VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

ROZŠÍŘENÍ UŽIVATELSKÉHO ROZHRANÍ NÁSTROJE NETBOX

NETBOX TOOL USER INTERFACE EXTENSION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **VIKTOR KUBEC**
AUTHOR

VEDOUCÍ PRÁCE **Ing. JIŘÍ MATOUŠEK, Ph.D.**
SUPERVISOR

BRNO 2025

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

161589

Ústav:	Ústav počítačových systémů (UPSY)
Student:	**Kubec Viktor**
Program:	Informační technologie
Název:	**Rozšíření uživatelského rozhraní nástroje NetBox**
Kategorie:	Uživatelská rozhraní
Akademický rok:	2024/25

Zadání:

1. Seznamte se s nástrojem NetBox určeným pro modelování a dokumentování síťové infrastruktury. Zaměřte se především na možnosti rozšíření jeho uživatelského rozhraní.
2. Analyzujte požadavky sdružení CESNET na rozšíření uživatelského rozhraní nástroje NetBox a rozdělte je do kategorií „nezbytné“ a „vhodné.“
3. Navrhněte rozšíření uživatelského rozhraní nástroje NetBox, které naplní všechny „nezbytné“ požadavky a umožní další snadné rozšíření pro naplnění „vhodných“ požadavků.
4. Implementujte navržené rozšíření uživatelského rozhraní nástroje NetBox.
5. Zdokumentujte implementované rozšíření uživatelského rozhraní nástroje NetBox. Podrobně se zaměřte na možnosti jeho dalšího rozšíření pro naplnění „vhodných“ požadavků.
6. Zhodnoťte dosažené výsledky práce a diskutujte možnosti jejího dalšího pokračování.

Literatura:

- Dle pokynů vedoucího práce.

Při obhajobě semestrální části projektu je požadováno:

- Splnění bodů 1 až 3 zadání.

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

Vedoucí práce:	**Matoušek Jiří, Ing., Ph.D.**
Konzultanti:	Ing. David Vodák
	Ing. Martin Špinler
Vedoucí ústavu:	Sekanina Lukáš, prof. Ing., Ph.D.
Datum zadání:	1.11.2024
Termín pro odevzdání:	14.5.2025
Datum schválení:	31.10.2024

Abstrakt

Tato bakalářská práce se věnuje rozšíření uživatelského rozhraní nástroje NetBox, který představuje jedno z nejrozšířenějších open-source řešení pro správu IP adres (IPAM) a infrastruktury datových center (DCIM). NetBox poskytuje velké množství funkcí pro evidenci síťových zařízení, správu adresního prostoru nebo dokumentaci fyzické topologie, avšak jeho standardní uživatelské prostředí nemusí vždy plně vyhovovat specifickým potřebám síťových administrátorů či IT specialistů. Práce se proto zaměřuje na úpravu uživatelského rozhraní podle požadavků oddělení nástrojů pro administraci a bezpečnost sdružení CESNET. Tyto požadavky zahrnují rozšíření NetBoxu o dosud chybějící funkce, které zaměstnancům CESNETu výrazně usnadní každodenní práci. Klíčovou součástí práce je analýza současných možností nástroje NetBox, identifikace potřebných vylepšení a následný návrh pluginů, jejichž cílem je zjednodušení rutinních úkonů při práci s aplikací. V teoretické části je nejprve představen koncept NetBoxu, popsána jeho architektura založená na frameworku Django, shrnuty možnosti jeho rozšiřování, požadavky CESNETu na rozšíření a analýza těchto požadavků. Praktická část práce se věnuje návrhu a implementaci rozšíření, problémům při vývoji a radám pro ty, kteří by mohli mít zájem o rozšíření NetBoxu. Závěr shrnuje dosažené výsledky, přínos práce a doporučení pro další vývoj.

Abstract

This bachelor's thesis focuses on extending the user interface of the NetBox tool, one of the most widely adopted open-source solutions for IP address management (IPAM) and data-center infrastructure management (DCIM). NetBox provides a wealth of features for recording network devices, managing address space, and documenting physical topology, yet its stock user interface does not always satisfy the specific needs of network administrators or IT specialists. Accordingly, this work concentrates on adapting the user interface to the requirements of the Administration and Security Tools Department of the CESNET association. Those requirements include the addition of functionality that is currently missing from NetBox and that will significantly streamline the daily work of CESNET's staff. A key component of the thesis is an analysis of NetBox's existing capabilities, the identification of needed improvements, and the subsequent design of plugins aimed at simplifying routine tasks within the application. In the theoretical part, we begin by introducing the NetBox concept and detailing its Django-based architecture. We then outline the available extension mechanisms, present the specific enhancement requests from CESNET, and analyze those requirements. Practical part covers the design and implementation of the plugins, discusses the development challenges encountered, and offers useful tips for others who wish to extend NetBox. Conclusion summarizes the outcomes and contributions of this work and provides recommendations for future development.

Klíčová slova

NetBox, Django, IPAM, DCIM, uživatelské rozhraní, správa sítě, plugin, rozšíření

Keywords

NetBox, Django, IPAM, DCIM, user interface, network management, plugin, extension

Citace

KUBEC, Viktor. *Rozšíření uživatelského rozhraní nástroje NetBox*. Brno, 2025. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Jiří Matoušek, Ph.D.

Rozšíření uživatelského rozhraní nástroje NetBox

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Jiřího Matouška, Ph.D. Další informace mi poskytli pánové Ing. Martin Špinler a Ing. David Vodák ze sdružení CESNET. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal. Ještě bych chtěl dodat, že jsem využíval generativní umělé inteligence a to konkrétně k refaktorizaci textu a rychlejšímu vyhledávání informací.

. . . . . .
Viktor Kubec
12. května 2025

Poděkování

Rád bych poděkoval vedoucímu práce panu Ing. Jiřímu Matouškovi, Ph.D. za jeho podporu, vedení a poskytování četných konzultací v průběhu tvorby mé bakalářské práce. Také bych rád poděkoval pánům Ing. Martinu Špinlerovi a Ing. Davidu Vodákovi ze sdružení CESNET za jejich poskytnutou pomoc, konzultace a vstřícnost. Rád bych ještě poděkoval mému otci za jeho cenné rady a podporu.

Obsah

Seznam obrázků

Kapitola 1

Úvod

Udržitelné a efektivní řízení rozsáhlých síťových prostředí se v posledních letech stává stále náročnější, a to zejména vzhledem k rozšíření cloudových technologií, virtualizace a rychle se měnících požadavků koncových uživatelů. NetBox, vyvíjený primárně v jazyce Python nad frameworkem Django, představuje v současné době jedno z předních open-source řešení pro správu síťových prvků. Je používán k přehledné evidenci adresního prostoru (IP Address Management, IPAM), zařízení a topologií (Data center infrastructure management, DCIM), přičemž jim poskytuje jednotný pohled na celou infrastrukturu. Kvůli svému rozšiřitelnému charakteru se stále častěji nasazuje i ve firmách, jež se doposud spoléhaly na vlastní nedostačující nástroje, tabulky, nebo dokumenty.

Výchozí uživatelské rozhraní NetBoxu však nemusí vždy pokrýt specifické potřeby organizací. V reálném provozu se často vyžaduje rychlejší přístup ke klíčovým informacím, pokročilé filtrování nebo intuitivnější vizualizace zařízení a síťových segmentů. Proto řada firem zvažuje tvorbu doplňků (tzv. pluginů) či přímo úpravu kódu uživatelského rozhraní. Tyto úpravy by usnadnily každodenní práci a administrativu. Nedostatečná uživatelská přívětivost může vést v praxi k delší době potřebné pro nalezení relevantních údajů, zvýšenému riziku vzniku chyb, a tím i ke zvýšení nákladů na provoz a správu sítě.

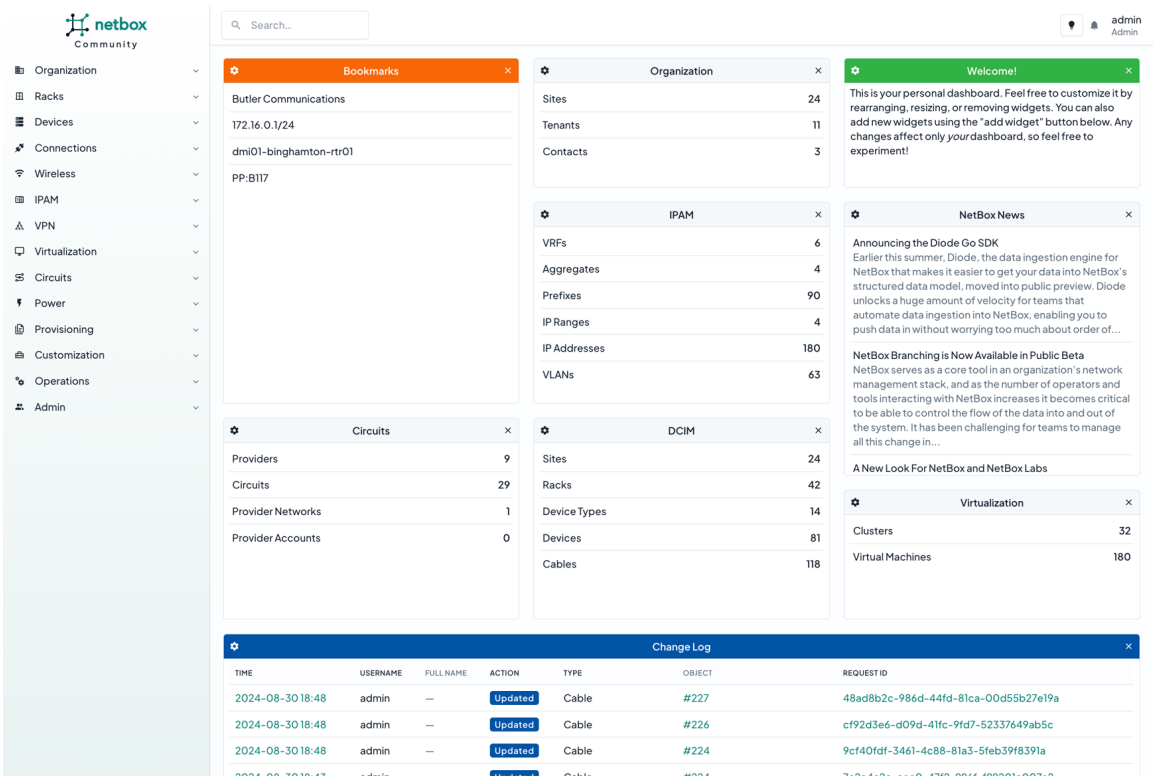
Cílem této bakalářské práce bylo provést návrh a implementaci rozšíření uživatelského rozhraní splňujícího požadavky oddělení nástrojů pro administraci a bezpečnost sdružení CESNET. V první fázi bylo nutné pochopit architekturu NetBoxu, jeho hlavní komponenty a možnosti, jak nástroj rozšířit efektivně a s ohledem na udržitelnost kódu. Následně bylo třeba provést analýzu požadavků od CESNETu a rozhodnout, která rozšíření jsou nezbytná a která pouze vhodná a mohou se implementovat nad rámec této práce. Po tomto seznámení a analýze bylo možné provést návrh pluginů. Bylo třeba navrhnout co jednotlivé pluginy budou mít za funkce, co bude třeba přidat do rozhraní a jak kód psát co nejvíc modulárně a přehledně. Pak bylo možné začít implementovat konkrétní rozšíření.

V kapitole 2 je představen NetBox. V kapitole 3 je popsán CESNET, požadavky na rozšíření NetBoxu a analýza požadavků. V kapitole 4 je popsán návrh architektury řešení, požadavky na nové prvky uživatelského rozhraní a návrh řešení jednotlivých požadavků. V kapitole 5 je popsána implementace jednotlivých rozšíření, problémům při vývoji a radám jak s vývojem začít. V kapitole 6 je shrnutí dosažených výsledků, jejich přínos a možnosti, jak dále nástroj NetBox rozšiřovat v budoucnu.

Kapitola 2

Přehled NetBoxu a jeho architektury

NetBox je open-source platforma vyvíjená v jazyce Python nad webovým frameworkem Django [7], určená primárně pro správu IP adresního prostoru (IPAM) a zařízení v datových centrech (DCIM). NetBox původně vyvinul Jeremy Stretch jako interní nástroj ve společnosti DigitalOcean. Poté co byl v červnu 2016 uvolněn pod open-source licencí jej dál rozvíjí komunita. Jejich cílem je vytvořit jednotné řešení pokrývající potřebu přesné evidence síťových prvků, zaznamenávání jejich fyzické i logické topologie a usnadnění správy IP adres. Projekt získal popularitu i díky relativně přívětivému uživatelskému rozhraní. To dokáže zjednodušit každodenní administrativu správců sítí. [17, 18]



Obrázek 2.1: Ukázka výchozího vzhledu NetBoxu.

2.1 Hlavní oblasti využití

NetBox se využívá především tam, kde je potřeba přehledné správy síťové infrastruktury, efektivní dokumentace a vizualizace vztahů mezi jednotlivými zařízeními. K hlavním oblastem využití patří:

Správa IP adres (IPAM)
NetBox poskytuje nástroje pro sledování a organizaci celého IP adresního prostoru. Dokáže zaznamenat nejen konkrétní IP adresy, ale i sítě (prefixy), VLANy (Virtual Local Area Network), VRF (Virtual Routing and Forwarding) a související objekty [17]. Správci tak mohou snadno zjistit přiřazené IP adresy, volné rozsahy nebo vzájemné překrývání jednotlivých segmentů sítí.

Správa infrastruktury datových center (DCIM)
Druhou klíčovou oblastí je správa fyzických zařízení a jejich umístění v rackových skříních, včetně kabeláže, napájecích zdrojů a dalších komponent [17]. NetBox umožňuje zaznamenávat i detailní informace o výrobcích a modelech zařízení, sériových číslech, záručních lhůtách či datech instalace. Tím přispívá k udržení aktuální a spolehlivé inventarizace i ve velkých a složitých prostředích.

Vizualizace topologie
NetBox umožňuje detailní evidenci a správu síťových zařízení, rozhraní, kabelových propojení a jejich fyzického zapojení. Uživatel může v rozhraní NetBoxu například:

- Přehledně sledovat obsazené porty a připojené kabely.
- Pomocí funkce *Cable Trace* zobrazit kompletní trasu propojení od počátečního portu až k cílovému.
- Využít *rack diagramy* znázorňující fyzické umístění zařízení v jednotlivých rackových skříních.

V základní instalaci ovšem NetBox nenabízí automatické generování komplexních topologických diagramů ani interaktivních map sítě (zejména na L2 a L3 vrstvě). Pro pokročilou vizualizaci síťové topologie je možné využít komunitní rozšíření, například plugin **NetBox Topology Views** [29], který z dat uložených v NetBoxu generuje přehledné grafické diagramy. Kvůli tomu je možné rychle identifikovat slabá místa nebo SPOF (Single Points of Failure), v síti. [17]

Plánování a dokumentace
NetBox nabízí velké množství možností přidávání metadat a vytváření vlastních poznámek ke každému objektu v systému [17]. Kvůli tomu může sloužit nejen k dokumentaci aktuálního stavu sítě, ale také jako detailní zdroj informací, kde lze snadno dohledat důvody pro konkrétní konfiguraci či historii provedených změn.

2.2 Výhody NetBoxu

Před nasazením NetBoxu je vhodné seznámit se nejen s oblastmi jeho využití, ale také s klíčovými výhodami, činícími z něj atraktivní řešení. Mezi jeho hlavní přínosy patří otevřenost systému, široká komunitní podpora a vysoká flexibilita, umožňující snadnou adaptaci v různorodých provozních podmínkách.

Open-source
Zdrojový kód NetBoxu je dostupný na GitHubu [18] pod licencí Apache 2.0 [2]. Komunita může opravovat chyby, vyvíjet nové pluginy nebo dále pokračovat ve vývoji NetBoxu. NetBox si tak může každý přizpůsobit dle specifických potřeb.

Jednotné řešení
V minulosti správa IP adres nebo dokumentace DCIM obvykle vyžadovala kombinaci různých nástrojů, například tabulkových procesorů, interních skriptů či specializovaných uzavřených systémů. NetBox tuto fragmentaci řeší sjednocením do jednoho centrálního systému, kde se data vzájemně propojují, což výrazně usnadňuje jejich správu [17].

Komunitní podpora
Jelikož má NetBox velkou uživatelskou základnu, existuje spousta zdrojů, v nichž lze snadno nalézt odpovědi na běžné problémy či návody k nasazení. K dispozici jsou také oficiální diskusní fóra a chatovací kanály (např. Slack), kde lze získat pomoc.[18].

Flexibilita nasazení
NetBox lze instalovat jak na fyzické servery a virtuální stroje, tak i do kontejnerizovaných prostředí (Docker) [17]. Díky tomu je vhodný pro malé i rozsáhlé podnikové infrastruktury, ať už provozované interně, nebo v cloudu.

2.3 Nevýhody NetBoxu

Přestože NetBox nabízí řadu výhod, je třeba zohlednit i jeho slabé stránky. Tyto nevýhody mohou ovlivnit rozhodnutí o jeho nasazení. Níže jsou uvedeny hlavní body:

Strmá křivka učení a méně intuitivní uživatelské rozhraní
Orientace v NetBoxu a správné nastavení datových modelů vyžaduje pokročilé znalosti síťových konceptů i samotné aplikace. Grafické uživatelské rozhraní (GUI) není zcela intuitivní. V některých případech je třeba prostudovat dokumentaci, vyhledat tutoriál nebo se obrátit na komunitní fóra [14]. U tutoriálů se stává, že nejsou vždy aktuální a mohou obsahovat zastaralé informace.

Komplexní instalace, údržba a proces upgradů
Provozování NetBoxu znamená také správu podpůrných komponent (PostgreSQL [23], Redis [27], Gunicorn [9], Celery [4]). Každá aktualizace vyžaduje kontrolu závislostí, migraci databáze a synchronizaci verzí pluginů.

Výkonnostní limity
Při správě desítek tisíc zařízení nebo při hromadných voláních API může docházet k výraznému zpomalení odezvy systému a vysokému zatížení CPU. [15, 16]

Citlivost na kompatibilitu pluginů
Přestože pluginy jsou silnou stránkou NetBoxu, každá nová hlavní verze může narušit funkčnost neaktualizovaných modulů nebo způsobit chyby ve vykreslování stránek [17]. Z tohoto důvodu vznikl certifikační program NetBox Labs. Tento program má za úkol stabilitu pluginů zlepšit, ale současně klade vyšší nároky na jejich údržbu [22].

2.4 Struktura a architektura NetBoxu

Architektura NetBoxu vychází z modelu klient–server, přičemž hlavní část logiky je realizována na straně serveru pomocí Django frameworku [7]. Vysoká modularita a přehledná hierarchie objektů umožňuje NetBoxu flexibilně reagovat na změny v infrastruktuře a zároveň usnadňuje další rozvoj či integrace s jinými systémy [17].

Hlavní komponenty

Webová aplikace (Django)

NetBox je vyvíjen v jazyce Python s využitím frameworku Django, který poskytuje strukturovaný přístup k tvorbě webových aplikací [7]. Django se stará o mapování URL adres na konkrétní funkce (views), které mají za úkol zpracovat požadavky uživatele a vrátit mu odpovídající HTML šablonu nebo API odpověď. Díky Django frameworku je k dispozici vestavěná administrace, která zjednodušuje úpravu modelů a jejich správu. Samotný NetBox však nabízí vlastní uživatelské rozhraní přizpůsobené potřebám datacenter a síťových specialistů [17].

Databázová vrstva

Oficiálně NetBox podporuje zejména PostgreSQL [23], která je zároveň díky svým schopnostem například full-text vyhledávání, JSONB pole doporučenou databází. Přestože existují návody, jak NetBox zprovoznit i s jinými databázemi, může to vyžadovat neoficiální úpravy a uživatel může narazit na nekompatibility. Databázová schémata v NetBoxu odrážejí klíčové entity, jako jsou Device, Rack, Site, IPAddress, VLAN atd. Modelová struktura je poměrně komplexní. Pokrývá široké spektrum objektů, od fyzického umístění serverů po virtuální logické segmenty či VLAN [17].

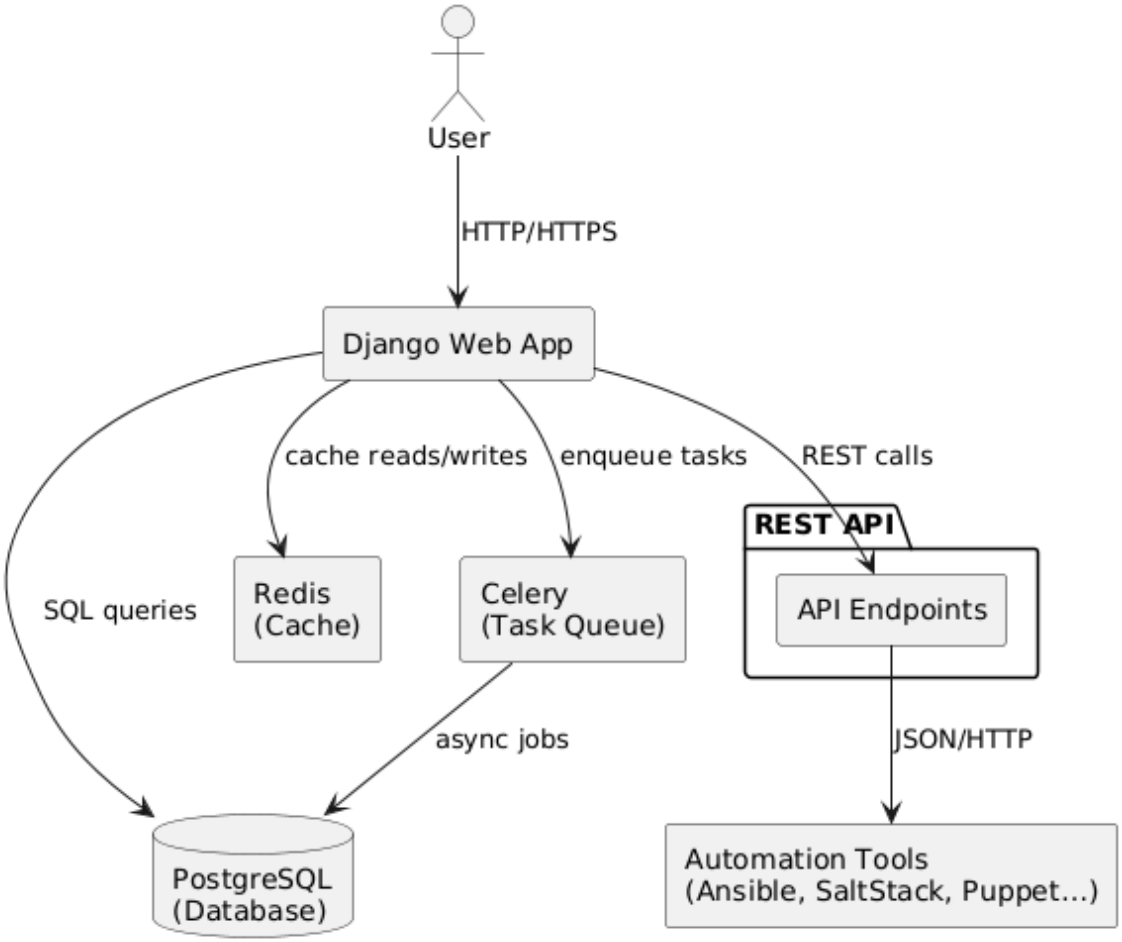
Kešování a fronty (volitelné komponenty)

Pro zvýšení výkonu lze využít keš například Redis [27], uchovávající nejčastěji dotazovaná data. Snižuje tak zatížení databáze [17]. Ve větších nasazeních bývají užitečné také fronty pro asynchronní zpracování dlouhotrvajících úloh například při generování reportů. Django pro tyto účely často využívá Celery [4].

REST API

NetBox je od počátku navržen tak, aby mimo webové rozhraní nabízel také RESTful API [20]. To umožňuje dalším aplikacím či skriptům rozumně přistupovat k datům v NetBoxu, vytvářet záznamy, aktualizovat je či je mazat. Tento přístup je zásadní pro integraci s nástroji pro automatizaci sítě například Ansible [1], SaltStack [28], Puppet [25], Chef [6] aj. a podporuje tzv. Infrastructure as Code, kdy se infrastruktura spravuje pomocí deklarativních konfiguračních souborů a NetBox slouží jako jediný zdroj pravdy (single source of truth) [17].

Na obrázku 2.2 je diagram znázorňující vztahy mezi hlavními komponentami NetBoxu. Na obrázku jdou vidět toky požadavků mezi uživatelem a Django webovou aplikací, její propojení s PostgreSQL databází, cache vrstvou Redis, asynchronními úlohami přes Celery a REST API integrací s nástroji pro automatizaci.



Obrázek 2.2: Diagram znázorňující vztah hlavních komponent.

Datový model a klíčové objekty

V NetBoxu je poměrně velké množství typů objektů, se kterými lze pracovat. Mezi hlavní patří:

Site
Základní prvek pro popis geografické lokality či obecného místa (budova, pobočka, datové centrum) [17].

Rack
Kontejner pro fyzické umístění zařízení v rámci Site. Zahrnuje informaci o rozměrech, kapacitě a dalších detailech [17].

Device a Device Role
Fyzické zařízení či virtuální stroj. Device Role pak charakterizuje účel (např. router, switch, server) [17].

Interface
Popisuje fyzické či virtuální rozhraní zařízení, jeho rychlost, MAC adresu, případnou vazbu na VLAN [17].

IP Address a Prefix
Slouží k popisu konkrétních IP adres a adresních rozsahů, které mohou být přiděleny jednotlivým rozhraním či síťovým segmentům [17].

VLAN, VRF
Logické vrstvy sítě, které umožňují oddělení provozu, případně vícenásobné použití stejného adresního prostoru v různých virtuálních kontextech [17].

Cable
Umožňuje mapovat fyzické propojení mezi porty na zařízeních či zásuvkami, což pomáhá sledovat skutečnou kabeláž a usnadňuje případné zásahy při poruše [17].

Díky tomuto rozsáhlému modelu lze v NetBoxu velmi přesně zachytit stav i konfiguraci velkých sítí a datacenter. Jednotlivé objekty navzájem vytvářejí vazby (např. Interface může být propojeno kabelem s jiným Interface), díky tomu pak lze v systému rychle dohledat jednotlivé souvislosti [17].

Logika a vrstvy v Django frameworku

Architekturu lze rozdělit do následujících vrstev, které se v Django terminologii často označují jako Model-View-Template (MVT) [7]:

Model (datová vrstva)
Obsahuje definice tříd, mapující se přímo na databázové tabulky například Device, Site. Zahrnuje validace, definici vztahů (OneToOne, ManyToOne, ManyToMany), dále metadata a pokročilejší logiku například signály při ukládání objektů [7].

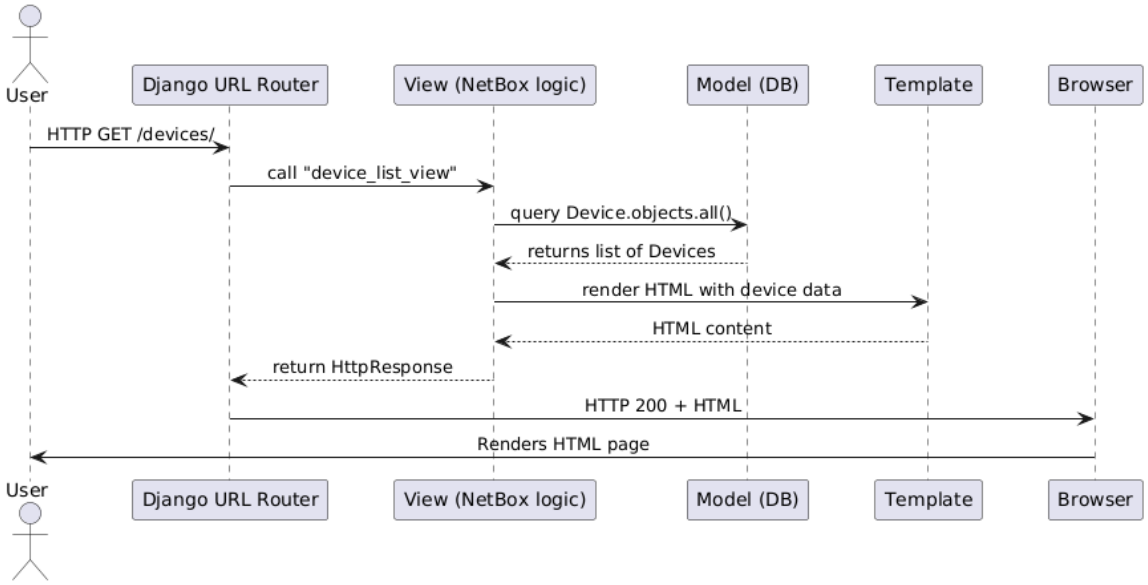
View (logická vrstva aplikace)
V Django kontextu představují funkce či třídy zpracovávající požadavky. Řídí načtení potřebných dat z modelů a volbu šablony, která se má vykreslit [7]. V NetBoxu existuje řada tzv. generic views, které ulehčují operace typu „vytvoření nového objektu“, „detail objektu“ či „vypsání všech objektů daného typu“ [17].

Template (prezentační vrstva)
HTML šablony, které zajišťují vizuální vzhled a zobrazování dat uživateli. NetBox využívá Django template engine, který umožňuje i sofistikované operace, jako je cyklování nad seznamy, práce s podmínkami či dědičnost šablon [7]. Pro styling NetBox obvykle využívá framework Bootstrap [3].

API (serializace, viewsety)
Logiku pro zpracování HTTP metod (GET, POST, PUT, DELETE) poskytuje Django REST Framework (DRF). V NetBoxu existují tzv. serializery, jež definují, jak se modelové objekty transformují do formátu JSON, a viewsets svázané s konkrétními endpointy [20].

Autentizace a oprávnění
Součástí architektury je i propracovaný systém oprávnění. Lze definovat, kteří uživatelé mohou číst, vytvářet nebo mazat data [17]. Pro integraci s podnikovým prostředím je dostupná autentizace přes LDAP nebo SSO, což usnadňuje správu uživatelských účtů v organizacích.

Na obrázku 2.3 je sekvenční diagram, který ilustruje, jak v rámci Django aplikace (zde NetBox) probíhá zpracování jednoho HTTP požadavku.



Obrázek 2.3: Sekvenční diagram Django MVT architektury.

2.5 Možnosti rozšíření

Jednou z hlavních výhod NetBoxu je jeho otevřenost vůči dalším úpravám a integracím [17, 21]. Pro uživatele i vývojáře je k dispozici několik cest, jak NetBox přizpůsobit konkrétním potřebám, aniž by museli hluboce zasahovat do jádra aplikace. Tyto pluginy se instalují samostatně a v konfiguračním souboru NetBoxu se aktivují pomocí deklarace v sekci PLUGINS. Velkou předností je, že i při aktualizacích jádra NetBoxu zůstávají pluginy oddělené a riziko konfliktů je tak minimalizováno, pokud se dodržují oficiální zásady tvorby pluginů [21].

Pluginy a jejich význam

Od verze 2.8 NetBox oficiálně podporuje koncept pluginů [21], což je mechanismus umožňující přidávat do aplikace nové funkce či upravovat stávající chování. Pluginy typicky:

Přidávají vlastní datové modely
Může jít o doplňující objekty, které NetBox neřeší nativně například sledování specifických hardwarových komponent, licencí, smluv či výstražných stavů.

Rozšiřují administrativní rozhraní
Plugin může obohatit menu o nové položky, přidávat formuláře pro zadávání dat či vytvářet nové pohledy (views).

Integrují NetBox s jinými systémy
Častým využitím je synchronizace dat s externími službami, například monitoring, helpdesk, inventurní systémy nebo CI/CD pipeliny.

Upravují existující objekty
Některé pluginy umožňují přidávat tzv. custom fields (vlastní pole) do standardních objektů či modifikovat jejich zobrazování.

Realizují speciální logiku
Například definují signály, které se spustí při vytváření, úpravě nebo mazání objektů, a následně provedou určitý úkon například odeslání notifikace.

Custom Scripts a schopnost automatizace

Mimo pluginy nabízí NetBox funkci Custom Scripts, která umožňuje jednorázově či pravidelně spouštět vlastní Python kód přímo v kontextu NetBoxu [17]. Tato funkce nachází uplatnění v následujících oblastech:

Hromadné operace
V případě, že je potřeba naimportovat velký objem nových zařízení či provést změnu ve všech VLAN, lze to řešit skriptem, který provede úpravy bez nutnosti manuálního klikání v GUI.

Synchronizace s externími zdroji
Pomocí custom skriptů lze načítat data z CSV, Excelu nebo REST API jiného nástroje například firewallu a integrovat je do NetBoxu.

Kontrolní mechanismy
Custom skripty mohou pracovat jako validátory konfigurací, hledat a upozorňovat na rozpory v záznamech například duplicity IP adres.

Výhodou tohoto přístupu je, že není nutné připravovat celý plugin, pokud jde o jednorázový či méně rozsáhlý úkon. Custom Scripts lze navíc snadno sdílet mezi uživateli NetBoxu, například prostřednictvím komunitních repozitářů [17].

Úprava vzhledu

Uživatelské rozhraní NetBoxu lze do určité míry přizpůsobit i po vizuální stránce [17]. Standardně NetBox využívá pro stylyzaci Bootstrap [3] , díky čemuž je možné měnit barvy, fonty či rozložení stránek. Pro většinu nasazení postačuje základní styl, avšak organizace, které si zakládají na korporátní identitě, mohou vytvořit vlastní styl a branding. Tato přizpůsobení se obvykle řeší pomocí:

- **Overridu šablon**: V kořenovém adresáři aplikace lze definovat alternativní Django šablony, které se použijí místo výchozích.
- **Vlastní CSS/JS**: Přidáním vlastních kaskádových stylů a JavaScript kódu je možné upravit drobné detaily, jako jsou layouty tabulek, barvy tlačítek nebo ikony.

Integrace s nástroji pro správu a automatizaci sítě

NetBox se díky svému REST API [20] často stává single source of truth pro nástroje jako Ansible [1], SaltStack [28] či Terraform [10], které se starají o deployment a konfiguraci síťových zařízení. Typickým scénářem je, že správce nadefinuje všechny sítě, VLAN, IP adresy a zařízení v NetBoxu. Automatizační systém pak tato data stahuje přes REST API a generuje konfigurace pro routery, switche či virtuální stroje. Jakékoliv změny provedené v NetBoxu se tímto způsobem mohou projevit i na fyzické infrastruktuře [17, 20]. Mezi hlavní výhody tohoto přístupu patří:

- **Eliminace chyb z duplikovaného zadávání dat**: Veškeré informace o infrastruktuře se spravují na jednom místě.
- **Zvýšení konzistence**: Aktualizace dat v NetBoxu se projeví v konfigurace produkčního prostředí [20].
- **Zrychlení procesů**: Namísto ruční editace konfiguračních souborů se vše řeší formou skriptů, což je klíčové ve velkých prostředích s velkým počtem zařízení.

Další cesty rozšíření

NetBox lze rozšířit pomocí komunitních nebo vlastních pluginů. Pluginy přidávají funkce nad rámec standardní instalace a mohou systém udělat ještě užitečnější. Níže jsou uvedeny nejčastěji využívané způsoby rozšíření:

Grafické pluginy pro topologii
Uživatelé vytvářejí pluginy, jež pomocí knihoven typu D3.js generují interaktivní grafy sítě. Uzly i hrany reagují na najetí myši či kliknutí, což usnadňuje orientaci ve složitých topologiích [21].

Reporty a exporty dat
Organizace mívají specifické požadavky na reporting (např. PDF výstupy pro management). Pluginy proto umožňují formátovat data a exportovat je do dalších systémů, jako jsou například IT service management (ITSM) či Configuration management database (CMDB).

Notifikační systémy
Pomocí webhooků lze NetBox propojit se Slackem, Microsoft Teams, e-mailem nebo ticketovacími nástroji. Systém pak může automaticky rozesílat upozornění při určitých událostech, například přidání nového zařízení nebo detekci konfliktu adres [17].

Kapitola 3

CESNET a požadavky na rozšíření

CESNET je sdružení českých vysokých škol a Akademie věd České republiky. Provozuje a rozvíjí národní e-infrastrukturu pro vědu, výzkum a vzdělávání. Jeho hlavním cílem je poskytovat univerzitám a výzkumným institucím špičkové služby v oblasti vysokorychlostní sítě, výpočetních kapacit, datových úložišť a bezpečnosti [5]. V rámci této práce je CESNET uživatel a zadavatel konkrétních rozšíření nástroje NetBox. Ty by měly usnadnit správu rozsáhlé infrastruktury, včetně specializovaných přenosových technologií.

3.1 Jak CESNET využívá NetBox

Dokumentace a evidence
Uchovávání přehledných informací o umístění, konfiguraci a historii síťových prvků, včetně modulů, optických vláken či karet. Vyhledávání adres, VLAN a dalších entit, s možností využít hromadné importy/exporty dat [17, 20].

Zefektivnění správy
NetBox slouží jako *single source of truth* (jediný zdroj pravdy) pro interní skripty, aby bylo možné provádět hromadné operace s minimem ruční práce. Také pro sdílení informací v rámci týmu, včetně přístupových oprávnění pro různé role uživatelů [17].

Rozšíření pro speciální případy
Vzhledem k modularitě síťových zařízení a často komplikované topologii nastaly problémy týkající se pokročilé práce s moduly, fyzickými kabely či zobrazením optických propojení. Právě proto CESNET potřebuje rozšířit NetBox o nové funkce, řešící tyto problémy.

3.2 Konkrétní problémy a jejich rozdělení

Zadavatel (CESNET) poskytl seznam rozšíření. Tyto rozšíření se musely rozdělit dle jejich důležitosti na nezbytné a volitelné. Nezbytné požadavky bylo potřeba vypracovat pro splnění zadání bakalářské práce a rozšíření NetBoxu o důležité funkce. Vhodné požadavky bylo možné vypracovat nad rámec zadání (jedná se o požadavky, které by bylo dobré implementovat, ale nemají takovou prioritu jako nezbytné požadavky).

Požadavky jsou rozděleny do tří skupin:

- M: Manipulace s moduly
- C: Manipulace s propoji
- P: Manipulace s rozhraními

Čísla u jednotlivých typů požadavků udávají v jakém pořadí měly být implementovány. Důvodem je, že jednotlivé požadavky na sobě závisí.

Nezbytné požadavky

M1) Jednoduchá možnost přesunutí karty mezi Module Bays
Uživatelé potřebují vyjmout modul (kartu) z jednoho *Device* a vložit jej do jiného (resp. „přesunout do skladu“). Toto by mělo jít provést bez smazáním a opětovného vytvoření modulu. NetBox by měl akci vyřešit jedním krokem z uživatelského rozhraní. V praxi tedy: vyjmutí karty ze starého *Device* a následné vytvoření nového modulu v cílovém *Device* [17].

M2) Uchování historie přesunu karty
Každý *Inventory Item*, který reprezentuje konkrétní kartu, by měl mít záznam o tom, kde se v průběhu své životnosti nacházel [17]. Tato historie je pro CESNET klíčová při zpětném dohledávání problémů, záruk a také pro inventurní účely.

P1) Možnost rozpadu Interface na RX a TX
V určitých případech je třeba spojit vstup (RX) a výstup (TX) se stejným protějším portem (např. některé optické topologie). Momentálně NetBox považuje port většinou za dvousměrný [17]. Rozpad Interface na RX/TX by měl být uživatelsky jednoduchý a neměl by narušit dosavadní koncepci síťových rozhraní.

C1) Potřeba uchovávat informace o fyzických kabelech
NetBox umí vytvářet a mazat kabely (*Cable* a *Connections*), ale práce s nimi může být pro rozsáhlé přenosové systémy příliš jednoduchá, tj. snadno lze kabel vymazat. Navíc formálně *Cable* vyžaduje zapojení do dvou koncových *Interfaces* [17]. Cílem je uchovat u kabelu vlastní identifikaci (např. sériové číslo optického patch kabelu) a historii, podobně jako u modulu v rámci *Inventory Items* (viz M2).

Volitelné požadavky

M3) Kontrola při manipulaci s modulem
Pokud uživatel zavádí kartu do *Module Bay* (tj. vytváří *Module*), NetBox nenabízí možnost propojit ji s odpovídajícím *Inventory Item*. CESNET navrhuje možnost označit některé *Module Types* tagem, který by při vytváření modulu vyžadoval zadání odpovídajícího *Inventory Item*.

C2) Svázání Cable s Inventory Item
Stejná logika jako u modulů: při vytváření kabelu by systém nabízel uživateli vybrat příslušný *Inventory Item*, pokud existuje. Na rozdíl od *Module* a *Module Type* nemá *Cable* definovaný *Cable Type*, takže bude třeba navrhnout, jak tuto vazbu realizovat.

C3) Zřetězení vybraných interface do řetězového propojení
Umožnit uživateli postupně vybrat více portů a automatizovaně je „spojit“ (např. RX1 → TX1, RX2 → TX2, ...). Tato funkce by zrychlila vytváření propojení v případech, kdy se jedna linka skládá z více optických vláken, TAPů apod.

C4) Reprezentace TAPu, spojky či kabelové „chobotnice“
CESNET by uvítal možnost snadno modelovat doplňková zařízení (TAP, spojka, Y-cable), což NetBox nativně neřeší. V čistém NetBoxu je potřeba pro takové scénáře různě improvizovat s *Front*, *Rear Ports* a *Cables*.

3.3 Analýza nezbytných požadavků

V této části analyzujeme klíčové požadavky M1, M2, P1 a C1. Popíšeme, proč jsou nutné, jaké problémy řeší, a jaké omezení či specifické vlastnosti je třeba při jejich implementaci zohlednit. Podrobné technické návrhy těchto řešení budou uvedeny v následující kapitole.

Analýza požadavků M1 a M2: Přesun a historie modulů

Momentálně NetBox moduly striktně váže ke konkrétnímu zařízení (*Device*). Pokud je třeba modul přesunout do jiného zařízení, je nutné ho odstranit a opětovně vytvořit, což má za následek ztrátu historie (změny, záznamy o konfiguracích, auditní logy). [17]

Proto je třeba zajistit:

- **Zachování historie modulů**: Historie u jednotlivých modulů bude zachována, jelikož dojde k přesunu a ne k mazání modulu, což by vedlo ke ztrátě historie.
- **Řízený přesun modulů**: Zajistit uživatelsky přívětivé rozhraní umožňující jednoduché přemístění modulů mezi zařízeními nebo jejich přesun do „skladu“. Sklad v NetBox neexistuje, ale lze ho vytvořit jako *Device*.

Kompatibilita s NetBoxem spočívá ve využití existujících struktur (*Inventory Item*), přičemž bude kladen důraz na to, aby se nezasáhlo do jádra systému. [17]

Analýza požadavku P1: Rozdělení Interface na RX a TX

NetBox ve výchozím stavu předpokládá symetrické (obousměrné) kabelové propojení rozhraní. Existují však situace, zejména při práci s optickými či specializovanými kabely, kdy je nutné rozlišit směr signálu na vysílací (TX) a přijímací (RX) část. [17]

Z analýzy vyplývají následující požadavky:

- **Explicitní rozdělení portů**: Umožnit rozpad portu na subporty RX a TX, přičemž původní port bude zachován a nové porty se na něj budou odkazovat jako na „parenta“.
- **Zpětná kompatibilita**: Integrace pluginu nesmí narušit výchozí chování NetBoxu. Plugin nesmí změnit žádné existující funkce.

Analýza požadavku C1: Uchovávání informací o fyzických kabelech

NetBox sice umožňuje evidenci kabelů, ale neposkytuje dostatečné informace o fyzických atributech (např. sériová čísla, historie revizí, opravy či výměny). Historie bude řešena pomocí propojení s *Inventory Item* v rámci požadavku C2. [17]

V analýze poždavku vyplývá:

- **Fyzická identifikace kabelů**: Je třeba umožnit uchování detailních informací, které budou přístupné pro audit, sledování a plánování údržby.
- **Možnost evidence dodatečných informací**: Zavést způsob zaznamenávání dodatečných metadat (např. stav revizí, poznámky). Takto bude k dispozici více informací. To povede k jednodušší identifikaci fyzických kabelů.

Je třeba respektovat zásady NetBoxu. NetBox se snaží udržet jednoduchost datového modelu a přehlednost uživatelského rozhraní. Každé rozšíření tedy musí být volitelné a zpětně kompatibilní.

Analýza s ohledem na budoucí rozšíření

Již v této analýze je nutné brát v úvahu volitelné požadavky M3, C2, C3 a C4. Požadavky na rozšiřitelnost řešení jsou:

- **Modularita**: Rozšiřování modelů a formulářů musí být navrženo tak, aby bylo snadno upravitelné (např. samostatné funkce pro *Cable*, moduly pro práci s *Inventory Item*).
- **Tagování a klasifikace**: Při návrhu nových rozhraní a formulářů předem analyzovat, jak bude v budoucnu možno snadno přidat nové parametry (např. nové atributy v *Inventory Item*, rozšíření *Module Type* o specifické tagy).

Kapitola 4

Návrh rozšíření

V této části práce se zaměříme na návrh nezbytných rozšíření NetBoxu. Začneme s návrhem architektury, kde se dozvíte jaké jsou možnosti rozšíření NetBoxu. Dále si stanovíme jak je nutné rozšířit databázový model. Teprve poté přistoupíme k podrobnému návrhu řešení, včetně nových částí uživatelského rozhraní, který cílí na „nezbytné“ i „doporučené“ požadavky vymezené v předchozí kapitole.

4.1 Návrh architektury řešení

Z technického pohledu existují dvě hlavní cesty, jak NetBox rozšířit:

- **Formou pluginu** (doporučované řešení) – zachová se separace kódu a menší riziko konfliktu při upgradu NetBoxu [21].
- **Přímými úpravami v jádře** – vyžaduje dobrou znalost vnitřní architektury NetBoxu a může být složitější pro budoucí aktualizace [18, 17].

S ohledem na udržitelnost a možnost rychlého nasazení je vhodnější použít pluginovou architekturu. Ta umožňuje definovat nové datové modely a s minimem zásahů do jádra rozšířit existující formuláře, API endpointy či šablony [21].

Struktura pluginu

Mezi základní složky a soubory patří:

- **init** – Soubor, ve kterém se definuje konfigurační třída pluginu.
- **models** – Soubor obsahující definice vlastních Django modelů rozšiřujících databázové schéma NetBoxu.
- **views** – Sobour definující všechny webové pohledy („views“), které obsluhují HTTP požadavky a vrací odpovědi (HTML šablony, JSON, přesměrování atp.).
- **urls** – Soubor mapující URL cesty pro webové pohledy a API endpointy pluginu.
- **templates** – Složka obsahující HTML šablony doplňující dosavadní vzhled NetBoxu nebo poskytující vlastní seznamy a detailní zobrazení.

Pluginy mohou samozřejmě obsahovat mnohem více souborů. Konkrétní struktury pluginů uvidíte v samotném řešení. Kromě samotného vytvoření pluginu je třeba ho zaregistrovat mezi používané pluginy. To se dělá v konfiguračním souboru NetBoxu. Plugin se přidá do sekce PLUGINS a definuje si vlastní PLUGIN CONFIG, pokud to je třeba [17].

Datový model pro historii a metadata

Pro M2 (historie modulů) lze zavést tabulku *ModuleHistory*, která bude mít klíč na *Inventory Item* a záznam pro každou změnu umístění. Pro C1 může vzniknout *CableItem* s doplňujícími atributy (sériové číslo atd.), navázaný 1:1 na existující *Cable*. Pro P1 (oddělení RX/TX) stačí rozšířit existující *Interface* o příznak (typ směru) nebo zavést speciální typ *InterfaceRX* a *InterfaceTX*.

Vazba na stávající NetBox API (volitelné)

Pokud CESNET využívá i REST API pro integraci, je třeba zajistit, aby nové funkce (přesun modulu, upravená práce s kabely) byly dostupné i programově [20]. Je vhodné zachovat konzistenci s pojmenováním a konvencemi NetBoxu [17].

Dopad na výkon a údržbu

Rozšíření by mělo minimalizovat dopady na výkon. Pro hromadné akce (např. import modulů, generování historie) lze v budoucnu navrhnout asynchronní zpracování (fronty typu Redis [27]/Celery [4]). Bude-li plugin správně navržen, udržování kompatibility s novějšími verzemi NetBoxu by mělo vyžadovat jen sledování oficiálních změn v plugin API [21].

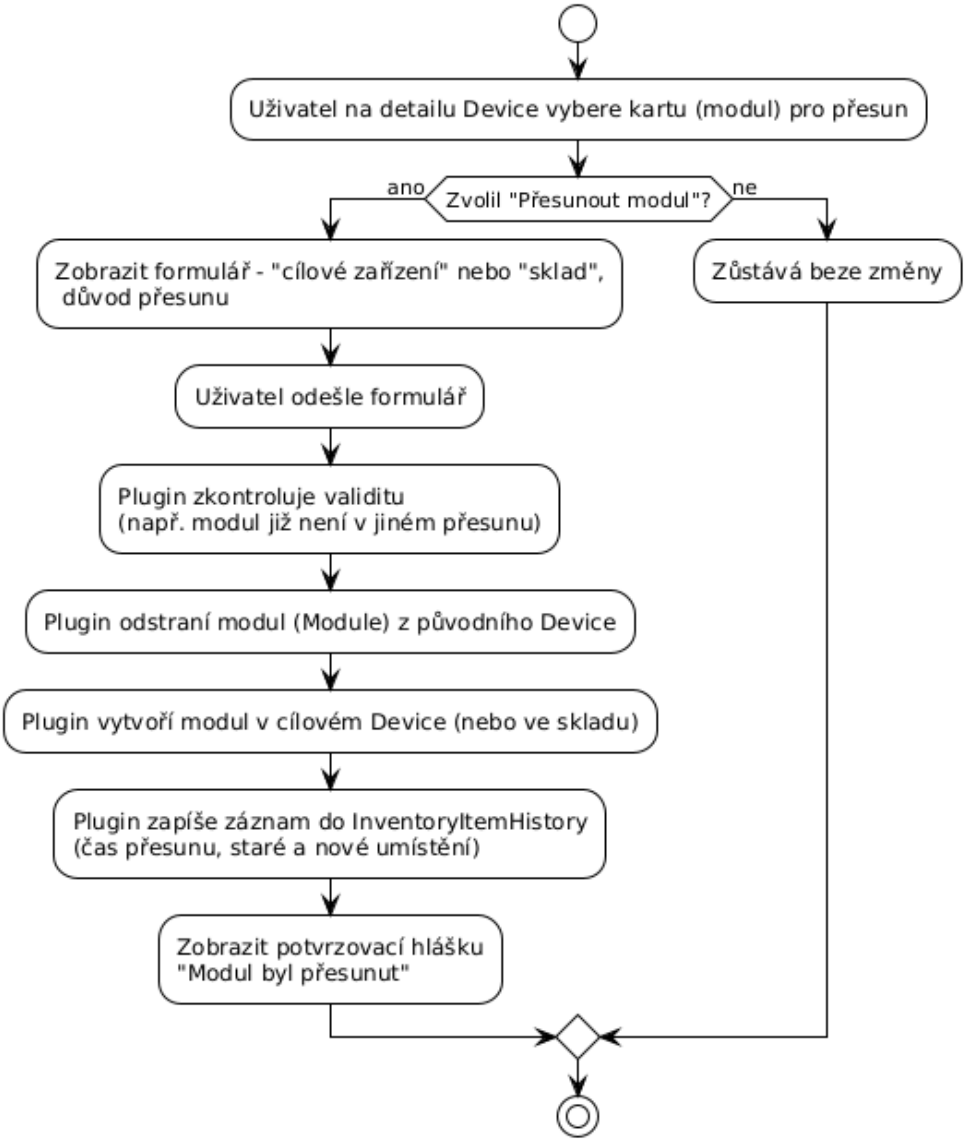
4.2 Požadavky na UI prvky a návrh řešení

Pro splnění definovaných cílů je potřeba správcům a technikům nabídnout jednoduchý a přehledný způsob, jak provádět požadované akce přímo z webového prostředí. Následující podsekce shrnují jak požadavky na nové UI prvky (M1, M2, P1, C1), tak navrhované řešení implementované formou pluginu pro NetBox.

M1 a M2: Přesun modulu a historie umístění

Požadavek: Uživatelé musí mít možnost přesunout modul (kartu) mezi zařízeními nebo do skladu a současně sledovat historii jeho umístění. NetBox již historii řeší. Samotné udržení této historie je řešeno přenosem modulu místo jeho mazání.

Návrh řešení: Prostřednictvím pluginu pro NetBox rozšířit modulovou logiku o akci *Module Swap*. Tlačítko *Přesunout modul* zobrazí formulář, v němž uživatel vybere nové *Device* (resp. sklad). Po potvrzení plugin automaticky odstraní modul z původního zařízení a vytvoří jej na cílovém, přičemž NetBox zajistí propsání informací do historie (changelogu). Sekvenční diagram akce Module Swap je na obrázku 4.1.

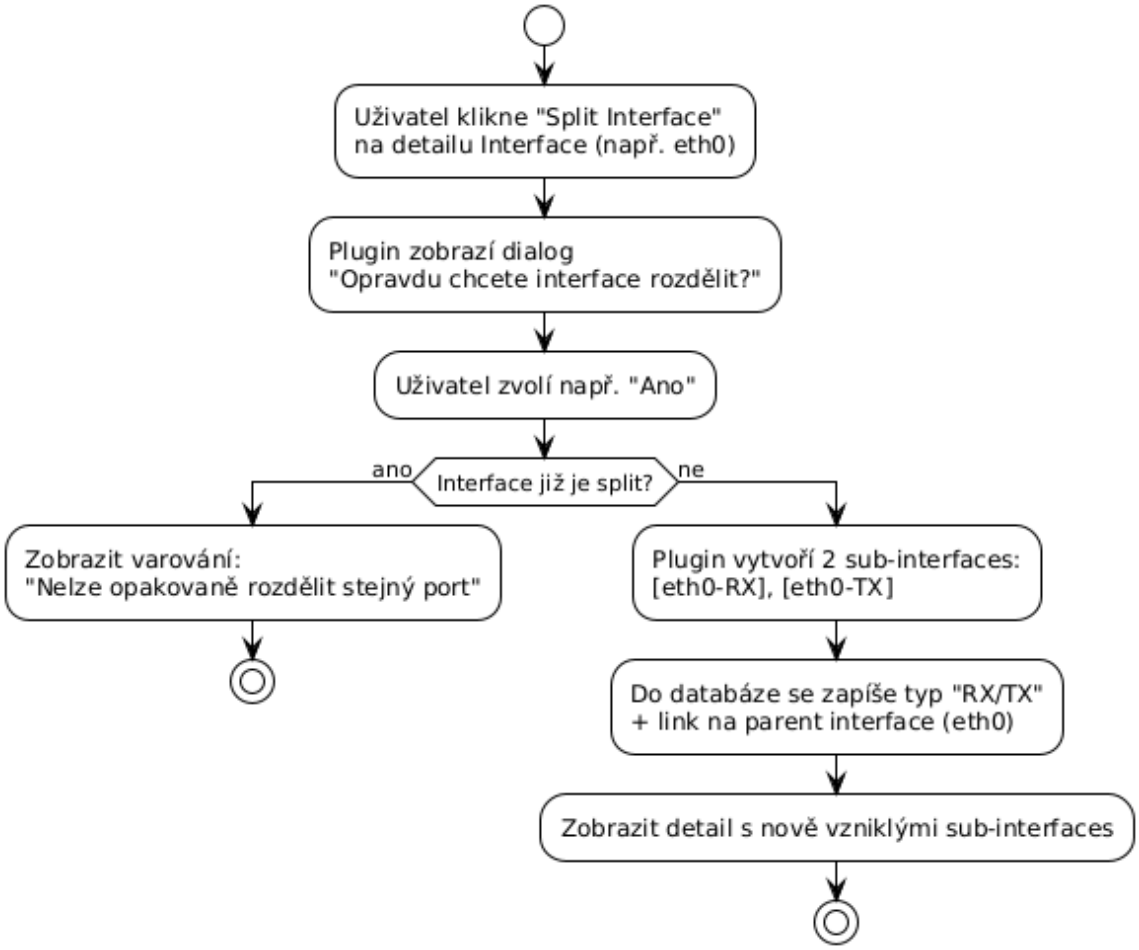


Obrázek 4.1: Sekvenční diagram akce „Module Swap“.

P1: Rozdělení Interface na RX, TX a Main

Požadavek: Umožnit rozpad fyzického portu na odvozené sub-interfaces RX a TX. Ve formuláři pro vytváření kabelu pak lze zvolit typ propojení (RX–TX, TX–TX atd.).

Návrh řešení: Do seznamu rozhraní přidat volbu *Split Interface*. Po aktivaci plugin ověří, zda port není již rozdělen, a následně vytvoří podrozdělení: *Interface-RX* a *Interface-TX*. Při opakované akci se uživateli zobrazí varovná hláška a proces se přeruší. Sekvenční diagram je uveden na obrázku 4.2.



Obrázek 4.2: Sekvenční diagram akce „Split Interface“.

C1: Rozšířená evidence kabelů

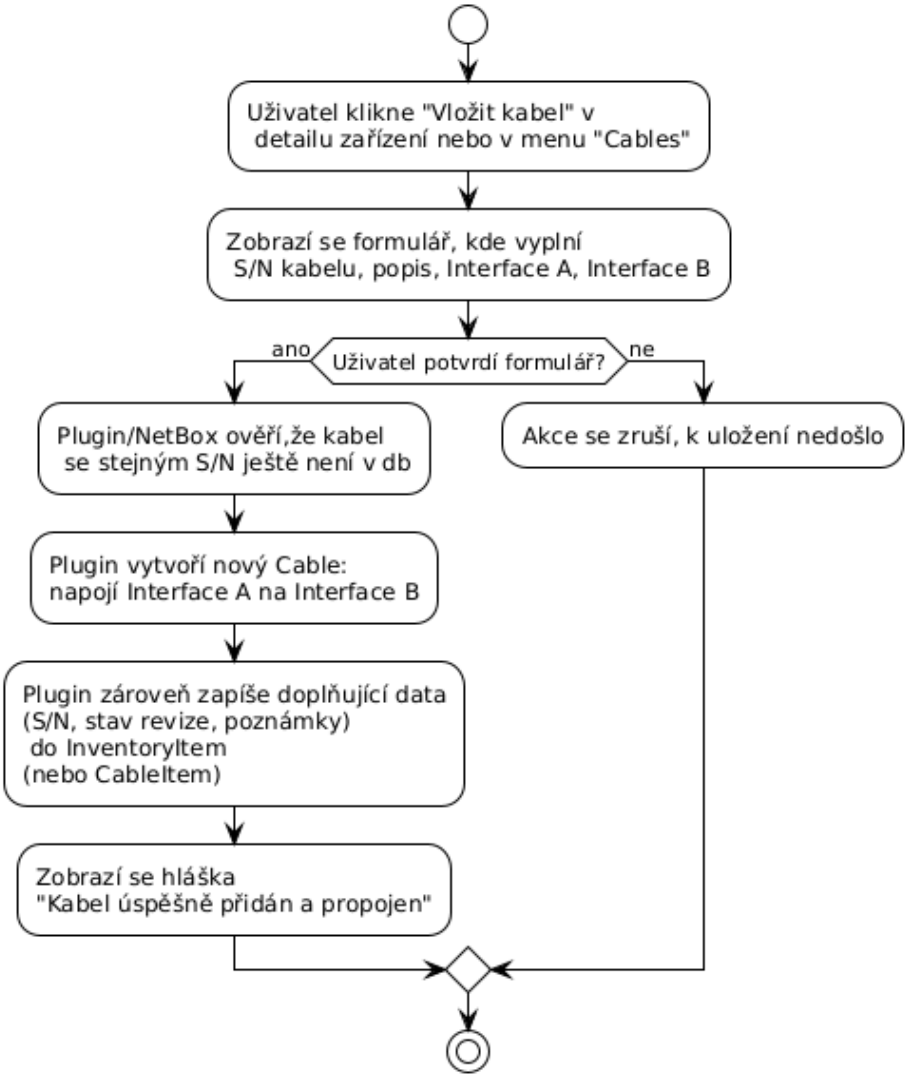
Požadavek: V detailu kabelu zaznamenávat metadata jako sériové číslo, stav revize a poznámky. NetBox standardně nepodporuje komplexní objekty kabelů.

Návrh řešení: Rozšířit nebo vytvořit entitu *CableItem* vázanou 1:1 na *Cable*. Při vložení kabelu („Add cable") plugin spustí formulář podobný vkládání modulu, ve kterém uživatel vyplní standardní propojení mezi rozhraními (Interface A, Interface B) a doplňující metadata. Po odeslání se vytvoří záznam kabelu i jeho *Inventory Item*, včetně historie zapojení. Sekvenční diagram procesu je na obrázku 4.3.

Zobrazení seznamu karet

Požadavek: Samostatná stránka vykreslující tabulky dle typu karty. Tabulky obsahují data o kartách daného typu např. umístění karty, sériové číslo, rok zavedení atd.

Návrh řešení: Vytvořit jednoduchý plugin na filtraci karet dle typu. Plugin vyfiltruje všechny karty daného typu, získá všechna jejich data, zpracuje je a vykreslí do přehledné tabulky. Takto může uživatel vidět všechny důležité informace na jednom místě.



Obrázek 4.3: Sekvenční diagram akce „Add cable“.

Základ pro volitelné rozšíření

Architektura pluginu je navržena modulárně:

- Samostatné třídy a služby pro manipulaci s *Inventory Item* (karty i kabely).
- Oddělené pluginy pro manipulaci s *Module*, *Interface* a *Cable*.
- Formuláře postavené tak, aby bylo možné snadno přidávat pole a kontrolní logiku pro budoucí funkce (M3, C2, C3, C4).

Tímto způsobem pokryjeme nezbytné požadavky (M1, M2, P1, C1) a připravíme základ pro volitelné požadavky.

Kapitola 5

Implementace rozšíření

Tato kapitola se věnuje implementaci navržených rozšíření. Na základě analýzy požadavků a návrhu řešení popsaných v předchozích kapitolách byly implementovány čtyři pluginy. Ty řeší nezbytné požadavky sdružení CESNET. V následujících podkapitolách je detailně popsán proces implementace včetně použitých technologií, přístupu k vývoji, datové vrstvy, aplikační logiky a uživatelského rozhraní. Pozornost je také věnována integraci s existujícím systémem NetBox a řešení problémů, které se během implementace vyskytly.

5.1 Vývojové prostředí a příprava

Prvním krokem bylo vybrání vhodných technologií a příprava vývojového prostředí. Vzhledem k tomu, že rozšíření musí být kompatibilní s existujícím systémem NetBoxu, bylo důležité dodržet využití stejných technologií. Důležité také bylo se rozhodnout jak kód verzovat a distribuovat, tak ať je volně dostupný pro kohokoliv.

Popis vývojového prostředí a použitých technologií

Pro implementaci všech čtyř rozšíření byly využity stejné technologie jako v případě samotné aplikace NetBox. Hlavním programovacím jazykem je Python. Ten je v současnosti jedním z nejpoužívanějších jazyků pro vývoj webových aplikací [26]. Konkrétně byla použita verze Pythonu 3.10.12. Ta poskytuje velké množství moderních funkcí a knihoven.

Jako webový framework bylo využito Django, tvořící základ architektury NetBoxu. Django je vysoce výkonný webový framework pro Python. Jeho filozofie je založena na principu „batteries included“ (poskytuje všechny potřebné nástroje v základní instalaci) a klade důraz na znovupoužitelnost kódu a princip „Don’t Repeat Yourself“ (DRY) [8]. Django poskytuje skvělou sadu nástrojů pro rychlý vývoj webových aplikací, což bylo zásadní pro efektivní implementaci.

Pro práci s databází bylo využito Django ORM (Object-Relational Mapping), umožňující definovat datové modely pomocí tříd Pythonu a následně s nimi pracovat bez nutnosti psaní SQL dotazů. Tato abstrakce umožnila snadnější implementaci databázových operací a integraci s existujícím datovým modelem. Jako databázový systém byl zvolen PostgreSQL. Ten je jedním z velmi používaných databázových systémů při práci s NetBoxem [19]. PostgreSQL nabízí dobrý výkon, spolehlivost a pokročilé funkce. Ty byly využity při implementaci komplexních dotazů potřebných pro některá z implementovaných rozšíření.

Pro implementaci uživatelského rozhraní byly využity Django šablony (templates), které jsou součástí Django frameworku. Tyto šablony umožňují kombinovat HTML s jednodu-

chým šablonovacím jazykem, což usnadňuje dynamické generování obsahu webových stránek. Pro zajištění konzistentního vzhledu s původním rozhraním NetBoxu byly využity existující CSS styly a komponenty.

Příprava vývojového prostředí

Pro vývoj rozšíření NetBoxu bylo vytvořeno lokální vývojové prostředí. Toto prostředí umožnilo efektivní implementaci a testování jednotlivých funkcí. Na rozdíl od kontejnerizovaného přístupu s využitím Docker kontejnerů byla zvolena přímá instalace NetBoxu v lokálním prostředí. [19]

Po úspěšné instalaci a konfiguraci lokální instance NetBoxu byl využit jednotný postup pro implementaci jednotlivých rozšíření. Tento postup zahrnoval vytváření separátních Python balíčků pro každé rozšíření a jejich následnou instalaci do lokálního instance NetBoxu. To umožnilo izolovaný vývoj a testování jednotlivých funkcí.

Pro ladění a testování byl využit vestavěný vývojový server Djanga. Ten poskytuje funkce jako automatické načítání změn v kódu a ladící informace. Toto prostředí umožnilo rychlý iterativní vývoj jednotlivých rozšíření.

Verzovací systém a distribuce rozšíření

Pro správu verzí zdrojového kódu byl využit systém Git, umožňující sledování změn a údržbu více verzí (větví) projektu. Každé ze čtyř implementovaných rozšíření bylo umístěno do vlastního repozitáře na GitHubu, což zajistilo jejich oddělený vývoj, správu a možnost sdílení. Rozšíření jsou implementována jako samostatné pluginy pro NetBox. Tento přístup je přímo doporučen vývojáři NetBoxu [21].

Pro distribuci pluginů byla využita platforma PyPI (Python Package Index). Rozšíření jsou zabalena jako samostatné Python balíčky doplněny o definované závislosti a metadata. Tato forma distribuce umožňuje snadnou instalaci rozšíření do jakékoliv instance NetBoxu jediným příkazem pip install.

Verzování balíčků se řídí pravidly sémantického verzování (Semantic Versioning). Ten definuje tři čísla verze (MAJOR.MINOR.PATCH) s jasnými pravidly pro jejich inkrementaci [24]. Díky tomu uživatelé snadno rozliší, zda nová verze přináší zpětně nekompatibilní změny, nové funkce, nebo pouze opravy chyb.

5.2 Implementace datové vrstvy

Datová vrstva tvoří základ každého z implementovaných rozšíření. Správný návrh datových modelů je klíčový pro zajištění efektivní práce s daty, jejich konzistence a integrace s existujícím datovým modelem NetBoxu. V této sekci je popsána implementace datové vrstvy pro jednotlivá rozšíření včetně návrhu nových modelů a jejich vazeb na existující datovou strukturu NetBoxu.

Datový model pro Interface Relationship Manager

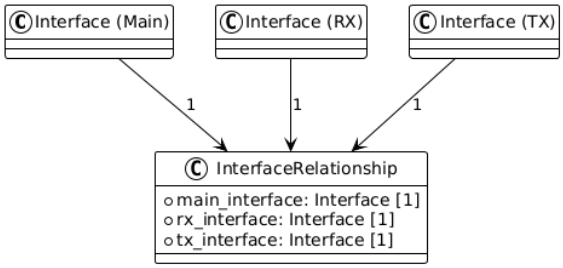
Rozšíření *Interface Relationship Manager* [12] umožňuje rozdělit jedno rozhraní na dvě samostatná logická rozhraní pro příjem (RX) a vysílání (TX) dat. Tento přístup je užitečný v prostředích, kde je nutné sledovat směr toku dat odděleně.

Hlavním datovým modelem je `InterfaceRelationship`, který definuje vztah mezi hlavním rozhraním a dvojicí RX a TX rozhraní. Všechny tři vazby jsou typu `OneToOneField`, což zajišťuje, že každé rozhraní se může v daném vztahu vyskytovat právě jednou.

```
class InterfaceRelationship(models.Model):
    main_interface = models.OneToOneField(
    Interface,
    on_delete=models.CASCADE
    )
    rx_interface = models.OneToOneField(
    Interface,
    on_delete=models.CASCADE
    )
    tx_interface = models.OneToOneField(
    Interface,
    on_delete=models.CASCADE
)
```

Výpis 5.1: Zjednodušený model InterfaceRelationship

Poznámka: Tento výřez slouží pouze k ilustraci vztahu mezi entitami. Neobsahuje další části modelu.
Tímto vzniká vztah:



Obrázek 5.1: Diagram vztahu mezi Interfacy

Datový model pro Module Swap

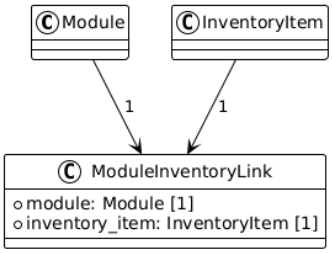
Rozšíření *Module Swap* [13] má dvě hlavní funkce: umožnit jednoznačné propojení mezi modulem (`dcim.Module`) a položkou inventáře (`dcim.InventoryItem`) a usnadnit výměnu modulů mezi zařízeními. Pro první z těchto funkcí plugin definuje vlastní datový model `ModuleInventoryLink`, který realizuje relaci 1:1 mezi modulem a inventární položkou.

Tento model zajišťuje, že každý modul může být spojen s právě jednou položkou inventáře a naopak. Díky tomu je možné uchovat informaci o fyzické komponentě modulu a její odpovídající evidenční položce v NetBoxu, což je důležité např. pro správu majetku nebo sledování záruk.

```
class ModuleInventoryLink(models.Model):
    module = models.OneToOneField(Module, on_delete=models.CASCADE)
    inventory_item = models.OneToOneField(
    InventoryItem,
    on_delete=models.CASCADE
)
```

Výpis 5.2: Zjednodušený model ModuleInventoryLink

Poznámka: Tento výřez slouží pouze k ilustraci vztahu mezi entitami. Neobsahuje další části modelu.
Tímto vzniká vztah:



Obrázek 5.2: Diagram vztahu mezi Module a InventoryItem

Druhou funkcí pluginu je samotná výměna (swap) modulů mezi zařízeními. Tento proces je realizován pomocí existujících datových modelů NetBoxu, zejména `dcim.Module`, `dcim.Device` a `dcim.ModuleBay`. Plugin poskytuje rozhraní pro přesun modulu mezi zařízeními bez ztráty dat. Pro tuto funkcionalitu není nutné zavádět žádný nový perzistentní model.

Datový model pro Cable Extension

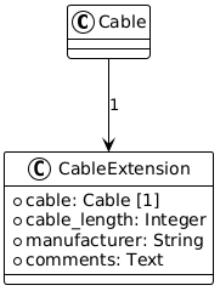
Rozšíření *Cable Extension* [11] doplňuje výchozí model `Cable` v NetBoxu o další užitečné atributy jako délku kabelu, výrobce nebo poznámky. Model `CableExtension` je s původním modelem propojen přes `OneToOneField`, což umožňuje rozšířit data bez zásahu do základního datového schématu.

```
class CableExtension(models.Model):
    cable = models.OneToOneField(
    Cable,
    on_delete=models.CASCADE
)
cable_length = models.PositiveIntegerField(blank=True, null=True)
manufacturer = models.CharField(max_length=100, blank=True, null=True)
comments = models.TextField(blank=True, null=True)
```

Výpis 5.3: Zjednodušený model CableExtension

Poznámka: Tento výřez slouží pouze k ilustraci vztahu mezi entitami. Neobsahuje další části modelu.

Tímto vzniká vztah:



Obrázek 5.3: Diagram vztahu mezi Module a InventoryItem

Poznámka k validitě dat: Plugin *Cable Extension* byl implementován jako experimentální řešení, protože původní požadavek nebylo možné v rámci architektury NetBoxu přesně naplnit. Některé atributy (např. `cable_length`) se tak v pluginu mohou duplikovat s již existujícími atributy základního modelu `Cable`. Cílem bylo ověřit možnosti rozšíření modelu bez zásahu do jádra NetBoxu. Výsledná data proto nemusí být vždy zcela konzistentní a je třeba k nim přistupovat s určitou mírou opatrnosti. Celkově se jedná spíš o důkaz toho, že ne vše se dá dělat pluginem nebo to je vhodné tak dělat.

Databázové migrace

Pro implementované datové modely byly vytvořeny migrační soubory pomocí Django migračního systému [8]. Tyto soubory jsou součástí každého GitHub repozitáře v adresáři migrations a zajišťují automatickou aktualizaci databázového schématu při instalaci nebo aktualizaci rozšíření. Tento přístup zajišťuje bezpečné změny databázového schématu a zároveň umožňuje zpětné migrace v případě potřeby.

Validace a integrita dat mimo modely

Validace vstupních dat a zajištění jejich integrity v rámci rozšíření neprobíhá na úrovni samotných datových modelů, ale je implementována na aplikační vrstvě – typicky ve formulářích (forms.py) a v některých případech i v řídicích strukturách (views.py). Tento přístup odpovídá doporučeným praktikám frameworku Django [8], který umožňuje víceúrovňovou validaci dat a poskytuje vývojářům flexibilitu při řízení toku dat mezi uživatelským rozhraním a databází.

Inventory Viewer a absence vlastní datové vrstvy

Plugin Inventory Viewer nevyužívá vlastní perzistentní datový model, a proto zde není popsán jako ostatní pluginy. Pracuje výhradně s existujícími entitami v NetBoxu (např. dcim.Module, dcim.Device, dcim.InventoryItem) prostřednictvím komplexních dotazů a filtrovacích mechanismů. Jeho implementace se tak nachází výhradně na úrovni zobrazení a logiky aplikace. Ta je popsána v další sekci.

5.3 Implementace aplikační logiky

Aplikační logika tvoří vrstvu mezi datovou vrstvou a uživatelským rozhraním. V kontextu frameworku Django, na kterém je NetBox postaven, zahrnuje aplikační vrstva především pohledy (`views`), formuláře (`forms`), ale také logiku pro zpracování dat, validaci vstupů nebo přesměrování.

NetBox poskytuje pro tvorbu pluginů užitečné generické komponenty, jako jsou třídy `ObjectListView`, `ObjectEditView` nebo `ObjectDeleteView`, které značně urychlují implementaci CRUD operací. Také samotné formuláře lze ve většině případů vytvořit jako potomky `NetBoxModelForm`, což zajišťuje jednotné chování celého systému [7, 21].

Každé rozšíření implementuje tuto logiku odlišným způsobem, v závislosti na tom, jaké akce a zpracování jsou od uživatele očekávány. V následujících podkapitolách je podrobně popsáno, jak aplikační logika funguje v jednotlivých pluginech.

Interface Relationship Manager – zpracování vztahů

Plugin *Interface Relationship Manager* [12] umožňuje vytvořit logický vztah mezi třemi rozhraními: hlavním, přijímacím (`RX`) a vysílacím (`TX`). Implementované pluginy nevyužívají generické pohledy z NetBoxu, tento plugin definuje vlastní pohled select_device_for_split, který zajišťuje celý proces výběru zařízení, rozhraní a vytvoření vztahu.

Základ aplikační logiky spočívá ve zpracování požadavků typu `GET` nebo `POST`. Uživatel nejprve vybere zařízení a následně rozhraní, které chce rozdělit. V případě platného výběru dojde k zavolání pomocné funkce `split_interface`, která provede samotné rozdělení:

- vytvoří dvě nová rozhraní `RX` a `TX` jako podřízená původnímu,
- zaznamená nově vzniklý vztah do modelu `InterfaceRelationship`,
- provede validaci, zda rozhraní nebylo rozděleno již dříve.

```
def split_interface(device_id, interface_name):
    device = Device.objects.get(pk=device_id)
    original = Interface.objects.get(device=device, name=interface_name)
    rx = Interface.objects.create(
        device=device, name=original.name+"_rx", parent=original
    )
    tx = Interface.objects.create(
        device=device, name=original.name+"_tx", parent=original
    )
    InterfaceRelationship.objects.create(
        main_interface=original, rx_interface=rx, tx_interface=tx
    )
```

Výpis 5.4: Zjednodušená funkce `split_interface`

Poznámka: Výpis je zjednodušený pro ilustraci základní aplikační logiky.

Pohled dále obsahuje zpracování chybových stavů, informuje uživatele o úspěchu či selhání operace pomocí systému zpráv (`messages.success`, `messages.error`) a nakonec přesměrovává zpět na formulář s možností další volby. Rozhraní, která byla již dříve rozdělena, jsou na základě dat v modelu `InterfaceRelationship` dynamicky vyloučena z výběru.

```
PluginMenuItem(
    link='plugins:interface_relationship_manager:select_device',
    link_text='Split Interface',
    permissions=['dcim.change_device']
)
```

Výpis 5.5: Zápis odkazu do menu pluginu

Tento přístup představuje kombinaci vlastního pohledu, utility funkce a šablony s jednoduchým formulářem, které společně pokrývají celý proces interaktivního vytváření vztahů mezi rozhraními. Implementace využívá osvědčené konstrukce Django frameworku a současně využívá specifické funkce NetBoxu, jako je registrace odkazu v menu pomocí PluginMenuItem nebo zpětné přesměrování pomocí `reverse()` [7, 21].

Module Swap – přesun modulů a vazba na inventář

Plugin *Module Swap* [13] poskytuje funkce pro dvě činnosti: umožňuje propojit modul s konkrétní inventární položkou a zároveň nabízí dvoukrokový proces pro přesun modulu mezi zařízeními. Aplikační logika pluginu je tedy rozdělena do dvou částí – přehled a správa vazby modul–inventář a přesun modulu. Přesun modulu je realizován pomocí dvou po sobě jdoucích kroků ve formě tříd `Step1SelectView` a `Step2BayView`. V prvním kroku uživatel vybere modul a cílové zařízení. Ve druhém kroku pak vybírá konkrétní slot (`ModuleBay`) v cílovém zařízení, kam má být modul přesunut.

Vybraný modul se v rámci databázové transakce nejprve odebere ze stávajícího slotu, následně se přesune do nového a případně se aktualizuje i zařízení, které je propojeno s odpovídající položkou inventáře.

```
with transaction.atomic():
    old_bay = ModuleBay.objects.filter(module=selected_module).first()
    if old_bay:
        old_bay.module = None
        old_bay.save()
    selected_module.module_bay = target_module_bay
    selected_module.device_id = target_module_bay.device_id
    selected_module.save()
    try:
        link = ModuleInventoryLink.objects.get(module=selected_module)
        inv_item = link.inventory_item
        inv_item.device_id = target_module_bay.device_id
        inv_item.save()
    except ModuleInventoryLink.DoesNotExist:
        pass
```

Výpis 5.6: Zkrácený přehled přesunu modulu ve třídě `Step2BayView`

Druhou část pluginu tvoří správa propojení mezi moduly a položkami inventáře. Ta je realizována pomocí formuláře `LinkModuleInventoryForm`, který kromě výběru položek zajišťuje validaci jejich jednoznačného propojení (1:1), a několika tříd pohledů, které zajišťují jejich vytvoření, úpravu, mazání a zobrazení přehledu.

```
if ModuleInventoryLink.objects.filter(module=module)
.exclude(pk=self.instance.pk).exists():
```

```
    self.add_error('module', 'This module is already linked.')
if ModuleInventoryLink.objects.filter(inventory_item=inventory_item)
.exclude(pk=self.instance.pk).exists():
    self.add_error('inventory_item', 'This item is already linked.')
```

Výpis 5.7: Validace vazby ve formuláři `LinkModuleInventoryForm`

Tímto způsobem plugin sjednocuje správu fyzických modulů a jejich evidenci, a zároveň výrazně zjednodušuje proces jejich přesunu mezi zařízeními. Využití více tříd pohledů, formulářů a přechodových stavů ukazuje dobrou separaci aplikační logiky a odpovídá principům návrhu v rámci frameworku Django [7].

Cable Extension – rozšířené vytváření kabelů

Plugin *Cable Extension* rozšiřuje výchozí možnosti NetBoxu při vytváření kabelů. Uživatelo umožňuje propojit dvě rozhraní prostřednictvím kabelu a zároveň nabízí možnost vytvořit odpovídající položky v inventáři (`InventoryItem`) pro oba konce spojení. Nad rámec základní funkcionality ukládá plugin do doplňkového modelu `CableExtension` rozšířené informace o kabelu, jako je výrobce nebo poznámky. Toto rozšíření ovšem nefunguje. Největším přínosem tohoto pluginu je možnost tvorby inventárních položek, což tvoří základ pro volitelné rozšíření C2 a možnost vytvoření kabelu zapojeného pouze na jedné straně [11, 21].

Aplikační logika je implementována ve formuláři `CableCreateForm`, který zachycuje vstupní údaje o propojených rozhraních, typu a délce kabelu a dalších volitelných identifikátorech (např. sériové číslo nebo interní kód dílu). Uživatel může také zvolit, zda si přeje vytvořit inventární položky na obou stranách kabelu.

Validace ve formuláři kontroluje, že zvolená rozhraní nejsou již propojena jiným kabelem:

```
if int_a and int_a.cable:
    self.add_error('interface_a', "Interface A~is already connected!")
if int_b and int_b.cable:
    self.add_error('interface_b', "Interface B is already connected!")
```

Výpis 5.8: Zjednodušený výpis validace dostupnosti rozhraní

V případě validního vstupu se v metodě `save()` provede uložení záznamů. Kabel je vytvořen spolu s připojením k rozhraním a volitelně se vytváří `InventoryItem` pro každý konec. Na závěr se uloží rozšířené informace do `CableExtension`:

```
CableExtension.objects.create(
    cable=cable,
    manufacturer=manufacturer,
    comments=comments
)
```

Výpis 5.9: Zjednodušený výpis uložení rozšířených dat o kabelu

Pro pohodlné vytváření nových kabelů je do menu pluginu registrován odkaz pomocí `PluginMenuItem`, který směřuje na pohled `CableCreateView`. Ten se stará o vykreslení formuláře a zpracování zaslaných dat [21]. Zobrazení rozšířených metadat na detailní stránce kabelu pomocí `PluginTemplateExtension` není funkční. Nepodařilo se mi provést override původní šablony. Jelikož jsem se rozhodl NetBox rozšiřovat čistě pomocí pluginů, tak jsem to nemohl vyřešit jinak. Zde je možné v budoucnu provést úupravy jádra aplikace.

Inventory Viewer – přehled modulů a propojení

Plugin *Inventory Viewer* slouží k přehlednému zobrazení modulů evidovaných v NetBoxu. Zobrazuje jejich základní metadata (např. sériové číslo, evidenční číslo, poznámky) i doplňkové informace získané z vlastních polí jako jsou rok zavedení a měřicí bod. Klíčovou součástí je přehled propojení kabeláží ve formátu „Zařízení/Port ↔ Zařízení/Port“, čímž poskytuje přímou vizualizaci fyzických spojení. Vypisuje i kabely zapojené pouze na jedné straně. V tomto případě je výpis ve formátu „Zařízení/Port ↔ (nezapojeno)“ [21].

Tabulka je implementována jako vlastní třída `ModulyCustomTable`, která rozšiřuje funkcionalitu `django-tables2`. Každý řádek představuje jeden modul, zatímco sloupce obsahují jednotlivé vlastnosti. Vlastní vykreslovací metody (`render_datum`, `render_umisteni`, `render_propojeni`) zajišťují přehledný výstup:

```
if len(real_term_objs) == 2:
    desc_a = describe_real_term(real_term_objs[0])
    desc_b = describe_real_term(real_term_objs[1])
    results.append(f"{desc_a} <-> {desc_b}")
```

Výpis 5.10: Zjednodušený výpis formátování propojení kabelů

Data jsou rozdělena podle typu modulu (`module.module_type`) a zobrazena ve více tabulkách na jedné stránce. To zajišťuje pohled `ModulyListView`, který připravuje seskupená data a předává je do šablony `inventory_view.html` [7, 21].

Odkaz na tuto stránku je přidán do hlavního menu pluginu pomocí `PluginMenuItem`.

5.4 Uživatelské rozhraní pluginů

Každý z implementovaných pluginů rozšiřuje standardní uživatelské rozhraní NetBoxu o nové pohledy a šablony. Tyto prvky jsou navrženy s důrazem na jednoduchost, přehlednost a zachování konzistence s existujícím vzhledem systému. Pluginy využívají vlastní šablony, využívající základní layout NetBoxu (`base/layout.html`) a přidávají nové formuláře nebo jiné komponenty. Cílem této kapitoly je představit, jak jednotlivé pluginy upravují nebo rozšiřují UI a jaké interakce uživatelům umožňují.

Interface Relationship Manager

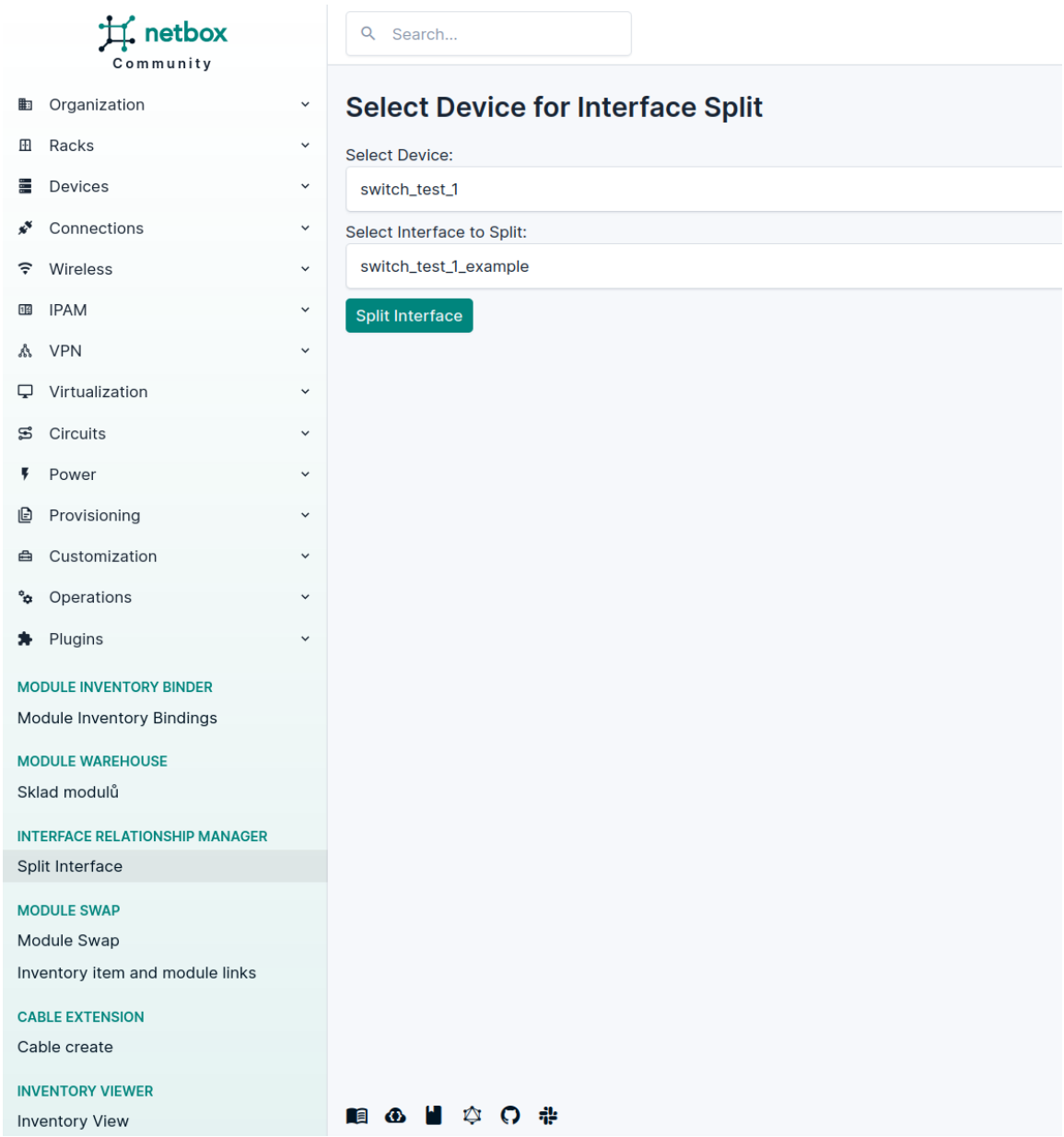
Plugin *Interface Relationship Manager* přidává do rozhraní jednoduchou stránku pro rozdělení síťového rozhraní na dvě podřízená (RX a TX). Tato stránka je dostupná z menu pluginu a je realizována jako vícekrokový formulář s interaktivním načítáním dat na základě uživatelského výběru.

Uživatel nejprve zvolí zařízení, pro které chce vytvořit vztah mezi rozhraními. Následně se mu zobrazí další část formuláře s výběrem dostupných rozhraní, které je možné rozdělit. Výběr rozhraní je podmíněn tím, že dané rozhraní ještě nebylo dříve rozděleno, což je ověřováno pomocí backendové logiky pluginu [12, 21].

Stránka je vytvořena pomocí vlastní šablony, která vychází z výchozí struktury NetBoxu. Formulář nevyužívá JavaScript, interakce je řešena prostým odesláním formuláře při změně výběru zařízení pomocí atributu `onchange="this.form.submit()"`. To zjednodušuje implementaci a zaručuje kompatibilitu s prostředím NetBoxu.

V případě úspěšného nebo neúspěšného rozdělení jsou uživateli zobrazeny zprávy o výsledku operace pomocí systémových hlášení frameworku Django [7].

Z hlediska vzhledu a struktury odpovídá stránka designovým konvencím NetBoxu, a přestože využívá vlastní šablonu, působí jako přirozená součást systému. Obrázek 5.4 znázorňuje výslednou podobu této stránky.



Obrázek 5.4: Ukázka uživatelského rozhraní Interface Relationship Manager pluginu

Module Swap

Plugin *Module Swap* rozšiřuje rozhraní NetBoxu o několik jednoduchých, ale praktických šablon pro správu vazby mezi moduly a inventárními položkami a také o dvoufázové rozhraní pro přesun modulů mezi zařízeními.

Pro správu vazeb plugin poskytuje stránku s přehledem propojených modulů a položek inventáře. Tato tabulka využívá standardní Bootstrap tabulku s možností přidání, úpravy

a odstranění záznamu pomocí tlačítek s ikonami. Významné je, že zobrazuje nejen název modulu a inventárního prvku, ale i příslušná zařízení na obou stranách:

```
<tr>
  <td>{{ link.module }}</td>
  <td>{{ link.inventory_item }}</td>
  <td>{{ link.inventory_item.device|default:"-" }}</td>
  <td>{{ link.module.device|default:"-" }}</td>
  <td>
    <a href="..." class="btn btn-sm btn-primary">Edit</a>
    <a href="..." class="btn btn-sm btn-danger">Delete</a>
  </td>
</tr>
```

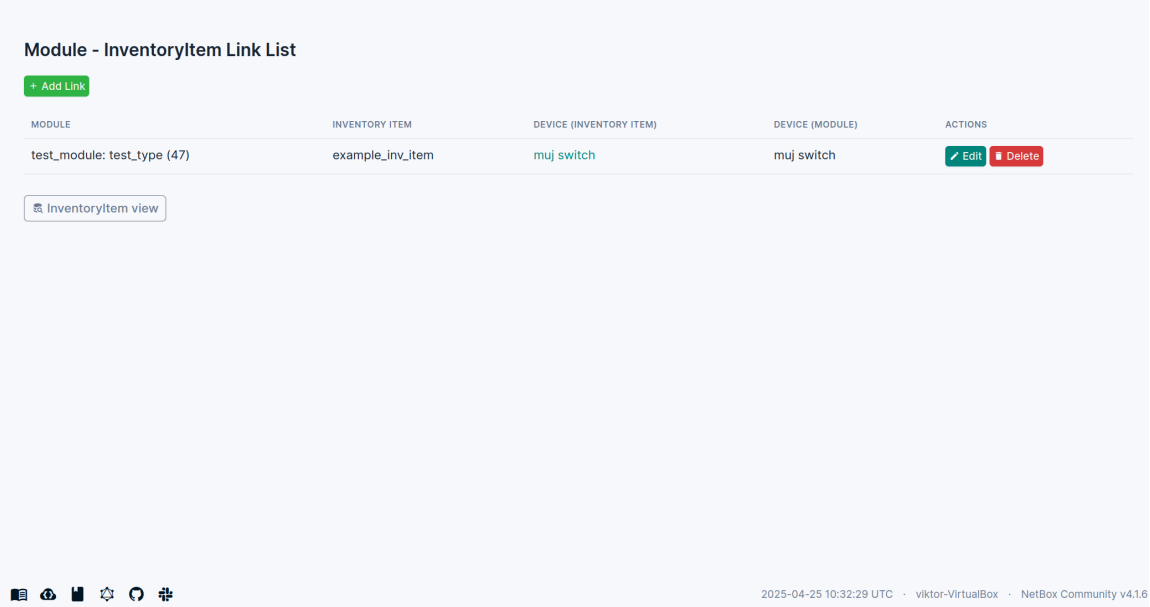
Výpis 5.11: Zjednodušený výpis části tabulky ve výpisu vazeb

Pro vytvoření nové vazby je k dispozici jednoduchý formulář zobrazující se na samostatné stránce. Po odeslání formuláře dojde k validaci a vytvoření nové instance objektu `ModuleInventoryLink`.

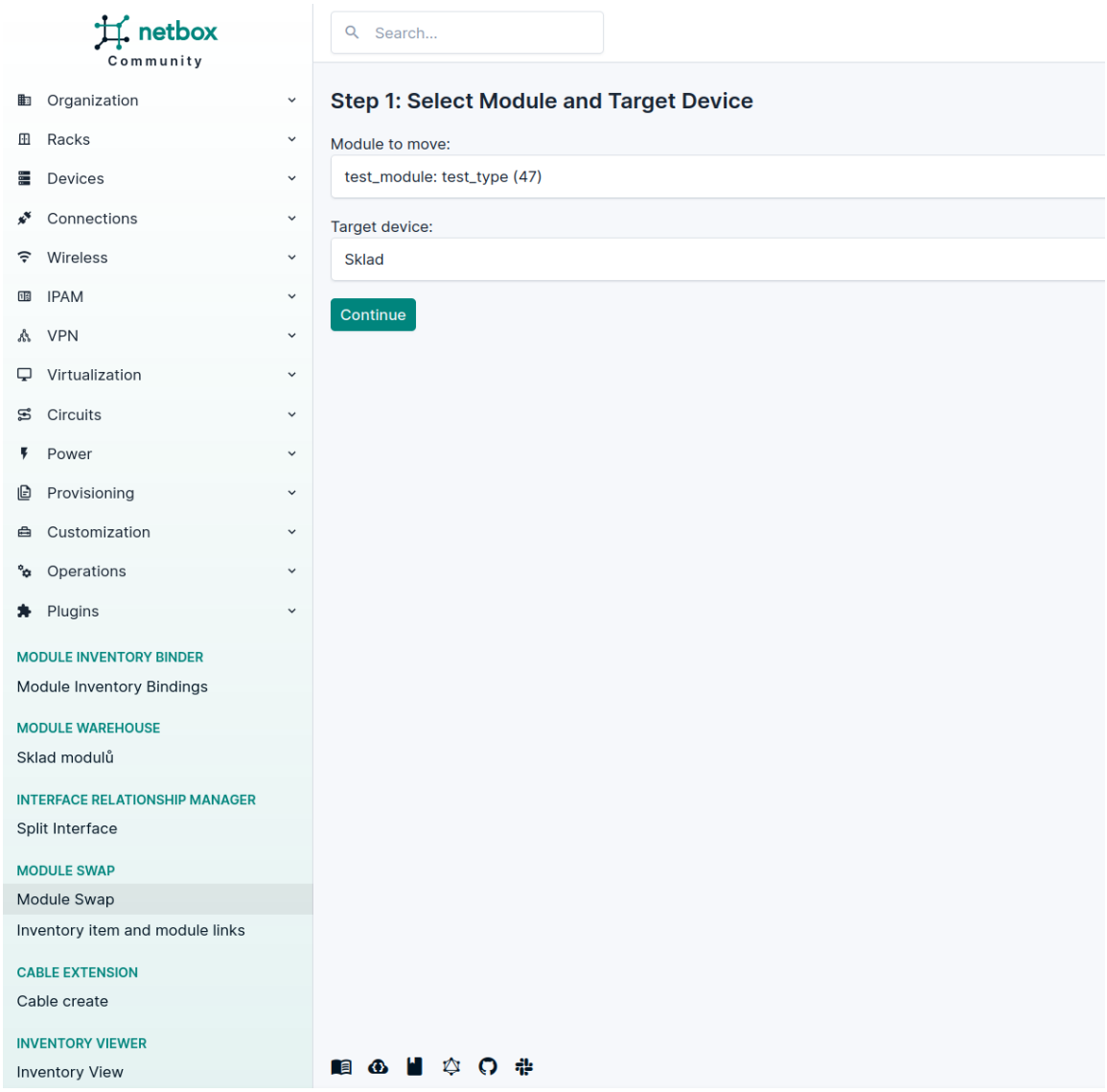
Součástí pluginu je také dvoukrokový formulář pro přesun modulu – nejprve je vybrán modul a cílové zařízení, následně konkrétní slot (`ModuleBay`). Každý krok využívá samostatnou HTML šablonu s jednoduchým formulářem, čímž je zajištěna přehlednost a oddělení jednotlivých kroků.

Zobrazené šablony jsou stylově sladěny s prostředím NetBoxu, což napomáhá snadné orientaci uživatele. Díky jednoduchosti, oddělenému vykreslování jednotlivých kroků a intuitivnímu rozhraní je plugin lehce použitelný. [13, 21].

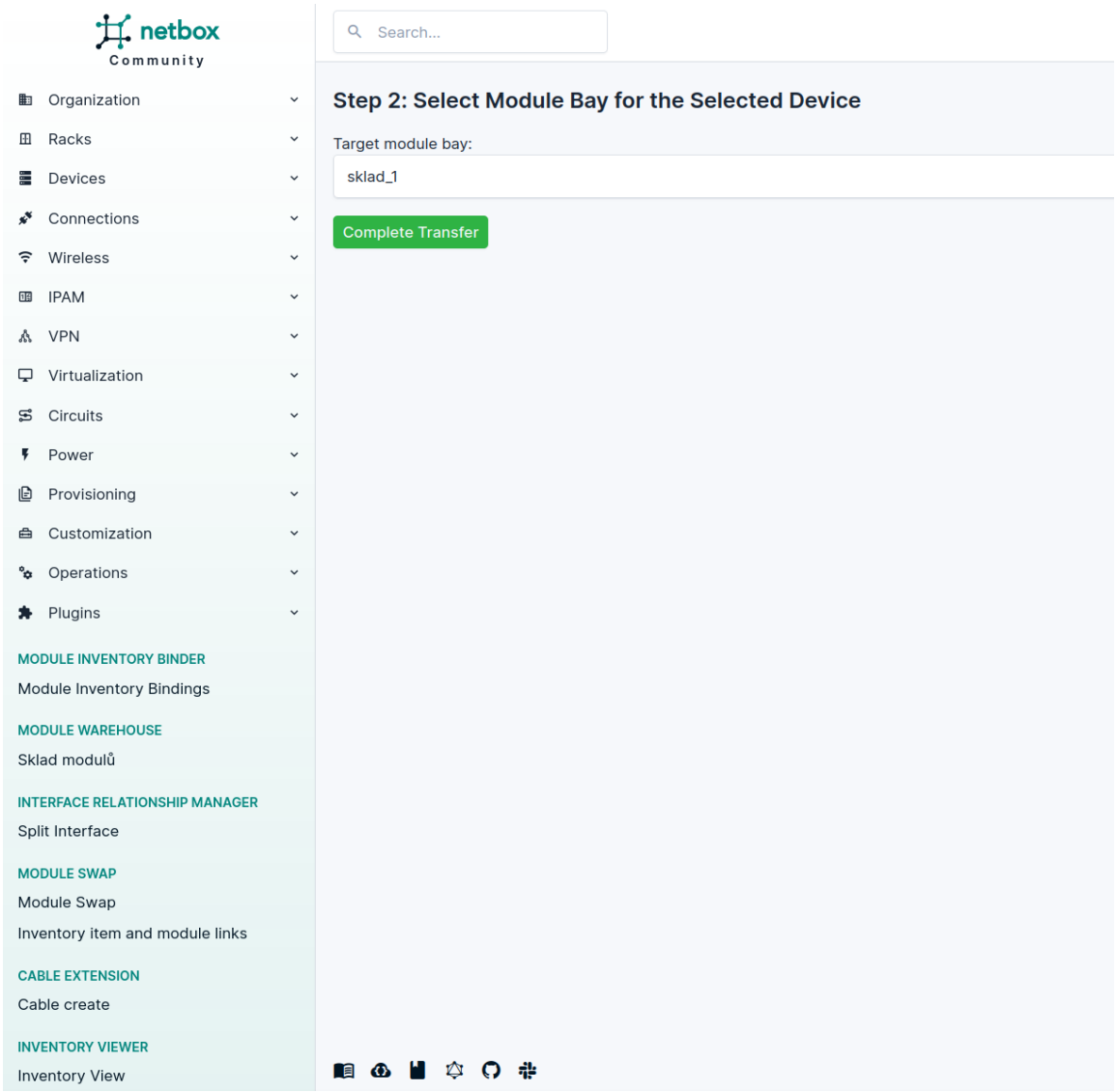
Obrázek 5.5 zachycuje seznam propojení mezi moduly a inventářem. Obrázky 5.6 a 5.7 ilustrují oba kroky procesu přesunu modulu mezi zařízeními.



Obrázek 5.5: Ukázka uživatelského rozhraní seznamu propojení mezi moduly a inventářem



Obrázek 5.6: Ukázka uživatelského rozhraní prvního kroku přenosu modulu



Obrázek 5.7: Ukázka uživatelského rozhraní druhého kroku přenosu modulu

Inventory Viewer

Plugin *Inventory Viewer* rozšiřuje uživatelské rozhraní NetBoxu o stránku, která slouží k přehlednému zobrazení všech modulů (např. FPGA karet), rozdělených podle jejich typu. Každý typ modulu (`ModuleType`) je zobrazen jako samostatná tabulka doplněná nadpisem a popisem daného typu. Zobrazení je realizováno pomocí knihovny `django-tables2` a vlastní tabulky `ModulyCustomTable`, která se vykresluje pomocí filtru `render_table`.

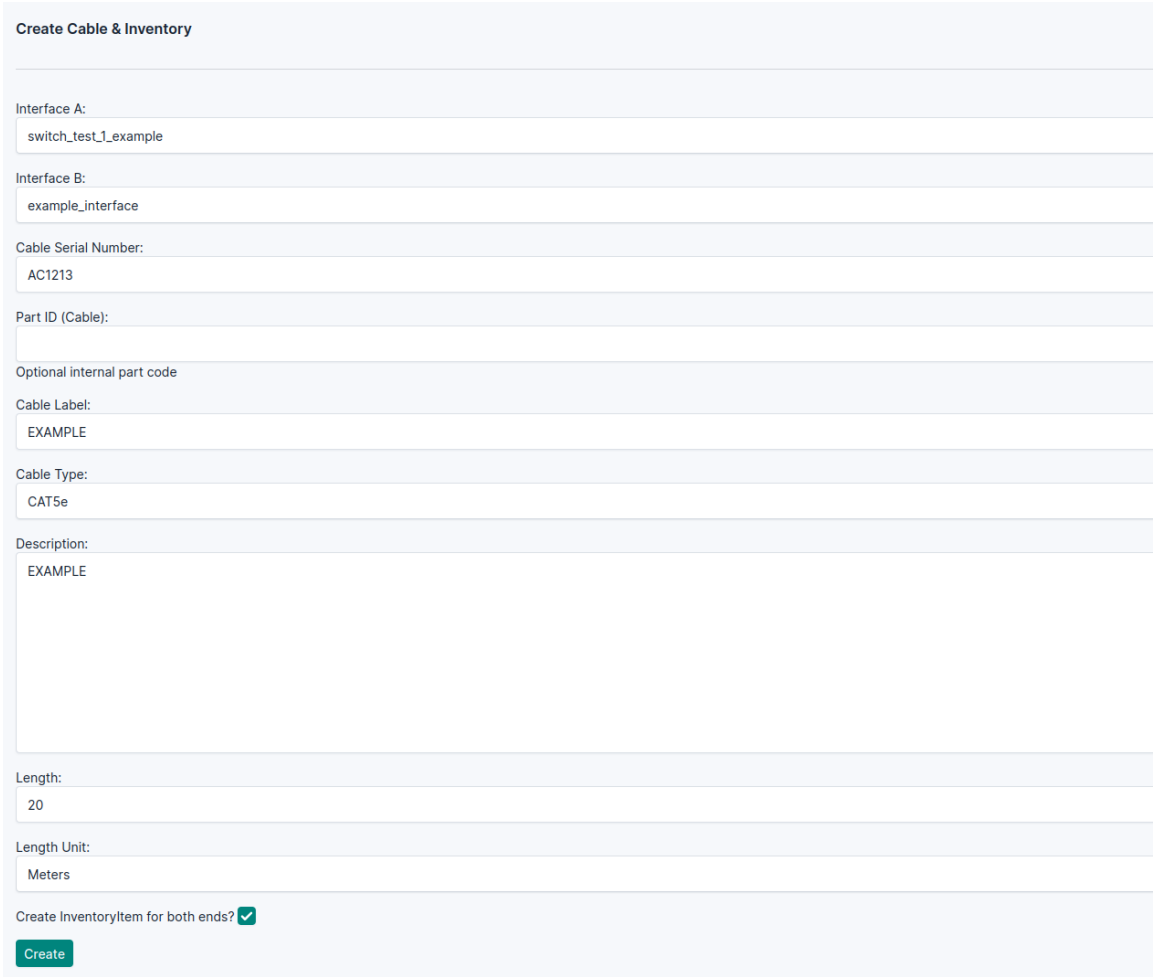
Tato stránka tak slouží jako přehledný výpis modulů, vhodný nejen pro správce systému, ale i pro inventarizační účely. Veškeré údaje pochází z interních datových struktur NetBoxu a doplňkových polí v modelech modulů. Obrázek 5.8 zachycuje příklad zobrazení tabulek na této stránce.

Obrázek 5.8: Ukázka uživatelského rozhraní pluginu Inventory Viewer

Cable Extension

Plugin *Cable Extension* přidává do uživatelského rozhraní dvě nové stránky – formulář pro vytvoření nového propojení kabelu mezi rozhraními (včetně případné tvorby inventárních položek), a editační formulář pro doplňkové metadata uložená v modelu `CableExtension`. Stránka pro vytvoření kabelu využívá jednoduchý formulář založený na NetBoxu, rozšířený o další pole, jako je sériové číslo, poznámka nebo volba, zda vytvořit inventární záznamy. Druhá stránka umožňuje editaci rozšířených metadat kabelu (např. výrobce, komentář). Kromě formulářů plugin měl zobrazovat informace o rozšíření přímo v detailu kabelu pomocí vlastní šablony vložené do stránky pomocí `PluginTemplateExtension`. Toto se mi však nepodařilo zprovoznit.

Obrázek 5.9 zachycuje příklad tvorby kabelu a inventárních položek pro kabely. [11, 21].



Obrázek 5.9: Ukázka uživatelského rozhraní pluginu Cable Extension

5.5 Dokumentace

Každý z implementovaných pluginů byl doplněn o základní technickou dokumentaci. Pluginy obsahují přehledně okomentovaný zdrojový kód a soubor README, který shrnuje jejich hlavní funkcionalitu, způsob použití, požadavky na prostředí a návod k instalaci. Dokumentace zároveň obsahuje seznam funkcí, poznámky k implementaci a přehled změn (changelog). Pro přehlednost jsou také uvedeny typické scénáře použití a struktura projektu. Soubor README je dostupný v každém veřejném repozitáři pluginu (GitHub a PyPi) a slouží jako vstupní bod pro vývojáře i uživatele. Základní orientace v uživatelském rozhraní je navíc podpořena integrací pluginových položek do hlavního menu NetBoxu [21].

5.6 Testování

Automatizované testování nebylo v rámci tohoto projektu implementováno. Funkčnost všech pluginů byla ověřena ručně v rámci vývojového a testovacího prostředí. Ruční testování zahrnovalo typické scénáře použití (např. vytvoření vazby, přesun modulu, vytvoření kabelu,

zobrazení dat v rozhraní). I přes absenci unit testů byla funkcionalita všech částí před odevzdáním práce prověřena.

5.7 Problémy a doporučení

Během vývoje pluginů se objevilo několik výzev. Významným problémem byla nedostatečná dokumentace k tvorbě pluginů pro NetBox, zejména v oblasti práce s REST API. Oficiální tutoriály často postrádají dostatečné vysvětlení a předpokládají jistou znalost vnitřní architektury systému. To výrazně zpomalilo práci, především ze začáku a také to vedlo k tomu, že se mi REST API nepodařilo použít.

Další komplikace nastaly při pokusu o rozšíření uživatelského rozhraní. Použití vlastních šablon pomocí `template overrides` nebo `PluginTemplateExtension` selhávalo, příčinu tohoto chování se mi nepodařilo zjistit. Problémy se objevily také při pokusech o tvorbu vlastních polí (*custom fields*) nebo odkazů (*custom links*) pomocí Django signals. Tyto úpravy by vedly k lepšímu výsledku, i přes hodiny snažení a debugování se mi tyto techniky nepodařilo použít.

Řešení uvedených problémů je ve většině případů možné provést přímou úpravou zdrojového kódu konkrétní NetBox instance nebo využitím možností dostupných přes administrační rozhraní NetBoxu. Tyto přístupy jsou ovšem v rozporu s filozofií pluginového vývoje, a proto nebyly využity.

Pro budoucí vývojáře pluginů v prostředí NetBoxu lze doporučit začít s menším a samostatným pluginem, který pomůže pochopit architekturu NetBoxu a datový model. Využití REST API a úprav existujících šablon pomocí overridu či extensionu je rozhodnutí závislé čistě na vývojáři a jeho sebevědomí a zkušenostech.

Určitě bych doporučil věnovat větší pozornost testování. Ruční ověření sice může být dostačující v rané fázi, ale pro dlouhodobou udržitelnost a stabilitu pluginu je vhodné zavést alespoň základní sadu automatizovaných testů. Toto je něco co jsem možná měl udělat, ale s ničím takovým nemám velkou zkušenost a času už taky bylo málo, proto jsem se rozhodl pro to nejzákladnější a nejlehčí řešení.

Za velmi přínosné považuji využívání komunitních zdrojů, open-source repozitářů a pluginů třetích stran jako inspirace při návrhu struktury, uživatelského rozhraní a rychlejšímu získání informací. Tento přístup může výrazně zkrátit dobu vývoje a předejít častým chybám. Lituji, že jsem tento přístup nezačal využívat dřív v průběhu vývoje, určitě by mi to ulehčilo práci a hodiny strávené laděním chyb, které často ani nebyly mé.

Kapitola 6

Závěr

Cílem této práce bylo analyzovat možnosti rozšíření nástroje NetBox pomocí vlastních pluginů, navrhnout a implementovat konkrétní rozšíření pokrývající potřeby sdružení CESNET a ověřit jejich funkčnost. Během vývoje byly vytvořeny čtyři samostatné pluginy, z nichž každý řešil specifický problém, který nebylo možné jednoduše pokrýt nativní funkcionalitou systému NetBox.

Z pohledu dosažených výsledků lze konstatovat, že všechny plánované cíle byly splněny. Pluginy byly plně integrovány do uživatelského rozhraní NetBoxu, využívají standardní komponenty frameworku Django a respektují konvence systému. Vznikly rozšiřující prvky pro správu vztahů mezi rozhraními, přesuny modulů, evidenci kabelů a přehled modulů v inventáři. Tyto komponenty byly funkčně otestovány a dokumentovány.

Přínosem práce je nejen implementace jednotlivých rozšíření, ale také ověření flexibility NetBoxu pomocí jeho pluginového systému. Práce mi rovněž přinesla nové praktické zkušenosti s vývojem webových aplikací, návrhem uživatelského rozhraní a integrací rozšíření do většího systému. Získané poznatky mohou další lidé využít při budoucím přizpůsobování systému.

Další vývoj se může zaměřit především na rozšíření podpory REST API pro jednotlivé pluginy, automatizaci testování a lepší integraci s dalšími systémy. U některých funkcí by se také vyplatilo zvážit jejich implementaci v jádře NetBoxu. Některé funkce se mi totiž nepodařilo naimplementovat pomocí pluginů. Rozšíření jádra NetBoxu by bylo v těchto případech jednodušší. Dobré by také mohlo být doplnění složitějších typů vztahů mezi komponentami nebo pokročilé vizualizační prvky v uživatelském rozhraní.

Celkově lze výsledky této práce hodnotit jako přínosné nejen z hlediska rozšíření konkrétní instalace systému NetBox, ale i jako praktickou ukázku práce s pluginy. Také jako zdroj nových zkušeností.

Literatura

[1] Ansible Community. *Ansible Documentation* [online]. Red Hat, Inc., 2025. Dostupné z: https://docs.ansible.com/. [cit. 2025-04-20].

[2] Apache Software Foundation. *Apache License, Version 2.0* [online]. Apache Software Foundation, 2025. Dostupné z: https://www.apache.org/licenses/LICENSE-2.0. [cit. 2025-03-03].

[3] Bootstrap Contributors. *Bootstrap Documentation* [online]. Bootstrap, 2025. Dostupné z: https://getbootstrap.com/docs/5.0/getting-started/introduction/. [cit. 2025-03-04].

[4] Celery Project. *Celery — Distributed Task Queue Documentation* [online]. Celery Project, 2025. Dostupné z: https://docs.celeryq.dev/. [cit. 2025-04-20].

[5] CESNET, z.s.p.o. *Webové sídlo* [online]. 2025. Dostupné z: https://www.cesnet.cz/. [cit. 2025-03-06].

[6] Chef Software, Inc. *Chef Documentation* [online]. Progress Software Corporation, 2025. Dostupné z: https://docs.chef.io/. [cit. 2025-04-20].

[7] Django Software Foundation. *Django Documentation* [online]. Django Software Foundation, 2025. Dostupné z: https://docs.djangoproject.com/en/5.2/. [cit. 2025-03-03].

[8] Django Software Foundation. *Django Web* [online]. Django Software Foundation, 2025. Dostupné z: https://www.djangoproject.com/. [cit. 2025-04-22].

[9] Gunicorn Project. *Gunicorn — WSGI Server Documentation* [online]. Gunicorn Project, 2025. Dostupné z: https://docs.gunicorn.org/. [cit. 2025-04-20].

[10] HashiCorp. *Terraform Documentation* [online]. HashiCorp, 2025. Dostupné z: https://developer.hashicorp.com/terraform/docs. [cit. 2025-04-20].

[11] Kubec, Viktor. *Cable Extension Plugin* [online]. GitHub, Inc., 2025. Dostupné z: https://github.com/vubeckubec/cable_extension. [cit. 2025-04-22].

[12] Kubec, Viktor. *Interface Relationship Manager Plugin* [online]. GitHub, Inc., 2025. Dostupné z: https://github.com/vubeckubec/interface_relationship_manager. [cit. 2025-04-22].

[13] Kubec, Viktor. *Module Swap Plugin* [online]. GitHub, Inc., 2025. Dostupné z: https://github.com/vubeckubec/Module_Swap. [cit. 2025-04-22].

[14] NetBox Community. *Deleting a connected port, interface or other connection on a device should not be possible if there's a connected cable — Issue #5418* [online]. GitHub, Inc., prosinec 2020. Dostupné z: https://github.com/netbox-community/netbox/issues/5418. [cit. 2025-04-20].

[15] NetBox Community. *NetBox consuming 100% CPU over time with LDAP authentication enabled — Issue #5194* [online]. GitHub, Inc., září 2020. Dostupné z: https://github.com/netbox-community/netbox/issues/5194. [cit. 2025-04-20].

[16] NetBox Community. *Horizontal scale for performance enhancement — Discussion #11744* [online]. GitHub, Inc., únor 2023. Dostupné z: https://github.com/netbox-community/netbox/discussions/11744. [cit. 2025-04-20].

[17] NetBox Community. *NetBox Documentation* [online]. NetBox labs, 2025. Dostupné z: https://docs.netbox.dev/en/stable/. [cit. 2025-03-03].

[18] NetBox Community. *NetBox GitHub Repository* [online]. GitHub, Inc., 2025. Dostupné z: https://github.com/netbox-community/netbox. [cit. 2025-03-03].

[19] NetBox Community. *NetBox Installation* [online]. NetBox labs, 2025. Dostupné z: https://docs.netbox.dev/en/stable/installation/. [cit. 2025-04-22].

[20] NetBox Community. *NetBox REST API* [online]. NetBox labs, 2025. Dostupné z: https://docs.netbox.dev/en/stable/api/. [cit. 2025-03-04].

[21] NetBox Community. *Plugin Development* [online]. NetBox labs, 2025. Dostupné z: https://docs.netbox.dev/en/stable/plugins/. [cit. 2025-03-04].

[22] NetBox Labs. *NetBox Labs Announces Certified and Supported NetBox Plugins* [online]. NetBox Labs, březen 2024. Dostupné z: https://netboxlabs.com/blog/netbox-labs-announces-certified-and-supported-netbox-plugins/. [cit. 2025-04-20].

[23] PostgreSQL Global Development Group. *PostgreSQL Documentation* [online]. PostgreSQL Global Development Group, 2025. Dostupné z: https://www.postgresql.org/docs/. [cit. 2025-03-04].

[24] Preston Werner, T. *Semantic Versioning Specification (SemVer 2.0.0)* [online]. semver.org, červen 2018. Dostupné z: https://semver.org/. [cit. 2025-04-22].

[25] Puppet, Inc. *Puppet Documentation* [online]. Puppet, Inc., 2025. Dostupné z: https://puppet.com/docs/. [cit. 2025-04-20].

[26] Python Software Foundation. *Python Web* [online]. Python Software Foundation, 2025. Dostupné z: https://www.python.org/. [cit. 2025-04-22].

[27] Redis Ltd. *Redis Documentation* [online]. Redis Ltd., 2025. Dostupné z: https://redis.io/docs/latest/. [cit. 2025-04-20].

[28] Salt Project. *Salt Project Documentation* [online]. VMware, Inc., 2025. Dostupné z: https://docs.saltproject.io/. [cit. 2025-04-20].

[29] Vanhaverbeke, Mattijs. *Netbox Topology Views Plugin* [online]. GitHub, Inc., 2025. Dostupné z: https://github.com/netbox-community/netbox-topology-views. [cit. 2025-04-29].