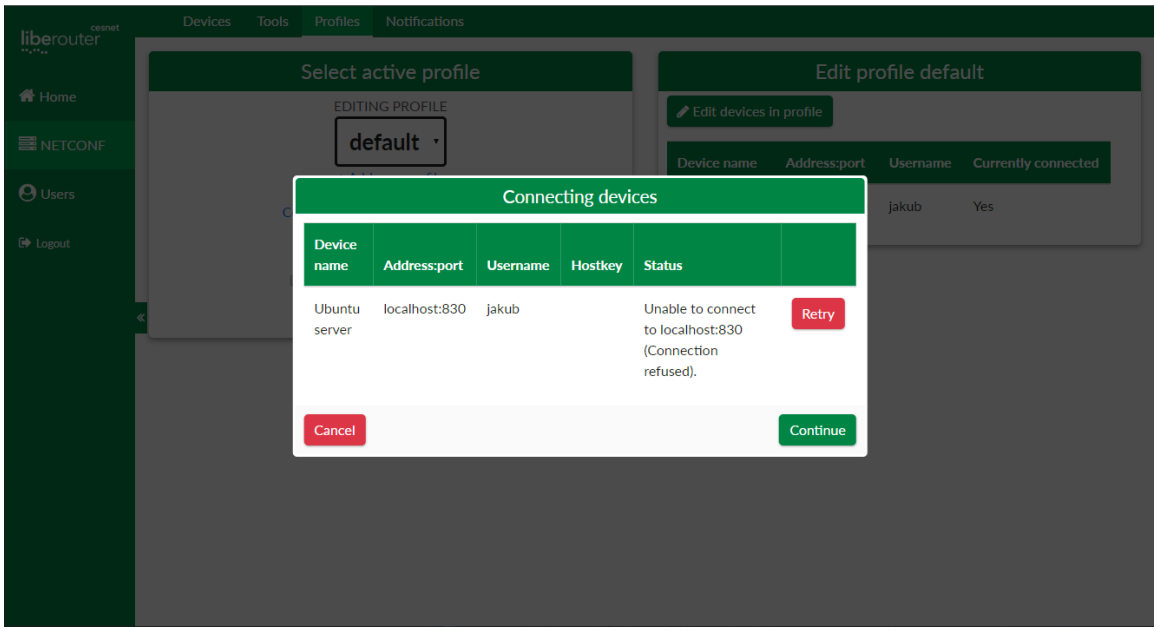

Figure 4.6: Now connecting component showing a connection error

Services

The tool library provides services for handling devices and device connections, as well as loading and handling device configuration.

The **device service** is responsible for creating device connection requests and for saving and loading saved devices from the database. The device service also contains the event emitter controlling the now connecting component. The function `createConnectionRequest` from the device service serves to add a device to the *waiting for connection* list. When the *waiting for connection* list changes, an event is emitted to summon the now connecting component and to connect the waiting devices.

The **session service** manages open connections to devices and stores loaded device configuration before the configuration is sent to the device. The session service also contains an event emitter, which outputs an event each time the list of sessions is changed. Developers can use the `sessionsChanged` event to reload components or warn user, that a device was disconnected. The sessions service also contains a function, that sends a request to the backend, to verify, that a session is still valid and the device did not disconnect.

The **schema service** handles loading and uploading YANG schemas. It also contains functions for formating YANG schemas before displaying them. The YANG schema parsing is simplified, as indicated in Figure 4.3.

The **socket service** provides functions for communicating with the backend using web sockets. During the initialization of this service, it opens a web socket connection. Functions from this service can then be called to send or listen to messages from this socket connection. Sockets are used during the device connection process because the device sends asynchronous messages if authentication or host key check is required.

4.4 Backend

The backend facilitates communication with the devices and with the saved devices database. The backend also handles user's profiles and uploaded schemas. Functions facilitating communication between the frontend and the backend are in four files:

- `init.py`: configures HTTP request paths and functions that handle these requests,
- `communications.py`: handles data from requests and calls appropriate functions,
- `sockets.py`: manages requests sent as a web socket instead of an HTTP request,
- `netconf.py`: handles requests related to sessions and device connections.

The `netconf.py` file, together with the `data.py` file, handle everything related to connecting to devices. Both of these files were heavily inspired by the Netopeer2GUI project and are used under the Apache 2.0 license.

The `devices.py` file handles saving, loading and searching devices in the database. All functions using a database connection have the database connection as a parameter. Having a database connection as a parameter helps with testing. When running tests, a new database is created, to prevent accidental modification of the production database.

All functions dealing with profiles are in the `profiles.py` file. The profiles are saved in a JSON file, and each user has a file in the "user files" folder for saving profile information. Each profile has two mandatory items: its name and whether or not it should connect on login. Profiles also contain a list of devices, that are included in the profile. Each device is a JSON object and has to contain a filed ID containing the ID of a device from the device database. The device object can contain a list of notification channels, that the user subscribed to on this device. Devices that are not saved in a database can not be included in a profile.

4.5 Fixed bugs in the libnetconf2 library

Two bugs were discovered in the *libnetconf2* library during the development of the UI. The first bug was in the Python wrapper for the `libnetconf2` library, in the `setAuthInte-`

`ractiveClb` function. The `setAuthInteractiveClb` function is responsible for sending authentication requests from the device to the frontend when no password is specified during a connection request. The first argument of this function is a function pointer to a callback, that sends the authentication request to the frontend. The second argument contains extra data, that are passed to the callback function. The bug prevented the `setAuthInteractiveClb` function to pass the second argument to the callback function. This bug prevented the frontend to detect, which device sent the authentication request since the device information was passed to the callback function in the second argument. This bug was not discovered before, because the Netopeer2GUI project, which used the *libnetconf2* Python wrapper too, could only connect to a single device at a time. Therefore distinguishing between devices when requesting a password was not needed. This bug has been reported to the development team and fixed by Radek Krejčí, who maintains the Python wrapper.

The second bug was discovered in the CMake configuration file responsible for installing the Python wrapper for the *libnetconf2* library. This bug caused the installation of the Python wrapper to fail if the path to the library source files contained a space. This bug was caused by an incorrect command in the installation script. A variable containing the path was not surrounded in quote marks, so the spaces in the file path were not properly escaped. This bug was discovered by chance when the library was built in my work directory, which contains a space in its name. I fixed this bug and submitted it to the *libnetconf2* repository as a pull request.

Chapter 5

Testing

Because the project is a UI, two types of testing were needed to test the system: unit testing and UI testing. Unit testing covers backend functions. UI testing will be done by users to test the accessibility of the UI. Because the goal of this thesis is to implement a system usable by inexperienced users, UI testing is especially important.

5.1 Test Environment

Testing was done on the Ubuntu version 18.04. Initially, the testing environment was planned to be the newest stable version at the time of writing, 20.04, but due to issues with `netopeer2-server`, version 18.04 had to be used instead. The issue with the `netopeer2-server` has been reported to the developers of the *Netopeer2* project.

Testing was done on a virtual machine with a clean installation of Ubuntu 18.04 to prevent intervention from unrelated programs during testing. The only running programs apart from the Ubuntu default daemons were MongoDB daemon, `netopeer2-server` and the backend and frontend of liberouter GUI with the Netconf module described in this thesis.

Testing was done on a virtual machine with a clean installation of Ubuntu 18.04 to prevent intervention from unrelated programs during testing. The only running programs apart from the Ubuntu default daemons were MongoDB daemon, netopeer2 server and the backend and frontend of liberouter GUI with the Netconf module described in this thesis. The Python programming language was in version 3.6.9, NodeJS in version 8.10.0 and npm in version 6.14.5.

To help setting up an environment similar to the testing environment, a Vagrantfile is included in the vagrant folder of the project. The Vagrant virtualization software uses the Vagrantfile configuration file to start a virtual machine and install all required libraries and programs on it. Users can then start the whole project by running the `vagrant up` command in the vagrant directory.

5.2 Unit testing

The backend functionality was tested using the Python unit test library. The tests can be run by executing the `runtest.py` file, which can be found in the backend folder.

Before starting the tests, a new database connection is established. Creating a new database connection prevents accidental modification of the production database during

the testing process. Before the testing process starts, three devices are inserted into the testing database. After the tests are done, all content is removed from the database, to prevent past tests affecting current test results.

Two files from the backend are covered by tests: `devices.py` and `profiles.py`. Developing functions to handle profiles and saved devices was quicker when using unit tests than when testing the functionality using frontend because these functions can be easily tested using automated tests.

Functions handling HTTP requests or working with external data were not covered by unit test, because writing tests for these functions was more complicated than using frontend to test their behaviour.

5.3 User interface testing

The first time that the UI was tested was during the design process. While designing the prototype, the design was discussed with potential users and edited based on their feedback. During the development process, the UI was tested to confirm basic functionality.

The netopeer2 server served as a testing NETCONF device. During the development process, a problem with the Windows Subsystem for Linux (WSL) caused the `netopeer2-server` to fail each time it received an RPC request. The WSL allows users to use a Linux terminal inside of the Windows operating system, and it was up until the `netopeer2-server` failure used to run the whole project. `Netopeer2-server` not working on WSL caused delay in development since the `netopeer2-server` had to be re-installed on a Ubuntu virtual machine. It also took some time to diagnose this error.

Because some of the features are not implemented in the UI yet, larger user testing was not done yet. A user testing with potential users will be done after more features are added.

Chapter 6

Conclusion

This thesis provides an overview of the NETCONF protocol and some of its implementations and overview of the Angular framework. Studying this topic was followed by designing a web UI to configure devices using the NETCONF protocol and implementation of the UI. The main requirement was the implementation of a plugin system, allowing developers to create tools, which simplify working with device configurations.

After studying a prototype was designed. First, a wireframe was drawn on a paper with a pencil, then a prototype was designed in the Adobe XD prototyping tool. The prototype demonstrated the functionality to other people, allowing me to make changes based on their feedback before implementing the UI. Then a list of features was discussed with people experienced with using NETCONF to configure devices. While some features were deemed necessary and it was settled that those features will be implemented as soon as possible, other features were decided to be implemented in future updates.

When the list of features and the design were decided, an implementation phase started. I used the liberouter GUI framework to implement the project. The implementation is divided into three parts: frontend, backend and tools. Frontend and tools are implemented in Angular, backend in Python. The *libnetconf2* library is handling the NETCONF connections to devices, and the *libyang* library is responsible for working with the YANG schemas.

The devices tab was implemented exactly as in the prototype. Design of the tools tab is also implemented as designed. The YANG Explorer tool was implemented, the YANG Configure tool was implemented in a simplified version, as it does not yet allow modifying the configuration tree structure, it only is capable of displaying the configuration in a tree structure and changing node values. The profiles tab was modified for better usability. The profile system was implemented as designed. A plugin system was implemented, allowing users to add new tools without having to recompile the whole application. The UI can connect to devices, both to those saved in a database and to new devices, and load their configuration.

Due to time constraints, the notification system and the colour picker tool were not implemented. These features are planned to be implemented soon.

The UI can be improved in many ways. The most crucial for further development is implementing a better system for handling device configuration in the tool library. Currently, only basic operations are supported. The YANG Configure tool can be improved to support configuring multiple devices at once and to better handle the configuration data, minimizing the use of the backend.

A notification system can be implemented to allow users to subscribe to notifications generated by devices. The design for the notification system is described in this thesis. A notification service is already implemented in the frontend, but it is currently not being used, because a backend implementation of notifications is missing.

The plugin system allows developers to implement tools for configuring NETCONF enabled devices much easier than if they had to create a new application for their use case. In the future, more tools can be implemented for various device types and specific uses. The colour picker tool can be implemented to demonstrate a use case to inspire other developers to use the solution developed in this thesis instead of them making their own. Currently, a new Python wrapper for the libyang library is being developed. After the new wrapper is tested and stable, it should replace the current Python wrapper.

Bibliography

[1] Adobe. *UI/UX design and collaboration tool | Adobe XD.* Accessed: 2020-05-10. Available at: https://www.adobe.com/products/xd.html.

[2] Angular. *Angular - Ahead-of-time (AOT) compilation.* Accessed: 2020-04-12. Available at: https://angular.io/guide/aot-compiler.

[3] Bjorklund, M. and Systems, T. f. *YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)* [Proposed standard]. RFC 6020. RFC Editor, October 2010. Available at: https://www.rfc-editor.org/rfc/rfc6020.txt.

[4] Bjorklund, M. and Systems, T. f. *The YANG 1.1 Data Modeling Language* [Proposed standard]. RFC 7950. RFC Editor, August 2016. Available at: https://www.rfc-editor.org/rfc/rfc7950.txt.

[5] Bjorklund, M., Tail-f Systems, Schoenwaelder, J. and Jacobs University. *Network Management Datastore Architecture (NMDA)* [Proposed standard]. RFC 8342. RFC Editor, March 2018. Available at: https://www.rfc-editor.org/rfc/rfc8342.txt.

[6] Cesnet. *Libnetconf2.* Available at: https://github.com/CESNET/libnetconf2.

[7] Cesnet. *Libyang.* Available at: https://github.com/CESNET/libyang.

[8] Cesnet. *Libyang: XPath Addressing.* Accessed: 2020-05-06. Available at: https://netopeer.liberouter.org/doc/libyang/devel/howtoxpath.html.

[9] Cesnet. *Netopeer2GUI.* Available at: https://github.com/CESNET/Netopeer2GUI.

[10] Claise, B., Clarke, J. and Lindblad, J. *Network programmability with YANG: the structure of network automation with YANG, NETCONF, RESTCONF, and gNMI.* Addison-Wesley, 2019.

[11] Enns, R., Bjorklund, M., Juniper Networks, Tail-f Systems, Schoenwaelder, J. et al. *Network Configuration Protocol (NETCONF)* [Proposed standard]. RFC 6241. RFC Editor, June 2011. Available at: https://www.rfc-editor.org/rfc/rfc6241.txt.

[12] Fette, I., Google, Melnikov, A. and Isode. *The WebSocket Protocol* [Proposed standard]. RFC 6455. RFC Editor, December 2011. Available at: https://www.rfc-editor.org/rfc/rfc6455.txt.

[13] Huawei Technologies Co., L. *CloudEngine 12800 and 12800E Configuration Guide - Netconf capabilities exchange*. Accessed: 2020-01-28. Available at: https://support.huawei.com/enterprise/en/doc/EDOC1100039539/8b7d49ad/netconf-capabilities-exchange.

[14] Kitware. *Overview | CMake*. Accessed: 2020-05-19. Available at: https://cmake.org/overview/.

[15] OpenJS Foundation. *What is npm? | Node.js*. Accessed: 2020-05-14. Available at: https://nodejs.org/en/knowledge/getting-started/npm/what-is-npm/.

[16] Pallets. *Flask Documentation*. Accessed: 2020-05-14. Available at: https://flask.palletsprojects.com/en/1.1.x/.

[17] R. Enns, E. *NETCONF Configuration Protocol* [Internet Requests for Comments]. RFC 4741. RFC Editor, December 2006. Available at: https://www.rfc-editor.org/rfc/rfc4741.txt.

[18] Zuev, A. *Building an extensible Dynamic Pluggable Enterprise Application with Angular*. Apr 2019. Accessed: 2020-01-20. Available at: https://medium.com/angular-in-depth/building-extensible-dynamic-pluggable-enterprise-application-with-angular-aed8979faba5.

List of Acronyms

UI User Interface

GUI Graphical User Interface

RPC Remote Procedure Call

SNMP Simple Network Management Protocol

CLI Command-line interface

MIB Management information base

AOT Ahead-of-Time

JIT Just-in-Time

URI uniform resource identifier

npm node package manager

WSL Windows Subsystem for Linux

Appendix A

Contents of the Attached CD

Following directories and files can be found on the CD:

- Directory `src` – A directory containing all project source files.
- Directory `vagrant` – A directory containing a Vagrantfile, that can be used to start the application using Vagrant by HashiCorp.
- Directory `text` – A directory containing LaTeX source files for this thesis.
- File `README.md` – A file describing the project and installation instructions
- File `xmanja00.pdf` – An electronic version of this thesis.