

Návrh systému pro inventarizaci a vizualizaci IT infrastruktury

Petr Přikryl

Bakalářská práce
2025



Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav informatiky a umělé inteligence

Akademický rok: 2024/2025

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení: Petr Příkryl
Osobní číslo: A22082
Studijní program: B0613A140020 Softwarové inženýrství
Forma studia: Prezenční
Téma práce: Návrh systému pro inventarizaci a vizualizaci IT infrastruktury
Téma práce anglicky: Design of a System for Inventory and Visualization of IT Infrastructure

Zásady pro vypracování

- Analyzujte problematiku a vypracujte literární rešerši na dané téma.
- Vypracujte stručný rozbor technologií které budou použity k návrhu systému.
- Provedte analýzu požadavků pro návrh systému.
- Navrhněte, realizujte a otestujte aplikaci s využitím vhodných technologií.
- Věnujte pozornost zabezpečení systému.

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

1. TRULOVE, James. Sítě LAN: hardware, instalace a zapojení. Profesionál. Praha: Grada, 2009. ISBN 9788024720982.
2. KUROSE, James F. a ROSS, Keith W. Počítačové sítě. Brno: Computer Press, 2014. ISBN 9788025138250.
3. TANENBAUM, Andrew S. a WETHERALL, D. Computer networks. Fifth edition. New Delhi: Dorling Kindersley, [2014]. ISBN 9789332518742.
4. HELALI, Saida. Systems and network infrastructure integration: design, implementation, safety and supervision. London: ISTE, 2020. ISBN 9781119779964.
5. POKORNÝ, Jaroslav a VALENTA, Michal. Databázové systémy. 2. přepracované vydání. Vysokoškolská učebnice. Praha: Česká technika – nakladatelství ČVUT, 2020. ISBN 978-80-01-06696-6.

Vedoucí bakalářské práce: **Ing. David Malaník, Ph.D.**
Ústav informatiky a umělé inteligence

Datum zadání bakalářské práce: **27. října 2024**
Termín odevzdání bakalářské práce: **2. června 2025**



L.S.

doc. Ing. Jiří Vojtěšek, Ph.D. v.r.
děkan

prof. Mgr. Roman Jašek, Ph.D., DBA v.r.
ředitel ústavu

Ve Zlíně dne 17. listopadu 2024

PROHLÁŠENÍ AUTORA BAKALÁŘSKÉ PRÁCE

Beru na vědomí, že

- odevzdáním bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb., v platném znění bez ohledu na výsledek obhajoby;
- bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému a bude dostupná k nahlédnutí;
- jedno vyhotovení bakalářské práce v listinné podobě bude ponecháno Univerzitě Tomáše Bati ve Zlíně k uložení;
- na moji bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- podle § 60 odst. 1 autorského zákona má Univerzita Tomáše Bati ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- podle § 60 odst. 2 a 3 mohu užít své dílo – bakalářskou práci – nebo poskytnout licenci k jejímu využití jen s předchozím písemným souhlasem Univerzity Tomáše Bati ve Zlíně, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše);
- pokud bylo k vypracování bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tj. k nekomerčnímu využití), nelze výsledky bakalářské práce využít ke komerčním účelům;
- pokud je výstupem bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá; neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji, že

- jsem na bakalářské práci pracoval(a) samostatně a použitou literaturu jsem řádně citoval(a); v případě publikace výsledků budu uveden(a) jako spoluautor;
- odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou obsahově totožné.

Ve Zlíně, dne

.....
podpis autora

ABSTRAKT

Tato bakalářská práce se zabývá návrhem a implementací webové aplikace pro správu síťových zařízení a jejich vizualizaci. Cílem bylo vytvořit moderní a uživatelsky přívětivé rozhraní pro sledování zařízení, jejich stavu (online/offline), statistiky a vzájemné propojení v síti.

Frontend aplikace je postaven pomocí knihovny React a Material UI, zatímco backend využívá Node.js, GraphQL a databázi PostgreSQL. Komunikace mezi klientem a serverem probíhá pomocí GraphQL API, které zajišťuje efektivní a flexibilní práci s daty. Bezpečnost přístupu je řešena pomocí autentizace pomocí JWT tokenů, přičemž systém podporuje přihlášení, správu uživatelů a role.

Aplikace dále umožňuje automatické sledování dostupnosti zařízení pomocí pravidelného pingování, zaznamenávání historie stavu zařízení, vzdálené probouzení zařízení přes Wake-on-LAN, export CSV a responzivní vizualizaci topologie sítě pomocí D3.js a knihoven pro automatické rozložení grafů.

Součástí práce je také testování funkčnosti, výkonu a bezpečnosti celého systému a vyhodnocení dosažených výsledků.

Klíčová slova: webová aplikace, vizualizace sítě, správa zařízení, React, Node.js, GraphQL, PostgreSQL, JWT autentizace, Wake-on-LAN, monitorování stavu zařízení

ABSTRACT

This bachelor's thesis focuses on the design and implementation of a web application for managing network devices and visualizing their topology. The goal was to create a modern and user-friendly interface for monitoring devices, their online/offline status, statistics, and interconnections within the network.

The frontend of the application is built using the React library and Material UI, while the backend utilizes Node.js, GraphQL, and a PostgreSQL database. Communication between the client and server is handled via a GraphQL API, ensuring efficient and flexible data handling. Secure access is enforced through JWT-based authentication, supporting login, user management, and user roles.

The application also enables automatic device status monitoring through regular pinging, logging of device status history, remote device wake-up via Wake-on-LAN, CSV export functionality, and responsive network topology visualization using D3.js and graph layout libraries.

The thesis also includes functionality, performance, and security testing of the entire system, along with an evaluation of the achieved results.

Keywords: web application, network visualization, device management, React, Node.js, GraphQL, PostgreSQL, JWT authentication, Wake-on-LAN, device status monitoring

Děkuji své rodině za trpělivost, podporu a víru ve chvílích, kdy i kompilátor ztrácel trpělivost. Bez vás by tato práce vznikala o poznání déle a s menším úsměvem.

Zvláštní poděkování patří mému vedoucímu práce, Ing. Davidu Malaníkovi, Ph.D., za cenné rady, vstřícnost a odborné vedení po celou dobu řešení této práce.

OBSAH

ÚVOD.....	15
I TEORETICKÁ ČÁST	16
1 INVENTARIZACE A VIZUALIZACE IT INFRASTRUKTURY	17
1.1 VÝZNAM A POTŘEBA SPRÁVY IT INFRASTRUKTURY	17
1.2 PŘÍSTUPY K INVENTARIZACI IT INFRASTRUKTURY	18
1.2.1 Manuální evidence	19
1.2.2 Automatizované skenování sítě.....	19
1.2.3 CMDB systémy	20
1.3 PŘÍSTUPY K VIZUALIZACI IT INFRASTRUKTURY	20
1.3.1 Statické diagramy a schémata	21
1.3.2 Dynamická topologická zobrazení	22
2 PŘEHLED EXISTUJÍCÍCH ŘEŠENÍ KE SPRÁVĚ IT INFRASTRUKTURY	23
2.1 KOMERČNÍ VS. OPEN-SOURCE SOFTWARE	23
2.1.1 NetBox	24
2.1.2 RackTables	25
2.1.3 Device42	25
2.1.4 ManageEngine OpManager	26
3 TECHNOLOGIE POUŽITÉ PŘI NÁVRHU SYSTÉMU	28
3.1 DOCKER	28
3.2 DATABÁZOVÝ SYSTÉM POSTGRESQL	28
3.3 TECHNOLOGIE PRO VIZUALIZACI IT INFRASTRUKTURY	29
3.3.1 D3.js	30
3.4 BACKENDOVÉ FRAMEWORKY	30
3.4.1 Node.js	31
3.4.2 Express.js	32
3.4.3 Apollo server	33
3.4.4 Pg (node-postgres)	33
3.5 FRONTENDOVÉ FRAMEWORKY	34
3.5.1 React.....	34
3.5.2 Apollo Client.....	35
3.5.3 Material UI	35
3.6 BEZPEČNOSTNÍ ASPEKTY IT SYSTÉMŮ	36
3.6.1 Bcrypt.....	37
3.6.2 JWT	38
3.7 TESTOVÁNÍ.....	39
3.7.1 Apollo Sandbox – interaktivní prostředí pro GraphQL	39
3.7.2 Lighthouse.....	40

II	PRAKTICKÁ ČÁST	42
4	NÁVRH.....	43
4.1	FUNKČNÍ POŽADAVKY	43
4.1.1	Evidence zařízení	43
4.1.2	Zobrazení online/offline stavu zařízení	43
4.1.3	Historie dostupnosti zařízení	43
4.1.4	Vyhledávání.....	44
4.1.5	Vizualizace sítě.....	44
4.1.6	Wake-on-LAN	44
4.1.7	Správa uživatelů	44
4.1.8	Role a oprávnění.....	44
4.1.9	Přehled statistik	44
4.1.10	Uživatelské rozhraní.....	44
4.2	NEFUNKČNÍ POŽADAVKY	45
4.2.1	Dostupnost systému.....	45
4.2.2	Výkon	45
4.2.3	Odezva.....	45
4.2.4	Zabezpečení dat.....	45
4.2.5	Oprávnění	45
4.2.6	Kompatibilita.....	45
4.2.7	Uživatelská přívětivost.....	45
4.3	PŘÍPADY UŽITÍ	46
4.4	SCÉNÁŘE PŘÍPADŮ UŽITÍ.....	47
4.5	DATABÁZOVÝ MODEL	54
5	NÁVRH SYSTÉMU	56
5.1	POUŽITÉ TECHNOLOGIE	56
5.2	STRUKTURA PROJEKTU	56
5.3	NÁVRH UŽIVATELSKÉHO ROZHRANÍ	57
5.3.1	Záložka Vizualizace	57
5.3.2	Správa zařízení	59
5.3.3	Správa uživatelů	61
5.3.4	Inventarizace	62
5.3.5	Statistiky	65
5.3.6	Uživatelské menu	65
5.3.7	Přihlášení	66
5.3.8	Wallboard	67
5.4	BEZPEČNOSTNÍ OPATŘENÍ.....	67
5.4.1	Backend	67
5.4.2	Frontend.....	68
5.4.3	Databáze	68
6	IMPLEMENTACE SYSTÉMU	69
6.1	VÝVOJ DATABÁZOVÉ VRSTVY	69

6.1.1	docker-compose.yml	69
6.1.2	DB.js.....	70
6.1.3	db_init.sql.....	70
6.2	BACKENDOVÁ ČÁST APLIKACE	71
6.2.1	GraphQL	71
6.2.2	index.js	73
6.2.3	Dockerfile a docker_compose.yml.....	75
6.3	FRONTENDOVÁ ČÁST APLIKACE	76
6.3.1	App.js	76
6.3.2	GraphSwitcher.js	78
6.3.3	NetworkGraph.js	78
6.3.4	NetworkGraph2.js	80
6.3.5	NetworkGraph3.js	81
6.3.6	DeviceManager.js.....	82
6.3.7	Login.js.....	83
6.3.8	UserManagement.js.....	84
6.4	ZABEZPEČENÍ SYSTÉMU.....	85
6.4.1	Security Headers	86
7	TESTOVÁNÍ A VYHODNOCENÍ.....	89
7.1	METODIKA TESTOVÁNÍ.....	89
7.2	FUNKČNÍ TESTY.....	89
7.3	VÝKONOVÉ TESTY.....	91
ZÁVĚR		93
SEZNAM POUŽITÉ LITERATURY.....		95
SEZNAM OBRÁZKŮ		99
SEZNAM TABULEK.....		101
SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK.....		102
SEZNAM PŘÍLOH.....		104

ÚVOD

Digitalizace a rostoucí komplexita IT prostředí kladou na správu infrastruktury stále vyšší nároky. Přehled o aktuálním stavu zařízení, jejich dostupnosti i propojení se tak stává klíčovým předpokladem pro efektivní řízení informačních systémů. Kromě technických parametrů je však důležité evidovat i majetkovou stránku zařízení, jejich fyzické umístění, odpovědné osoby a stav z hlediska inventarizace.

Cílem této bakalářské práce je návrh a realizace webového systému, který umožní nejen správu a vizualizaci IT infrastruktury, ale také její efektivní inventarizaci. Výsledná aplikace nabízí moderní a přehledné uživatelské rozhraní, podporu uživatelských rolí, statistiky, historii dostupnosti zařízení, nástroje pro automatizované sledování stavu i funkci Wake-on-LAN. Součástí je rovněž samostatný modul pro inventuru, který umožňuje evidenci zařízení i mimo aktivní síť.

Práce se opírá o moderní technologie jako jsou React, Material UI, GraphQL, Node.js a PostgreSQL, přičemž je kladen důraz na přehlednost, bezpečnost a rozšiřitelnost systému. Projekt je zároveň připraven pro nasazení v prostředí Docker.

I TEORETICKÁ ČÁST

1 INVENTARIZACE A VIZUALIZACE IT INFRASTRUKTURY

V oblasti informačních technologií se pod pojmem infrastruktura rozumí soubor základních technologických prostředků a systémů, které tvoří nezbytné zázemí pro provoz IT služeb a aplikací. Zahrnuje jak fyzické komponenty, jako jsou servery, síťová zařízení nebo datová úložiště, tak i softwarové nástroje, operační systémy a další podpůrné technologie. [1]

V prostředí datových center spadá pod infrastrukturu například výpočetní technika, síťová konektivita, napájecí jednotky či systémy chlazení. V oblasti webhostingu pak infrastruktura tvoří technologický základ, který zajišťuje spolehlivý chod webových stránek, databází či cloudových platforem, a to jak z hlediska výkonu, tak bezpečnosti a dostupnosti. [1]

1.1 Význam a potřeba správy IT infrastruktury

S rostoucí mírou digitalizace organizací získává IT infrastruktura stále větší význam. Je nezbytná nejen pro každodenní provoz, ale i pro rozvoj a zajištění technologické připravenosti organizace. Správa IT prostředí tak vyžaduje důraz na stabilitu, přizpůsobitelnost, bezpečnost i hospodárnost. [2]

Důležitou vlastností infrastruktury je možnost škálování a přizpůsobení měnícím se požadavkům. Systémy by měly umožnit jednoduché rozšiřování výpočetních zdrojů, úložišť nebo síťových kapacit bez nutnosti přerušení provozu. To se často řeší implementací modulárních prvků nebo využitím cloudových služeb, které umožňují pružnou reakci na aktuální potřeby. [2]

Monitorování síťové infrastruktury slouží k průběžnému vyhodnocování jejího technického stavu a výkonnosti, čímž umožňuje včasné odhalení přetížených spojů, chybné dimenzace nebo možných poruch zařízení. Zahrnuje dohled nad aktivními prvky (routery, switche, přístupové body), servery a jejich službami, koncovými zařízeními, operačními systémy i samotnými aplikacemi, včetně sledování jejich dostupnosti a využití systémových prostředků. [40]

Dostupnost a stabilita infrastruktury je dalším důležitým aspektem. Zahrnuje využití záloh, systémů obnovy po výpadku a pravidelného monitorování. Cílem je zamezit ztrátě dat a minimalizovat provozní přerušení. Sledování provozních metrik navíc umožňuje včasné odhalení a nápravu nedostatků. [2]

Nezbytnou součástí správy IT prostředí je také zabezpečení a soulady s předpisy. Týká se jak ochrany fyzického zařízení, tak dat a přístupů. Pro zajištění bezpečného provozu je

potřeba zavést šifrování, řízení oprávnění, pravidelné audity a proškolení zaměstnanců. Tato opatření napomáhají snížení rizik spojených s lidským faktorem a naplnění legislativních požadavků. [2]

Rychlost reakce na síťový problém záleží pouze na tom, jak rychle dokážeme chybu najít. [39]

Díky správě a monitorování IT infrastruktury, lze ihned poznat, že například přestala fungovat internetová karta a údržba ji může okamžitě vyměnit. [38]

Plánování topologie sítě je velmi podstatné pro návrh a správné fungování celé sítě. [37]

Významnou roli dnes hrají cloudové platformy, které organizacím umožňují snížit náklady na vlastní infrastrukturu, zjednodušit přístup ke službám a podpořit práci na dálku. Volba mezi veřejným, privátním nebo hybridním modelem závisí na charakteru provozu a požadavcích na bezpečnost a výkon. [2]

Moderní správa IT často zahrnuje i automatizaci běžných operací. Nástroje využívající umělou inteligenci nebo strojové učení usnadňují analýzu dat, správu provozu a zavádění úprav. Díky tomu je možné lépe reagovat na změny a optimalizovat využití dostupných zdrojů. [2]

1.2 Přístupy k inventarizaci IT infrastruktury

Inventarizace IT prostředků představuje zásadní nástroj pro zajištění přehledu o technologickém vybavení organizace. Systematická evidence zařízení, jejich technických parametrů, lokalizace a vztahů k dalším prvkům infrastruktury je nezbytná nejen pro každodenní správu, ale také pro plánování, bezpečnostní dohled a auditní účely. [26]

V prostředí, kde je infrastruktura složitější a častěji se mění, se ruční vedení inventáře ukazuje jako časově náročné, náchylné k chybám a obtížně udržitelné. Fragmentace dat mezi různými systémy nebo odděleními pak výrazně komplikuje správu i rozhodovací procesy. Z tohoto důvodu se stále častěji uplatňují přístupy založené na digitalizaci a automatizaci evidence. Tyto systémy umožňují centralizovaný přístup k aktuálním informacím, čímž snižují administrativní zátěž a zvyšují přesnost záznamů. [26]

K efektivní inventarizaci lze přistupovat několika způsoby v závislosti na potřebách organizace, úrovni technické vyspělosti a dostupných nástrojích. [26]

1.2.1 Manuální evidence

Mezi nejzákladnější přístupy k evidenci IT majetku patří ruční správa záznamů pomocí tabulkových editorů. V tomto modelu je každé nové zařízení při pořízení zaznamenáno do předem připraveného formuláře, typicky v podobě sdílené tabulky. Záznamy zahrnují základní informace, jako je název zařízení, typ, umístění nebo přidělený uživatel, a spravují je pověření zaměstnanci. [27]

Tento způsob vedení inventáře je technicky nenáročný, avšak v praxi vykazuje omezenou efektivitu. Správa dat tímto způsobem bývá časově náročná, přičemž ruční zadávání informací zvyšuje riziko chybovosti. V případě většího počtu zařízení může být obtížné udržet záznamy aktuální a konzistentní, zejména pokud dochází ke změnám v majetkové struktuře nebo přemisťování zařízení mezi uživateli a lokalitami. [27]

Manuální evidence je obvykle využívána v menších organizacích, kde je rozsah spravované infrastruktury omezený. Ve větších podnicích, kde je požadována vyšší míra přesnosti a pravidelná aktualizace, je tento způsob obtížně udržitelný. [27]

1.2.2 Automatizované skenování sítě

Automatizované skenování sítě představuje moderní přístup k inventarizaci IT infrastruktury, který umožňuje efektivní a pravidelný sběr dat o síťových zařízeních bez nutnosti manuálního zásahu. Tento přístup je založen na použití specializovaných nástrojů, které jsou schopny automaticky detekovat všechna aktivní zařízení v síti a shromažďovat o nich potřebné informace. [28]

Skenování probíhá typicky dvěma způsoby – bezagentově a s využitím agentů. Při bezagentovém přístupu není třeba na jednotlivá zařízení instalovat žádný doplňkový software. Identifikace zařízení a jejich parametrů probíhá pomocí standardních protokolů, jako jsou například WMI, SNMP nebo SSH. Agentové řešení naproti tomu spoléhá na malé komponenty nainstalované přímo na cílových strojích, které následně poskytují detailnější údaje. [28]

Výsledkem automatizovaného skenování je komplexní souhrn informací o síťové infrastruktuře – včetně počítačů, serverů, tiskáren, síťových prvků i softwarového vybavení. Systém může data pravidelně aktualizovat a archivovat změny, čímž poskytuje aktuální a přesný inventární přehled. To výrazně snižuje časovou náročnost oproti manuálním metodám a zároveň minimalizuje riziko lidské chyby. [28]

Automatizované síťové skenování je tak vhodným řešením pro střední a větší organizace, kde je třeba spravovat rozsáhlý a dynamicky se měnící soubor zařízení a komponent infrastruktury. [28]

1.2.3 CMDB systémy

Konfigurační databáze (CMDB) představuje specifický nástroj pro evidenci a sledování IT infrastruktury z technického hlediska. Na rozdíl od systémů zaměřených na majetkovou evidenci, které se zabývají vlastnictvím, náklady a účetní stránkou IT prostředků, se CMDB soustředí na uchování informací o technických součástech infrastruktury a jejich vzájemných vztazích. [29]

Databáze tohoto typu obsahuje přesná a aktuální data o jednotlivých prvcích IT prostředí, například o serverech, aplikacích, síťových zařízeních, databázích nebo poskytovaných službách. Klíčovým rysem CMDB je schopnost uchovávat i informace o tom, jak spolu jednotlivé položky souvisejí. Díky tomu je možné snadněji porozumět tomu, jaké následky by mohl mít výpadek jednoho prvku na ostatní části infrastruktury nebo přímo na běžící IT služby. [29]

CMDB je užitečná zejména v oblastech, kde je potřeba sledovat dopady selhání, analyzovat příčiny incidentů nebo plánovat technologické změny. Správně vedená databáze pomáhá zvyšovat přehled o prostředí, zkracuje čas potřebný pro řešení provozních problémů a podporuje rozhodování na základě znalosti reálných vazeb mezi jednotlivými komponentami. [29]

1.3 Přístupy k vizualizaci IT infrastruktury

Síťová vizualizace, známá také jako grafová vizualizace nebo analýza propojení, představuje způsob grafického zobrazení sítí složených z propojených prvků. V tomto kontextu uzly reprezentují jednotlivé datové body a spoje mezi nimi znázorňují vzájemné vztahy. [32]

Cílem vizualizace je překročit omezení lineárních nebo tabulkových modelů a umožnit lepší pochopení souvislostí v datech. S využitím specializovaných nástrojů je možné takové vztahy zachytit a analyzovat vizuální formou. [32]

Výhody síťové vizualizace:

- Intuitivnost

- Uzly a spojení mezi nimi působí přirozeně a jsou snadno srozumitelné i pro uživatele bez předchozích zkušeností. Vizualizace pomáhá zjednodušit složité struktury a zpřehlednit jejich uspořádání. [32]
- Rychlost porozumění
 - Díky vizuálnímu formátu lze snadno rozpoznat vzorce, odchylky a klíčová spojení. To výrazně zrychluje orientaci v datech. [32]
- Flexibilita
 - Interaktivní zobrazení umožňuje zaměřit se na konkrétní části sítě nebo získat celkový přehled. Vizualizace usnadňuje uchopení komplexních datových struktur. [32]
- Hlubší vhled do souvislostí
 - Interaktivní práce s daty umožňuje rychleji pochopit jejich kontext a vztahy, které by ve statickém nebo agregovaném zobrazení zůstaly skryté. [32]

1.3.1 Statické diagramy a schémata

Statické diagramy představují důležitý nástroj pro vizualizaci IT infrastruktury. Jejich cílem je přehledně a jednoznačně zobrazit strukturu sítě, rozmístění zařízení a vztahy mezi nimi. Typicky se jedná o ručně nebo nástrojem vytvořené schéma, které zachycuje síťovou topologii v určitém čase a slouží jako dokumentace skutečného stavu prostředí. [30]

Existují dva hlavní typy síťových diagramů: fyzické a logické. Fyzický diagram znázorňuje konkrétní fyzické umístění zařízení včetně kabeláže a síťových spojení. Lze zde najít například servery, přepínače, směrovače nebo racky a jejich vzájemné propojení. Tento typ diagramu je užitečný zejména při fyzickém plánování nebo údržbě infrastruktury. [30]

Naopak logický diagram se soustředí na logickou strukturu sítě, tedy na to, jak data proudí mezi jednotlivými částmi systému. Zachycuje například IP adresy, VLANy, podsítě, firewallové zóny nebo další logické komponenty. Pomáhá lépe porozumět komunikaci mezi zařízeními a podpořit správné návrhy síťové architektury. [30]

Dobře navržený diagram zjednodušuje složité infrastruktury a pomáhá technickému týmu i dalším zainteresovaným stranám rychle porozumět aktuálnímu stavu sítě. Usnadňuje také řešení provozních problémů, plánování změn nebo komunikaci v rámci IT oddělení. [30]

1.3.2 Dynamická topologická zobrazení

Na rozdíl od statických schémat, která zobrazují navrženou podobu sítě nebo její stav v určitém bodě minulosti, poskytují dynamická topologická zobrazení aktuální pohled na to, jak síť v daném okamžiku skutečně funguje. Tento typ mapování zobrazuje změny v síti v reálném čase a umožňuje tak rychlou orientaci v aktuální infrastruktuře. [31]

Dynamické mapy jsou zpravidla interaktivní – správci mají možnost podrobně prozkoumat jednotlivé části sítě, například spojení, zařízení nebo toky dat. Díky tomu lze snadno dohledat zdroje problémů, přetížení nebo nečekané změny konfigurace. [31]

Automatizace procesu zjišťování a vykreslování zajišťuje vysokou úroveň detailu, která by byla při ručním vytváření velmi časově náročná. Zatímco manuálně tvořené diagramy často končí na úrovni přepínačů, protože podrobné zakreslení koncových bodů není praktické, dynamická vizualizace běžně zachycuje i tyto části – a to průběžně, bez nutnosti zásahu ze strany uživatele. [31]

Díky těmto vlastnostem jsou nástroje pro síťové zobrazení s automatickým mapováním a průběžnou aktualizací výrazně účinnější při správě změn, analýze provozu a řešení incidentů než klasické statické mapy, které je třeba neustále ručně udržovat. [31]

2 PŘEHLED EXISTUJÍCÍCH ŘEŠENÍ KE SPRÁVĚ IT INFRASTRUKTURY

Informační technologie tvoří nedílnou součást provozu většiny moderních organizací. V případě větších podniků se správa digitální infrastruktury často stává komplexním úkolem, zejména pokud jde o hardwarové komponenty, jako jsou servery, pracovní stanice či síťová zařízení. Právě v tomto kontextu vstupují do hry nástroje a platformy pro správu IT infrastruktury. [33]

Tyto systémy umožňují správcům monitorovat a udržovat klíčové prvky jejich sítí. Poskytují detailní přehled o stavu zařízení a dokážou v reálném čase upozornit na vzniklé problémy, jako je výpadek serveru, nefunkční směrovač nebo nedostupnost připojení u konkrétní stanice. Správce tak získává možnost včasně reagovat a předejít rozsáhlejším incidentům. [33]

Hlavním přínosem těchto řešení je zajištění provozní stability bez ohrožení bezpečnosti spravovaných prvků. Díky centralizované správě a přímému dohledovému rozhraní lze infrastrukturu efektivněji řídit, rychle odhalovat slabá místa a plánovat zásahy s minimálním dopadem na běžný chod organizace. [33]

2.1 Komerční vs. open-source software

V oblasti softwaru rozlišujeme dvě základní kategorie: open-source, tedy aplikace s volně dostupným zdrojovým kódem, a komerční řešení s uzavřeným vývojem, často spojená s licenčními poplatky. Oba přístupy mají své výhody i omezení, a jejich volba závisí na konkrétních potřebách uživatele nebo organizace. [22]

Rozdíly mezi komerčními a open-source softwary:

- cenová dostupnost:
 - Open-source nástroje jsou zpravidla dostupné bezplatně nebo s minimálními náklady. Naopak komerční software bývá spojen s pravidelnými poplatky – formou předplatného nebo jednorázové licence. Bezplatné verze komerčních produktů bývají omezeny funkčně nebo obsahují reklamy. [22]
- stabilita a spolehlivost:
 - U komerčních produktů je stabilita často garantována, protože za vývojem stojí profesionální týmy s jasně definovanou odpovědností. Naproti tomu

open-source projekty mohou obsahovat nedokumentované chyby nebo být neplně kompatibilní s určitou konfigurací. Často se předpokládá technická zdatnost uživatele. [22]

- možnosti přizpůsobení:
 - Komerční software je zpravidla navržen pro snadné použití, ale bývá méně flexibilní. Pokročilé úpravy chování aplikace jsou obvykle omezeny. Naopak open-source nástroje umožňují široké možnosti úprav – za předpokladu, že má uživatel dostatečné znalosti a přístup ke kódu. [22]

Oba přístupy mají své místo – open-source ocení technicky orientovaní uživatelé hledající přizpůsobení, zatímco komerční software nabízí pohodlí a profesionální podporu. Výběr by měl vycházet z konkrétní situace, požadované úrovně kontroly, nároků na údržbu a rozpočtových možností. [22]

2.1.1 NetBox

NetBox je moderní open-source platforma určená pro správu, dokumentaci a modelování síťové infrastruktury. Od svého vzniku v roce 2016 se stal klíčovým nástrojem pro inženýry a správce sítí po celém světě. Slouží jako centrální zdroj pravdy o síťových prostředcích a jejich topologii. [24]

NetBox neslouží k přímému řízení zařízení, ale poskytuje rozhraní a API, které zjednodušují automatizaci, monitoring a integraci s dalšími nástroji. [24]

Klíčové vlastnosti:

- databáze síťových prvků – evidence zařízení, racků, kabelů, IP rozsahů, VLANů, obvodů, napájení a dalších prvků,
- API a automatizace – REST API a webhooks umožňují napojení na nástroje jako Ansible, Terraform nebo monitoring,
- rozšiřitelnost – podpora vlastních polí, štítků a pluginů pro přizpůsobení konkrétním potřebám,
- role a oprávnění – detailní řízení přístupů pro jednotlivé uživatele a týmy,
- konfigurace zařízení – generování konfigurací pomocí šablon (např. Jinja2) a možnost je zpřístupnit přes API,

- skriptování a události – vlastní skripty a akce spouštěné na základě změn v systému,
- záznam změn – kompletní historie všech úprav, včetně informací o tom, kdo je provedl. [24]

2.1.2 RackTables

RackTables je open-source systém určený k evidenci a správě fyzických i virtuálních IT prostředků. Pomáhá organizacím udržet přehled o síťových zařízeních, serverech, IP adresách a rack skříních. Díky tomu umožňuje efektivní organizaci, plánování a vizualizaci infrastruktury v datových centrech i menších provozech. RackTables je distribuován pod licencí GPLv3 a je dostupný zdarma. [23]

Klíčové vlastnosti:

- správa zařízení a sítí – umožňuje zaznamenávat informace o síťových prvcích, serverech, připojeních i přidělených IP adresách,
- přehledná vizualizace – nabízí možnosti grafického zobrazení racků a jejich obsazení,
- pokročilé filtrování – výkonné vyhledávání a třídění dat podle různých kritérií,
- modulární a škálovatelné řešení – vhodné pro menší firmy i rozsáhlá datová centra. [23]

2.1.3 Device42

Device42 je komerční platforma určená pro komplexní správu IT prostředí, která kombinuje nástroje pro automatické objevování zařízení, správu prostředků, mapování závislostí a řízení datových center. Je navržena s ohledem na potřeby hybridních infrastruktur, tedy prostředí kombinujících on-premise, virtualizované i cloudové komponenty. Díky rozsáhlé funkcionalitě je Device42 vhodný zejména pro větší organizace, které vyžadují přesný přehled o infrastruktuře, snadnější plánování migrací, optimalizaci využití zdrojů, efektivní dokumentaci a dodržování auditních a bezpečnostních požadavků. [25]

Klíčové vlastnosti:

- automatizované objevování a mapování – platforma bez potřeby instalace agentů zajišťuje průběžné skenování a identifikaci fyzických i virtuálních serverů, síťových

prvků, úložišť, kontejnerů, pracovních stanic a dalších zařízení. Zároveň zobrazuje jejich vzájemné závislosti,

- IT Asset management a CMDB – Device42 poskytuje robustní správu prostředků a databázi konfiguračních položek (CMDB), která uchovává přesná a aktuální data o celé infrastruktuře včetně vztahů mezi objekty,
- správa softwarových licencí – umožňuje sledovat nasazené aplikace na úrovni serverů i klientských zařízení a spravovat software včetně informací o verzích, výrobcích nebo koncích podpory,
- vizualizace a DCIM nástroje – součástí jsou nástroje pro vizualizaci racků, místností a celých budov s možností přímé správy napájení, kabeláže, teploty i zátěže zařízení,
- IPAM a síťové přehledy – vestavěný systém pro správu IP adres poskytuje jednotný pohled na adresní prostor, využití, rezervace a síťové rozsahy,
- nadstavbové funkce s využitím AI – moduly Insights+, InsightsAI a EnrichAI umožňují vytvářet vlastní přehledy, obohacovat data z externích zdrojů a získávat doporučení pro migraci do cloudu i správu životního cyklu zařízení. [25]

2.1.4 ManageEngine OpManager

ManageEngine OpManager je komerční řešení určené pro monitoring výkonu sítě a infrastruktury, které se řadí mezi přední nástroje v oblasti IT Operations Management (ITOM). Platforma je navržena tak, aby organizacím poskytla přehled o zdraví a dostupnosti jejich síťových prvků, serverů a aplikací v reálném čase. Díky své modulární a škálovatelné architektuře je vhodná jak pro menší IT týmy, tak pro rozsáhlá podniková prostředí. Mezi nejčastější scénáře nasazení patří optimalizace síťové výkonnosti, dohled nad servery a podpora kontinuálního provozu. [26]

Klíčové vlastnosti:

- monitoring síťových zařízení – sleduje výkon směrovačů, prepínačů, přístupových bodů a dalších prvků infrastruktury. Pomáhá identifikovat přetížení a úzká místa v síti,
- správa serverů – OpManager podporuje dohled nad fyzickými i virtuálními servery (Hyper-V, VMware, Citrix, Nutanix) a sleduje klíčové metriky jako dostupnost, vytížení CPU, paměti a diskových operací,

- vizualizace a mapování sítě – nabízí topologické mapy (Layer 2), zobrazení racků, 3D půdorysy i pohledy na obchodní procesy (Business Views) pro lepší orientaci v infrastruktuře,
- monitoring RTT a VoIP – měří odezvu WAN spojení a sleduje kvalitu hlasových služeb v síti, což je klíčové pro stabilní provoz VoIP řešení,
- přizpůsobitelné dashboardy – umožňuje tvorbu vlastních panelů s metrikami dle potřeb administrátora nebo konkrétního týmu. [26]

3 TECHNOLOGIE POUŽITÉ PŘI NÁVRHU SYSTÉMU

Volba správné technologie představuje klíčové rozhodnutí, které může zásadně ovlivnit úspěch nebo neúspěch celého projektu. Výběr konkrétního technologického řešení totiž výrazně přispívá k efektivitě vývoje, schopnosti systému škálovat s rostoucími požadavky a k celkovému fungování výsledného softwarového produktu. [35]

3.1 Docker

Docker je platforma pro vytváření a spouštění aplikací v tzv. kontejnerech – izolovaných prostředích, která obsahují veškeré komponenty potřebné pro běh softwaru. Kontejnery umožňují provoz více aplikací na jednom systému bez vzájemného rušení a s minimálními nároky na výkon. [3]

Na rozdíl od tradičních virtuálních strojů kontejnery nesimulují celý operační systém, ale sdílejí jádro hostitele. Díky tomu jsou výrazně lehčí, rychlejší a snadněji nasaditelné. Vývojář tak získává jistotu, že aplikace poběží konzistentně v jakémkoli prostředí. [3]

Docker je složen z:

1. Docker Image – šablona, podle které vzniká kontejner. Obsahuje operační prostředí, knihovny a aplikační kód,
2. Dockerfile – textový soubor s instrukcemi pro vytvoření image,
3. Kontejner – běžící instance image, izolovaná, ale efektivní na prostředky. [3]

Hlavní výhody:

- přenositelnost – aplikaci lze spustit lokálně, na serveru i v cloudu bez úprav,
- izolace – každá aplikace má vlastní prostředí, což eliminuje konflikty,
- rychlost a efektivita – spuštění kontejneru trvá řádově sekundy a vyžaduje méně zdrojů než virtuální stroj. [3]

3.2 Databázový systém PostgreSQL

PostgreSQL je vyspělý databázový systém s otevřeným kódem, který v sobě spojuje principy relačního modelu s podporou objektových konceptů. Uživatelům nabízí širokou škálu funkcí, jako je práce s vlastními datovými typy, podpora JSON formátu, pokročilé indexování nebo transakce splňující ACID vlastnosti. Díky těmto schopnostem se

PostgreSQL stal vyhledávaným řešením jak pro běžné webové aplikace, tak i pro rozsáhlé podnikové systémy s vysokými nároky na zátěž a bezpečnost. [4]

Významnou výhodou je jeho škálovatelnost a možnost rozšiřování prostřednictvím modulů a rozšíření, která umožňují například práci s geografickými daty (PostGIS), připojení externích databází či integraci s nástroji pro strojové učení. Systém je vyvíjen komunitou, která neustále reaguje na aktuální trendy v oblasti datových technologií. [4]

V prostředí cloudových služeb, zejména pak na platformě Microsoft Azure, je PostgreSQL dostupný jako plně spravovaná databáze. To znamená, že uživatelé nemusejí řešit správu infrastruktury, zálohování, bezpečnostní aktualizace nebo škálování. Tato služba umožňuje soustředit se čistě na vývoj a správu dat, přičemž systém zajišťuje vysokou dostupnost, zabezpečení i automatickou údržbu. [4]

Nabízí také další typy objektů jako jsou trigger, sekvence nebo synonyma. [36]

Díky těmto vlastnostem je PostgreSQL v současnosti považován za jeden z nejvyspělejších systémů pro správu databází, který nachází uplatnění jak v soukromém, tak veřejném sektoru, včetně financí, zdravotnictví, vědy i státní správy. [4]

3.3 Technologie pro vizualizaci IT infrastruktury

Vzhledem k rostoucí složitosti a rozsahu podnikových sítí je vizualizace IT infrastruktury klíčovým nástrojem pro efektivní správu. Pomocí grafů, diagramů a topologických map lze snadno získat přehled o architektuře sítě, jejím aktuálním stavu a vztazích mezi jednotlivými zařízeními. Tato vizualizace umožňuje rychle identifikovat chybně nakonfigurované prvky, poruchy nebo podezřelou síťovou aktivitu, což zjednodušuje diagnostiku a zkracuje reakční dobu při řešení incidentů. [34]

Síťová vizualizace navíc poskytuje cenné informace pro kapacitní plánování. Zobrazení aktuálních dat ve formě přehledných grafů a diagramů napomáhá při hodnocení využití prostředků, objemu přenesených dat nebo vytíženosti jednotlivých segmentů infrastruktury. Díky tomu lze předcházet přetížení a optimalizovat výkon celého systému. [34]

Ačkoli je možné síťové mapy vytvářet manuálně, jedná se o časově náročný proces, který často nezachycuje aktuální změny v topologii. Automatizované nástroje pro vizualizaci proto představují efektivnější řešení, protože poskytují aktuální pohled na síť v reálném čase a usnadňují každodenní provozní správu. [34]

3.3.1 D3.js

D3.js, neboli Data-Driven Documents, představuje efektivní knihovnu pro programovací jazyk JavaScript, určenou k vykreslování vizuálních struktur dat na webu. Slouží jako prostředek pro transformaci datových vstupů do grafické podoby přímo ve struktuře HTML dokumentu. Díky propojení s technologiemi SVG a CSS nabízí vývojářům flexibilní nástroj pro tvorbu vlastních vizualizací, které mohou být statické či dynamické. [5]

Na rozdíl od jiných knihoven, D3.js pracuje na principu manipulace s prvky DOM, a to na základě datových struktur. To znamená, že umožňuje přiřazení dat k HTML prvkům, jejich automatickou aktualizaci, a dokonce i animaci v reálném čase. D3.js je vhodná zejména tehdy, když je požadována vysoká míra přizpůsobení, což zahrnuje i tvorbu nestandardních grafů a zcela originálních vizualizačních komponent. [5]

Mezi zásadní přednosti této knihovny patří:

- plná kontrola nad výsledným výstupem vizualizace,
 - interaktivní chování prvků na základě akcí uživatele,
 - přístup k širokému spektru datových zdrojů – od jednoduchých souborů CSV po API rozhraní,
 - komunita vývojářů, která neustále knihovnu rozvíjí a poskytuje příklady použití.
- [5]

Použití D3.js sahá od analytických dashboardů přes datové reporty až po vizualizaci sítí, vztahů nebo hierarchických struktur. Umožňuje nejen zobrazovat statistické informace, ale také sledovat vývojové trendy nebo prezentovat velké datové objemy přehledně a interaktivně. [5]

Vzhledem ke své síle, flexibilitě a podpoře otevřeného kódu se D3.js stala oblíbenou volbou ve světě webového vývoje, kde vizualizace hraje klíčovou roli při interpretaci složitých datových celků. [5]

3.4 Backendové frameworky

Frameworky pro backend představují prostředky, které usnadňují programátorům návrh stabilních i rozšiřitelných internetových systémů. Tyto systémy zahrnují sadu předpřipravených nástrojů, modulů či metod, které zjednodušují tvorbu i správu serverové části softwaru. Používají se k různorodým činnostem, jako je manipulace s databázemi,

transformace informací, ověřování údajů, tvorba odpovědí nebo zabezpečení aplikace. Mezi PHP frameworky dominují Laravel spolu se Symfony. Laravel nabízí čisté rozhraní a spoustu vestavěných schopností, jako je mapování objektů na tabulky, práce s prostředními vrstvami, ukládání do cache nebo systém pro ověřování uživatelů. Symfony naopak staví na modulární architektuře, která dovoluje použití jeho částí nezávisle nebo jako celek. Důležité rysy zahrnují správu závislostí, práci s událostmi a formulářové komponenty. [6]

3.4.1 Node.js

Node.js představuje volně dostupné prostředí, které umožňuje provádět JavaScriptové skripty mimo klientský prohlížeč, konkrétně na úrovni serveru. Ačkoli byl jazyk JavaScript původně určen pro běh ve webových rozhraních, jeho využití díky Node.js přesáhlo tuto doménu a umožnilo vývoj výkonných a flexibilních serverových aplikací. Platforma staví na vysoce optimalizovaném enginu V8 od Googlu, jenž kompiluje kód přímo do instrukcí srozumitelných procesoru, čímž umožňuje rychlou exekuci výpočetně náročných operací, například při práci s daty nebo při zajištění síťové komunikace. [7]

Z hlediska architektury využívá Node.js asynchronní systém založený na událostech, který umožňuje efektivní paralelní zpracování velkého množství požadavků, aniž by bylo nutné čekat na ukončení jednotlivých akcí. Tato vlastnost je zásadní zejména pro nástroje vyžadující obsluhu více současných připojení, jako jsou chatovací rozhraní, rozhraní API nebo streamovací systémy. Node.js funguje v režimu s jediným vláknem, což zjednodušuje správu paměti a zároveň snižuje režii na systémové prostředky. Neoddělitelnou součástí je také nástroj NPM, který poskytuje vývojářům přístup k rozsáhlé knihovně balíčků a modulů pro urychlení a zefektivnění implementace. [7]

Mezi hlavní oblasti nasazení platformy Node.js patří real-time aplikace, jako jsou interaktivní komunikátory, notifikační systémy nebo online herní prostředí, dále rozhraní typu API pro rychlé zpracování požadavků a služby pro přenos multimediálního obsahu. Díky své univerzálnosti se Node.js často využívá nejen pro vývoj serverových částí systémů, ale také jako součást plnohodnotných full-stack řešení v kombinaci s moderními frontendovými technologiemi. [7]

Výhody:

- škálovatelnost: Umožňuje efektivní zpracování velkého množství současných požadavků,

- znalost jazyka JavaScript: Většina vývojářů jej již ovládá, není třeba se učit nový jazyk,
- jednotné použití JS: Možnost psát jak backend, tak frontend ve stejném jazyce,
- silná komunita: Aktivní vývoj a velké množství dostupných knihoven a modulů,
- nízké nároky na paměť: Asynchronní model snižuje spotřebu systémových prostředků. [8]

Nevýhody:

- nestabilita API: Časté změny v rámci vývoje mohou narušit zpětnou kompatibilitu,
- nutnost přepisování kódu: Aktualizace Node.js nebo knihoven často vyžadují úpravy existujícího řešení. [8]

3.4.2 Express.js

Express představuje široce využívaný framework založený na platformě Node.js, určený pro vývoj serverových aplikací. Je charakterizován svou jednoduchostí, výkonností a flexibilitou. Díky své nenucené filozofii (unopinionated) vývojářům nevnučuje konkrétní metodiky či struktury, ale dává prostor pro vlastní rozhodnutí ohledně architektury a použitých komponent. [9]

Jeho obliba vyplývá z nenáročné instalace, intuitivního použití a snadné integrace s rozsáhlým ekosystémem JavaScriptových knihoven. Díky těmto vlastnostem se Express často uplatňuje při návrhu rozhraní API, vývoji menších projektů a tvorbě funkčních prototypů. [9]

Hlavní funkce:

- směrování požadavků: umožňuje přehledné definování pravidel pro zpracování různých HTTP metod a URL cest,
- middleware vrstvy: podporuje zřetězení funkcí mezi přijetím požadavku a odesláním odpovědi (např. autentizace, logování, úprava vstupních dat),
- šablonová podpora: zajišťuje generování dynamických stránek,
- obsluha statických dat: zpřístupňuje soubory jako CSS, JavaScript nebo obrázky přímo klientovi,

- modularita: lze jej snadno kombinovat s dalšími balíčky pro rozšíření funkcionality – od databází až po autentizační mechanismy. [9]

3.4.3 Apollo server

Apollo Server je open-source server GraphQL vyhovující specifikacím, který bezproblémově spolupracuje s jakýmkoli klientem GraphQL, včetně klienta Apollo. Tento nástroj umožňuje vytvářet robustní, snadno dokumentovatelná a produkčně použitelná rozhraní, nezávislá na konkrétním datovém zdroji. [10]

Klíčové vlastnosti:

- jednoduché spuštění: umožňuje rychlé sestavení základního serveru pro vývoj datových dotazů,
- postupná integrace: uživatelé mohou přidávat nové funkce podle potřeby – vhodné pro malé i rozsáhlé projekty,
- široká kompatibilita: nezávislý na konkrétním úložišti dat, nástroji pro sestavení ani klientovi GraphQL,
- připravenost pro nasazení: architektura je navržena pro stabilní provoz v reálném prostředí. [10]

3.4.4 Pg (node-postgres)

Node-postgres je knihovna určená pro práci s databází PostgreSQL v prostředí Node.js. Umožňuje provádět databázové operace pomocí více stylů – podporuje zpětná volání, promise i moderní asynchronní zápis pomocí `async/await`. Nabízí také funkce jako sdružování připojení (connection pooling), připravené SQL dotazy, práci s kurzory, streamování výsledků nebo napojení na nativní rozšíření v C/C++. Poradí si i s pokročilou analýzou typů dat. [11]

Knihovna je navržena tak, aby usnadnila rychlé nasazení a zároveň nabídla nástroje pro řešení složitějších situací. Dokumentace poskytuje nejen základní přehled, ale také rozšiřující návody pro využití pokročilých funkcí PostgreSQL přímo z prostředí Node.js. [11]

3.5 Frontendové frameworky

Front-end frameworky představují sadu nástrojů a předpřipravených prvků, které vývojářům usnadňují návrh uživatelských rozhraní v rozsáhlých a udržitelných aplikacích. Obsahují znovupoužitelné moduly psané v jazycích HTML, CSS a JavaScript, což přispívá k jednotnému stylu a přehledné architektuře kódu. [12]

Tato řešení poskytují vývojovou kostru, která urychluje práci, podporuje přehlednost a snižuje chybovost při vývoji klientské části aplikací. Frameworky typicky nabízejí nástroje pro:

- generování šablon a stylů,
- návrh responzivního rozhraní kompatibilního s různými zařízeními,
- zajištění vizuální i funkční konzistence napříč platformami,
- automatizaci procesů,
- správu kódu a jeho aktualizace. [12]

Každý framework má specifické vlastnosti, které jej odlišují od ostatních. Výběr konkrétní technologie by měl vycházet z povahy projektu, očekávané škálovatelnosti a požadavků na funkčnost i design. [12]

3.5.1 React

React, vyvinutý společností Facebook v roce 2011, je open-source knihovna v jazyce JavaScript určená pro tvorbu interaktivních uživatelských rozhraní. Zaměřuje se na vizuální vrstvu aplikace a umožňuje vývoj opakovaně využitelných komponent, které usnadňují strukturovaný návrh rozhraní. [12]

Jednou z hlavních výhod Reactu je využití virtuálního DOMu, tedy interní paměťové reprezentace skutečného Document Object Modelu. Tato technika umožňuje rychlé porovnání a efektivní vykreslení změn, čímž dochází ke zlepšení výkonu při aktualizaci uživatelského rozhraní. Jednosměrný tok dat zároveň zjednodušuje správu aplikačního stavu. React se navíc snadno učí a je doplněn o nástroje jako Redux, který přináší centralizovanou správu dat. [12]

Facebook také poskytuje React Native, samostatný nástroj pro vývoj nativních mobilních aplikací, využívající stejnou architekturu komponent jako React pro webové prostředí.

I přes své výhody má React určité slabiny. Komplexní struktura komponent může vést k obtížnějšímu ladění a méně přehledné kódové základně. Framework nepřináší správu směrování nebo stavů jako výchozí součást – je nutné doplnit externími knihovnami, např. React Router či Redux. [12]

Rychlý vývoj knihovny způsobuje, že dokumentace nemusí být vždy aktuální, což komplikuje začátky novým uživatelům. Kromě toho může být pro začátečníky nezvyklá syntaxe JSX, která kombinuje JavaScript s HTML strukturou přímo v kódu komponenty. [12]

3.5.2 Apollo Client

Apollo Client je komplexní knihovna pro správu stavů v JavaScriptových aplikacích, která umožňuje efektivní práci s daty z GraphQL rozhraní i lokálního úložiště. Je optimalizována pro integraci s moderními frontendovými technologiemi, zejména s knihovnou React a poskytuje vývojářům sadu nástrojů pro práci s daty. [10]

Klíčové vlastnosti:

- deklarativní přístup k datům: dotazy lze definovat přímo uvnitř komponent, což zjednodušuje logiku zobrazení,
- automatické ukládání do mezipaměti: výsledky dotazů jsou ukládány a opětovně využívány, což vede ke snížení počtu požadavků a zrychlení aplikace,
- jednotná správa stavů: umožňuje sjednotit práci s lokálními hodnotami i vzdálenými daty v rámci jednoho systému,
- vývojářská podpora: obsahuje nástroje pro ladění a optimalizaci dotazů nad GraphQL rozhraním. [10]

3.5.3 Material UI

Material UI (MUI) je knihovna stylovacích a komponentových nástrojů pro React, která implementuje vizuální jazyk Material Design, původně představený společností Google v roce 2014. Framework poskytuje sadu předdefinovaných a snadno přizpůsobitelných prvků uživatelského rozhraní, které umožňují rychlý vývoj profesionálně vypadajících webových a mobilních aplikací. Cílem tohoto systému je vizuální a funkční konzistence napříč různými platformami i zařízeními. [13]

MUI vychází z detailně definované návrhové metodiky, která klade důraz na prvky jako typografie, struktura rozložení, barevná schémata, měřítko a hierarchii obsahu. Díky tomu je možné navrhovat rozhraní, která jsou intuitivní, esteticky sladěná a optimalizovaná pro obchodní využití. [13]

Vývojáři oceňují MUI pro jeho bohatou nabídku připravených komponent, které lze snadno začlenit do projektu bez nutnosti psát rozhraní od základu. Kromě toho je k dispozici systém tvorby motivů, který umožňuje upravovat globální vzhled – například barevné palety, vlastnosti povrchů či stíny – a tím dosáhnout jednotného vizuálního stylu celé aplikace. [13]

Díky kvalitní dokumentaci, která obsahuje praktické příklady a podrobné návody, je práce s touto knihovnou přístupná i méně zkušeným uživatelům. Kromě technické stránky hraje významnou roli i aktivní komunita a vývojářský tým, který pravidelně publikuje aktualizace, opravuje chyby a sbírá zpětnou vazbu od uživatelů. [13]

3.6 Bezpečnostní aspekty IT systémů

Ochrana informačních systémů představuje klíčový prvek v rámci správy digitální infrastruktury organizací. Cílem je snížení pravděpodobnosti hrozeb a zajištění spolehlivého a nepřerušovaného fungování. Tohoto stavu se dosahuje zaváděním bezpečnostních politik, sledováním datových toků v síti a prováděním pravidelných kontrolních mechanismů. [16]

Hlavní oblasti kybernetického zabezpečení:

- ochrana sítě se zaměřuje na obranu proti neoprávněnému přístupu do technické infrastruktury, která tvoří základ provozu digitálních služeb,
- zabezpečení dat zahrnuje opatření na ochranu důvěrnosti, konzistence a dostupnosti uchovávaných informací,
- reakce na bezpečnostní události zahrnuje schopnost rozpoznat, vyhodnotit, zvládnout a obnovit provoz v případě incidentu,
- etické útoky známé jako penetrační testy umožňují odhalit slabá místa systému ještě předtím, než je zneužije reálný útočník. [16]

Kybernetické hrozby zahrnují široké spektrum technik, mezi něž patří škodlivý software (například viry, červi či ransomware), podvodné e-maily (phishing), zahlcující útoky na služby (DoS), nebo zachytávání komunikace třetí stranou. Bezpečnostní specialisté proto

musejí průběžně sledovat nově vznikající metody útoků a přizpůsobovat obranné mechanismy. [16]

Role člověka v kybernetickém zabezpečení:

Zajištění ochrany systémů není pouze otázkou technologií. Uživatelé představují významný faktor – často právě oni bývají nejslabším článkem. Z toho důvodu je nezbytné investovat do školení v oblasti bezpečného chování, například správného vytváření hesel, rozpoznávání podvodných zpráv nebo práce s citlivými daty. [16]

Právní rámec a bezpečnostní normy:

Kybernetická ochrana není pouze technická otázka – významnou roli hrají i legislativní požadavky a mezinárodní standardy. Nařízení jako GDPR definují povinnost chránit osobní údaje a zajišťovat jejich důvěrnost. Směrnice NIS2, která se promítá i do českých zákonů, přináší přísnější pravidla pro organizace ve strategických oblastech. [16]

Z hlediska řízení bezpečnosti se uplatňují normy jako ISO/IEC 27001, které stanovují rámec pro ochranu informací v organizacích. Dodržování těchto standardů zvyšuje důvěryhodnost firmy, zlepšuje její připravenost na incidenty a pomáhá plnit legislativní povinnosti. [16]

3.6.1 Bcrypt

Algoritmus bcrypt slouží k ochraně hesel pomocí kryptografického hashování. Vytvořili ho Niels Provos a David Mazières jako odolné řešení založené na šifře Blowfish. Samotný název spojuje označení „b“ pro Blowfish a „crypt“, což je tradiční hashovací funkce využívaná v UNIX systémech. [14]

Hlavním důvodem vzniku bcrypt byla potřeba odolnější ochrany přihlašovacích údajů vzhledem k rostoucí výpočetní síle moderního hardwaru. Oproti starším metodám byl bcrypt záměrně navržen tak, aby výpočet hashe trval déle – čímž komplikuje útoky typu brute-force. [14]

Klíčové vlastnosti algoritmu bcrypt:

- **Adaptivní návrh:** Jednou z hlavních výhod bcrypt je schopnost přizpůsobit se vývoji výpočetních technologií. Algoritmus umožňuje zvyšovat náročnost výpočtu podle potřeby, což znamená, že čím je hardware výkonnější, tím lze navýšit úroveň ochrany. Tato vlastnost zásadně ztěžuje útoky metodou hrubé síly. [15]

- **Salted by Default:** Při každém zpracování hesla generuje bcrypt jedinečný náhodný řetězec – tzv. salt. Tento doplněk se následně připojí k původnímu vstupu před samotným hashováním. Díky tomu nelze použít předem vypočítané tabulky (rainbow tables), protože stejné heslo nikdy nevede ke stejnému výstupu. [15]
- **Vysoká výpočetní složitost:** Ve srovnání s jednoduššími funkcemi jako MD5 nebo SHA-1 je bcrypt výrazně náročnější na zpracování. Vyžaduje větší množství systémových prostředků, což zpomaluje pokusy o prolomení, a tím zvyšuje úroveň zabezpečení uložených údajů. [15]

3.6.2 JWT

JWT (JSON Web Token) je otevřený standard určený k bezpečnému přenosu informací mezi systémy pomocí krátkých, samostatných tokenů. Tyto tokeny se běžně využívají při autentizaci a autorizaci uživatelů, protože mohou obsahovat údaje ověřující identitu a oprávnění konkrétní osoby. [17]

Z hlediska ověřování uživatele JWT pomáhá navázat důvěru mezi serverem a neznámým klientem bez potřeby ukládání relace na straně serveru. [17]

Struktura tokenu JWT:

1. Hlavička (Header) – určuje typ tokenu a použitý algoritmus pro digitální podpis (např. HS256 nebo RS256),
2. Tělo (Payload) – obsahuje vlastní data, tzv. nároky (claims), například uživatelské jméno, e-mail nebo role,
3. Podpis (Signature) – vzniká kryptografickým podepsáním hlavičky a těla pomocí tajného klíče nebo páru veřejného a soukromého klíče. [17]

Všechna tato data jsou zakódována ve formátu Base64Url, což umožňuje jejich snadné přenášení přes HTTP. [17]

Jak JWT funguje v praxi:

1. Aplikační server vytvoří token, do kterého vloží zvolené informace.
2. Token je podepsán definovaným klíčem.
3. Výsledný token je odeslán klientovi, který ho ukládá (např. do localStorage).

4. Při každém požadavku klient připojí token k hlavičce HTTP (např. Authorization: Bearer ...).
5. Server ověří platnost podpisu a pokud je vše v pořádku, pokračuje v obsluze požadavku. [17]

Díky tomuto mechanismu je možné vytvářet škálovatelné aplikace bez potřeby uchovávat relace, protože token nese vše potřebné pro identifikaci uživatele. [17]

Výhody použití JWT:

- úspora místa: díky tomu, že JWT využívá datový formát JSON, jsou výsledné tokeny menší než například SAML, který používá XML. Kompaktnost je výhodná zejména při přenosu přes síť nebo v prostředí webových prohlížečů,
- vysoká úroveň zabezpečení: JWT podporuje podepisování pomocí sdíleného tajného klíče (HMAC) i asymetrických klíčů, například prostřednictvím X.509 certifikátů. Na rozdíl od XML podpisů, které mohou být náchylné ke složitým chybám, je digitální podpis v JSON jednodušší a méně zranitelný,
- široká podpora: formát JSON je přirozeně podporován ve většině programovacích jazyků, protože odpovídá interním datovým strukturám. Oproti tomu zpracování XML často vyžaduje složitější mapování, což činí práci s JWT přímější,
- rychlé zpracování: JWT je navrženo pro provoz ve velkém měřítku a je vhodné i pro zařízení s omezeným výkonem, jako jsou mobilní telefony. Jeho jednoduchá struktura usnadňuje validaci a zpracování na straně klienta i serveru. [18]

3.7 Testování

Testování softwaru je systematický proces, jehož cílem je ověřit, zda výsledný produkt odpovídá zadaným funkčním i nefunkčním požadavkům, a zároveň minimalizovat množství chyb, které by mohly negativně ovlivnit jeho použití v praxi. Provádí se jak manuálně, tak s využitím automatizovaných nástrojů a zahrnuje spouštění jednotlivých komponent i celých systémů za účelem identifikace odchylek, chyb nebo nedostatečného chování vůči specifikacím. [19]

3.7.1 Apollo Sandbox – interaktivní prostředí pro GraphQL

Apollo Sandbox je webový nástroj, který slouží jako interaktivní prostředí pro testování, vývoj a dokumentaci GraphQL API. Tento nástroj je součástí platformy Apollo a nabízí

vývojářům pohodlný způsob, jak zkoumat schéma, psát dotazy a ověřovat funkcionality API bez nutnosti externího prostředí. [20]

Hlavní funkce:

- editor dotazů: pokročilé rozhraní pro psaní GraphQL dotazů a mutací s podporou automatického doplňování, zvýraznění syntaxe a validace,
- prohlížeč schématu: grafické zobrazení typů, polí a jejich vztahů napříč API – usnadňuje orientaci v komplexních strukturách,
- mocking: možnost generování falešných dat podle definovaného schématu, což umožňuje testování bez napojení na reálný backend,
- spolupráce v týmu: sdílení sandboxu s kolegy, společná úprava dotazů a okamžité sledování změn,
- integrovaná dokumentace: možnost přidávat poznámky a popisky ke schématu i k odpovědím, což usnadňuje sdílení znalostí o API. [20]

Apollo Sandbox funguje jako vývojový a testovací nástroj, ale zároveň jako samodokumentační prostředí, které zlepšuje komunikaci mezi členy týmu a zvyšuje přehlednost API. [20]

3.7.2 Lighthouse

Microsoft 365 Lighthouse je nástroj navržený pro poskytovatele spravovaných služeb (MSP), kteří spravují více zákaznických tenantů současně. Umožňuje sjednocený dohled nad prostředím malých a středních firem, zjednodušuje správu zabezpečení a automatizuje běžné úlohy – vše z jednoho přehledného rozhraní. [21]

Díky tomu mohou IT týmy efektivněji řešit rizika, zajišťovat konzistenci konfigurace napříč organizacemi a škálovat své služby bez zvýšení provozní zátěže. Lighthouse je plně integrován s Microsoft 365 a dostupný bez dalších nákladů pro partnery v programu Cloud Solution Provider (CSP). [21]

Výhody:

- zabezpečení ve velkém měřítku: implementace doporučených bezpečnostních politik a nástrojů jako MFA či SSPR pro více tenantů najednou,

- standardizované nasazení: konzistentní postupy pro onboarding a konfiguraci zákaznických prostředí podle osvědčených scénářů pro SMB,
- přehled a kontrola: monitoring souladu s bezpečnostními zásadami, porovnání konfigurací napříč tenanty a snadná identifikace odchylek,
- doporučení na bázi AI: inteligentní přehledy a návrhy pomáhají zvyšovat spokojenost zákazníků, rozvíjet obchodní vztahy a předcházet problémům,
- efektivní správa: zrychlené řešení rutinních úloh (reset hesla, kontrola rizikových přihlášení, zpracování incidentů) bez nutnosti ručního zásahu v každém tenantovi.

[21]

II PRAKTICKÁ ČÁST

4 NÁVRH

V dnešní době je pro správu rozsáhlejších sítí nezbytné mít přehled o aktuálním stavu zařízení a jejich dostupnosti. Mým cílem bylo vytvořit efektivní systém pro správu síťové infrastruktury, který umožní administrátorům snadno evidovat zařízení, sledovat jejich stav online/offline a zároveň poskytne přehledné uživatelské rozhraní pro každodenní práci.

Systém umožňuje vizualizaci topologie sítě, včetně hierarchie zařízení a jejich propojení. Dále je možné zařízení vyhledávat, filtrovat a třídit podle různých parametrů jako je typ, název, MAC adresa nebo IP adresa. Každé zařízení má navíc historii online stavu za posledních 24 hodin, která je přehledně zobrazena pomocí minigrafu.

Součástí systému je i rozhraní pro správu uživatelů a rolí, díky čemuž lze snadno rozlišovat běžné uživatele od administrátorů. Administrátoři mají možnost zařízení upravovat, probouzet přes Wake-on-LAN, a měnit jejich základní parametry.

Hlavním cílem je navrhnout systém, který je přehledný, jednoduchý na ovládání a zároveň dostatečně výkonný, aby pokryl běžné potřeby síťových administrátorů při každodenní práci.

4.1 Funkční požadavky

4.1.1 Evidence zařízení

- Aplikace bude schopna evidovat síťová zařízení (např. switche, routery, firewally, servery).
- Uživatel bude moci přidat nové zařízení, upravit jeho údaje nebo ho odstranit ze systému.

4.1.2 Zobrazení online/offline stavu zařízení

- Aplikace bude u každého zařízení zobrazovat, zda je aktuálně online či offline.
- Stav bude pravidelně aktualizován a ukládán do historie.

4.1.3 Historie dostupnosti zařízení

- Aplikace bude ukládat historii online stavů zařízení po dobu minimálně 24 hodin.
- Uživatel bude mít možnost zobrazit přehled v podobě minigrafu přímo u zařízení.

4.1.4 Vyhledávání

- Aplikace umožní uživateli vyhledávat zařízení podle názvu, typu, MAC adresy, IP adresy nebo vlastníka.

4.1.5 Vizualizace sítě

- Aplikace nabídne grafickou síťovou mapu, ve které budou znázorněna zařízení a jejich hierarchické propojení.
- Mapa umožní přiblížení, oddálení a posun.

4.1.6 Wake-on-LAN

- Aplikace umožní uživatelům odeslat magický paket pro probuzení zařízení pomocí funkce Wake-on-LAN.

4.1.7 Správa uživatelů

- Aplikace nabídne rozhraní pro správu uživatelů, včetně jejich přidávání, mazání a změny hesel.
- Správu bude moci provádět pouze uživatel s rolí administrátora.

4.1.8 Role a oprávnění

- Aplikace bude rozlišovat alespoň dvě role: běžný uživatel a administrátor.
- Administrátor bude mít vyšší oprávnění, včetně správy zařízení a uživatelů.

4.1.9 Přehled statistik

- Aplikace bude poskytovat souhrnné statistiky o stavu zařízení – celkový počet, kolik je online/offline, a přehled podle typů zařízení.
- Tyto statistiky budou vizualizovány formou grafů.

4.1.10 Uživatelské rozhraní

- Aplikace nabídne přehledné, moderní a responzivní uživatelské rozhraní s podporou světlého i tmavého režimu.
- Při chybě aplikace zobrazí uživateli jasné a srozumitelné chybové hlášení.

4.2 Nefunkční požadavky

4.2.1 Dostupnost systému

- Aplikace musí být dostupná minimálně 99,9 % času během běžného provozu.

4.2.2 Výkon

- Aplikace musí být schopna obsloužit alespoň 1000 zařízení bez znatelného zpoždění při načítání nebo interakci.

4.2.3 Odezva

- Odezva systému na běžné uživatelské akce (např. přidání zařízení, zobrazení seznamu) nesmí překročit 1 sekundu.

4.2.4 Zabezpečení dat

- Hesla uživatelů budou ukládána ve formě hashů (např. bcrypt).
- Databáze bude zabezpečena heslem

4.2.5 Oprávnění

- Každá akce v systému bude ověřena podle role přihlášeného uživatele. Běžní uživatelé nebudou mít přístup ke správě ostatních uživatelů.

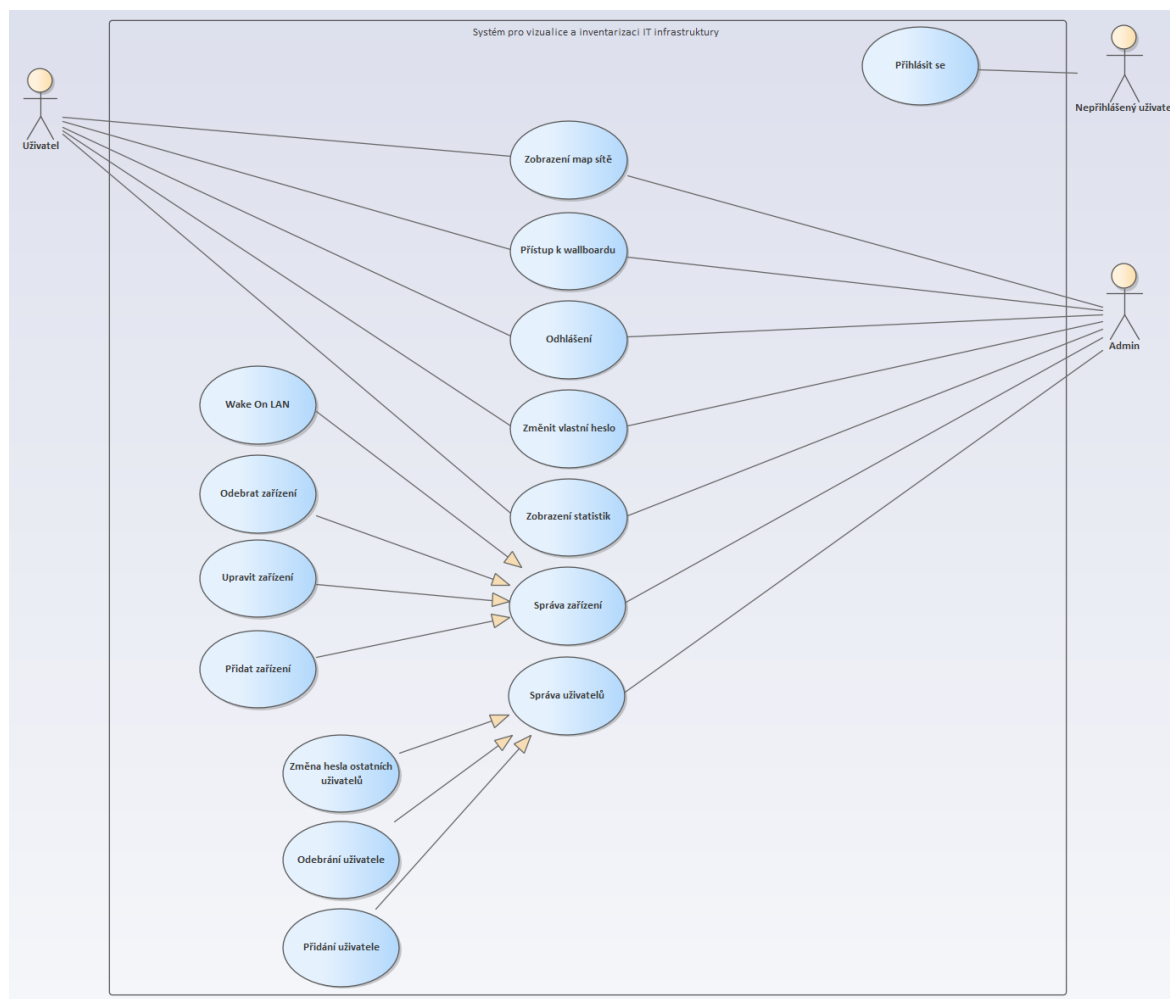
4.2.6 Kompatibilita

- Aplikace bude kompatibilní s moderními webovými prohlížeči (Chrome, Firefox, Edge) a optimalizovaná i pro zobrazení na tabletech.

4.2.7 Uživatelská přívětivost

- Rozhraní musí být intuitivní a srozumitelné i pro méně technicky zdatné uživatele.
- Aplikace nabídne responzivní design a přehledné zobrazení i na menších obrazovkách.

4.3 Případy užití



Obrázek 1 - Use case model

(zdroj: vlastní zpracování v programu Enterprise Architect)

Diagram případů užití na předchozím obrázku ukazuje 3 aktéry. Uživatele, Admina a Nepřihlášeného uživatele. Nepřihlášený uživatel se může pouze přihlásit. Uživatel má už více případů užití. Může si zobrazit mapu sítě, zobrazit statistiky, změnit své heslo, zobrazovat wallboard a odhlásit se. Admin může to, co běžný uživatel, ale navíc má přístup ke správě zařízení a správě uživatelů, tedy může odesílat wake on lan pakety, odebrat, přidat a upravit zařízení, odebrat a přidat uživatele a měnit jim heslo.

4.4 Scénáře případů užití

Tabulka 1 - Scénář případů užití UC001

Název: Přihlášení		
ID: UC001		
Charakteristika:	Uživatel se přihlásí pomocí emailu a hesla	
Aktér:	Nepřihlášený uživatel	
Vstupní podmínky:	Nejsou	
Hlavní scénář:		
Krok	Aktér/systém	Popis
1	Aktér	Vyplní email a heslo
2	Aktér	Stiskne tlačítko přihlásit
3	Aktér	Je přihlášen a přesměrován na domovskou stránku
Alternativní scénáře:		
1	Aktér	Vyplní nesprávné údaje
2	Systém	Zobrazí chybovou hlášku

(zdroj: vlastní zpracování)

Tabulka 2 - Scénář případů užití UC002

Název: Změna vlastního hesla		
ID: UC002		
Charakteristika:		Uživatel si změni přihlašovací heslo
Aktér:		Admin/Uživatel
Vstupní podmínky:		Uživatel je přihlášen
Hlavní scénář:		
Krok	Aktér/systém	Popis
1	Aktér	Vyplní staré a nové heslo
2	Aktér	Stiskne tlačítko změnit
3	Systém	Změní heslo a odhlásí uživatele
Alternativní scénáře:		
1	Aktér	Vyplní nesprávné údaje
2	Systém	Zobrazí chybovou hlášku

(zdroj: vlastní zpracování)

Tabulka 3 - Scénář případů užití UC003

Název: Zobrazení statistik		
ID: UC003		
Charakteristika:		Uživatel zobrazí sekci statistiky
Aktér:		Admin/Uživatel
Vstupní podmínky:		Uživatel je přihlášen
Hlavní scénář:		
Krok	Aktér/systém	Popis
1	Aktér	Klikne na statistiky
2	System	Vykreslí statistiky
Alternativní scénáře:		

(zdroj: vlastní zpracování)

Tabulka 4- Scénář případů užití UC004

Název: Zobrazení Wallboardu		
ID: UC004		
Charakteristika:		Uživatel zobrazí Wallboard
Aktér:		Admin/Uživatel
Vstupní podmínky:		Uživatel je přihlášen
Hlavní scénář:		
Krok	Aktér/systém	Popis
1	Aktér	Napiše /wallboard za base url
2	System	Vykreslí Wallboard
Alternativní scénáře:		

(zdroj: vlastní zpracování)

Tabulka 5- Scénář případů užití UC005

Název: Zobrazení map sítě		
ID: UC005		
Charakteristika:		Uživatel zobrazí mapu sítě
Aktér:		Admin/Uživatel
Vstupní podmínky:		Uživatel je přihlášen
Hlavní scénář:		
Krok	Aktér/systém	Popis
1	Aktér	Klikne na záložku vizualizace
2	Systém	Vykreslí mapu
Alternativní scénáře:		

(zdroj: vlastní zpracování)

Tabulka 6- Scénář případů užití UC006

Název: Odhlášení		
ID: UC006		
Charakteristika:		Uživatel se odhlásí
Aktér:		Admin/Uživatel
Vstupní podmínky:		Uživatel je přihlášen
Hlavní scénář:		
Krok	Aktér/systém	Popis
1	Aktér	Klikne na odhlásit se
2	Systém	Odhlásí uživatele a přesměruje na přihlášení
Alternativní scénáře:		

(zdroj: vlastní zpracování)

Tabulka 7- Scénář případů užití UC007

Název: Wake On Lan		
ID: UC007		
Charakteristika:	Administrátor odešle magic paketu pro probuzení uživatele	
Aktér:	Admin	
Vstupní podmínky:	Uživatel je přihlášen a má roli administrátora	
	Existuje alespoň jedno zařízení v databázi	
Hlavní scénář:		
Krok	Aktér/systém	Popis
1	Aktér	Klikne na tlačítko se symbolem zapnutí u požadovaného zařízení
2	Systém	Proběhne kontrola MAC adresy a pošle se paketa
3	Systém	Potvrdí odeslání pakety
Alternativní scénáře:		
1	Aktér	Vybere zařízení s neplatnou MAC adresou
2	Systém	Zobrazí chybovou hlášku

(zdroj: vlastní zpracování)

Tabulka 8- Scénář případů užití UC008

Název: Odebrat zařízení		
ID: UC008		
Charakteristika:	Administrátor odebere zařízení	
Aktér:	Admin	
Vstupní podmínky:	Uživatel je přihlášen a má roli administrátora	
	Existuje alespoň jedno zařízení v databázi	
Hlavní scénář:		
Krok	Aktér/systém	Popis
1	Aktér	Klikne na tlačítko se symbolem koše u požadovaného zařízení
2	Systém	Zařízení se vymaže z databáze a aktualizuje se seznam
Alternativní scénáře:		

(zdroj: vlastní zpracování)

Tabulka 9- Scénář případů užití UC009

Název: Upravit zařízení		
ID: UC009		
Charakteristika:		Administrátor upraví zařízení
Aktér:		Admin
Vstupní podmínky:	Uživatel je přihlášen a má roli administrátora	
	Existuje alespoň jedno zařízení v databázi	
Hlavní scénář:		
Krok	Aktér/systém	Popis
1	Aktér	Klikne na tlačítko se symbolem tužky u požadovaného zařízení
2	Aktér	Uživatel přepíše požadované pole a klikne na Uložit Změny
3	System	System zpracuje změny a aktualizuje seznam
Alternativní scénáře:		

(zdroj: vlastní zpracování)

Tabulka 10- Scénář případů užití UC010

Název: Přidat zařízení		
ID: UC010		
Charakteristika:		Administrátor přidá zařízení
Aktér:		Admin
Vstupní podmínky:		Uživatel je přihlášen a má roli administrátora
Hlavní scénář:		
Krok	Aktér/systém	Popis
1	Aktér	Klikne na tlačítko přidat zařízení
2	Aktér	Uživatel vyplíše požadované pole a klikne na Přidat
3	System	System zpracuje změny a aktualizuje seznam
Alternativní scénáře:		
1	Aktér	Uživatel zadá nejedinečné pole
2	System	System zařízení nepřidá

(zdroj: vlastní zpracování)

Tabulka 11 - Scénář případů užití UC011

Název: Změna hesla jiného uživatele		
ID: UC011		
Charakteristika:		Administrátor změní jinému uživateli heslo
Aktér:		Admin
Vstupní podmínky:		Uživatel je přihlášen a má roli administrátora
		Existuje nějaký jiný uživatel
Hlavní scénář:		
Krok	Aktér/systém	Popis
1	Aktér	Klikne na tlačítko změnit heslo u daného uživatele
2	Aktér	Uživatel vyplní nové heslo a potvrdí tlačítkem změnit
3	Systém	Systém zpracuje změny
Alternativní scénáře:		

(zdroj: vlastní zpracování)

Tabulka 12- Scénář případů užití UC012

Název: Odebrání uživatele		
ID: UC012		
Charakteristika:	Administrátor odebere uživatele	
Aktér:	Admin	
Vstupní podmínky:	Uživatel je přihlášen a má roli administrátora	
	Existuje nějaký jiný uživatel	
Hlavní scénář:		
Krok	Aktér/systém	Popis
1	Aktér	Klikne na tlačítko smazat u daného uživatele
2	Aktér	Uživatel potvrdí stiskem tlačítka OK smazání.
3	Systém	Systém zpracuje změny a aktualizuje seznam.
Alternativní scénáře:		

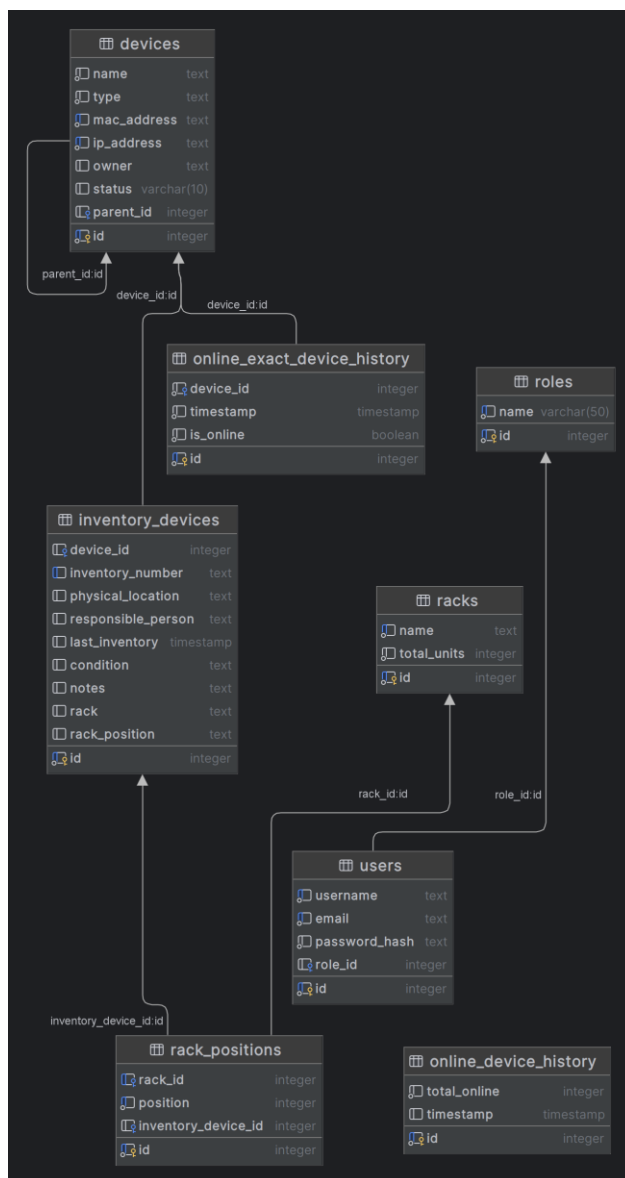
(zdroj: vlastní zpracování)

Tabulka 13- Scénář případů užití UC013

Název: Přidání uživatele		
ID: UC013		
Charakteristika:	Administrátor přidá uživatele	
Aktér:	Admin	
Vstupní podmínky:	Uživatel je přihlášen a má roli administrátora	
Hlavní scénář:		
Krok	Aktér/systém	Popis
1	Aktér	Klikne na tlačítko přidat uživatele
2	Aktér	Uživatel vyplní pole a potvrdí stiskem tlačítka přidat.
3	Systém	Systém zpracuje změny a aktualizuje seznam.
Alternativní scénáře:		

(zdroj: vlastní zpracování)

4.5 Databázový model



Obrázek 2 - Databázový model

(zdroj: vygenerováno v aplikaci WebStorm)

V databázovém modelu se nachází 8 hlavních tabulek, které vzájemně spolupracují pro správu a vizualizaci IT infrastruktury:

- Tabulka **users** slouží k ukládání informací o uživateli systému – obsahuje uživatelské jméno, e-mail, zahashované heslo a roli (napojenou přes **role_id** na tabulku **roles**).
- Tabulka **roles** definuje možné role uživatelů (např. administrátor nebo běžný uživatel) a je propojena s tabulkou **users** pomocí primárního klíče **id**.

- Tabulka `devices` uchovává detailní informace o síťových zařízeních, včetně názvu, typu, MAC a IP adresy, vlastníka, stavu (online nebo offline) a případného nadřazeného zařízení (`parent_id`, který umožňuje hierarchické vztahy mezi zařízeními).
- Tabulka `online_device_history` ukládá historické informace o celkovém počtu online zařízení v daný čas. Slouží ke sledování agregovaných statistik.
- Tabulka `online_exact_device_history` eviduje historii online stavů jednotlivých zařízení. Pomocí `device_id` je napojena na tabulku `devices` a umožňuje detailní časové sledování jejich dostupnosti (`is_online`).
- Tabulka `inventory devices` slouží k inventurní funkci aplikace, evidují se zde zařízení a informace k inventuře, je zvlášť, aby mohla sledovat i zařízení které se v síti už nenachází, ale je potřeba je inventarizovat.
- Tabulka `racks` obsahuje názvy a kapacity racků.
- Tabulka `rack_positions` slouží k evidování obsazenosti různých racků. Funkcionalita pak nedovolí přidat zařízení tam, kde je už jiné.

5 NÁVRH SYSTÉMU

V této části bakalářské práce je popsán postup při implementaci návrhu aplikace.

5.1 Použité technologie

Pro vývoj této aplikace byla zvolena moderní open-source knihovna React, která umožňuje efektivní vývoj interaktivních webových rozhraní. Díky své komponentové architektuře nabízí přehlednou strukturu kódu a snadné opakované použití prvků v rámci celého projektu.

Jako základní návrhový framework pro rozhraní bylo využito Material UI, které poskytuje sadu profesionálně vypadajících a responzivních komponent v souladu s principy Material Designu. Díky tomu je uživatelské rozhraní přehledné, konzistentní a snadno rozšiřitelné.

Pro práci s daty bylo použito GraphQL ve spojení s Apollo Clientem, což umožňuje efektivní získávání jen těch dat, která komponenty skutečně potřebují. Backend aplikace je postaven v Node.js s databází PostgreSQL a běží v prostředí Docker, což usnadňuje jak vývoj, tak i nasazení do produkce.

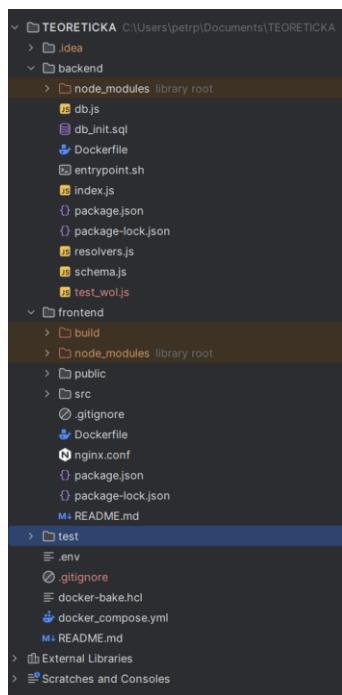
Aby bylo možné vizuálně zobrazit statistiky a historii dostupnosti zařízení, byla použita knihovna react-chartjs-2, která umožňuje snadno tvořit interaktivní grafy na základě knihovny Chart.js.

Celý projekt je navržen tak, aby byl snadno rozšiřitelný, bezpečný a dobře použitelný nejen pro správce sítě, ale i pro běžné uživatele s omezenými oprávněními.

5.2 Struktura projektu

Základem kvalitního softwarového řešení je dobře navržená a přehledná adresářová struktura, která usnadňuje nejen samotný vývoj, ale i budoucí údržbu a rozšiřování projektu.

Základním rozdělením je rozdělení složek na Backend a Frontend. Ve složce backend se nachází soubory, které slouží k nastavení a inicializaci databáze, a soubory sloužící ke GraphQL. Frontend složka zase obsahuje hlavně složku src ve které se nachází samotné komponenty a ikony. Některé ikony se nachází i ve složce public.



Obrázek 3 - Struktura Projektu

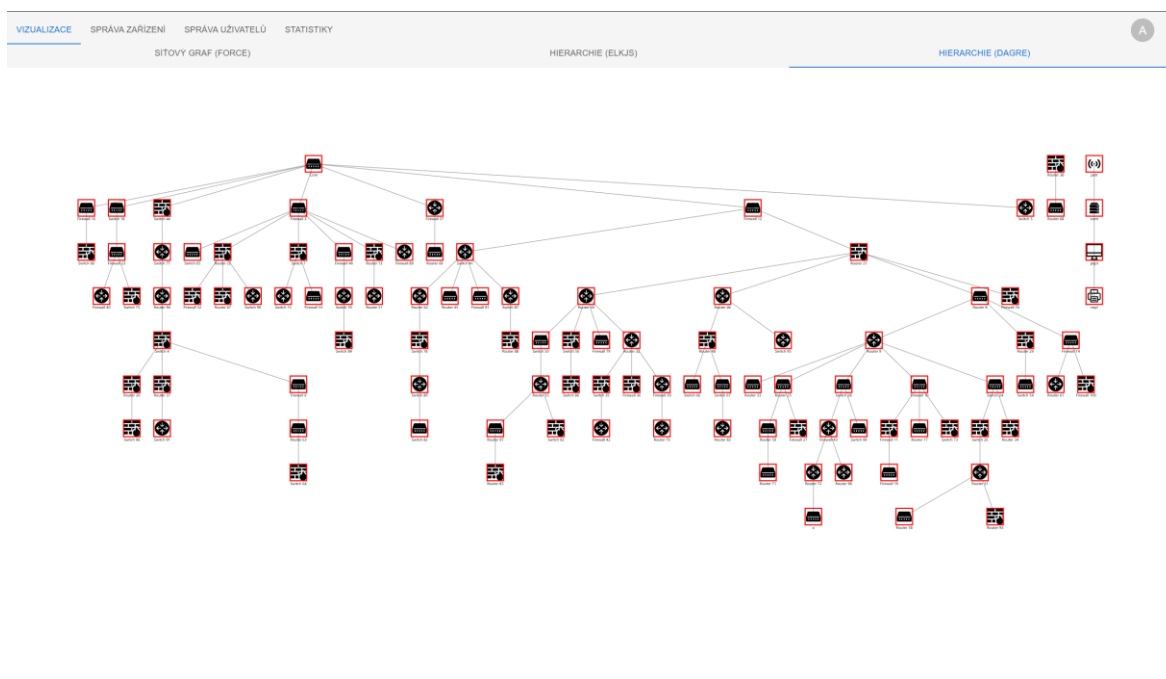
(zdroj: struktura projektu v IDE WebStorm)

5.3 Návrh uživatelského rozhraní

Uživatelské rozhraní tohoto systému bylo navrženo tak, aby bylo přehledné a intuitivní. Na horní části obrazovky se nachází záložky, pomocí kterých lze přepínat jednotlivé součásti aplikace, a také ikonka uživatele s rozbalovacím menu pro odhlášení a změnu hesla. Součástí aplikace je spousta piktogramů a emotikonů pro lepší orientaci a vzhled.

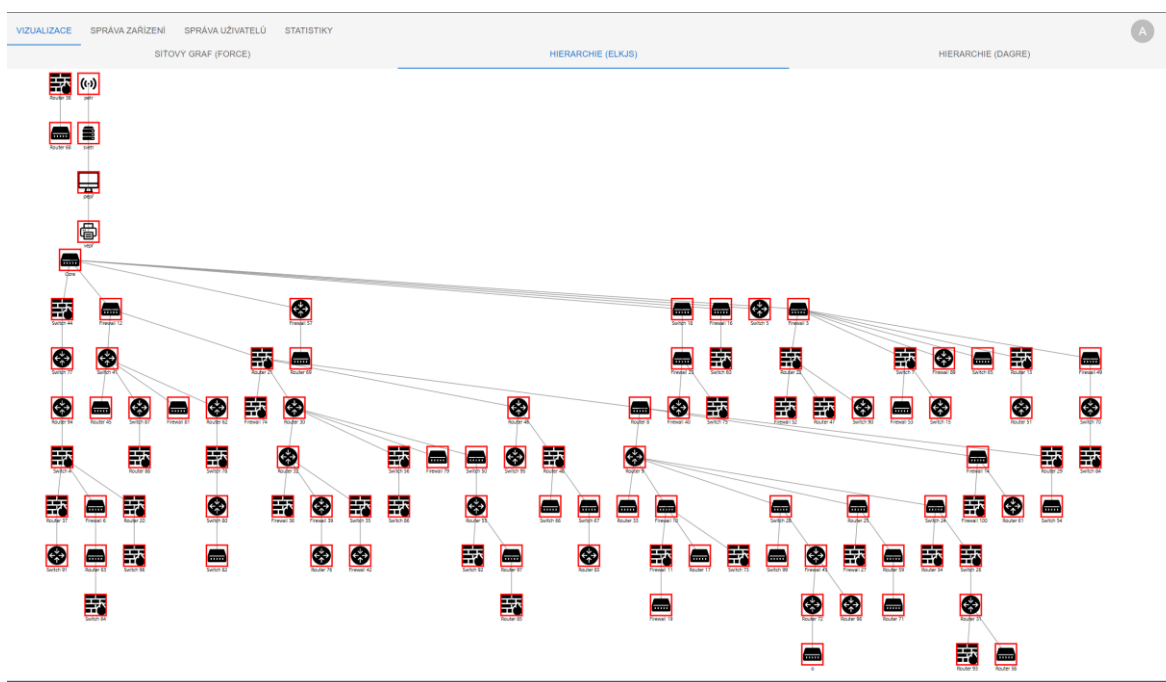
5.3.1 Záložka Vizualizace

V záložce vizualizace se nachází samotná mapa zařízení. Zařízení jsou propojené čarami kolem zařízení se vykresluje obrys čtverce, který mění barvu podle toho, zdali je zařízení online (zelená) nebo off-line (červená). Uživatel má také možnost volby mezi třemi možnostmi vizualizace. A to force-directed layout, Eclipse Layout Kernel a dagre. Každý z těchto grafů má své výhody i nevýhody. Jako nejpřehlednější se jeví dagre, který je rozvržen nejvíc přehledně ze všech tří map. Nejzábavněji pak ale vypadá force-directed mapa, která nabízí fyzikální simulaci. Jednotlivé zařízení v ní je možné táhnout a pustit a tím se celá mapa pohybuje a přeorganizuje, nenabízí tak stále stejnou strukturu jako ostatní modely, ale při každém načtení je mírně odlišná. Bohužel díky této simulaci se někdy linie spojení překrývají a mapa se jeví velice nepřehledná.



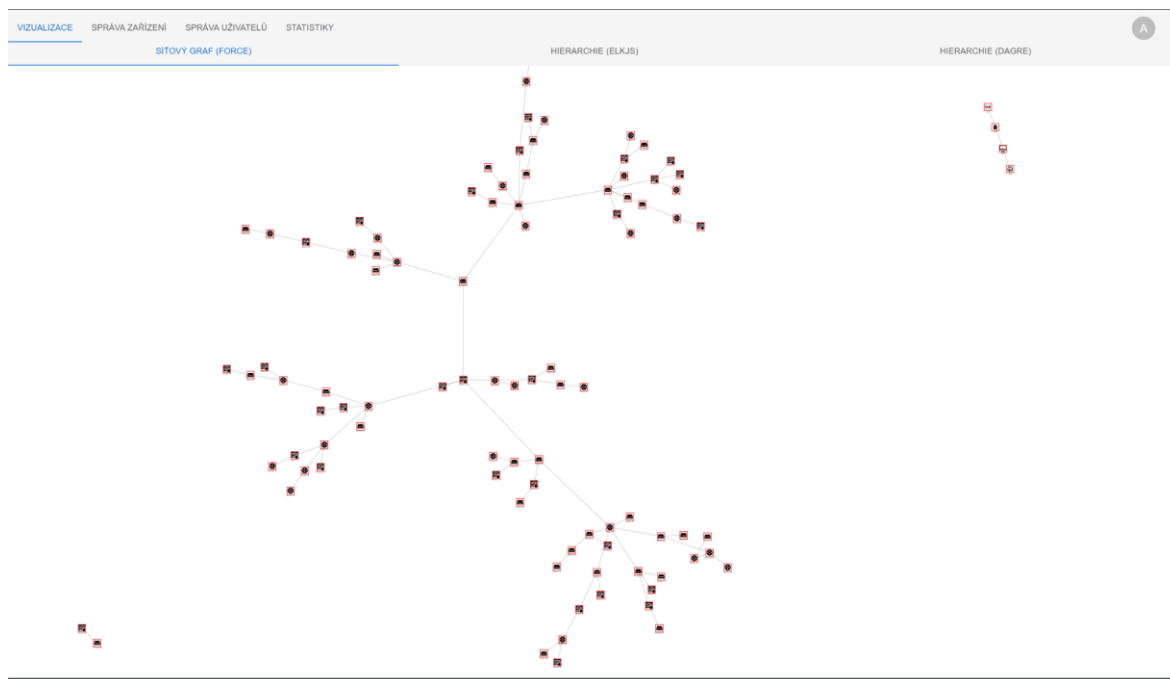
Obrázek 4 - Vizualizace – dagre

(zdroj: ukázka aplikace vlastní tvorby)



Obrázek 5 - Vizualizace – ELKJS

(zdroj: ukázka aplikace vlastní tvorby)



Obrázek 6 - Vizualizace – FORCE

(zdroj: ukázka aplikace vlastní tvorby)

Všechny tyto mapy umožňují funkce jako přiblížení a posun, force navíc umožňuje i posun a tah jednotlivých zařízení, u ostatních map jsou zařízení vykreslené pevně.

5.3.2 Správa zařízení

V záložce správa zařízení se nachází výpis všech zařízení. V této záložce je také možnost přidat zařízení, odstranit zařízení, upravit zařízení a vyhledat zařízení. Dále je možné zařízení probudit zasláním příkazu WOL a nabízí se zde i možnost filtrace záznamů podle kteréhokoli sloupce a export dat ve formátu CSV.

VIZUALIZACE
SPRÁVA ZAŘÍZENÍ
SPRÁVA UŽIVATELŮ
STATISTIKY

Správa zařízení

+ PŘIDAT ZAŘÍZENÍ
EXPORT CSV

Hledat zařízení

Název	Typ	MAC	IP	Vlastník	Rodič	Stav	Akce
Router 31	Router	00:DC:BD:2C:40:C4	192.168.0.7	Networking Team	Switch 26	offline	  
Firewall 23	Switch	00:DA:FA:47:DB:52	192.168.5.220	Networking Team	Switch 18	offline	  
Router 25	Switch	00:95:DB:A2:65:28	192.168.3.30	Security Team	Router 9	offline	  
Firewall 6	Switch	00:0C:D1:EF:5C:15	192.168.5.193	Networking Team	Switch 4	offline	  
Firewall 14	Switch	00:E3:0E:91:9C:DC	192.168.5.151	IT Department	Router 8	offline	  
Router 30	Router	00:94:82:49:FA:ED	192.168.4.166	Security Team	Router 21	offline	  
Router 20	Firewall	00:80:CF:FA:85:6C	192.168.1.33	Security Team	Switch 4	offline	  
Switch 67	Switch	00:E5:A7:C9:26:36	192.168.5.29	Networking Team	Router 48	offline	  
Switch 65	Switch	00:11:A4:EF:55:76	192.168.3.79	IT Department	Firewall 3	offline	  
Firewall 3	Switch	00:3C:F6:DB:26:5B	192.168.4.6	Security Team	Core	offline	  
Router 46	Router	00:9E:E1:CC:B6:F9	192.168.9.7	Security Team	Router 21	offline	  

Obrázek 7 - Správa zařízení

(zdroj: ukázka aplikace vlastní tvorby)

VIZUALIZACE
SPRÁVA ZAŘÍZENÍ
SPRÁVA UŽIVATELŮ
STATISTIKY

Správa zařízení

+ PŘIDAT ZAŘÍZENÍ
EXPORT CSV

Hledat zařízení

Název	Typ	MAC	IP	Vlastník	Rodič	Stav	Akce
Router 30	Router	00:94:82:49:FA:ED				offline	  
Router 58	Switch	00:9C:53:52:E0:71				offline	  
Firewall 6	Switch	00:0C:D1:EF:5C:15				offline	  
Firewall 40	Router	00:4F:1C:9E:AD:F3				offline	  
Router 31	Router	00:DC:BD:2C:40:C4				offline	  
Router 33	Switch	00:28:4F:57:26:9E				offline	  
Router 29	Firewall	00:87:25:6B:00:A4				offline	  
Switch 67	Switch	00:E5:A7:C9:26:36	192.168.5.29	Networking Team	Router 48	offline	  
Router 62	Router	00:41:EF:58:39:6E	192.168.6.216	IT Department	Switch 41	offline	  
Router 45	Switch	00:8C:11:DD:20:80	192.168.3.141	Security Team	Switch 41	offline	  
Core	Switch	00:07:0B:78:FF:05	192.168.10.69	IT Department	Žádný	offline	  

Přidat zařízení

Název

Typ

MAC Adresa

IP Adresa

Vlastník

Rodič

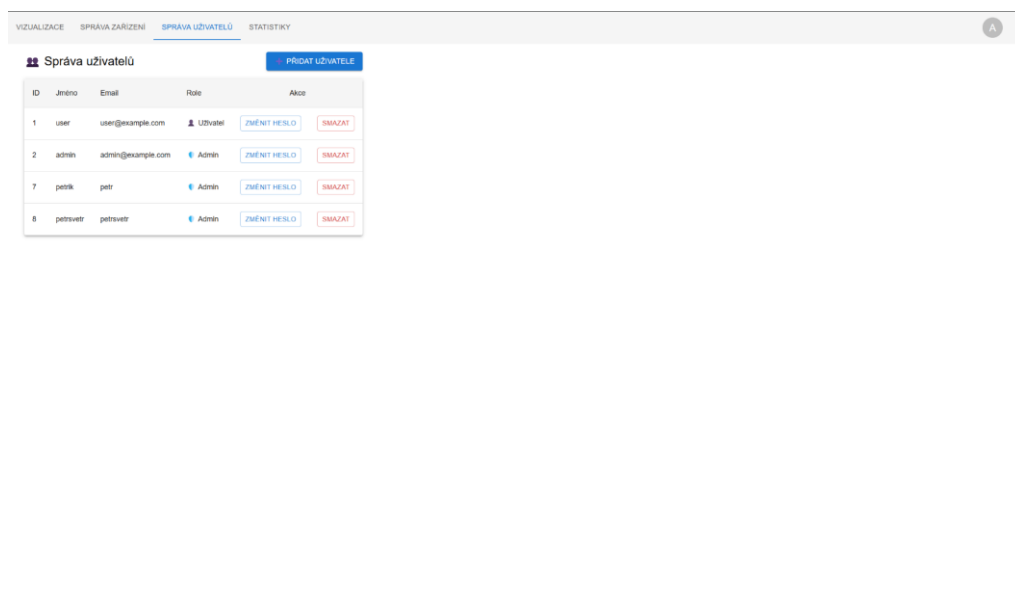
ZRUŠIT
PŘIDAT

Obrázek 8 - Modální okno přidat zařízení

(zdroj: ukázka aplikace vlastní tvorby)

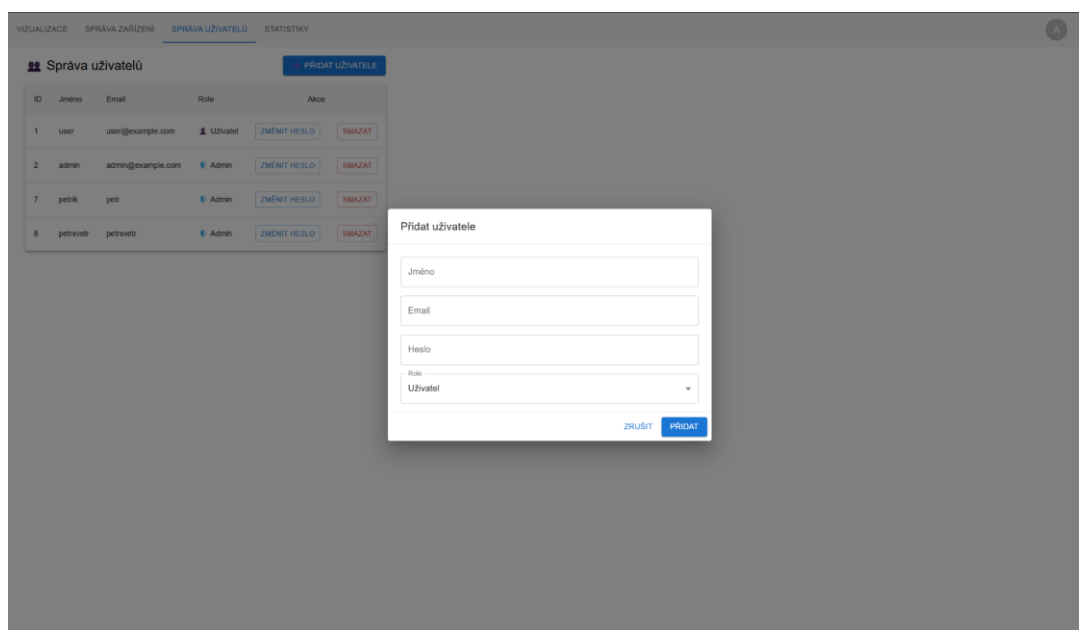
5.3.3 Správa uživatelů

Záložka správa uživatelů ukazuje výpis všech uživatelů v databázi. Nabízí možnost přidání a odebrání uživatele a také změnu hesla. U každého uživatele jsou také vypsány role doplněné emotikonu buďto štítu (admin) nebo lidí jakožto běžných uživatelů.



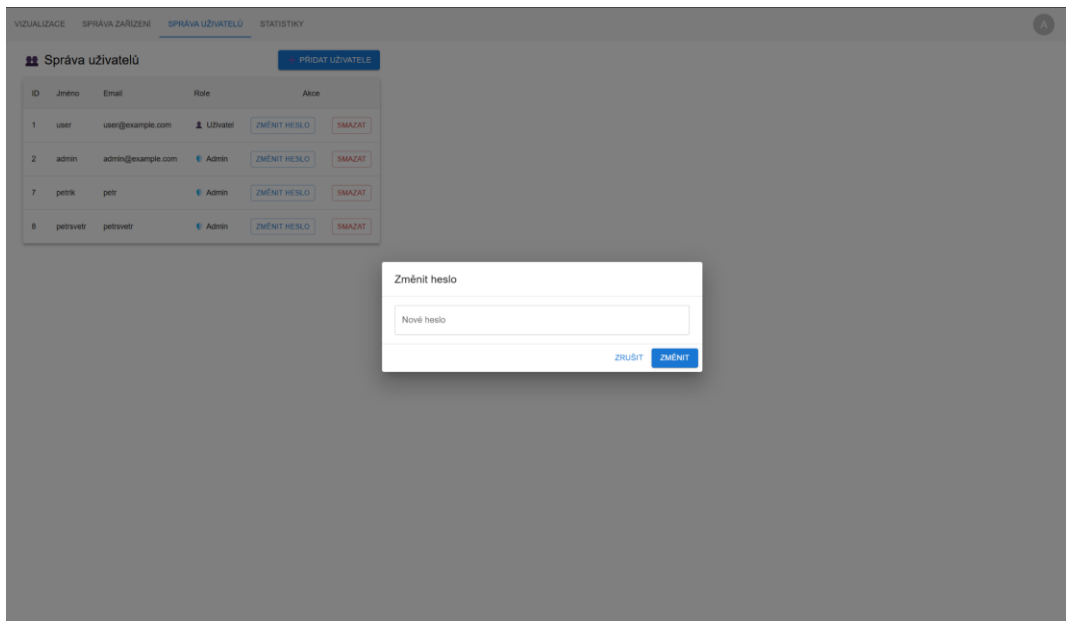
Obrázek 9 - Správa uživatelů

(zdroj: ukázka aplikace vlastní tvorby)



Obrázek 10 - Modální okno Přidat uživatele

(zdroj: ukázka aplikace vlastní tvorby)

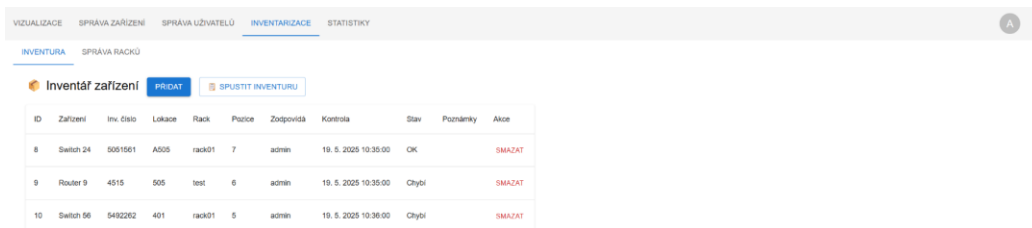


Obrázek 11 - Modální okno Změnit heslo

(zdroj: ukázka aplikace vlastní tvorby)

5.3.4 Inventarizace

Záložka inventarizace slouží k inventarizaci zařízení, nachází se zde seznam zařízení s informacemi pro inventarizaci, funkce pro přidání nového zařízení, která je napojená na tabulku devices a users, tak se propisují jednotlivé pole navzájem. Dále je tam také funkce pro inventuru, kde se dá postupně projít zařízení po zařízení a provést inventuru. Dále je zde záložka pro správu racků.



Obrázek 12 - Inventář zařízení

(zdroj: ukázka aplikace vlastní tvorby)

The screenshot shows the 'Přidat položku' (Add item) form in the inventory management application. The form is a modal window with the following fields:

- Zařízení (Device): Dropdown menu
- Inventární číslo (Inventory number): Text input
- Fyzická lokace (Physical location): Text input
- Rack: Dropdown menu
- Pozice v racku (Position in rack): Dropdown menu
- Zodpovědná osoba (Responsible person): Dropdown menu
- Kontrola (Check): Text input with a date/time value of 19.05.2025 10:37
- Stav (Status): Dropdown menu
- Poznámky (Notes): Text area

At the bottom of the form are two buttons: 'ZRUŠIT' (Cancel) and 'PŘIDAT' (Add).

In the background, a table titled 'Inventář zařízení' (Device inventory) is visible, showing columns for ID, Zařízení, Inv. číslo, Lokace, Rack, Pozice, Zodpovědná, and Kontrola. The table contains three rows of data:

ID	Zařízení	Inv. číslo	Lokace	Rack	Pozice	Zodpovědná	Kontrola
8	Switch 24	5051561	A505	rack01	7	admin	19. 5. 2025 10:37
9	Router 9	4515	505	test	6	admin	19. 5. 2025 10:37
10	Switch 56	5492262	401	rack01	5	admin	19. 5. 2025 10:37

Obrázek 13 - Přidání položky do inventáře

(zdroje: ukázka aplikace vlastní tvorby)

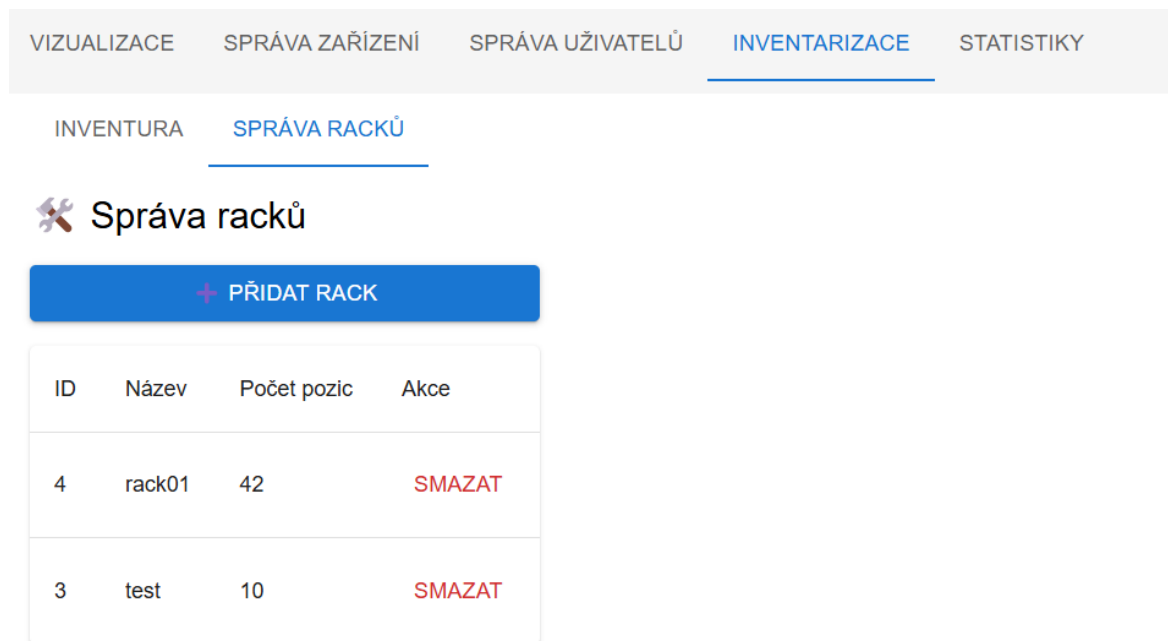
The screenshot shows the 'Provedení inventory' (Perform inventory) form in the inventory management application. The form is a modal window with the following fields:

- Hledat (název, osoba, inventární číslo...) (Search (name, person, inventory number...)): Search input
- Filtr podle stavu (Filter by status): Dropdown menu
- Filtr podle osoby (Filter by person): Dropdown menu
- 5051561 Switch 24: Item details with dropdowns for Stav (OK), Místnost (A505), Rack (rack01), and Pozice (7). Below are fields for Zodpovědná (osoba) (admin) and Poznámky.
- 4515 Router 9: Item details with dropdowns for Stav (Chybí), Místnost (505), Rack (test), and Pozice (6). Below are fields for Zodpovědná (osoba) (admin) and Poznámky.
- 5492262 Switch 56: Item details with dropdowns for Stav (Chybí), Místnost (401), Rack (rack01), and Pozice (5). Below are fields for Zodpovědná (osoba) (admin) and Poznámky.

At the bottom of the form are two buttons: 'ZRUŠIT' (Cancel) and 'ULOŽIT ZMĚNY' (Save changes).

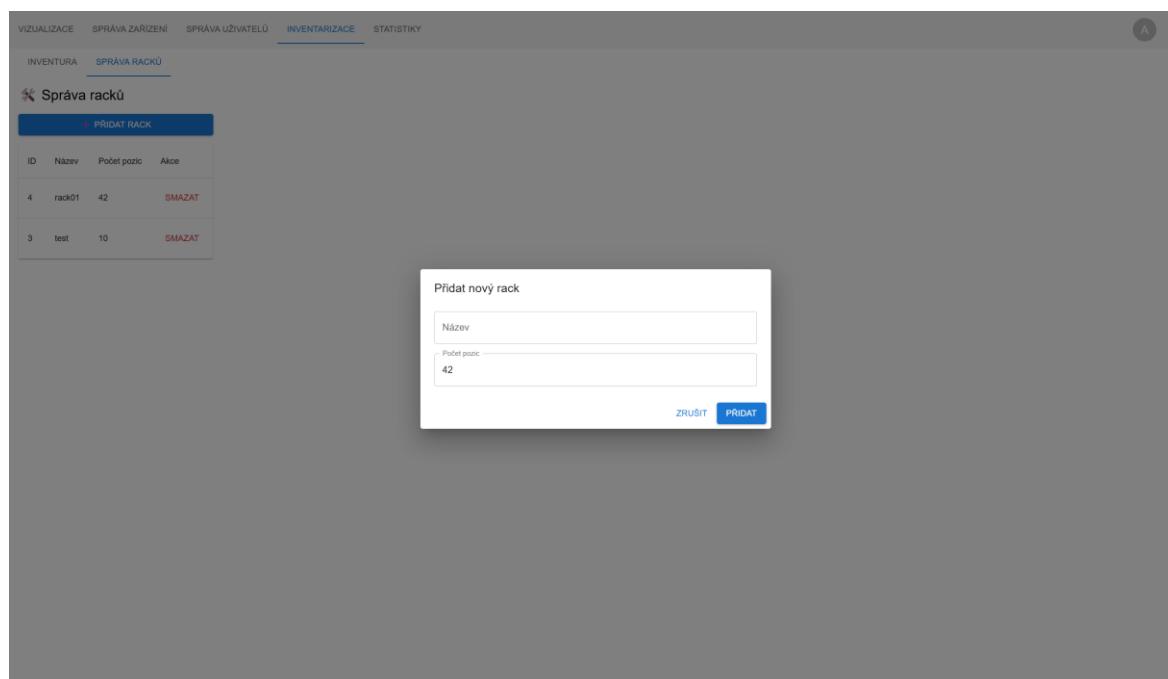
Obrázek 14 - Provedení inventory

(zdroje: ukázka aplikace vlastní tvorby)



Obrázek 15 - Správa racků

(zdroje: ukázka aplikace vlastní tvorby)



Obrázek 16 - Přidat nový rack

(zdroje: ukázka aplikace vlastní tvorby)

5.3.5 Statistiky

V záložce statistiky nalezneme grafy a statistky ohledně stavu zařízení. Najdeme zde historii online zařízení, celkový počet zařízení nebo zařízení v kategoriích.

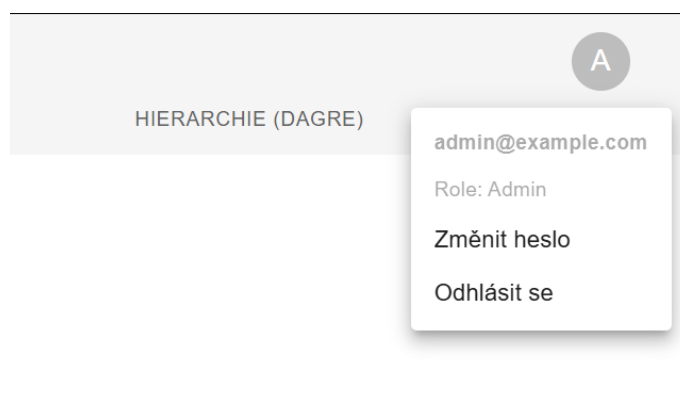


Obrázek 17 – Statistiky

(zdroj: ukázka aplikace vlastní tvorby)

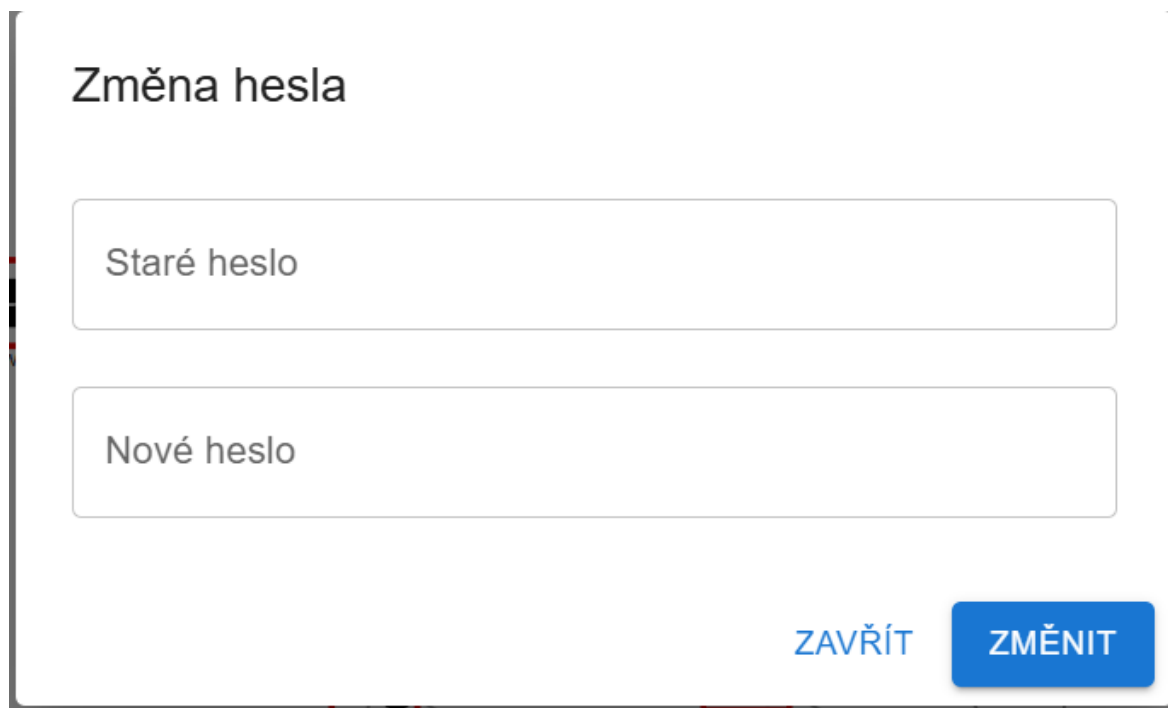
5.3.6 Uživatelské menu

V aplikaci se nachází v rohu tlačítko s iniciálou přihlášeného uživatele, které po kliknutí vyvolá okno s informací o uživateli, roli a také nabídne možnost odhlášení se nebo tlačítko pro změnu hesla.



Obrázek 18 – Uživatelské menu s iniciálou

(zdroj: ukázka aplikace vlastní tvorby)

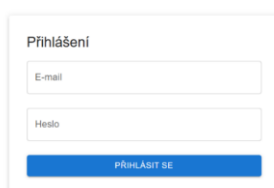


Obrázek 19 – Dialog pro změnu hesla

(zdroj: ukázka aplikace vlastní tvorby)

5.3.7 Přihlášení

Obrazovka pro přihlášení je jednoduchá, obsahuje pouze pole pro identifikaci a tlačítko pro potvrzení.



Obrázek 20 - Přihlašovací obrazovka

(zdroj: ukázka aplikace vlastní tvorby)

Tokán má i čas platnosti, a dokonce se při každém přihlášení mění. To je výhodné třeba oproti přístupu ověřování přes API klíč, který je často trvalý nebo má platnost v rámci dní, měsíců nebo dokonce roků.

- Hashování hesel – Ukládaná hesla nejsou v databázi viditelná, nachází se tam pouze hash hesla pomocí nástroje bcrypt, který velmi těžké prolomit, pokud uživatel používá správně vybrané heslo.
- Role-based resolvery – Jedná se hlavně o zabezpečení komponent, kde se vyžaduje mít roli administrátora. Pokud uživatel není administrátor nemůže měnit data.
- Bezpečné mutace změny hesla – Aby uživatel mohl změnit své heslo, musí znát to předchozí, to je vhodné například pokud uživatel zůstane někde přihlášený účtu se nemůže zmocnit jiná osoba na delší dobu.
- Dotazování přes předpřipravené dotazy – V aplikaci se používají parametrizované SQL dotazy. Díky tomuto přístupu je aplikace imunní proti SQL injection

5.4.2 Frontend.

Frontend je velmi kritická část aplikace, jelikož s ní do přímého styku přijdou veškerí uživatelé systému. Proto bylo nutné zavést opatření, které zabraňují jak neúmyslnému, tak úmyslnému proniknutí k neoprávněným datům.

- JWT Token – Tento token slouží k přihlašování a ověřování uživatelů. Při odhlášení se deaktivuje a smaže. Po novém přihlášení se vygeneruje nový. Dále se také posílá pro všechny požadavky vůči backendu.
- Role-based přístup – Uživatelům je dovoleno přístup jen do některých nekritických částí aplikace. Pokud se přihlásí administrátor objeví se jinak skryté nabídky.
- Odhlášení po změně hesla – po změně hesla je uživatel odhlášen, aby znova ověřil svou identitu.

5.4.3 Databáze

Databáze je zabezpečena jak heslem, tak i vnitřně pomocí různých omezení chránících data, jejich duplicitu a chyby v databázovém modelu.

6 IMPLEMENTACE SYSTÉMU

6.1 Vývoj databázové vrstvy

Databáze funguje stejně jako ostatní části aplikace v Dockeru, to nám pomáhá při rychlém nasazování a díky kontejnerizaci také jistou izolaci od zbytku systému.

6.1.1 docker-compose.yml

```
1  version: '3.8'
2
3  services:
4    postgres:
5      image: postgres:17
6      container_name: postgres
7      restart: always
8      networks:
9        - app_network
10     environment:
11       POSTGRES_USER: postgres
12       POSTGRES_PASSWORD: password
13       POSTGRES_DB: inventory
14     ports:
15       - "5432:5432"
16     volumes:
17       - postgres_data:/var/lib/postgresql/data
```

Obrázek 22 - Docker Compose pro postgres

(zdroj: vlastní tvorba)

Výše uvedený obrázek nám ukazuje konfigurační docker-compose.yml soubor pro vytvoření kontejneru. V poli image je vidno, že používáme verzi 17. V poli environment pak že vytváříme uživatele postgres s heslem password jméno databáze je inventory. Níže v obrázku je informace, jaké porty se nám převádí. Výrok 5432:5432 znamená, že na port 5432 který je otevřený na zařízení na kterém tento kontejner běží se nám přeposílají data které posílá kontejner zevnitř také na port 5432. První část před dvojtečkou můžeme,

jakkoliv změnit v rozsahu portů a na tom portu který před dvojtečku napíšeme se pak dostaneme k databázi.

6.1.2 DB.js

Tento soubor slouží k nastavení a exportu připojení k PostgreSQL databázi pomocí knihovny PG. Definujeme zde jméno uživatele, jméno databáze heslo a port na kterém kontejner poslouchá. Dále je zde políčko host, kde by se standartně psala IP adresa nebo URL serveru kde je databáze nastavena. Díky Dockeru však stačí napsat pouze jméno kontejneru kde databáze běží, v našem případě postgres, a nastavíme tím pevný přístup k databázi nehlavně na ip adresu.

```
1  import pkg from 'pg';
2  const { Pool } = pkg;
3  import dotenv from 'dotenv';
4
5  dotenv.config();
6
7  const pool = new Pool({
8    user: process.env.DB_USER || 'postgres',
9    host: process.env.DB_HOST || 'postgres',
10   database: process.env.DB_NAME || 'inventory',
11   password: process.env.DB_PASSWORD || 'password',
12   port: process.env.DB_PORT || 5432,
13 });
14
15 export default pool; Show usages petr-prikryl
16
```

Obrázek 23 - DB.js

(zdroj: vlastní tvorba)

6.1.3 db_init.sql

Tento soubor obsahuje příkazy pro vytvoření tabulek v databázi. Navíc taky obsahuje příkazy INSERT pro ukázkové data.

```

1 CREATE TABLE IF NOT EXISTS devices (
2     id SERIAL PRIMARY KEY,
3     name TEXT NOT NULL,
4     type TEXT NOT NULL,
5     mac_address TEXT UNIQUE NOT NULL,
6     ip_address TEXT UNIQUE NOT NULL,
7     owner TEXT,
8     status VARCHAR(10) DEFAULT 'offline',
9     parent_id INT REFERENCES devices(id) ON DELETE SET NULL
10 );
11
12 CREATE TABLE IF NOT EXISTS roles (
13     id SERIAL PRIMARY KEY,
14     name VARCHAR(50) NOT NULL UNIQUE
15 );
16
17 CREATE TABLE IF NOT EXISTS users (
18     id SERIAL PRIMARY KEY,
19     username TEXT NOT NULL UNIQUE,
20     email TEXT NOT NULL UNIQUE,
21     password_hash TEXT NOT NULL,
22     role_id INTEGER REFERENCES roles(id)
23 );
24
25 CREATE TABLE IF NOT EXISTS online_device_history (
26     id SERIAL PRIMARY KEY,
27     total_online INT NOT NULL,
28     timestamp TIMESTAMP DEFAULT NOW()
29 );
30
31 CREATE TABLE online_exact_device_history (
32     id SERIAL PRIMARY KEY,
33     device_id INTEGER NOT NULL,
34     timestamp TIMESTAMP NOT NULL DEFAULT NOW(),
35     is_online BOOLEAN NOT NULL,
36     FOREIGN KEY (device_id) REFERENCES devices(id) ON DELETE CASCADE
37 );
38
39 INSERT INTO roles (name) VALUES ('admin'), ('viewer');
40
41 INSERT INTO devices (name, type, mac_address, ip_address, owner, parent_id) VALUES
42     ('Core', 'Switch', '08:00:27:00:78:03:00', '192.168.10.49', 'owner: IT Department', (parent_id NULL)),
43     ('Firewall 1', 'type:Switch', 'mac_address: 00:1C:F6:0B:26:58', 'ip_address: 192.168.4.6', 'owner: Security Team', (parent_id 1)),
44     ('Switch 4', 'type:Firewall', 'mac_address: 00:22:38:AD:0F:4B', 'ip_address: 192.168.5.129', 'owner: Networking Team', (parent_id 2)),
45     ('Switch 5', 'type:Router', 'mac_address: 08:0A:AA:93:83:89', 'ip_address: 192.168.2.102', 'owner: IT Department', (parent_id 1)),
46     ('Firewall 6', 'type:Switch', 'mac_address: 00:00:00:00:00:00', 'ip_address: 192.168.5.193', 'owner: Networking Team', (parent_id 2)),
47     ('Switch 7', 'type:Firewall', 'mac_address: 00:60:14:9F:2A:00', 'ip_address: 192.168.4.252', 'owner: Networking Team', (parent_id 2)),
48     ('Router 8', 'type:Switch', 'mac_address: 08:07:8C:88:1C:07', 'ip_address: 192.168.8.258', 'owner: IT Department', (parent_id 2)),
49     ('Router 9', 'type:Router', 'mac_address: 00:9A:52:3E:52:7F', 'ip_address: 192.168.4.219', 'owner: Security Team', (parent_id 7)).

```

Obrázek 24 - db__init.sql

(zdroj: vlastní tvorba)

6.2 Backendová část aplikace

6.2.1 GraphQL

GraphQL je dotazovací jazyk pro API. Jeho výhoda spočívá ve flexibilitě obdržených dat, můžeme endpointy nastavit tak aby nám poskytly pouze potřebné informace, ne víc, ne méně.

Z pohledu architektury se GraphQL skládá ze dvou základních částí:

- Schéma (schema.js) – definuje datové typy, dotazy (queries) a operace (mutations), které API poskytuje.

```
import { gql } from 'apollo-server-express';

const typeDefs : DocumentNode = gql`
  type Device {
    id: ID!
    name: String!
    type: String!
    mac_address: String!
    ip_address: String!
    owner: String
    status: String!
    parent: Device
  }
  type DeviceOnlinePoint {
    timestamp: String!
    is_online: Boolean!
  }
  type AuthPayload {
    token: String!
    user: User!
  }
  type User {
    id: ID!
    username: String!
    email: String!
    role_id: ID!
  }
`
```

Obrázek 25 – ukázka schema.js

(zdroje: vlastní zpracování)

- Resolvery (resolvers.js) – implementují konkrétní chování jednotlivých dotazů. V praxi představují propojení mezi GraphQL vrstvou a databází nebo jiným zdrojem dat.

```
const resolvers = {} = {
  Query: {
    users: async (_, __, context) => Promise.any() => {
      if (!context.user || context.user.role !== 1) {
        throw new Error("Přístup odepřen. Jen admin může vidět uživatele.");
      }
      const result = await context.pool.query("SELECT id, username, email, role_id FROM users");
      return result.rows;
    },

    devices: async (_, __, context) => Promise.any() => {
      if (!context.user) throw new Error("Neautorizováno");
      const res = await context.pool.query('SELECT * FROM devices');
      return res.rows;
    },

    device: async (_, { id }, context) => Promise.any() => {
      if (!context.user) throw new Error("Neautorizováno");
      const res = await context.pool.query('SELECT * FROM devices WHERE id = $1', [id]);
      return res.rows[0];
    },

    onlineDeviceHistory: async (_, __, context) => Promise.any() => {
      if (!context.user) throw new Error("Neautorizováno");
      const res = await context.pool.query("SELECT total_online, timestamp FROM online_device_history ORDER BY timestamp ASC");
      return res.rows.map(row => ({
        total_online: row.total_online,
        timestamp: row.timestamp.toISOString()
      }));
    },

    deviceOnlineHistory: async (_, { deviceId }, context) => Promise.any() => {
      if (!context.user) throw new Error("Neautorizováno");
      const result = await context.pool.query(
        `SELECT timestamp, is_online FROM online_exact_device_history
        WHERE device_id = $1 AND timestamp >= NOW() - INTERVAL '24 HOURS'
        ORDER BY timestamp ASC`,
        [deviceId]
      );
      return result.rows.map(row => ({
        timestamp: row.timestamp.toISOString(),
        is_online: row.is_online,
      }));
    },
  },
};
```

Obrázek 26 - Ukázka resolvers.js

(zdroj: vlastní zpracování)

6.2.2 index.js

Tento soubor je hlavní vstupní bod backendu. Slouží ke spuštění serveru, připojení ke GraphQL serveru, zabezpečení pomocí JWT a propojení s databází. V hlavičce souboru se nachází importy pro knihovny:

- express – Webový framework pro Node.js
- apollo-server-express – umožňuje napojit GraphQL server na express
- pool – import připojení k databázi
- typeDefs + resolvers – Definice GraphQL schématu a jeho funkční logika
- dotenv – možnost načtení proměnných z env souboru
- cors – umožňuje přístup z frontendu v našem případě z Reactu
- jwt – pro dekodování a ověřování JWT tokenů

Základní nastavní frontend aplikace probíhá pomocí funkce

```
const app = express ();
app.use(express.json());
app.use(express.urlencoded({ extended: true }));
app.use(cors());
```

Ta tedy spustí express a aktivuje middleware a to konkrétně

express.json, který povolí práci s JSON v požadavcích, express.urlencoded, to umožňuje aplikaci zpracovávat form-data a nakonec cors který nastavuje server, aby povoloval požadavky z jiných domén, protože frontend je v jiném kontejneru.

Dále je část kódu

```
const server = new ApolloServer({
  typeDefs,
  resolvers,
  context: ({ req }) => {
    const auth = req.headers.authorization || '';
    const token = auth.replace("Bearer ", "");

    let user = null;

    if (token) {
      try {
        user = jwt.verify(token, process.env.JWT_SECRET);
        console.log("☑ Uživatelský token v kontextu:", user);
      } catch (err) {
        console.error("✗ Neplatný token:", err.message);
      }
    } else {
      console.warn("⚠ Token není přítomen.");
    }

    return { pool, user };
  }
});
```

Kde se už přímo konfiguruje GraphQL server. Tento kód načte typeDefs a resolvers, předává context, který je potřeba pro autentizaci a přístup k databázi a v contextu dekoduje token,

Zbývá poslední důležitá část kódu :

```
async function startServer() {
  await server.start();
```

```

server.applyMiddleware({ app });

app.listen(4000, () => {
  console.log('🚀 Server běží na http://localhost:4000/graphql');
});
}

startServer().catch(console.error);

```

Tento kód spouští Apollo GraphQL server na portu 4000, zároveň aplikuje GraphQL middleware do Expressu, tím vzniká endpoint /graphql. Pokud vše proběhne podle očekávání do konzole je vypsána zpráva o úspěšném startu.

6.2.3 Dockerfile a docker_compose.yml

V této části pro chod backendu v Dockeru jsou důležité dva soubory a to dockerfile, který přímo určuje, jak se kontejner spouští a docker-compose.yml který ho spouští.

```

# Zde se použije NODE jako základní obraz
FROM node:18
#Zde se nastaví pracovní složka
WORKDIR /app
#Zde se kopírují dva soubory, které obsahují seznam závislostí
COPY package.json package-lock.json ./

#zde se ty závislosti z předchozího řádku nainstalují
RUN npm install

# zde se kopíruje samotný kód aplikace
COPY . .
# zde se převádí CRLF na LF, protože některé komponenty jsou náchylné na správné ukončení souboru
RUN apt-get update && apt-get install -y dos2unix
RUN dos2unix /app/entrypoint.sh
# instalace dalších závislostí
RUN apt-get update && apt-get install -y postgresql-client dos2unix iputils-ping net-tools iproute2 nmap

# zde se nastavuje oprávnění pro hlavní spouštěcí skript
RUN chmod +x /app/entrypoint.sh

# zde se povoluje port 4000
EXPOSE 4000

# a zde se spouští spouštěcí skript
CMD ["/bin/bash", "/app/entrypoint.sh"]

```

Obrázek 27 – DOCKERFILE

(zdroj: vlastní zpracování)

```

backend:
  build:
  container_name:
  restart:

```

./backend
backend
always

```

depends_on:
  -
                                postgres
environment:
  POSTGRES_USER:                postgres
  POSTGRES_PASSWORD:            password
  DB_HOST:                      postgres
  POSTGRES_DB:                  inventory
  JWT_SECRET:                   tajnyklic123
  TZ:                           Europe/Prague
ports:
  -
                                "4000:4000"
networks:
  -
                                app_network
volumes:
  - ./backend:/app

```

V souboru `docker-compose.yml` se definují proměnné, porty i síť a tím Dockeru vědět, a také je zde informace o tom, že se nepoužívá image z repozitáře, ale že musí být sám sestaven.

6.3 Frontendová část aplikace

6.3.1 App.js

Jedná se o hlavní část frontendu zde se napojuje na backend a definují se důležité nastavení.

```

const httpLink : ApolloLink = createHttpLink({ linkOptions: {
  uri: "http://localhost:4000/graphql",
}});

const authLink : ApolloLink = setContext( setter: (_ : GraphQLRequest , { headers }) : {headers: ...} => {
  const token : string = localStorage.getItem( key: "token");
  return {
    headers: {
      ...headers,
      authorization: token ? `Bearer ${token}` : "",
    },
  },
};

```

Obrázek 28 - App.js const httpLink a authLink

(zdroj: vlastní tvorba)

Ve výše uvedeném kódu je definice, kde se nachází backend, je zde uvedena URL adresa. V další části se nastavuje context tak, aby vkládal do hlaviček každého požadavku token.

```

const client : ApolloClient<NormalizedCacheObject> = new ApolloClient({ options: {
  link: authLink.concat(httpLink),
  cache: new InMemoryCache(),
}});

```

Obrázek 29 - App.js Apollo klient

(zdroj: vlastní tvorba)

V této části kódu se napojuje Apollo client na backend a nastavuje se ukládání dat do mezipaměti. Data se ukládají do mezipaměti, aby se aplikace vykreslovala rychleji.

Ve funkci

```
function App() {
  const [loggedIn, setLoggedIn] =
    useState(!!localStorage.getItem("token"));
```

definuje se zde stav loggedIn a ten říká, zdali se nachází token v localStorage, pokud ano, je uživatel přihlášen.

V další části kódu je navigace pomocí react routeru. Je zde hlavní stránka dashboard, ke které má přístup pouze přihlášený uživatel. Pak je tu i wallboard, kde se nachází statistiky.

```
<Routes>
  <Route path="/" element={<Dashboard onLogout={() =>
    setLoggedIn(false)} />} />
  <Route path="/wallboard" element={<Wallboard />} />
</Routes>
```

Dále přímo na dashboard komponentě se vykresluje obsah menu podle toho, jakou má uživatel roli.

```
const role = localStorage.getItem("role");
const tabs = [
  { label: "Vizualizace", component: <GraphSwitcher /> },
  ...(role === "1"
    ? [
        { label: "Správa zařízení", component:
          <RequireAdmin><DeviceManager /></RequireAdmin> },
        { label: "Správa uživatelů", component:
          <RequireAdmin><UserManagement /></RequireAdmin> },
      ]
    : []),
  { label: "Statistiky", component: <Statistics /> },
];
```

V poslední řadě vykreslujeme také menu uživatelského profilu

```
<UserProfileMenu onLogout={onLogout} />
```

A UI.

6.3.2 GraphSwitcher.js

Tato komponenta velice jednoduše nabízí další podmenu pro výběr konkrétního stylu mapy, přesměrovává na NetworkGraph, NetworkGraph2 a NetworkGraph3

```
1 // GraphSwitcher.js
2 import React, { useState } from "react";
3 import { Tabs, Tab, Box } from "@mui/material";
4 import NetworkGraph from "../NetworkGraph";
5 import NetworkGraph2 from "../NetworkGraph2";
6 import NetworkGraph3 from "../NetworkGraph3";
7
8 const GraphSwitcher = () => { Show usages  petr-prikryl
9   const [tab : number, setTab] = useState( initialState: 0);
10
11   return (
12     <Box sx={{ width: "100%", height: "100vh" }}>
13       <Tabs
14         value={tab}
15         onChange={(e : SyntheticEvent, newValue) : void => setTab(newValue)}
16         indicatorColor="primary"
17         textColor="primary"
18         variant="fullWidth"
19         sx={{ backgroundColor: "#f5f5f5" }}
20       >
21         <Tab label="Síťový graf (force)" />
22         <Tab label="Hierarchie (elkjs)" />
23         <Tab label="Hierarchie (dagre)" />
24       </Tabs>
25
26       {tab === 0 && <NetworkGraph />}
27       {tab === 1 && <NetworkGraph2 />}
28       {tab === 2 && <NetworkGraph3 />}
29     </Box>
30   );
31 };
32
33 export default GraphSwitcher; Show usages  petr-prikryl
```

Obrázek 30 - GraphSwitcher.js

(zdroj: vlastní tvorba)

6.3.3 NetworkGraph.js

V této komponentě probíhá pokročilá vizualizace síťových zařízení pomocí knihovny D3.js. Vykresluje síť jako interaktivní graf s ikonami tooltipy, přiblížením a možnostmi táhnutí. V první části souboru probíhá import potřebných knihoven a dotazů dále probíhá

import ikonek ze složky icons. Ikony jsou ve formátu svg. Poté již useEffect, vykresluje síťový graf. Důležité jsou následující konfigurace:

```
svg.call(d3.zoom().scaleExtent([0.1, 3]).on("zoom", ...));  
svg.selectAll("*").remove();
```

Aktivuje zoom a panování celého svg, poté se vše vyčistí, aby se předešlo duplicitám.

```
const tooltip = d3.select("body").append("div")  
  .attr("class", "tooltip")  
  .style("opacity", 0);
```

Zde se vytváří tooltipy. To je plovoucí element s informacemi o zařízení (jméno ip adresa a podobné) Zobrazuje se po najetí myši.

```
const nodes = data.devices.map(...);  
const links = data.devices.filter(d => d.parent)...
```

Zde se definují uzly (nodes) a vazhy (links). Každé zařízení je uzel a pokud má parent, vytvoří se mezi nimi vztah

```
const simulation = d3.forceSimulation(nodes)  
  .force("link", d3.forceLink(links)...)  
  .force("charge", d3.forceManyBody().strength(-500))  
  .force("center", d3.forceCenter(...))  
  .force("collide", d3.forceCollide(60));
```

Zde se nastavuje fyzikální simulace. Zařízení se díky tomu odpuzují, spojují a vyrovnávají do sítě.

```
const link = g.selectAll(".link")...  
const node = g.selectAll(".node")...
```

Dále se vykreslují linky a uzly. Linky spojují rodiče a děti. Uzly je skupinový (g) element který obsahuje

- Obrázek zařízení
- Rámeček podle online stavu (červená/ zelená)
- Jméno
- Tooltip

Dále se v souboru nachází nastavení interakce drag and drop

```
.on("start", dragStart)  
.on("drag", dragging)  
.on("end", dragEnd)
```

A třeba aktualizace pozic při každém tiku simulace

```
simulation.on("tick", () => {  
  ...  
});
```

Nakonec se celá komponenta vykreslí do boxu.

```
return (  
  <Box sx={{ width: "100%", height: "calc(100vh - 48px)" }}>  
    <svg ref={svgRef} width="2000" height="2000" />  
  </Box>  
);
```

6.3.4 NetworkGraph2.js

Tato komponenta je velmi podobná té předchozí, avšak navíc vykresluje graf hierarchicky podle ELK.js (Eclipse Layout Kernel) - uzly mají pevnou velikost a čtvercový layout. Linky mezi zařízeními jsou přímé a ne fyzikální.

```
const NetworkGraph2 = () => { Show usages ⓘ petr-prikryl
const { loading : boolean, error : ApolloError, data : MaybeMasked<any> } = useQuery(GET_DEVICES);
const svgRef : MutableRefObject<undefined> = useRef();

useEffect( effect : () :void => {
  if (!data || !window.ELK) return;

  const elk = new window.ELK();

  const nodes = data.devices.map(d => ({
    id: String(d.id),
    width: 60,
    height: 60,
    name: d.name,
    type: d.type,
    status: d.status,
    mac_address: d.mac_address,
    ip_address: d.ip_address,
    owner: d.owner,
  }));

  const edges = data.devices
    .filter(d => d.parent)
    .map(d => ({ id: `${d.parent.id}->${d.id}`, sources: [String(d.parent.id)], targets: [String(d.id)] }));

  const graph : {} = {
    id: "root",
    layoutOptions: {
      "elk.algorithm": "layered",
      "elk.direction": "DOWN",
      "elk.spacing.nodeNode": "50",
      "elk.layered.spacing.nodeNodeBetweenLayers": "80"
    },
    children: nodes,
    edges: edges,
  };

  elk.layout(graph).then(layout : Omit<...> & {...} => {
    const svg = d3.select(svgRef.current);
    svg.selectAll("*").remove();

    const zoomGroup = svg.append("g");

    svg.call(d3.zoom()
      .scaleExtent([0.2, 2])
      .on("zoom", (event) => zoomGroup.attr("transform", event.transform)));

    const tooltip = d3.select("body").append("div")
      .attr("class", "tooltip")
      .style("position", "absolute")
      .style("background-color", "#fff");
```

Obrázek 31 - Ukázka kódu NetworkGraph.js

(zdroj : vlastní tvorba)

6.3.5 NetworkGraph3.js

Tato třetí komponenta je velice podobná té předchozí, používá Dagre pro automatické rozvržení uzlů hierarchicky. Ze všech výše uvedených se jeví jako nejpřehlednější pro zobrazování sítě.

```

const NetworkGraph3 = () => { Show usages  ⚙️ petr-prikryl
  const { loading : boolean , error : ApolloError , data : MaybeMasked<any> } = useQuery(GET_DEVICES);
  const svgRef : MutableRefObject<undefined> = useRef();

  useEffect( effect: () : undefined | function(): any => {
    if (!data) return;

    const g : dagre.graphlib.Graph = new dagre.graphlib.Graph();
    g.setGraph({ rankdir: "TB", nodesep: 50, ranksep: 100 });
    g.setDefaultEdgeLabel(() => ({}));

    data.devices.forEach(device => {
      g.setNode(String(device.id), {
        label: device.name,
        width: 60,
        height: 60,
        type: device.type,
        status: device.status,
        mac_address: device.mac_address,
        ip_address: device.ip_address,
        owner: device.owner,
      });
    });

    data.devices.forEach(device => {
      if (device.parent) {
        g.setEdge(String(device.parent.id), String(device.id));
      }
    });

    dagre.layout(g);

    const svg = d3.select(svgRef.current);
    svg.selectAll("*").remove();

    const zoomGroup = svg.append("g");

    svg.call(d3.zoom()
      .scaleExtent([0.2, 2])
      .on("zoom", (event) => zoomGroup.attr("transform", event.transform)));
  }, [data]);
}

```

Obrázek 32 - Ukázka kódu NetworkGraph3.js

(zdroj: vlastní tvorba)

6.3.6 DeviceManager.js

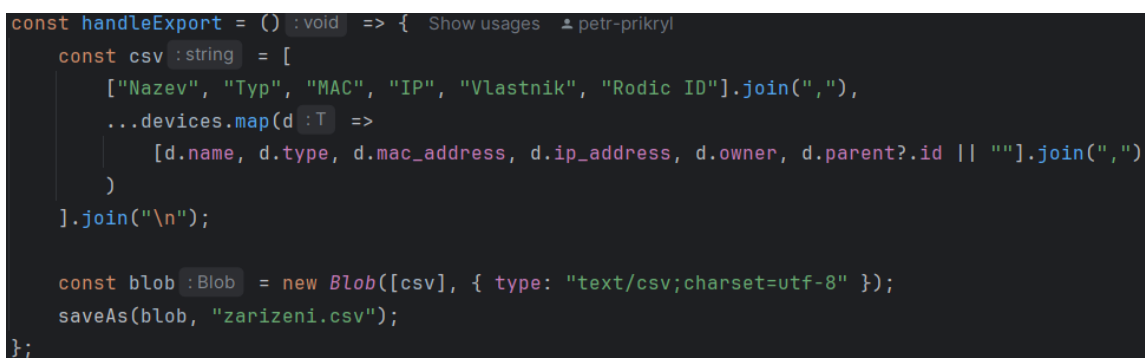
Tato komponenta slouží ke správě, editaci, přidávání a monitorování zařízení. Nachází se zde tabulka s výčtem všech zařízení a důležitých informací. Nachází se zde import potřebných mutací například:

```

const GET_DEVICES = gql`
  query {
    devices {
      id
      name
      type
      mac_address
      ip_address
      owner
      status
      parent {
        id
        name
      }
    }
  }
`;

```

Dále se zde nachází funkce pro export:



```

const handleExport = () :void => { Show usages  ▲ petr-prikryl
  const csv :string = [
    ["Nazev", "Typ", "MAC", "IP", "Vlastnik", "Rodic ID"].join(","),
    ...devices.map(d :T =>
      [d.name, d.type, d.mac_address, d.ip_address, d.owner, d.parent?.id || ""].join(",")
    )
  ].join("\n");

  const blob :Blob = new Blob([csv], { type: "text/csv;charset=utf-8" });
  saveAs(blob, "zarizeni.csv");
};

```

Obrázek 33 - Funkce pro export

(zdroj: vlastní tvorba)

Mimo jiné se v této komponentě nachází funkce pro sestavování seznamu zařízení a logika WOL.

6.3.7 Login.js

Komponenta login.js slouží k přihlašování do systému. Používá mutaci login, aby získala token pomocí hesla a emailu. Token pak uloží i s rolí a emailem do localStorage.

```

import React, { useState } from "react";
import { useMutation, gql } from "@apollo/client";
import { TextField, Button, Typography, Paper, Box } from "@mui/material";

const LOGIN :DocumentNode = gql`
  mutation Login($email: String!, $password: String!) {
    login(email: $email, password: $password) {
      token
      user {
        id
        email
        role_id
      }
    }
  }
`;

const Login = ({ onLogin }) => {
  Show usages  petr-prikryl *
  const [form :{email: string, password: string}], setForm] = useState( initialState: { email: "", password: "" });
  const [login, { error : ApolloError | undefined }] = useMutation(LOGIN, options: {
    onCompleted: (data : MaybeMasked<NoInfer<...>>) :void => {
      localStorage.setItem("token", data.login.token);
      localStorage.setItem("role", data.login.user.role_id);
      localStorage.setItem("email", data.login.user.email);
      onLogin(); // callback do App
    },
  });

  const handleChange = (e) :void => setForm( value: { ...form, [e.target.name]: e.target.value }); Show usages  petr-prikryl

  const handleSubmit = () :Promise<FetchResult<...>> => login( options: { variables: form }); Show usages  petr-prikryl

  return (
    <Paper elevation={3} style={{ padding: 30, maxWidth: 400, margin: "100px auto" }}>
      <Typography variant="h5">Přihlášení</Typography>
      <TextField
        label="E-mail"
        name="email"
        fullWidth
        margin="normal"
        value={form.email}
        onChange={handleChange}
      />
      <TextField
        label="Heslo"
        name="password"
        type="password"
        fullWidth
        margin="normal"

```

Obrázek 34 – Ukázka komponenty login.js

(zdroj: vlastní tvorba)

6.3.8 UserManagement.js

V poslední řadě zdrojový kód obsahuje i komponentu UserManagement.js, která má na starosti správu uživatelů pomocí ní frontend komunikuje vytváření a mazání uživatelů nebo změny hesel.

```
const ADD_USER :DocumentNode = gql`
  mutation AddUser($username: String!, $email: String!, $password: String!, $role_id: Int!) {
    addUser(username: $username, email: $email, password: $password, role_id: $role_id) {
      id
    }
  }
`;

const DELETE_USER :DocumentNode = gql`
  mutation DeleteUser($id: ID!) {
    deleteUser(id: $id)
  }
`;

const CHANGE_USER_PASSWORD :DocumentNode = gql`
  mutation ChangeUserPassword($userId: ID!, $newPassword: String!) {
    changeUserPassword(userId: $userId, newPassword: $newPassword)
  }
`;

const UserManager = () => { Show usages  ⚡ petr-prikryl
  const { data : MaybeMasked<any> , loading : boolean , error : ApolloError , refetch } = useQuery(GET_USERS);
  const [addUser] = useMutation(ADD_USER, options: { onCompleted: () : Promise<ApolloQueryResult<...>> => refetch() });
  const [deleteUser] = useMutation(DELETE_USER, options: { onCompleted: () : Promise<ApolloQueryResult<...>> => refetch() });
  const [changePasswordAsAdmin] = useMutation(CHANGE_USER_PASSWORD, options: { onCompleted: () : Promise<ApolloQueryResult<...>> => refetch() });

  const [open : boolean , setOpen] = useState( initialState: false);
  const [form : {...} , setForm] = useState( initialState: { username: "", email: "", password: "", role_id: 2 });
  const [changePasswordOpen : boolean , setChangePasswordOpen] = useState( initialState: false);
  const [selectedUserId , setSelectedUserId] = useState( initialState: null);
  const [newPassword : string , setNewPassword] = useState( initialState: "");
  const handleChange = e => { Show usages  ⚡ petr-prikryl
    setForm( value: { ...form, [e.target.name]: e.target.value });
  };

  const handleAddUser = () :void => { Show usages  ⚡ petr-prikryl
    addUser( options: { variables: { ...form, role_id: parseInt(form.role_id) } });
    setOpen( value: false);
  };

  const handleDelete = (id) :void => { Show usages  ⚡ petr-prikryl
    if (window.confirm("Opravdu smazat uživatele?")) {
      deleteUser( options: { variables: { id } });
    }
  };

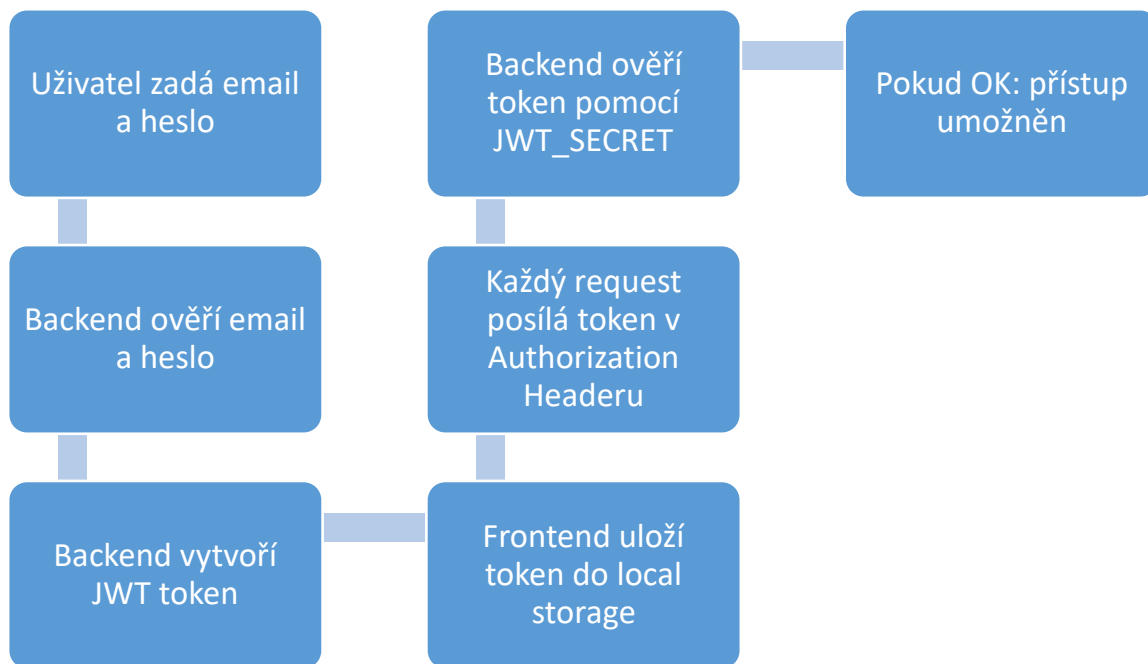
  const handleChangePassword = () :void => { Show usages  ⚡ petr-prikryl
    if (selectedUserId && newPassword) {
      changePasswordAsAdmin( options: { variables: { userId: selectedUserId, newPassword } });
      setChangePasswordOpen( value: false);
    }
  }
};
```

Obrázek 35 - Ukázka kódu UserManagement.js

(zdroj: vlastní tvorba)

6.4 Zabezpečení systému

Jak již bylo zmiňováno výše, aplikace klade důraz na zabezpečení. V aplikaci se používá autentizace pomocí JWT tokenu, která zabezpečuje veškeré požadavky od frontendu na backend. Na obrázku níže lze vidět základní proces autentizace požadavků



Obrázek 36 - Proces JWT

(zdroj: vlastní tvorba)

Token uživatel obdrží jako odpověď na mutaci Login:

```
const token = jwt.sign(
  { userId: user.id, role: user.role_id },
  process.env.JWT_SECRET,
  { expiresIn: "1h" }
);
```

Uživatel obdrží token, který platí 1 hodinu. Z důvodu autentizace je nutné mít správně synchronizovaný čas skrze všechny kontejnery a počítač ze kterého se přistupuje.

6.4.1 Security Headers

Pro zvýšení bezpečnosti webové aplikace byly implementovány tzv. bezpečnostní hlavičky HTTP (Security Headers). Těmito hlavičkami je prohlížeč instruován, jakým způsobem má s webovou aplikací nakládat, a jsou tak minimalizována některá rizika, jako je například cross-site scripting (XSS), clickjacking nebo podstrčení obsahu.

Byla použita hlavička Content-Security-Policy, kterou lze omezit načítání skriptů, stylů, obrázků a dalších zdrojů pouze na důvěryhodné domény. Dále byly nasazeny hlavičky

X-Content-Type-Options, X-Frame-Options a Referrer-Policy, kterými je omezena možnost zneužití MIME typů, vložení stránky do rámu (iframe) a únik referenčních údajů.

Dále byly povoleny pouze protokoly TLS 1.2 a 1.3.

Security Report Summary



Site: <https://bp.prikryl.cz/>

IP Address: 104.21.96.1

Report Time: 19 May 2025 12:09:31 UTC

Headers: ✔ Strict-Transport-Security ✔ Content-Security-Policy ✔ X-Frame-Options ✔ X-Content-Type-Options ✔ Referrer-Policy ✔ Permissions-Policy

Advanced: Wow, amazing grade! Perform a deeper security analysis of your website and APIs: [Try Now](#)

Raw Headers

HTTP/2	200
date	Mon, 19 May 2025 12:09:31 GMT
content-type	text/html
server	cloudflare
nel	{ "report_to": "cf-nel", "success_fraction": 0.0, "max_age": 604800 }
last-modified	Fri, 16 May 2025 15:40:50 GMT
vary	accept-encoding
strict-transport-security	max-age=31536000; includeSubDomains; preload
content-security-policy	default-src 'self'; connect-src 'self' https://graphql.prikryl.cz; script-src 'self' https://unpkg.com; style-src 'self' 'unsafe-inline'; object-src 'none'; base-uri 'self';
x-frame-options	SAMEORIGIN
x-content-type-options	nosniff
referrer-policy	strict-origin-when-cross-origin
permissions-policy	geolocation=(), microphone=(), camera=()
x-served-by	bp.prikryl.cz
report-to	{ "group": "cf-nel", "max_age": 604800, "endpoints": [{ "url": "https://a.nel.cloudflare.com/report/v4?s=1G1TgWBWd3FR4MSveHc8CFtWHn25qWwS5Udh5%2FXu615ERgxikE6b2M%2BpgDUFO86ngm1UM8PNq0vdYesT%2BwdKLjxUzRjSw6VBHxvVMRI%3D" }] }
cf-cache-status	DYNAMIC
content-encoding	gzip
cf-ray	94237c435932bdb8-DUB
alt-svc	h3=":443"; ma=86400

Obrázek 37 - Výsledek testu ze stránky securityheaders.com

(zdroj: stránka securityheaders.com)


[Home](#) [Projects](#) [Qualys Free Trial](#) [Contact](#)

 You are here: [Home](#) > [Projects](#) > [SSL Server Test](#) > bp.prikryl.cc

SSL Report: bp.prikryl.cc

 Assessed on: Mon, 19 May 2025 12:22:23 UTC | [Hide](#) | [Clear cache](#)
[Scan Another >>](#)

	Server	Test time	Grade
1	2606:4700:3030:0:0:0:6815:6001 Ready	Mon, 19 May 2025 12:11:54 UTC Duration: 44.944 sec	A+
2	2606:4700:3030:0:0:0:6815:4001 Ready	Mon, 19 May 2025 12:12:39 UTC Duration: 44.481 sec	A+
3	2606:4700:3030:0:0:0:6815:2001 Ready	Mon, 19 May 2025 12:13:24 UTC Duration: 44.252 sec	A+
4	2606:4700:3030:0:0:0:6815:7001 Ready	Mon, 19 May 2025 12:14:08 UTC Duration: 44.463 sec	A+
5	2606:4700:3030:0:0:0:6815:1001 Ready	Mon, 19 May 2025 12:14:52 UTC Duration: 44.135 sec	A+
6	2606:4700:3030:0:0:0:6815:5001 Ready	Mon, 19 May 2025 12:15:36 UTC Duration: 44.200 sec	A+
7	2606:4700:3030:0:0:0:6815:3001 Ready	Mon, 19 May 2025 12:16:21 UTC Duration: 51.489 sec	A+
8	104.21.112.1 Ready	Mon, 19 May 2025 12:17:12 UTC Duration: 44.621 sec	A+
9	104.21.80.1 Ready	Mon, 19 May 2025 12:17:57 UTC Duration: 44.449 sec	A+
10	104.21.64.1 Ready	Mon, 19 May 2025 12:18:41 UTC Duration: 44.726 sec	A+
11	104.21.96.1 Ready	Mon, 19 May 2025 12:19:26 UTC Duration: 44.2 sec	A+
12	104.21.16.1 Ready	Mon, 19 May 2025 12:20:10 UTC Duration: 44.224 sec	A+
13	104.21.48.1 Ready	Mon, 19 May 2025 12:20:54 UTC Duration: 44.336 sec	A+
14	104.21.32.1 Ready	Mon, 19 May 2025 12:21:39 UTC Duration: 44.121 sec	A+

SSL Report v2.4.0

Obrázek 38 - Výsledky testů ze stránky ssllabs.com

(zdroj: výsledky testů stránky ssllabs.com)

7 TESTOVÁNÍ A VYHODNOCENÍ

7.1 Metodika testování

Testování probíhalo na počítači se systémem Windows 11 Pro se sestavením 22631.5189. Jako webový prohlížeč byla použita Opera GX, verze 118.0.5461.50.

Frontendová část byla testována pomocí nástroje Lighthouse (verze 100.0.0.4), který hodnotil výkonnost, přístupnost, SEO a další metriky uživatelského rozhraní. Byly sledovány ukazatele jako First Contentful Paint, Largest Contentful Paint, Total Blocking Time a Speed Index. Výsledky těchto testů poskytují přehled o tom, jak rychle a plynule se aplikace načítá a reaguje.

Backendová část byla testována přes GraphQL Sandbox (verze 6a1de4) odesíláním jednotlivých dotazů a mutací. Byl sledován čas odpovědi jednotlivých požadavků a správnost vrácených dat. V rámci testování byla ověřena i správná práce s chybami, např. při neautorizovaném přístupu nebo neexistujícím záznamu.

Součástí testování bylo i ruční testování funkčnosti jednotlivých komponent, jako jsou přihlašování, správa zařízení, vizualizace sítě, historie online stavů a další.

Pro ověření bezpečnosti byla provedena kontrola JWT tokenů – ověření jejich správného vytvoření, uchování a vypršení platnosti.

7.2 Funkční testy

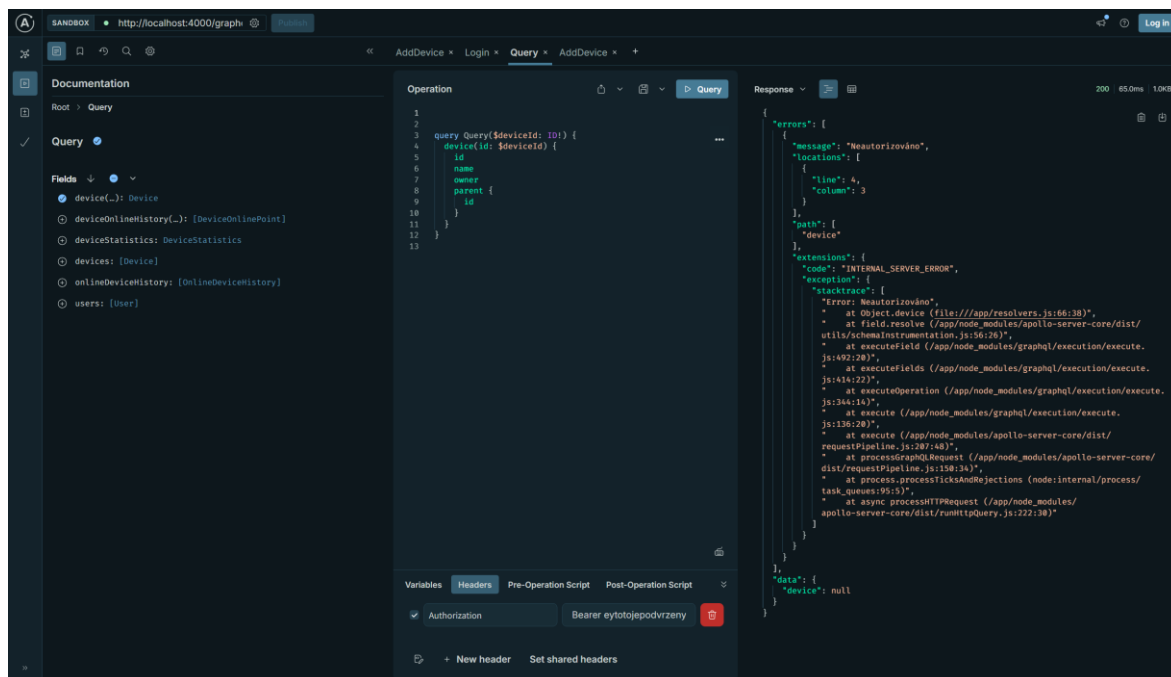
Funkční testy byly prováděny manuálně za pomoci webového prohlížeče i Apollo Sandboxu. Veškeré funkce plní účel podle očekávání. Byla otestována i bezpečnost aplikace. Aplikace se jeví jako zabezpečená a kompletní.

Tabulka 14 - Funkční testy

Funkce	Očekávaný výsledek	Je výsledek podle očekávání?
Přihlášení	Uživatel je přihlášen a přesměrován na hlavní stránku	ANO

Odhlášení	Uživatel je odhlášen a token zmizí z localStorage	ANO
Změna hesla	Heslo změněno a uživatel odhlášen, staré heslo nefunguje nové ano	ANO
Přidání zařízení	Zařízení je přidáno a zobrazeno v seznamu	ANO
Probuzení zařízení (WoL)	Zařízení je fyzicky probuzeno	ANO
Export zařízení	CSV je správně exportováno	ANO
Vizualizace	Graf s ikonami a tooltips je zobrazen	ANO
Správa uživatelů	Uživatele lze smazat, je odebrán z databáze	ANO
Odhlášení po vypršení tokenu	Po uplynutí 1 h od přihlášení proběhne odhlášení	ANO
Řízení přístupu k seznamu zařízení bez tokenu	API neposkytne odpověď, ale vypíše chybu Neautorizováno	ANO
Řízení přístupu k seznamu zařízení s vypršeným tokenem	API neposkytne odpověď, ale vypíše chybu Neautorizováno	ANO
Řízení přístupu k seznamu zařízení s náhodným tokenem	API neposkytne odpověď, ale vypíše chybu Neautorizováno	ANO

(zdroj: vlastní zpracování)



Obrázek 39 - Testování v Apollo GraphQL Sandboxu

(zdroj: vlastní zpracování)

7.3 Výkonové testy

Tabulka 15 - Výkonové testy

Frontend/Backend	Co testujeme?	Jak testujeme	Výsledný čas	Hodnocení
Backend	Výpis 4 uživatelů	GraphQL Sandbox	36ms	Výborný
Backend	Výpis 103 zařízení	GraphQL Sandbox	210ms	Uspokojivý
Backend	Výpis deviceOnlineHistory v	GraphQL Sandbox	85ms	Výborný
Backend	Vypsání DeviceStatistics	GraphQL Sandbox	34ms	Výborný
Backend	Plný výpis OnlineDeviceHistory	GraphQL Sandbox	39ms	Výborný

Backend	Výpis jednoho zařízení	GraphQL Sandbox	62ms	Velmi dobrý
Frontend	First Contentful Paint	Lighthouse	0,8s	Výborný
Frontend	Largest Contentful Paint	Lighthouse	1,7s	Výborný
Frontend	Total Blocking time	Lighthouse	20ms	Výborný
Frontend	Speed Index	Lighthouse	1,4s	Výborný

(zdroj: vlastní zpracování)

ZÁVĚR

Cílem této bakalářské práce bylo navrhnout a implementovat moderní systém pro evidenci, vizualizaci a inventarizaci IT infrastruktury. Výsledná aplikace úspěšně spojuje správu zařízení se zobrazením jejich síťových vazeb, automatickým sledováním dostupnosti a možností inventurní evidence nezávislé na aktivním stavu zařízení.

Během vývoje byl kladen důraz na bezpečnost (JWT, hashování hesel), uživatelskou přívětivost (responzivní UI v Reactu a Material UI) a efektivní práci s daty (GraphQL). Implementace inventarizačního modulu umožnila rozšířit systém i na evidenci zařízení mimo aktuální síťové prostředí, čímž se aplikace stává užitečným nástrojem pro správce IT v reálném provozu.

Všechny funkce byly úspěšně otestovány a výsledky potvrdily stabilitu, výkonnost i přehlednost systému. Aplikace je navržena jako otevřený a škálovatelný nástroj, který lze dále rozvíjet o další funkce dle specifických potřeb organizace.

SEZNAM POUŽITÉ LITERATURY

- [1] WEBGLOBE. Co je infrastruktura (v kontextu IT). Online. 2025. Dostupné z: <https://www.webglobe.cz/poradna/co-je-infrastruktura>. [cit. 2025-04-20].
- [2] CDC DATA S.R.O. Klíčové pilíře moderní IT infrastruktury. Online. 2023. Dostupné z: <https://www.cdc.cz/cs/klicove-pilire-moderni-it-infrastruktury/>. [cit. 2025-04-20].
- [3] DIGITÁLNÍ ARCHITEKTI S.R.O. Docker jako významná složka moderní datové infrastruktury. Online. © 2025. Dostupné z: <https://digitalniarchitekti.cz/clanek/docker/>. [cit. 2025-04-19].
- [4] MICROSOFT. Co je PostgreSQL? Online. © 2025. Dostupné z: <https://azure.microsoft.com/cs-cz/resources/cloud-computing-dictionary/what-is-postgresql>. [cit. 2025-04-13].
- [5] EXPERT DEV, S.R.O. D3. Online. © 2024. Dostupné z: <https://www.expert-dev.cz/slovnicek-pojmu/d3/>. [cit. 2025-04-13].
- [6] DCX TRADE S.R.O. Backendové frameworky. Online. 2025. Dostupné z: <https://webovestudio.com/clanky/tvorba-webu/backend/backendove-frameworky>. [cit. 2025-04-13].
- [7] WEBGLOBE. Co je Node.js. Online. 2025. Dostupné z: <https://www.webglobe.cz/poradna/co-je-node-js>. [cit. 2025-04-13].
- [8] RASCASONE, S.R.O. Proč k vývoji webových aplikací použít technologii NodeJS? Online. 2024. Dostupné z: <https://www.rascasone.com/cs/blog/node-js-architektura-moduly-npm>. [cit. 2025-04-13].
- [9] EXPERT DEV, S.R.O. Express. Online. 2024. Dostupné z: <https://www.expert-dev.cz/slovnicek-pojmu/express/>. [cit. 2025-04-13].
- [10] MEDIUM. Understanding Apollo Server and Apollo Client: A Comprehensive Guide. Online. 2024. Dostupné z: <https://techdynasty.medium.com/understanding-apollo-server-and-apollo-client-a-comprehensive-guide-b98dd4630f0c>. [cit. 2025-04-13].
- [11] NODE-POSTGRES. Node-postgres. Online. 2025. Dostupné z: <https://node-postgres.com>. [cit. 2025-04-13].
- [12] REVELO. Front End Frameworks: What They Are, and Best Options. Online. 2025. Dostupné z: <https://www.revelo.com/blog/front-end-frameworks>. [cit. 2025-04-13]

- [13] FLATLOGIC, LLC. What is Material UI? Online. 2022. Dostupné z: <https://flatlogic.com/blog/what-is-material-ui/>. [cit. 2025-04-13].
- [14] LOGROCKET, INC. Password hashing in Node.js with bcrypt. Online. 2024. Dostupné z: <https://blog.logrocket.com/password-hashing-node-js-bcrypt/>. [cit. 2025-04-19].
- [15] NOVE-HESLO.CZ. Slovník pojmů – Bcrypt. Online. 2025. Dostupné z: <https://www.nove-heslo.cz/slovník/bcrypt/>. [cit. 2025-04-19].
- [16] O2 CYBERNEWS. Kybernetická bezpečnost. Online. © 2025. Dostupné z: <https://o2cybernews.cz/slovník/kyberneticka-bezpecnost>. [cit. 2025-04-19].
- [17] DESCOPE INC. What Is a JWT & How It Works. Online. 2024. Dostupné z: <https://www.descope.com/learn/post/jwt>. [cit. 2025-04-19].
- [18] OKTA, INC. JSON Web Tokens. Online. © 2025. Dostupné z: <https://auth0.com/docs/secure/tokens/json-web-tokens#benefits-of-jwts>. [cit. 2025-04-19].
- [19] SKILLMEA DIGITAL S.R.O. Co je testování softwaru? Online. 2021. Dostupné z: <https://skillmea.cz/blog/co-je-testovanie-softveru>. [cit. 2025-04-20].
- [20] FREE CODE CAMP, INC. The GraphQL API Handbook – How to Build, Test, Consume and Document GraphQL APIs. Online. 2023. Dostupné z: <https://www.freecodecamp.org/news/building-consuming-and-documenting-a-graphql-api/>. [cit. 2025-04-19].
- [21] MICROSOFT. Přehled Microsoft 365 Lighthouse (pro partnery). Online. 2024. Dostupné z: <https://learn.microsoft.com/cs-cz/microsoft-365/lighthouse/m365-lighthouse-overview?view=o365-worldwide>. [cit. 2025-04-19].
- [22] I60 PUBLISHERS, S.R.O. Open-source vs. uzavřený, komerční software: Jaké jsou výhody a nevýhody? Online. 2023. Dostupné z: <https://www.i60.cz/clanek/detail/32786/open-source-vs-uzavreny-komerčni-software-jake-jsou-vyhody-a-nevyhody>. [cit. 2025-04-19].
- [23] MICROSOFT. Racktables. Online. © 2025. Dostupné z: <https://azuremarketplace.microsoft.com/en-us/marketplace/apps/bcloudllc1671615348068.racktables1?tab=overview>. [cit. 2025-04-19].

- [24] GITHUB, INC. NetBox The cornerstone of every automated network. Online. © 2025. Dostupné z: <https://github.com/netbox-community/netbox?tab=readme-ov-file>. [cit. 2025-04-19].
- [25] FRESHWORKS INC. What is Device42? Online. © 2025. Dostupné z: <https://docs.device42.com/what-is-device42/>. [cit. 2025-04-19].
- [26] ALVAO S.R.O. Inventura, nejen IT majetku, nemusí být úmorná. Online. 2020. Dostupné z: <https://www.alvao.com/cs/blog/inventura-nejen-it-majetku-nemusi-byt-umorna>. [cit. 2025-04-20].
- [27] NINJAONE LLC. IT Asset Inventory: Definition, Management, & Best Practices. Online. 2024. Dostupné z: <https://www.ninjaone.com/blog/how-to-do-an-it-asset-inventory/>. [cit. 2025-04-20].
- [28] SOFTINVENTIVE LAB. Jak skenovat síťová zařízení. Online. © 2006-2025. Dostupné z: <https://www.softinventive.com/cz/how-to-scan-network-devices>. [cit. 2025-04-20].
- [29] ALVAO S.R.O. Jaký je rozdíl mezi ITAM a CMDB? Online. 2024. Dostupné z: <https://www.alvao.com/cs/blog/jaky-je-rozdil-mezi-itam-cmdb>. [cit. 2025-04-20].
- [30] AUVIK NETWORKS INC. Network Diagrams 101: Definitions, Examples & How-To's. Online. 2024. Dostupné z: <https://www.auvik.com/franklyit/blog/effective-network-diagrams/>. [cit. 2025-04-20].
- [31] AUVIK NETWORKS INC. What is Network Visualization? Online. 2024. Dostupné z: <https://www.auvik.com/franklyit/blog/what-is-network-visualization/>. [cit. 2025-04-20].
- [32] CAMBRIDGE INTELLIGENCE. Network visualization: A detailed guide. Online. © 2025. Dostupné z: <https://cambridge-intelligence.com/why-visualize-networks/>. [cit. 2025-04-20].
- [33] FUTURE US INC. Best IT infrastructure management service of 2025. Online. 2024. Dostupné z: <https://www.techradar.com/best/best-infrastructure-management-service>. [cit. 2025-04-20].
- [34] SOLARWINDS WORLDWIDE, LLC. What Is Network Visualization? Online. © 2025. Dostupné z: <https://www.solarwinds.com/resources/it-glossary/network-visualization>. [cit. 2025-04-20].

- [35] MEDIUM. Choosing the Right Technology for Your IT Projects. Online. 2023. Dostupné z: <https://medium.com/@sunmait/choosing-the-right-technology-for-your-it-projects-213feedcfe5c>. [cit. 2025-04-20].
- [36] POKORNÝ, Jaroslav a VALENTA, Michal. Databázové systémy. 2. přepracované vydání. Vysokoškolská učebnice. Praha: Česká technika - nakladatelství ČVUT, 2020. ISBN 978-80-01-06696-6.
- [37] TANENBAUM, Andrew S. a WETHERALL, D. Computer networks. Fifth edition. New Delhi: Dorling Kindersley, [2014]. ISBN 9789332518742.
- [38] KUROSE, James F. a ROSS, Keith W. Počítačové sítě. Brno: Computer Press, 2014. ISBN 9788025138250.
- [39] TRULOVE, James. Sítě LAN: hardware, instalace a zapojení. Profesionál. Praha: Grada, 2009. ISBN 9788024720982.
- [40] HELALI, Saida. Systems and network infrastructure integration: design, implementation, safety and supervision. London: ISTE, 2020. ISBN 9781119779964. Dostupné také z: <https://proxy.k.utb.cz/login?url=https://onlinelibrary.wiley.com/doi/book/10.1002/9781119779964>

SEZNAM OBRÁZKŮ

Obrázek 1 - Use case model	46
Obrázek 2 - Databázový model	54
Obrázek 3 - Struktura Projektu	57
Obrázek 4 - Vizualizace – dagre	58
Obrázek 5 - Vizualizace – ELKJS	58
Obrázek 6 - Vizualizace – FORCE	59
Obrázek 7 - Správa zařízení	60
Obrázek 8 - Modální okno přidat zařízení	60
Obrázek 9 - Správa uživatelů	61
Obrázek 10 - Modální okno Přidat uživatele	61
Obrázek 11 - Modální okno Změnit heslo	62
Obrázek 12 - Inventář zařízení	62
Obrázek 13 - Přidání položky do inventáře	63
Obrázek 14 - Provedení inventury	63
Obrázek 15 - Správa racků	64
Obrázek 16 - Přidat nový rack	64
Obrázek 17 – Statistiky	65
Obrázek 18 – Uživatelské menu s iniciálou	65
Obrázek 19 – Dialog pro změnu hesla	66
Obrázek 20 - Přihlašovací obrazovka	66
Obrázek 21 – Wallboard	67
Obrázek 22 - Docker Compose pro postgres	69
Obrázek 23 - DB.js	70
Obrázek 24 - db__init.sql	71
Obrázek 25 – ukázka schema.js	72
Obrázek 26 - Ukázka resolvers.js	73
Obrázek 27 – DOCKERFILE	75
Obrázek 28 - App.js const httpLink a authLink	76
Obrázek 29 - App.js Apollo klient	76
Obrázek 30 - GraphSwitcher.js	78
Obrázek 31 - Ukázka kódu NetworkGraph.js	81
Obrázek 32 - Ukázka kódu NetworkGraph3.js	82
Obrázek 33 - Funkce pro export	83
Obrázek 34 – Ukázka komponenty login.js	84

Obrázek 35 - Ukázka kódu UserManagement.js	85
Obrázek 36 - Proces JWT	86
Obrázek 37 - Výsledek testu ze stránky securityheaders.com	87
Obrázek 38 - Výsledky testů ze stránky sslabs.com	88
Obrázek 39 - Testování v Apollo GraphQL Sandboxu.....	91

SEZNAM TABULEK

Tabulka 1 - Scénář případů užití UC001	47
Tabulka 2 - Scénář případů užití UC002	47
Tabulka 3 - Scénář případů užití UC003	48
Tabulka 4- Scénář případů užití UC004	48
Tabulka 5- Scénář případů užití UC005	49
Tabulka 6- Scénář případů užití UC006	49
Tabulka 7- Scénář případů užití UC007	50
Tabulka 8- Scénář případů užití UC008	50
Tabulka 9- Scénář případů užití UC009	51
Tabulka 10- Scénář případů užití UC010	51
Tabulka 11 - Scénář případů užití UC011	52
Tabulka 12- Scénář případů užití UC012	52
Tabulka 13- Scénář případů užití UC013	53
Tabulka 14 - Funkční testy	89
Tabulka 15 - Výkonové testy	91

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

IT	Information Technology
JWT	JSON Web Token
CSV	Comma-Separated Values
LAN	Local Area Network
MUI	Material UI
WoL	Wake-on-LAN
API	Application Programming Interface
WMI	Windows Management Instrumentation
SNMP	Simple Network Management Protocol
SSH	Secure Shell
CMDB	Configuration Management Database
REST	Representational State Transfer)
AI	Artificial Intelligence
RTT	Round Trip Time
VoIP	Voice over IP
WAN	Wide Area Network
JSON	JavaScript Object Notation
ACID	Atomicity, Consistency, Isolation, Durability
SVG	Scalable Vector Graphics
CSS	Cascading Style Sheets
DOM	Document Object Model
HTML	HyperText Markup Language
URL	Uniform Resource Locator
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure

GDPR	General Data Protection Regulation
ISO	International Organization for Standardization
IEC	International Electrotechnical Commission
MD5	Message-Digest Algorithm 5
SHA	Secure Hash Algorithm
XML	eXtensible Markup Language
HMAC	Hash-Based Message Authentication Code
CSP	Content Security Policy
MSP	Managed Service Provider
SMB	Server Message Block
MAC	Media Access Control address
SQL	Structured Query Language
XSS	Cross-Site Scripting
MIME	Multipurpose Internet Mail Extensions
SEO	Search Engine Optimization

SEZNAM PŘÍLOH

Příloha P I: Zdrojový kód