ZÁPADOČESKÁ UNIVERZITA V PLZNI
FAKULTA ELEKTROTECHNICKÁ
KATEDRA ELEKTRONIKY A INFORMAČNÍCH TECHNOLOGIÍ

DIPLOMOVÁ PRÁCE

Systém pro správu a administraci IoT Wi-Fi

Autor práce: **Bc. Kryštof Trkovský**
Vedoucí práce: **Ing. Petr Weissar, Ph.D.**

2024

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta elektrotechnická
Akademický rok: 2023/2024

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Kryštof TRKOVSKÝ**
Osobní číslo:	**E21N0045P**
Studijní program:	**N0714A060013 Elektronika a informační technologie**
Specializace:	**Elektronika**
Téma práce:	**Systém pro správu a administraci IoT Wi-Fi**
Zadávající katedra:	**Katedra elektroniky a informačních technologií**

Zásady pro vypracování

1. Zmapujte prostředí Wi-Fi sítí na ZČU a možnosti připojování IoT zařízení.
2. Navrhněte systém pro správu registrací a administraci připojování IoT zařízení do Wi-Fi sítě.
3. Systém realizujte, zohledněte požadavek snadné správy a údržby systému a bezpečnostní hlediska.

Rozsah diplomové práce: **40-60**
Rozsah grafických prací:
Forma zpracování diplomové práce: **elektronická**

Seznam doporučené literatury:

1. Identity PSK Manager for Cisco ISE – https://github.com/CiscoDevNet/iPSK-Manager
2. Identity PSK Feature Deployment Guide – https://www.cisco.com/c/en/us/td/docs/wireless/controller/technotes/8-5/b_Identity_PSK_Feature_Deployment_Guide.html
3. Dokumentace možných technologií pro realizaci:

- https://symfony.com/
- https://getbootstrap.com/
- https://jquery.com/

Vedoucí diplomové práce: **Ing. Petr Weissar, Ph.D.**
Katedra elektroniky a informačních technologií

Datum zadání diplomové práce: **6. října 2023**
Termín odevzdání diplomové práce: **24. května 2024**

Západočeská univerzita v Plzni
Fakulta elektrotechnická

L.S.

Prof. Ing. Zdeněk Peroutka, Ph.D.
děkan

Doc. Ing. Jiří Hammerbauer, Ph.D.
vedoucí katedry

V Plzni dne 6. října 2023

Abstrakt

V práci je řešen návrh a implementace systému pro správu a administraci připojování IoT zařízení do Wi-Fi sítě. Internet věcí je perspektivní oblast, ve které bylo vyvinuto mnoho komunikačních protokolů. Tato zařízení je potřeba provozovat v existujících infrastrukturách Wi-Fi sítí, což ale přináší výzvy v oblasti zabezpečení. Proto je potřebný systém, který umožní nastavit pravidla komunikace v této síti a kde budou zařízení přehledně zobrazena a přidělena uživatelům. Je dostupné nepřeberné množství technologií a způsobů realizace. Vždy je nutné zvážit přínosy a nevýhody jednotlivých řešení. Navrhovaný systém byl realizován jako jednostránková aplikace pro webové prohlížeče a mobilní zařízení s OS Android v technologiích PHP, Symfony, TypeScript, Vue a Ionic. Cílem bude zajistit bezpečnou komunikaci IoT zařízení.

Klíčová slova

webová SPA, mobilní aplikace, Vue, Symfony, Wi-Fi, internet věcí, GraphQL

Abstract

The thesis is focused on the design and implementation of a system for the management and administration of IoT devices connecting to a Wi-Fi network. IoT is a promising area in which many communication protocols have been developed. These devices need to operate in existing Wi-Fi network infrastructures, but this brings security challenges. Therefore, a system is needed to set up the rules for communication in this network and where devices are displayed and assigned to users. A variety of technologies and implementation methods are available. There are always benefits and drawbacks to be weighed. The proposed system has been implemented as a single-page application for web browsers and mobile devices running Android OS in technologies PHP, Symfony, TypeScript, Vue and Ionic. This will enable secure communication of IoT devices.

Keywords

web SPA, mobile app, Vue, Symfony, Wi-Fi, Internet of things, GraphQL

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně, s použitím odborné literatury a pramenů uvedených v seznamu, který je součástí této diplomové práce.

Dále prohlašuji, že veškerý software, použitý při řešení této diplomové práce, je legální.

V Plzni dne 24. května 2024

Bc. Kryštof Trkovský

. . . . . .

Podpis

Obsah

Seznam obrázků

Seznam tabulek

Seznam symbolů a zkratek

IoT	Internet Of Things - internet věcí
TLS	Transport Layer Security - zabezpečená přenosová vrstva
DoS	Denial of Service - odepření služby
MITM	Man-in-the-Middle - muž uprostřed
RFID	Radio Frequency Identification - identifikace na rádiové frekvenci
BLE	Bluetooth Low Energy - nízkoenergetický bluetooth
LoRa	Long range - dlouhý dosah
MQTT	MQ Telemetry Transport - MQ telemetrický přenos
DDS	Data Distribution Service - služba distribuce dat
AMQP	Advanced Message Queuing Protocol - pokročilý protokol front zpráv
SMTP	Simple Mail Transfer Protocol - jednoduchý protokol pro přenos mailů
Wi-Fi	Wireless-Fidelity - bezdrátová věrnost
IEEE	Institute of Electrical and Electronics Engineers - institut elektrotechnických a elektronických inženýrů
MU-MIMO	Multi User-Multiple Input Multiple Output - více uživatelů - více vstupů a více výstupů
WEP	Wired Equivalent Privacy - soukromí ekvivalentní drátovým sítím
RC4	Rivest Cipher 4 - Rivestova šifra 4
WPA	Wi-Fi Protected Access - chráněný přístup k Wi-Fi
AES	Advanced Encryption Standard - standard pokročilého šifrování
TKIP	Temporal Key Integrity Protocol - protokol časové integrity klíče
CCMP	Counter Cipher Mode with Block Chaining Message Authentication Code Protocol - režim čítačové šifry s blokovým řetězením protokolu pro ověřování pravosti zpráv
RADIUS	Remote Authentication Dial-In User Service - uživatelská vytáčená služba pro vzdálenou autentizaci
PSK	Pre-shared key - předsdílený klíč
WPS	Wi-Fi Protected Setup - nastavení chráněné Wi-Fi
SoC	System on a chip - systém na čipu

GPIO	General-purpose input/output - univerzální vstupní/výstupní pin
I2C	Inter-Integrated Circuit - interintegrovaný obvod
I2S	Inter-IC Sound - Zvuk Inter-IC
SPI	Serial Peripheral Interface - sériové periferní rozhraní
UART	Universal asynchronous receiver-transmitter - univerzální asynchronní přijímač-vysílač
ROM	Read-Only Memory - paměť pouze ke čtení
SRAM	Static Random Access Memory - statická paměť
RISC	Reduced Instruction Set Computer - počítač s redukovanou instrukční sadou
iPSK	Identity PSK - identita PSK
AAA	Authentication, authorization and accounting - ověřování, autorizace a účtování
DHCP	Dynamic Host Configuration Protocol - protokol dynamické konfigurace hostitele
NAT	Network Address Translation - překlad síťových adres
DNS	Domain Name System - systém doménových jmen
PAT	Port address translation - překlad adres portů
OS	Operating system - operační systém
SPA	Single Page Application - jednostránková aplikace
HTML	Hypertext Markup Language - značkovací jazyk pro tvorbu stránek s hypertextovými odkazy
JS	JavaScript
CSS	Cascading Style Sheets - kaskádové styly
MPA	Multi Page Application - vícestránková aplikace
PWA	Progressive Web Application - progresivní webová aplikace
CMS	Content Management Systems - systém pro správu obsahu
RIA	Rich Internet Applications - bohatá internetová aplikace
SEO	Search engine optimization - optimalizace pro vyhledávače
PHP	PHP: Hypertext Preprocessor - PHP: hypertextový preprocesor
ASP	Active Server Pages - aktivní serverové stránky
JSP	JavaServer Pages - Stránky JavaServer
API	Application Programming Interface - rozhraní pro programování aplikací
ECMA	European Computer Manufacturers Association - evropská asociace výrobců počítačů
JIT	Just in time - právě v čas
OOP	Object oriented programming - objektově orientované programování
GUI	Graphical user interface - grafické uživatelské rozhraní
NPM	Node Package Manager - systém pro zprávu balíčků Node.js

REST	Representational State Transfer - přenos reprezentačního stavu
ORM	Object Relational Mapping - objektově relační mapování
DRY	Don’t Repeat Yourself - neopakuj se
KISS	Keep it simple, Stupid! - Zachovejte jednoduchost, hlupáku!
SOLID	Single responsibility, Open-closed, Liskov substitution, Interface segregation, Dependency inversion - jednoduchá odpovědnost, Otevřeno-uzavřeno, Liskova substituce, Segregace rozhraní, Inverze závislosti
MVC	Model View Controller
JWT	JSON Web Token - JSON webový token
CLI	Command Line Interface - rozhraní příkazového řádku
DOM	Document Object Model - objektový model dokumentu
UI	User Interface - uživatelské rozhraní
JSON	JavaScript Object Notation - javaScriptový objektový zápis
XML	eXtensible Markup Language - rozšiřitelný značkovací jazyk
HTTP	Hypertext Transfer Protocol - protokol pro přenos hypertextových dokumentů
HTTPS	Hypertext Transfer Protocol Secure - zabezpečený protokol pro přenos hypertextových dokumentů
SOAP	Simple Object Access Protocol - Jednoduchý protokol přístupu k objektům
RPC	Remote Procedure Call - vzdálené volání procedur
TCP	Transmission Control Protocol - protokol pro řízení přenosu
IP	Internet Protocol - internetový protokol
PSR	PHP Standards Recommendations - doporučení k normám PHP
EER	Enhanced entity-relationship - vylepšený model vztahů mezi entitami
CORS	Cross-origin resource sharing - sdílené zdroje odjinud
IDE	Integrated Development Environment - vývojové prostředí
TS	TypeScript
ES	ECMA Script

1

Úvod

Zařízení pro internet věcí jsou obvykle používána ke sběru dat nebo umožňují vzdálené nastavení své funkce. Pro dosažení svého účelu potřebují další prvky, které budou schopné dekódovat a uložit tato sesbíraná data. Tato data je vždy potřeba nějakým způsobem přenášet, obvykle obousměrně. Pro tento účel bylo vyvinuto více druhů sítí. Všechny se však potýkají s náklady na vybudování a následnou správou této sítě, proto je výhodné využít již existující infrastrukturu, například Wi-Fi sítě.

V případě organizace, jako je Západočeská univerzita, je zcela nezbytné dodržovat nejvyšší úroveň bezpečnosti počítačových sítí. Proto není možné dovolit uživatelům připojovat libovolná zařízení nebo vytvářet vlastní podsítě. IoT zařízení nemusí vždy podporovat požadované protokoly zabezpečení nebo je jejich implementace příliš náročná. Proto je potřeba vytvořit systém, který umožní nastavit zařízením pravidla komunikace a zároveň umožní řízení jejich připojování do sítě. Zároveň musí být kladen důraz na bezpečnost. Například nelze vytvořit jednu síť s jedním heslem, do které se připojí všechna zařízení.

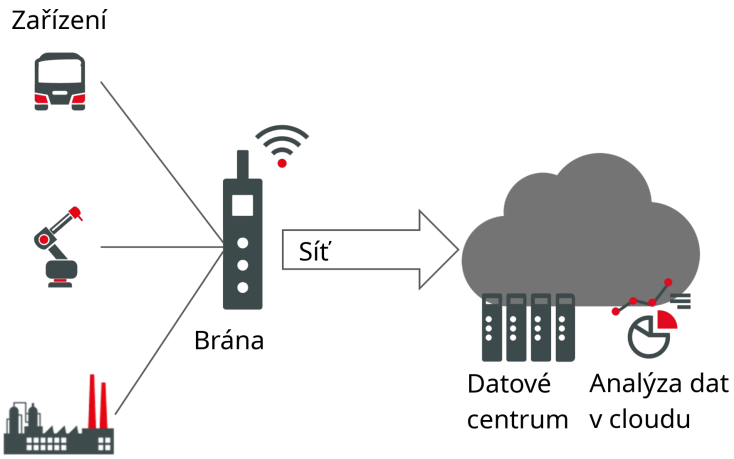
Práce se zabývá návrhem systému pro správu registrací a administraci připojování zařízení internetu věcí do Wi-Fi sítě. Je potřeba zvolit vhodný druh aplikace a vhodné technologie pro realizaci. Na základě požadavků danými IoT zařízeními a Wi-Fi sítěmi. Důležitou součástí je výběr vývojových nástrojů a metodiky vývoje (DevOps) a nakonec i vhodných vývojových a provozních prostředí. Tyto požadavky na aplikaci jsou velmi specifické a nebyla nalezena vhodná alternativa v prostředí aplikací s otevřeným kódem.

2

Internet věcí

Internet věcí je definován Mezinárodní telekomunikační unií jako globální infrastruktura pro informační společnost, která umožňuje poskytovat pokročilé služby. Toho lze docílit propojením zařízení na základě stávajících a rozvíjejících se informačních a komunikačních technologií. [1]

Obecně je tedy síť internetu věcí složena z fyzických zařízení, která si mezi sebou mohou předávat informace. Mezi tato zařízení může patřit jakákoliv věc opatřená elektronikou, softwarem a konektivitou do sítě. Obvykle bude obsahovat senzory nebo aktuátory, které realizují samotnou funkci zařízení. Jedná se například o domácí spotřebiče, vozidla, osobní elektroniku nebo průmyslové stroje. Dále budou v síti přítomny prvky pro analýzu, zpracování a prezentaci posbíraných dat.



Obrázek 2.1: Schéma jednoduché IoT sítě. Převzato z [2].

2.1 Využití internetu věcí

Dnes jsou tato zařízení rozšířena hlavně v domácnostech, podnicích, školách a průmyslu. [3]

Spotřební elektronika zahrnuje chytrá zařízení, například televize, žárovky nebo domácí asistenty. Díky nim je možné integrovat do domácnosti funkce chytré domácnosti, jako je nastavení automatického zhasínání světel, pokud uživatel opustí dům nebo automatické zalévání trávníku podle předpovědi počasí.

V podnicích je možné implementovat řešení jako chytré senzory, RFID tagy, chytrou energetiku a tím šetřit provozní náklady.

V oblasti průmyslu se vyskytují zařízení pro monitorování výrobních linek a procesů. Díky tomu je možné jednodušeji diagnostikovat problémy a některým dokonce i předcházet. [3] Tímto je minimalizována doba případného výpadku výroby. Zařízení v této oblasti mají speciální požadavky na odolnost a zabezpečení.

2.2 Přenos dat

Při návrhu připojení do sítě je potřeba zvážit parametry jako spotřeba energie nebo dosah pokud bude zařízení napájeno z baterií a vzdáleno od infrastruktury [3].

IoT sítě mohou využívat řadu protokolů podle požadovaných vlastností. Bluetooth má krátký dosah, používá se například pro přenos audia. BLE je optimalizovaný pro nižší spotřebu energie, používá se například pro fitness náramky. Mobilní sítě LTE nebo 5G jsou celoplošně dostupné, ale pojí se s nimi nutná periodická platba operátorovi sítě. LoRa a LoRaWAN jsou bezdrátové technologie umožňující komunikaci na velkou vzdálenost, ale menší objemy dat. Wi-Fi umožňuje rychlý přenos velkého objemu dat, ale vyžaduje vyšší spotřebu energie ve srovnání s technologií LoRa. Ethernet je omezen nutným připojením vodičem. Zigbee je mesh protokol, nejčastěji používaný pro chytrou domácnost, má kratší dosah než LoRa, ale delší než BLE. Zároveň má nižší energetické nároky než Wi-Fi. [4]

Tyto protokoly jsou obvykle kombinovány s dalšími vrstvami jako MQTT, AMQP, DDS [4].

Vzhledem k tomu, že na univerzitě je vybudována kvalitní síť Wi-Fi a IoT zařízení, která se na ni mohou připojit, jsou dostupná a vhodná pro účely výuky. Její využití je v tomto případě optimální.

2.3 Bezpečnostní rizika

Útoky na síťovou infrastrukturu se mohou dělit do tří hlavních kategorií. První je odepření služby (DoS), dále manipulace s daty (data manipulation) a nakonec zpřístupnění údajů (data disclosure). [5]

DoS útok spočívá v zahlcení zařízení komunikací, která nemá žádný užitek. Tento útok obvykle končí pádem zařízení. Cílem útoku je znepřístupnit službu a tím způsobit škodu uživatelům. [5] Do sítě IoT mohou být připojena zařízení s omezeným výkonem z důvodu větší výdrže na baterie nebo nižší výrobní ceny. Pokud je takové zařízení vystaveno v nezabezpečené síti, není problém tento útok realizovat i s nevelkými prostředky.

Útok manipulace s daty spočívá ve změně přenášených dat ze zařízení, tak aby byl splněn cíl útočníka. Pokud například brána nevyžaduje ověření uživatele před započetím komunikace je možné, že útočník bude moci přistupovat k dalším zařízením v síti až za koncovým zařízením. Tento útok se nazývá MITM - muž uprostřed.[5]

Zpřístupněním údajů je myšlen útok, který umožní útočníkovi bez autorizace získat přístup k citlivým datům, jako jsou osobní údaje, hesla a další. [5] Připojená zařízení do sítě nemusí vždy disponovat možnostmi pro využití zabezpečeného přenosu, například přes TLS. Často není možné aktualizovat software, ať už z důvodu chybějící podpory výrobce nebo technické náročnosti.

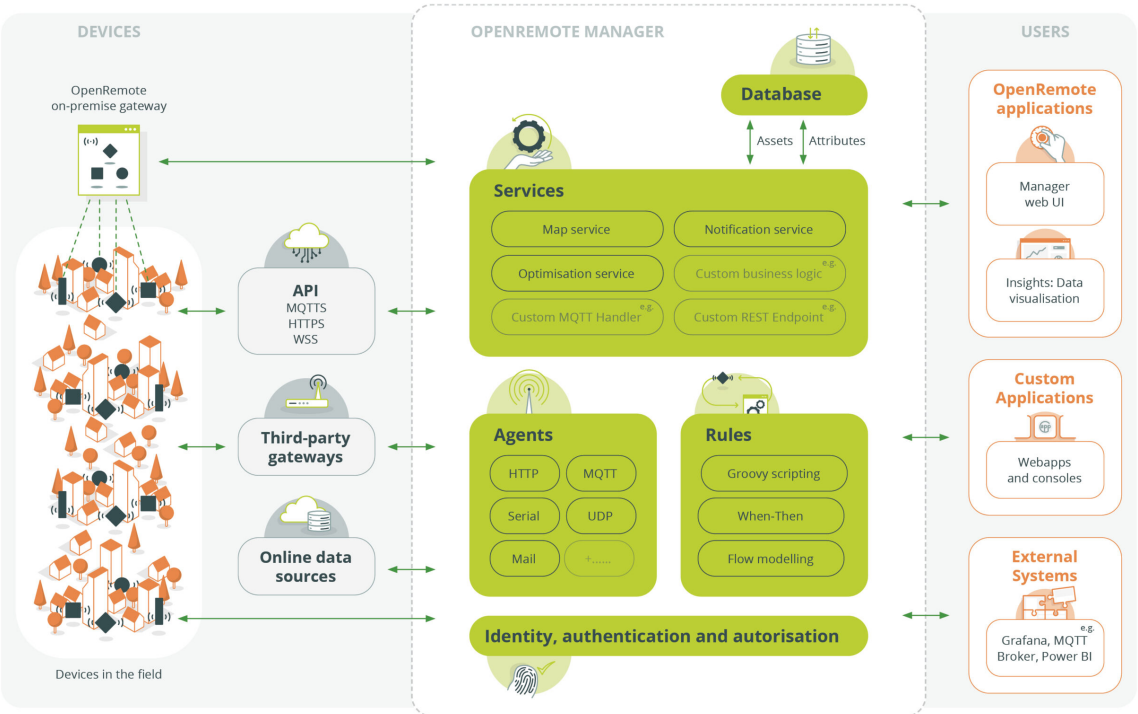
Z těchto důvodů je nutné IoT sítě navrhovat tak, aby k zmiňovaným útokům nemohlo docházet z podstaty fungování sítě.

2.4 Dostupné systémy pro správu IoT

Pro správu zařízení v síti LoRaWAN existuje projekt ChirpStack. Systém poskytuje webové rozhraní, prostřednictvím kterého lze spravovat síťové prvky, které komunikují pomocí protokolu MQTT. [6]

Pro vizualizaci a zpracování dat je k dispozici projekt ThingsBoard. Podporuje širokou škálu protokolů a správu zařízení. Umožňuje vytváření grafů a dalších prvků pro sledování nasbíraných dat. Pomocí řetězů pravidel (rule chains) je možné implementovat logiku, jako například upozornění v případě nadměrné teploty nebo výpadku přenosu dat.[7]

Obdobný projekt s názvem OpenRemote je zaměřen na implementaci IoT ve městech. Dokáže přehledně zobrazit údaje z měřených bodů na mapě a umožňuje nastavovat pravidla a podmínky vázané na přijatá data. Po zpracování pravidel a podmínek je vykonána definovaná akce.[8]



Obrázek 2.2: Struktura systému OpenRemote. Převzato z [8].

Jak je vidět na obrázku 2.2 systémy pro sběr a vizualizaci IoT dat se obvykle skládají ze zařízení, která jsou připojena různými protokoly ke službám (agents). Ty zajišťují vyčítání dat. Data jsou předávána a ukládána do databáze a probíhá nad nimi analýza nastavených notifikací a pravidel. Uživatelé mají přístup na webové rozhraní, které umožňuje zpracovat data do přehledné formy. Tyto systémy bývají napojeny na další služby, jako je odesílání mailů přes SMTP nebo SMS bránu.

3

Síť Wi-Fi

Wi-Fi je rodina bezdrátových protokolů, která svým vznikem umožnila bezdrátové připojení k internetu mnoha přenosným zařízením. Díky telefonům, notebookům, tabletům a potřebě neustálého přístupu k internetu se Wi-Fi stala integrální součástí života většiny lidí ve vyspělých zemích. Tyto sítě jsou dostupné ve většině domácností, firmách, školách i na veřejných prostranstvích. Skládají se z přístupového bodu a klientů, kteří se na něj připojují.

Wi-Fi využívá mikrovlnné rozsahy $2{,}4$ GHz, 5 GHz nebo nejmodernější zařízení 6 GHz. Za účelem zvýšení rychlosti přenosu dat lze zvětšit šířku kanálu v pásmu $2{,}4$ GHz z 20 MHz na 40 MHz a v pásmu 5 GHz z 20 MHz na 160 MHz. [9]

3.1 Historie, pojmenování, vlastnosti

Wi-Fi je definována standardem IEEE 802.11. Průběžně vzniklo značné množství verzí. V následující tabulce je shrnuto označení současně používaných generací Wi-Fi podle standardu, jak jej definovala Wi-Fi Alliance.

Technologie	Generace	Frekvence
IEEE 802.11be	Wi-Fi 7	$2{,}4$ GHz, 5 GHz, 6 GHz
IEEE 802.11ax	Wi-Fi 6	$2{,}4$ GHz, 5 GHz, 6 GHz
IEEE 802.11ac	Wi-Fi 5	$2{,}4$ GHz, 5 GHz
IEEE 802.11n	Wi-Fi 4	$2{,}4$ GHz, 5 GHz

Tabulka 3.1: Přehled označení technologií a generací Wi-Fi z [10]

Původní standard IEEE 802.11 byl vytvořen již v roce 1997. Podporoval maximální rychlost až 54 Mbit s^{-1} a mohl být provozován na frekvenci $2{,}4$ GHz. IEEE 802.11a následoval v roce 1999. Podporoval 54 Mbit s^{-1}, kvůli nižšímu vzájemnému ovlivňování fungoval na frekvenci 5 GHz. Ve stejném roce vznikl IEEE 802.11b s maximální rychlostí 11 Mbit s^{-1} na $2{,}4$ GHz. V roce 2003 pokračoval IEEE 802.11g s maximální rychlostí 54 Mbit s^{-1} na $2{,}4$ GHz. [11]

Následující standardy už všechny podporovaly pásma $2{,}4\,\text{GHz}$ i $5\,\text{GHz}$. S IEEE 802.11n (Wi-Fi 4) v roce 2009 přišlo rozšíření rychlosti na $600\,\text{Mbit}\,\text{s}^{-1}$ a s IEEE 802.11ac (Wi-Fi 5) v roce 2014 až $1300\,\text{Mbit}\,\text{s}^{-1}$. V současné době se jedná o nejpoužívanější protokol. [11]

Od standardu Wi-Fi 5 je podporována technologie MU-MIMO, která umožňuje využití více antén. S jejich pomocí lze přenášet data na více klientských zařízení najednou. Tím se zlepší efektivita přenosu a klienti podporující jen pomalejší přenosy neruší rychlejší provoz. [12]

V následujících standardech přibyla podpora frekvence $6\,\text{GHz}$. Nyní probíhá přechod na IEEE 802.11ax (Wi-Fi 6), který vznikl v roce 2019 s teoretickou propustností $9{,}6\,\text{Gbit}\,\text{s}^{-1}$. Do prodeje nastupují zařízení podporující dokonce IEEE 802.11be (Wi-Fi 7) s maximální propustností $40\,\text{Gbit}\,\text{s}^{-1}$. [11]

Toto jsou důležité milníky vývoje, ve skutečnosti standardů existuje mnohem více.

Je nutné zmínit, že maximální reálná rychlost v době rozšíření standardu nemusela být dosahována většinou zařízení, které standard podporovala. Záleží totiž na konkrétní hardwarové konfiguraci přístupových bodů a klientů.

Jaké standardy komerční zařízení podporují, by mělo být zřejmé z vizuálního značení, které doporučuje Wi-Fi Alliance.

Obrázek 3.1: Doporučené vizuální značení Wi-Fi. Převzato z [10].

3.2 Zabezpečení

Vzhledem k bezdrátové povaze sítě je snadné odposlouchávat nebo pozměnit veškerý datový provoz. Wi-Fi tak za cenu snížení rychlosti přenosu dat poskytuje možnosti ověřování uživatelů a šifrování komunikace.

Prvním zabezpečením byl WEP představený s IEEE 802.11b. Jeho použití je však dnes velmi nevhodné, protože je snadno prolomitelný kvůli zranitelnosti algoritmu distribuce klíče RC4. Prolomení probíhá odposlechnutím dostatečného počtu rámců - inicializačních vektorů. [9]

Jako náhradu za WEP navrhla Wi-Fi Alliance WPA. Používá šifrování TKIP (Temporal Key Integrity Protocol) a kontrolní součet MIC (Message Integrity Code) ke kontrole integrity rámců. TKIP využívá funkci mixování klíčů a kombinuje kořenový klíč s inicializačním vektorem předtím, než ho předá algoritmu RC4. WEP pouze spojil inicializační vektor s kořenovým klíčem a výsledek předal algoritmu RC4 [9]. Později byl klíč TKIP nahrazen za AES [13].

Toto zabezpečení se opět ukázalo jako nedostatečné, proto byl vydán protokol WPA2. Hlavní rozdíl oproti WPA je nutnost použití delšího klíče AES a CCMP, které nahradily TKIP [13].

Zatím poslední verzí zabezpečení Wi-Fi je WPA3. Opravuje zranitelnosti WPA2. Zvyšuje délku klíče AES a přidává další vlastnosti, které poskytují vyšší kryptografickou sílu a umožňují silnější ověřování. [9]

WPA1, 2 nebo 3 mohou být provozovány ve 2 módech. Osobní (personal) a firemní (Enterprise).

Osobní mód, také zvaný WPA-PSK (pre-shared key), poskytuje jedno heslo, kterým se uživatelé přihlašují do sítě [9]. Toto heslo se v případě kompromitace musí změnit pro všechny uživatele. Použití tohoto módu je v domácnostech nebo malých sítích. Konfigurace je nenáročná, a proto se s ním lze setkat častěji.

Firemní mód je určený pro větší sítě. Umožňuje uživatelům používat vlastní přístupové údaje, které jsou uloženy na RADIUS serveru [9]. Pokud by došlo k úniku hesla u jednoho uživatele, je možné jeho údaje zneplatnit bez ovlivnění dalších uživatelů.

Další možností pro autorizaci je funkce WPS. Po stisknutí tlačítka na přístupovém bodu se funkce aktivuje a klienta je možné připojit i bez zadání často dlouhého PSK. S povolením této funkce, ale přicházejí další rizika. Kdokoliv, kdo se fyzicky dostane k zařízení, má možnost se k němu připojit. U starších přístupových bodů šlo prolomit přístup zkoušením PINu WPS. [14]

V některých, obvykle firemních nebo univerzitních (eduroam) sítích je využíván pro autentizaci uživatele protokol IEEE 802.1X. Na rozdíl od předchozích protokolů je navržen pro využití i na drátových sítích. Po připojení zařízení drátem do sítě jsou přepínačem povoleny pouze autentizační rámce. Ověření proběhne přes autentizační server RADIUS. Po úspěšném ověření přepínač odblokuje port pro komunikaci. Poté může proběhnout vyjednávání o adresách s DHCP serverem. [15]

4

Návrh systému

Analýza ukázala, že většina otevřených projektů se zaměřuje spíše na prezentaci a sběr dat. Požadavky na systém pro správu Wi-Fi IoT v prostředí univerzity jsou velmi specifické a dostupné varianty s otevřeným kódem nevyhovují.

4.1 Podpora Wi-Fi u IoT zařízení

První z kategorií zařízení připojených do Wi-Fi IoT sítě budou projekty pro účely výuky na univerzitě nebo zařízení vyvinutá na univerzitě. Může se jednat o mikroprocesory s integrovanou Wi-Fi konektivitou. Obvykle se budou vyznačovat malým výpočetním výkonem a omezenými možnostmi připojení. Dále je nutné počítat i se staršími zařízeními, která už jsou přítomná a bude je potřeba připojit do sítě.

Častým řešením jsou procesory od společnosti Espressif. Jedná se o čínskou firmu, jejíž produkty jsou cenově dostupnější. Vyrábí SoC řady ESP32 nebo ESP8266, které integrují rádiový vysílač a přijímač, GPIO, I2C, I2S, SPI, UART a 32 bit jádro Xtensa nebo RISC-V.

ESP8266 je dostupnější varianta, obvykle s menším počtem vyvedených GPIO, pamětí až 512 kB SRAM, 384 kB ROM a 4 MB flash. Obsahuje jednojádrový 32 bit L106 CPU s frekvencí až do 160 MHz. Konektivitu podporuje pouze IEEE 802.11b/g/n, na frekvenci $2{,}4$ GHz do rychlosti 75 Mbit s^{-1}.[16]

ESP32 disponuje pamětí až 520 kB SRAM, 576 kB ROM a 8 MB flash. Obsahuje dvoujádrový 32 bit LX7 CPU s frekvencí až do 240 MHz. Dále může mít BLE 5.3 nebo BLE 4.2. Wi-Fi konektivita je IEEE 802.11b/g/n na frekvenci $2{,}4$ GHz do rychlosti 150 Mbit s^{-1}. Nejnovější verze ESP32-C6 má jednojádrové CPU 32 bit RISC-V s frekvencí 160 MHz a je vybavena Wi-Fi konektivitou IEEE 802.11ax (Wi-Fi 6) na frekvenci $2{,}4$ GHz. [16]

Všechny SoC mají podporu zabezpečení WEP, WPA, WPA2. Pravděpodobně pouze ESP32 podporuje WPA3 [17]. Na internetu jsou dohledatelné příklady, které ukazují, že je možné fungovat i v enterprise módu WPA2 [18]. Kvůli jednodušší konfiguraci a primárně pro výukové účely je jistě vhodnější WPA2-PSK.

Další příklad SoC s podporou Wi-Fi může být WFI32E02UC od firmy Microchip. Je

určen do průmyslového prostředí a disponuje pamětí 512 kB SRAM, 2 MB flash. Wi-Fi IEEE 802.11b/g/n na frekvenci 2,4 GHz. SoC podporuje až WPA3. [19]

Druhou kategorií zařízení jsou sériově vyráběná zařízení. Zde je třeba brát v potaz, že tato zařízení nejsou důvěryhodná a je potřeba omezit jejich přístup do internetu. Například z důvodu zamezení odesílání dat na servery výrobce. Podporované zabezpečení často není výrobcem uvedeno a je pravděpodobné, že nebude podporovat enterprise režim. Příkladem může být například chytrá zásuvka TP-Link Tapo P110M, která dle datasheetu [20] podporuje IEEE 802.11b/g/n a zabezpečení: žádné, WEP, WPA-PSK, WPA2-PSK, WPA3.

4.2 Požadavky na síť

Z možností IoT zařízení a Wi-Fi sítí vyplývají nutné požadavky. Vzhledem k rychlému rozvoji Wi-Fi je potřeba zajistit zpětnou kompatibilitu vysílačů s alespoň IEEE 802.11n (Wi-Fi 4) při použití frekvence 2,4 GHz. Běžně toto podporují i nové přístupové body. Použití WPA2-PSK k zabezpečení se jeví jako vhodný kompromis. Je podporováno IoT zařízeními a zároveň dosahuje dostatečné míry bezpečnosti.

Autentizační protokol IEEE 802.1X je sice některými IoT zařízeními podporován, ale jeho nastavení nemusí být vždy jednoduché. Proto je potřeba najít jiné řešení.

Síť na ZČU využívá přístupové body od společnosti CISCO, které podporují technologii iPSK. Díky tomu je možné do Wi-Fi sítě s jedním SSID přihlásit zařízení s různými přístupovými údaji. Zařízení poté nemusí sdílet PSK, což je výhodné pro bezpečnost celé sítě v případě úniku hesla. Hesla jsou spravována AAA serverem, což může být RADIUS. [21]

Pro monitoring a správu tohoto řešení je možné využít Identity PSK Manager for Cisco ISE [22].

V síti je potřeba realizovat bránu, která implementuje základní služby jako DHCP server a DNS server. Zařízení pod Wi-Fi IoT sítí budou připojeny a autentizovány přes Wi-Fi a iPSK. DHCP server přidělí zařízení IPv4 adresu. Pro další konfiguraci komunikace může být využit linuxový nástroj iptables.

4.3 Požadavky na systém

Vzhledem k zabezpečení sítě a provozu na ní je vyžadována funkcionalita, která umožní jednotlivým IoT zařízením nastavovat odlišná oprávnění komunikace mezi sebou a omezit nebo naopak povolit komunikaci do internetu. Komunikace s internetem může být vítaná, pokud bude senzor odesílat data do veřejného IoT cloudu, ale může být i nechtěná, pokud zařízení odesílá telemetrická data třetím stranám. Pro nastavení komunikace mezi zařízeními v jedné síti přijde vhod funkcionalita skupin, díky které půjde nastavit, jaká zařízení mají mezi sebou povolenou komunikaci. Bude tak umožněno zařízením, které pro svou funkci vyžadují data ze vzdálených senzorů, aby získaly hodnoty přímo ze sítě bez prostředníka.

Některá zařízení pro svou správu nebo zobrazení stavu a dat vyžadují, aby uživatel mohl přistoupit k webovému rozhraní. V jiném případě bude systém pro vyčítání dat potřebovat přistoupit přímo na zařízení. Zároveň je potřeba brát v potaz, že uživatel nebo systém pro vyčítání nemusí být vždy připojen do stejné sítě. Toto je řešitelné překladem portů (PAT) nebo překladem síťových adres (NAT) 1:1.

Překladem portů je možné dosáhnout připojení na více zařízení v síti na různých portech. Například lze na bránu 10.0.0.1 a provoz na portu 8080 přesměrovat na adresu ve vnitřní síti 192.168.1.10 a port 80. Pro zařízení 192.168.1.11 a port 80 lze na bráně využít např port 8081. Tímto způsobem lze přesměrovat mnoho portů.

Naproti tomu použitím NAT 1:1 lze přesměrovat pouze veškerý provoz na jedno zařízení uvnitř sítě. [23]

Vzhledem k nasazení v prostředí počítačové sítě je příhodná realizace s využitím webových technologií. Díky tomu je běh aplikace nezávislý na operačním systému uživatele. Uživatelé si nemusí aplikaci instalovat a není potřeba řešit vzdálené aktualizace, protože stačí změnit kód na serveru. Kvůli dosažení co největší flexibility při konfiguraci nových zařízení do IoT sítě je třeba zhodnotit roli mobilních zařízení s OS Android. Dnes je běžné, že prvotní nastavení a nastavení přístupu do Wi-Fi u komerčních zařízení probíhá přes mobilní aplikaci pomocí bluetooth. Je to účelné, výrobci mnohdy využívají ESP32, které má bluetooth integrovaný.

4.4 Technologie pro realizaci

Vývoj webových aplikací je velmi dynamická oblast. Dle několika zdrojů lze aplikace rozdělit až na 11 druhů. Toto je výběr z nich: Statické, dynamické, jednostránkové (SPA), vícestránkové (MPA), progresivní (PWA), systémy pro správu obsahu (CMS), internetové obchody, bohaté internetové aplikace, javascriptové aplikace a nakonec animované aplikace. Každý zástupce má jiné cíle a různé vlastnosti z pohledu komunikace mezi serverem a klientem nebo optimalizace vyhledávačů (SEO) [24] [25] [26]. V tomto dělení se některé vlastnosti prolínají. Například dynamická aplikace může být zároveň SPA. Proto bude vhodné toto dělení dále upřesnit.

Základní dělení může být na statické a dynamické. Statické jsou vytvořeny pouze v klientských jazycích jako HTML, CSS, JS a ke své funkci potřebují pouze software realizující server jako například Nginx nebo Apache. Dynamické aplikace přidávají vrstvu logiky, která je realizována serverovými skriptovacími jazyky. Obvykle spolupracují s databází nebo dalšími externími zdroji, kde se získávají dynamická data. K funkci je opět nutný Nginx nebo jiný server, kterému jsou předávána dynamická data. Například v případě PHP přes FastCGI [27].

Další dělení může být podle toho, jak probíhá komunikace mezi klientem a serverem na MPA a SPA.

V případě MPA je vykreslení obsahu v podobě HTML, CSS, JS prováděno na straně serveru a při přechodu na jiný pohled je vždy odeslán celý kód. Tento druh aplikace je nejběžnější a prohlížeče jsou pro něj optimalizovány například ukládáním opakujících se částí stránky do cache. Obsah je snadno zpracovatelný roboty, kteří indexují pro prohlížeče.

SPA se oproti tomu načte pouze jednou a mezi serverem a klientem se předávají už jen nutná data pro vykreslení nového obsahu. Server je omezen pouze na API. Část zátěže se tedy předá klientovi, který pomocí JavaScriptu stránku vykresluje. Tím je dosaženo větší plynulosti při prohlížení. Doba prvního načtení sice může být delší, ale obvykle se při ní načte velká část datově náročných prostředků. Aby byla stránka vhodně zaznamenávána vyhledávači, musí se využít speciálních technik. [24] [25] [26]

Zbývající kategorie popisují různé speciální vlastnosti. Hybridní aplikace PWA lze nainstalovat na klientské zařízení a dle druhu může být spustitelná bez internetového připojení nebo může synchronizovat data na pozadí. Obvykle bude spadat do kategorie dynamická SPA. CMS je nástroj pro tvorbu obsahu bez nutné znalosti programování. Typickým zástupcem je WordPress [28]. Vždy bude realizován jako dynamická SPA nebo MPA. Internetový obchod umožňuje prezentovat zboží, vkládat ho do košíku a pomocí napojení na platební brány a dopravce uzavřít obchod. Realizace je opět dynamická SPA nebo MPA. Jeden ze zástupců z této kategorie je Prestashop [29]. Bohaté internetové aplikace jsou dnes méně časté. Jako klientské technologie jsou použity Adobe Flash, JavaFX nebo Microsoft Silverlight. Jejich ústup nastal kvůli problémům s bezpečností a kompatibilitou. Byly tak postupně překonány moderním JavaScriptem, CSS3 a HTML5. JavaScriptové aplikace využívají pro své vykreslování u klienta obvykle jednu ze tří vývojových platforem React, Angular nebo Vue.js. Jsou takto realizovány statické i dynamické SPA. Animované aplikace se vyznačují využitím pohybujících se grafických prvků a propracovaným grafickým návrhem. Mohou být realizovány jako statické, dynamické, SPA i MPA. [24] [25] [26]

Pro realizaci interního systému pro správu zařízení na WiFi IoT síti se jeví jako vhodná volba dynamická SPA. Díky tomu, že není nutné ani žádoucí řešit optimalizaci vyhledávačů, zůstane výhoda lepší odezvy při používání. Dále lze z SPA pomocí dalších technologií vytvořit mobilní aplikaci, což přístupem MPA není vhodné. Z tohoto výběru vyplývá, že server je potřeba řešit jako API a klienta jako JavaScriptovou aplikaci.

4.5 Server

4.5.1 Programovací jazyk

Serverovou část (backend) je možné realizovat prakticky v libovolném jazyce. Vzhledem k náročnosti vývoje webových aplikací a mnoha společným vlastnostem u většiny projektů, vznikly v rámci programovacích jazyků vývojové platformy a sady znovupoužitelných komponent. Výběr jazyka je tak primárně závislý na omezení běhového prostředí a znalostech vývojáře. Díky dostačujícímu výkonu běžně dostupného hardwaru a relativně levné RAM paměti nemusí být vždy preferovány nízkoúrovňové jazyky. Běžně používané jazyky jsou JavaScript (Node.js), Python, PHP, Ruby, Java, Rust, C#, Go. Webové aplikace, stejně jako jiné moderní aplikace, se neobejdou bez velkého množství závislostí, proto většina z nich využívá balíčkovacího systému s online repozitářem dostupných knihoven. Všechny zmíněné jazyky a

nástroje mají otevřený zdrojový kód.

JavaScript je jazyk, který vznikl pro použití ve webových prohlížečích. S růstem jeho popularity se rozšířil a byly vyvinuty technologie jako Node.js umožňující jeho běh i mimo prohlížeč [30]. Jazyk samotný je podrobněji popsán v následující kapitole. Node.js je postaveno nad V8 JavaScript engine, což je také jádro známého prohlížeče Chrome. Node.js bylo navrženo jako asynchronní, událostmi řízené běhové prostředí. Díky tomuto návrhu je aplikace spuštěna v jednom procesu a nevytváří nové vlákno pro každý dotaz. Při vykonávání I/O operací, jako je čtení z databáze nebo souborového systému, neblokuje celé vlákno, ale po vrácení výsledku ihned pokračuje v operacích. Díky tomu aplikace mohou zvládat tisíce klientů v jeden okamžik. [31] V prostředí Node.js jsou rozšířené vývojové platformy Express nebo Koa [32]. V těchto případech se nejedná o celé systémy balíčku, proto je na vývojáři vhodné doplnění o další balíčky [32].

```
1 const express = require('express')
2 const app = express()
3
4 app.get('/', (req, res) => {
5     res.send('Test')
6 })
7
8 app.listen(80)
```

Obrázek 4.1: Příklad serveru s použitím vývojové platformy Express

Python má široké uplatnění v mnoha oblastech, od programování GUI aplikací pro OS Windows po oblast datové analýzy a strojového učení. Jeho syntaxe je úspornější než u jazyků, které mají syntaxi podobnou jazyku C (C-like). Je interpretovaný s dynamickým typováním. Což ho činí vhodným pro psaní skriptů. Pro webový vývoj je převážně využívána vývojová platforma Django. Programovací jazyk má širokou komunitu, která tvoří balíčky z nejrůznějších oborů. Ty jsou distribuovány přes portál pypi.org. [33] V Tiobe indexu, který komplexně hodnotí oblíbenost programovacích jazyků, je Python nejpopulárnější [34]. Toto hodnocení je obecné a není zaměřené pouze na webový vývoj.

```
1 from django.http import HttpResponse
2
3 def index(request):
4     return HttpResponse("Test")
```

Obrázek 4.2: Příklad view v jazyce python s použitím vývojové platformy Django

PHP vzniklo se zaměřením na webové aplikace, které jazyku dodnes zůstalo. Jazyk je interpretovaný a dynamicky typovaný. Stále prochází vývojem a s časem jsou přidávány nové vlastnosti, například před verzí 4 ještě nepodporoval OOP. Poslední verze je nyní 8.3, ve které

je jedním ze zlepšení možnost určit typ konstanty ve třídě. Typ proměnné není nutné určovat, ale pokud je zadán, kontrola správnosti typu probíhá za běhu programu. [27]

K jazyku vzniklo velké množství vývojových platforem velmi populární je Symfony [35]. Pro správu balíčků se používá projekt zvaný Composer. Tento jazyk je zdaleka nejpoužívanější pro vývoj backendu [36].

```php
<?php
namespace App\Controller;

use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Attribute\Route;

class TestController
{
    #[Route('/test', name: 'app_test')]
    public function test(): Response
    {
        return new Response(
            'Test'
        );
    }
}
```

Obrázek 4.3: Příklad controlleru v jazyce PHP s použitím vývojové platformy Symfony

Ruby je opět interpretovaný, dynamicky typovaný jazyk. Pro webový vývoj je populární vývojová platforma Ruby on Rails. Balíčky pro tento systém se označují jako RubyGems. Vše v tomto jazyce se dá chápat jako objekt ve smyslu OOP. Jakýkoliv prvek v kódu tedy může mít vlastnosti a akce.

```ruby
class TestController < ApplicationController
    def new
        render html: 'Test'.html_safe
    end
end
```

Obrázek 4.4: Příklad controlleru v jazyce Ruby s použitím vývojové platformy Ruby on Rails

Java je velmi rozšířený a univerzální jazyk. Je staticky typovaný a využívá JIT kompilátor. Pro vývoj webového serveru je dostupný projekt Spring Boot, což je soubor mnoha knihoven pro REST API, cloud, bezpečnost, GraphQL a další [37]. Pro odesílání obsahu je ve spojení s Javou využíván server Apache Tomcat. Veřejná online služba pro stahování a správu balíčků v podobném smyslu, jako mají předešlé jazyky, nebyla nalezena. Java je využívána hlavně v prostředích, kde je celý firemní systém založen na tomto jazyku a lze tak sdílet kód.

```java
import org.springframework.boot.*;
import org.springframework.boot.autoconfigure.*;
import org.springframework.web.bind.annotation.*;

@RestController
@SpringBootApplication
public class Test {

    @RequestMapping("/")
    String home() {
        return "Test";
    }

    public static void main(String[] args) {
        SpringApplication.run(Example.class, args);
    }
}
```

Obrázek 4.5: Příklad controlleru v jazyce Java s použitím vývojové platformy Spring Boot

Rust je relativně nový jazyk, který vznikl až v roce 2015. Je zaměřen na stabilitu, spolehlivost a produktivitu [38]. Je kompilovaný a vhodný i pro nízkoúrovňové programování [38]. Jeho hlavním rozdílem oproti ostatním kompilovaným jazykům je paměťová bezpečnost bez garbage collectoru [38]. Takto vytvořené aplikace mohou dosahovat velmi dobrého výkonu [38]. Pro webový vývoj je možné využít vývojovou platformu Rocket [39]. Jazyk integruje balíčkovací systém s názvem Cargo.

```rust
#[macro_use] extern crate rocket;

#[get("/")]
fn test(name: &str, age: u8) -> String {
    format!("Test")
}

#[launch]
fn rocket() -> _ {
    rocket::build().mount("/", routes![test])
}
```

Obrázek 4.6: Příklad serveru v jazyce Rust s použitím vývojové platformy Rocket

Jazyk C# vyvinutý ve společnosti Microsoft je obvykle svázaný s platformou .NET. Je velmi univerzální. Pro webový vývoj lze příhodně využívat ASP.NET. Jedná se o kompletní nástroj,

který obsahuje vše potřebné pro vývoj. Jazyk se ideologicky podobá Javě, svým statickým typováním a využitím JIT. Platforma ASP.NET dnes není spojená pouze s OS Windows, ale může být provozována na Linuxu i macOS. Balíčkovací systém se jmenuje NuGet. [40]

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace AttributeRoutingSample.Controllers
{
    [RoutePrefix("Test")]
    public class TestController : Controller
    {
        [Route("~/")]
        [Route]
        [Route("Index")]
        public ActionResult Index()
        {
            return View();
        }
    }
}
```

Obrázek 4.7: Příklad controlleru v jazyce C# s použitím vývojové platformy ASP.NET

Go byl vyvinut společností Google. Je to staticky typovaný kompilovaný jazyk s garbage collectorem. Jedna z webových vývojových platforem se jmenuje Buffalo a obsahuje mnoho balíčků, jako je ORM nebo šablonovací systém. Balíčky lze stahovat pomocí příkazu go get. [41]

```
package actions
func Home(c buffalo.Context) error {
    return c.Render(200, r.HTML("Test"))
}
```

Obrázek 4.8: Příklad action v jazyce GO s použitím vývojové platformy Buffalo

Vzhledem k tomu, že u této aplikace se nepočítá s velkým vytížením tisíců uživatelů a s ohledem na možný budoucí vývoj, byl vybrán jazyk PHP. Je totiž velmi oblíbený a tak je předpoklad, že je možné najít vývojáře, kteří mají s tímto jazykem zkušenosti.

4.5.2 Vývojová platforma pro PHP

Použití vývojové platformy přispívá k dobré udržitelnosti kódu nastavením pravidel vývoje, vzhledu kódu a dobrých programátorských technik, jako jsou správné návrhové vzory OOP a programátorské principy DRY, KISS, SOLID. Všechny platformy vynucují rozdělení logiky aplikace na části, jiné části tak získávají, zpracovávají a zobrazují data. Díky tomu je možné psát čistější kód. Použití vývojové platformy šetří práci, protože není třeba psát znovu již vymyšlený kód.

Zajímavé a populární projekty jsou Laravel, Symfony, Laminas a nakonec je potřeba zmínit český projekt Nette. Některé projekty jsou aktivní už od roku 2005 [35]. Vývoj tedy probíhá již dlouhou dobu a je ovlivněn konkurenčními projekty a i vývojem samotného PHP. Projekty tedy mají mnoho společných vlastností a pro vývojáře je možné po prostudování dokumentace přecházet mezi jednotlivými platformami bez velkého úsilí. Mezi společné vlastnosti patří použití ORM pro přístup k databázím. Architektura MVC - Model View Controller. Někdy je část controller nahrazena slovem presenter, ale funkcionalita je velmi podobná.

Odlišnosti mezi vývojovými platformami jsou primárně množství modulů, které obsahují. Výkonnost respektive zpomalení zpracování požadavků oproti kódu v čistém PHP. Podpora posledních vlastností PHP. Oblíbenost a podpora komunity. V neposlední řadě hraje významnou roli kvalita dokumentace. [35]

Pro vývoj této aplikace byl zvolen projekt Symfony. Má velmi kvalitní dokumentaci, je spojen s výborným ORM Doctrine, podporuje poslední verzi PHP 8.3 a hlavně má připravené knihovny (zde označené jako bundles) pro JWT autentizaci a GraphQL, což jak bude dále popsáno, jsou další vybrané technologie pro toto řešení.

4.5.3 Databáze

Databáze se dělí na relační a nerelační (NoSQL). Výběr vhodné databáze je podmíněn daty, která do ní budou ukládána.

Data mohou být strukturovaná s předdefinovaným schématem [42]. Pro tento druh dat je jasná volba relační databáze [42]. Veškeré vazby mezi objekty jsou jasně definované. Databáze tím může zajišťovat například kontrolu cizích klíčů [42]. Nevýhodou tohoto přístupu oproti NoSQL je nutný složitější návrh databázové struktury. Každá změna ve schématu znamená nutnost napsání migračního skriptu, který se musí spustit ve všech instancí. [43]

Pro částečně strukturovaná (semi-structured) data jsou vhodné NoSQL databáze [42]. Vyskytují se zde vnořené objekty a pole [42]. Dokumentově orientované databáze jako například MongoDB jsou potom více flexibilní z pohledu vývoje i možnosti škálovatelnosti [43]. Lze totiž aplikovat horizontální škálování, což znamená, že se data replikují do jiné instance databáze, například na jiném serveru [43]. MongoDB zajistí aktuálnost dat a zvýší se tím výkon pro čtení i zápis [43].

Další druhy dat jsou nestrukturovaná, například soubory. Pro jejich ukládání se využívá souborový systém. Dále existují časové řady, například hodnoty z měření, data v podobě grafu

jako sociální sítě nebo systémy pro doporučování. Dalším druhem jsou páry klíč-hodnota, jako například data pro cache. Nakonec jsou geoprostorová data jako geografické informace.

Vzhledem k jasně dané struktuře projektu a tomu, že není předpoklad velmi výrazného vytížení databáze, tak se jeví relační databáze jako vhodný nástroj pro uchování dat. Konkrétně byl vybrán projekt s otevřeným kódem MariaDB. Vzhledem k použití ORM Doctrine, je možné nahradit za jakoukoliv jinou, podporovanou relační databázi.

4.6 Klient

4.6.1 Programovací jazyky

Na klientské straně (frontend) je situace více přímočará. Výběr je totiž omezen podporou prohlížečů na HTML, CSS a JS. Existuje další alternativa k JavaScriptu a tou je WebAssembly.

Jedná se o binární instrukční formát, který je v prohlížeči spuštěn v prostředí sandboxu [44]. Díky tomu je možné v prohlížeči po zkompilování do WebAssembly spouštět kód napsaný v jiných jazycích, jako například Rust [44]. WebAssembly není navržen jako náhrada JavaScriptu, ale jeho doplněk pro účel spouštění výpočetně náročného kódu téměř nativní rychlostí [44]. Pro WebAssembly zatím není k dispozici takové množství nástrojů a knihoven jako pro JavaScript, proto je využíván jen v případech, kde by byl při náročných výpočtech přínosem.

HTML je základní stavební prvek stránky, který určuje význam jednotlivých prvků a strukturu na webu [45].

```
1  <!DOCTYPE html>
2  <html lang="cs">
3      <head>
4          <meta charset="utf-8" />
5          <title>WiFi IoT</title>
6      </head>
7
8      <body>
9          <div id="app"></div>
10     </body>
11 </html>
```

Obrázek 4.9: Základní struktura stránky v HTML

Kód se skládá z tagů, které obalují obsah, jako je v příkladu ukázáno na řádce 5. Tagy mohou mít atributy, například atribut id na řádce 9. Tímto způsobem je definována struktura stránky. Vzhled a chování stránky je potom možné ovlivňovat dalšími jazyky.

CSS slouží k definici vzhledu, barev, rozložení stránky [46].

```css
div {
    background-color: #98D098;
}
```

Obrázek 4.10: Ukázka kódu v CSS

Kód na obrázku 4.10 nastavuje všem tagům div barevné pozadí s barvou #98D098. V tomto případě je div na řádce 1 selektor (selector), background-color na řádce 2 vlastnost (property) a #98D098 hodnota vlastnosti (property value).

JavaScript byl standardizován asociací ECMA. Běží v jednom vlákně, je interpretovaný a překládán pomocí mechanismu JIT. Díky tomu je možné distribuovat přímo zdrojové kódy, což je vhodné pro použití ve světě webových technologií. Má prvotřídní funkce (First-class functions), což znamená, že funkce jsou brány jako proměnná. Lze je tak například přiřadit do proměnné nebo vrátit jako výsledek funkce nebo předat jako parametr funkce [47].

```javascript
const fceJakoPromenna = (vstup) => {
    alert(vstup)
}

fceJakoPromenna('Test')
```

Obrázek 4.11: Příklad použití funkce jako proměnné v JS

Toho je při programování klientských aplikací vzhledem k jejich asynchronní povaze hojně využíváno. JavaScript používá jako balíčkovací systém NPM [48].

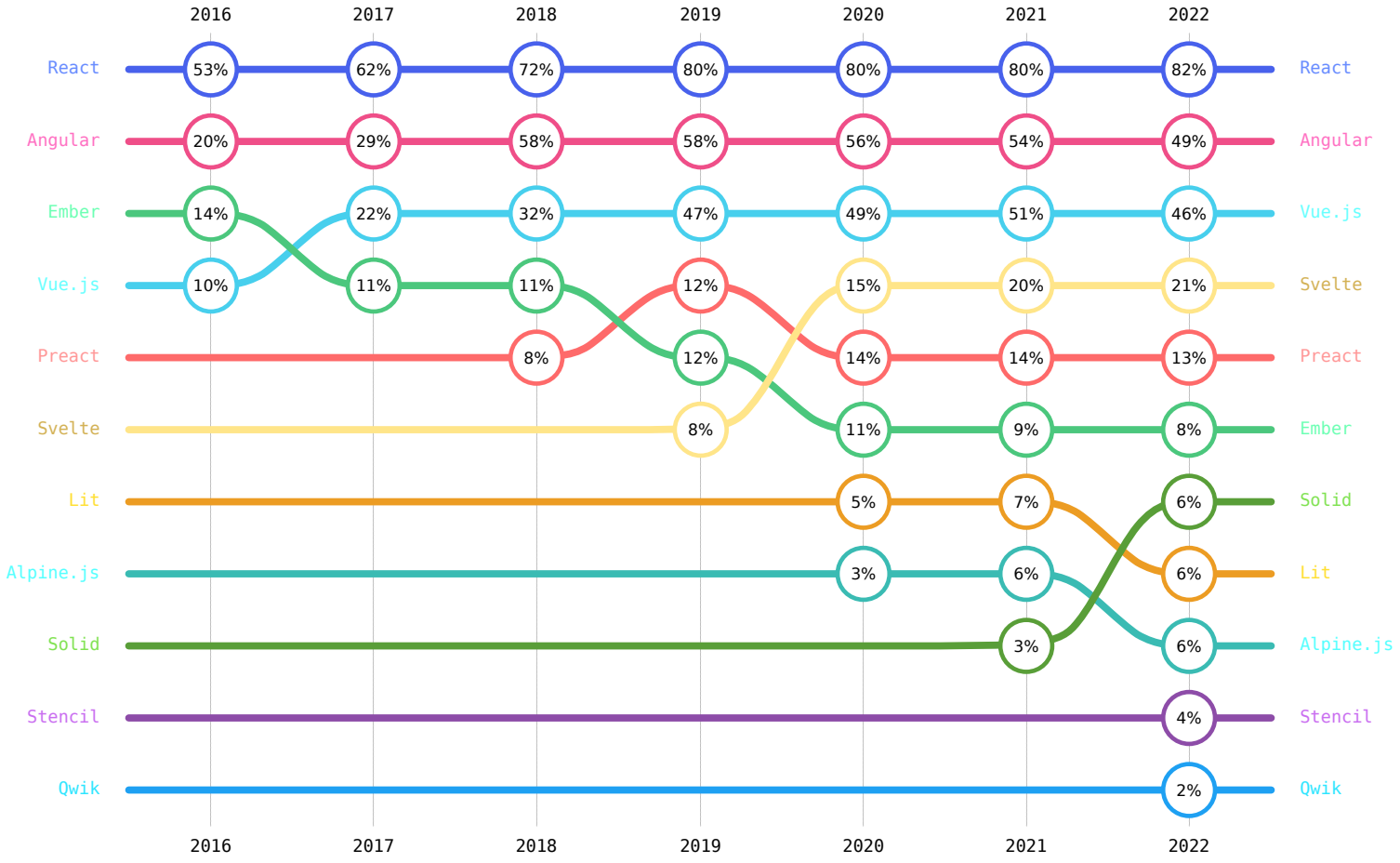
JS nepodporuje definici typů proměnných, což znemožňuje statickou analýzu kódu. Tuto vlastnost JavaScriptu se snaží opravit jazyk TypeScript. Je to silně typovaný jazyk vyvinutý společností Microsoft, který je překládán do JavaScriptu. Programování v TypeScriptu je plynulejší, protože vývojové prostředí může lépe napovídat například parametry při volání funkce. Před přeložením se provede typová analýza, která odhalí chyby ještě před spuštěním kódu. Tím je možné tvořit kvalitnější kód a předcházet nečekaným chybám v produkčním nasazení. [49]

```typescript
const fceJakoPromenna = (vstup: string): void => {
    alert(vstup)
}

fceJakoPromenna('Test')
```

Obrázek 4.12: Příklad použití funkce jako proměnné v TS

4.6.2 Vývojová platforma pro JavaScript

Pro vývoj SPA se dnes převážně využívají reaktivní vývojové platformy. Podle průzkumu mezi vývojáři na stateofjs.com jsou nejčastěji používané platformy React, Angular, Vue.js a Svelte [50]. Popularita knihovny je důležitým měřítkem. V prostředí jazyka JavaScript vzniká každý den mnoho knihoven a neustále dochází k jejich bouřlivému vývoji. Projekty jsou často komunitní s otevřeným kódem a pokud nejsou populární, nemají tendenci se dále rozvíjet a nedochází k případnému záplatování v budoucnosti. React a Angular byly původně vydány velkými společnostmi a pravděpodobně si tím získali důvěru vývojářské komunity. Nyní ovšem mají otevřený zdrojový kód. Všechny platformy mají k dispozici CLI pro snadné založení projektu a rychlý start vývoje.



Obrázek 4.13: Popularita JS frontend vývojových platforem v čase. Převzato z [50].

React byl vyvinut společností Facebook v roce 2013. Dlouhodobě se jedná o nejpoužívanější frontendovou platformu a probíhá jeho rychlý vývoj. Je hodnocen jako snadno naučitelný. [51]

```
import React from 'react';

function App() {
   return (
      <div>
      Test
      </div>
   );
}
```

Obrázek 4.14: Příklad kódu ve fronted vývojové platformě React

Angular je projekt vyvinutý společností Google v roce 2016. Předpokládá se, že bude dlouhodobě vyvíjen, ale to není nikdy jisté [52]. Jeho předchozí verze byla označená AngularJS, což byl relativně odlišný projekt. Angular je vyvíjen v jazyce TypeScript a je to také jediný jazyk, který podporuje. Je velmi rozsáhlý a je hodnocen jako složitější k naučení. [51]

```
import { Component } from '@angular/core';

@Component ({
   selector: 'test-app',
   template: `<div>Test</div>`,
})

export class AppComponent;
```

Obrázek 4.15: Příklad kódu ve fronted vývojové platformě Angular

Vue vznikl v roce 2014. Jeho vývojáři se inspirovali u tehdy existujících platforem jako AngularJS a React. Komunitou je ceněn pro svou srozumitelnost a má díky tomu krátkou křivku učení. Není svázán s definovanými knihovnami a má tak větší flexibilitu při výběru dalších potřebných komponent. [51] V testech rychlosti obvykle poráží Angular a React [53].

```
<template>
    <div>Test</div>
</template>

<script setup>
</script>
```

Obrázek 4.16: Příklad kódu ve fronted vývojové platformě Vue

Svelte se od předchozích projektů liší svou architekturou. U Reactu, Angularu a Vue je knihovna nejdříve celá načtena a potom vykonává vývojářem napsaný kód. Oproti tomu

kód napsaný pro Svelte je potřeba zkompilovat. Do prohlížeče jsou potom načteny jen nutné komponenty. Není to tedy framework v pravém slova smyslu. Zatímco React, Angular a Vue využívají pro změnu stránky takzvaný virtuální DOM, Svelte upravuje přímo DOM v prohlížeči [54]. Díky tomu je načítaná velikost menší a lze dosáhnout lepšího výkonu [51]. Je ovšem méně populární než předchozí jmenovaní zástupci.

Pro tento projekt byla zvolena vývojová platforma Vue z důvodu dostatečné popularity, volnosti při vývoji a osobní preferenci.

4.6.3 UI knihovna komponent a mobilní aplikace

Knihovnou komponent je někdy také myšlena CSS vývojová platforma. Vzhledem k nastaveným prioritám pro vývoj tohoto projektu bylo potřeba najít knihovnu, která umožní vývoj webové aplikace a zároveň bude kód snadné optimalizovat pro mobilní aplikaci. Tyto vlastnosti splňuje projekt Ionic [55].

Podporuje vývoj na platformách Angular, React i Vue [55]. Nejedná se pouze o knihovnu komponent, ale i o CLI, díky kterému lze snadno vygenerovat nový projekt s přednastavenými nástroji pro vývoj [55]. Pro tvorbu nativní mobilní aplikace podporuje technologii Capacitor nebo Cordova. Dnes je ovšem preferováno použití modernější technologie Capacitor, kterou vyvíjí stejný tým jako Ionic [56].

Spojením Ionic a Capacitor je možné vyvinout webovou aplikaci, optimalizovat její zobrazení pro mobilní zařízení a poté přes standardní vývojové nástroje pro Android nebo iOS vytvořit mobilní aplikaci, která bude sdílet identický kód jako webová aplikace. Přes API projektu Capacitor je umožněno využívat nativní funkce mobilních zařízení nezávisle na platformě.

4.7 Komunikace mezi serverem a klientem

Vzhledem k výběru dynamické SPA je nutné server realizovat jako API. Na výběr je z možností REST, SOAP, GraphQL a gRPC [57]. API se realizují jako bezstavové, což znamená, že každý požadavek musí obsahovat všechny důležité informace k jeho zpracování a nezávisí tak na předchozích požadavcích. Přenos dat obvykle probíhá po protokolu HTTP nebo zabezpečeném HTTPS. Další možností je protokol WebSockets, což je odlišný komunikační protokol, postavený přímo nad TCP. Umožňuje obousměrný přenos dat jedním TCP spojením. Tím lze dosáhnout lepšího výkonu, ale nepodporuje kompresi dat, neumí sám znovu navázat přerušené spojení a nepodporuje standardní způsoby ověřování uživatele [58]. Pokud nejsou přenášena živá data s nutností nízké latence, je využit protokol HTTP.

Po REST API je možné přenášet data ve struktuře JSON nebo XML. Jednotlivé operace jsou dány metodami HTTP GET, POST, DELETE, PATCH, OPTIONS [57]. Dotaz pak probíhá tak, že je odeslán například POST na adresu `https://example.net/v1/users`.

SOAP API podporuje přenos pomocí dokumentů formátovaných jako XML. V protokolu je

integrovaná kontrola chyb a je využíváno v případech, kde je potřeba zajistit vysokou bezpečnost [57]. S užitečnými daty je ovšem přenášeno mnoho informací navíc, která slouží k popisu dotazů. Samotná struktura XML vyžaduje více nadbytečných dat oproti JSON.

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
 <soap:Body>
  <IsValidISBN10 xmlns="http://webservices.daehosting.com/ISBN">
   <sISBN>0-19-852663-6</sISBN>
  </IsValidISBN10>
 </soap:Body>
</soap:Envelope>
```

Obrázek 4.17: Příklad SOAP dotazu. Převzato z [59]

GraphQL byl vyvinut a je používán společností Facebook. Přenos dat probíhá ve formátu JSON. Hlavní výhodou je možnost při výběru dat vydefinovat pouze požadovaná pole, tím se minimalizuje množství přenášených dat a je umožněna efektivnější komunikace mezi serverem a klientem. Pro získávání dat se vždy zasílá HTTP POST na server s GraphQL dotazem. [57]

```
query AppUserList {
    appUserList(first: 10) {
        edges {
            node {
                id
                username
                name
                deviceList {
                     id
                     name
                }
            }
        }
    }
}
```

Obrázek 4.18: Příklad GraphQL dotazu.

API gRPC používá HTTP GET k získání informací a POST pro vše ostatní. Obvykle je napsané přímo na míru danému systému a proto je jeho flexibilita menší a vývoj složitější. Vyvinula ho společnost Google a používá ho například pro komunikaci mezi mikroslužbami (microservices). [57]

GraphQL se zdá tedy být vhodným kompromisem s mnoha výhodami oproti REST API. Hlavní výhodou je silné typování schématu a data vracená API jsou jasně definována právě ve schématu i s vazbami objektů mezi sebou [60].

Pokud je potřeba získat data o uživateli s ID 1 v REST API, byl by odeslán dotaz na adresu `/user/1`, který by vrátil všechna dostupná data. V GraphQL lze přesně definovat, jaká data jsou žádaná, například id, uživatelské jméno a libovolné další. [60]

Výhodou je možnost využívat vnořených dotazů [60]. Data mají obvykle mezi sebou vazby, například uživatel má svá zařízení. V REST API by bylo nutné udělat dotaz nejdříve na `/user/1`, který by vrátil, že uživatel má zařízení s ID 10 a 11 a poté na `/device/10` a `/device/11` [60]. V GraphQL lze pomocí vnořeného dotazu získat všechna potřebná data, jako je ukázáno na obrázku 4.18. Toto se nazývá N+1 problém a tímto je přesunut na stranu backendu, kde může být vyřešen efektivněji nebo alespoň rychleji. Databáze zpracuje větší množství dotazů rychleji, než když jsou odeslány přes HTTPS na server. Efektivně lze na straně serveru řešit metodami jako dychtivé načítání dat (eager loading) nebo hromadné načítání dat (batch loading) [61]. REST API lze samozřejmě navrhnout tak, aby například při dotazu na uživatele zároveň vracelo i všechna data o zařízeních, nebo definovat pro speciální adresu pro jeden účel, kdy jsou potřeba data o zařízení i uživateli. To ovšem velmi snižuje univerzálnost API a zvyšuje komplexnost její implementace.

Další výhoda GraphQL přichází v případě přidání další funkcionality, například nového pole s daty. Potom není nutné vytvářet novou verzi pro zachování zpětné kompatibility, jako je tomu u REST API, kdy jsou běžné adresy jako `https://example.net/v1/users`, `https://example.net/v2/users`.

5

Realizace systému

Pro realizaci serveru byl vybrán jazyk PHP s použitím vývojové platformy Symfony. Bude komunikovat s databází MariaDB pomocí ORM doctrine. Pro realizaci klienta byla vybrána vývojová platforma Vue s UI knihovnou Ionic. Pomocí technologie Capacitor bude vytvořena mobilní aplikace. Přenos mezi klientem a serverem bude probíhat přes GraphQL API pomocí protokolu HTTP.

Aplikace byla rozdělena do logických celků, podle jejich funkcionality. Názvy se promítají do pojmenování tabulek v databázi, doctrine entit, jednotlivých tříd, které pracují s těmito zdroji, dále do konečných bodů GraphQL schématu až k pojmenování jednotlivých pohledů v klientské aplikaci. Tyto celky jsou

- AppUser - uživatel systému,
- Device - zařízení přihlášené do sítě nebo příprava pro přihlášení,
- DeviceGroup - skupina zařízení, která mezi sebou mohou komunikovat,
- DevicePhoto - fotky zařízení,
- DevicePort - přesměrování portů k zařízení na hub,
- RefreshToken - tokeny pro obnovu JWT,
- StaticIp - dostupné statické veřejné adresy k přiřazení,
- UserNotification - notifikace pro uživatele.

Uživatel je v systému vytvořen na základě přidání správcem. Mezi jeho vlastnosti patří jméno, uživatelské jméno (sloužící k ověření) a příznak, zda je administrátor. K uživateli lze přiřadit zařízení, skupiny zařízení, tokeny sloužící k obnově JWT a notifikace. Příznak administrátora určuje jeho roli v systému. Administrátor má navíc přístup k úpravě seznamu veřejných adres a k úpravě všech uživatelů. Administrátor má možnost upravovat skupiny a zařízení přidružená i ostatním uživatelům. Uživatel bez administrátorských práv může vytvářet skupiny a zařízení, které jsou následně přiřazené pouze jemu samotnému. Může také upravit

zatím nepřiřazené zařízení a tím si ho přivlastnit. Uživatel může mít účet ve stavech aktivní, bez přístupu, neaktivní, neaktivní - porušení pravidel. Tyto stavy slouží k rozlišení důvodu, proč byl danému uživateli odepřen přístup k systému.

Zařízení je v systému vytvořeno spuštěním příkazu, například při jeho přihlášení do sítě nebo může být přidáno uživatelem, jako příprava pro jeho přihlášení. Jeho vlastnosti zahrnují jméno (pro snadné rozpoznání), MAC adresu (pevně danou adresu výrobcem zařízení), IP adresu (adresu ve vnitřní síti, přidělenou buď dopředu nebo při přihlášení do sítě) a příznak, zda je zařízení známo, který určuje, zda je povolena komunikace. Tento příznak může mít také datum expirace, pokud bude zařízení používáno jen dočasně.

Zařízení může mít přiděleného majitele, což je uživatel systému, a příznak, zda má přístup do sítě internetu. Tento příznak může mít také nastavený datum expirace, aby byla průběžně ověřována zařízení, která mohou přistupovat do sítě internetu. Zařízení může mít přidělenou statickou IPv4 adresu ze seznamu uloženého v databázi a speciální přihlašovací údaje pro wifi (jméno a heslo). Poslední příznak určuje, zda je zařízení povoleno, což signalizuje, že bylo zařízení úmyslně zakázáno například kvůli nevhodné komunikaci v síti. K zařízením lze načíst skupiny, ve kterých jsou obsažena, porty přesměrované na nadřazené bráně a seznam fotek přiřazených k zařízení.

Skupina zařízení slouží k definici pravidel, která určují, jaká zařízení mohou mezi sebou komunikovat. Pokud zařízení nejsou ve skupině, nesmí vidět na ostatní zařízení v síti. Skupina má jméno pro snadnější rozlišení, majitele skupiny a příznak, zda je aktivní. Ke skupině lze načíst seznam všech zařízení v této skupině.

Fotka zařízení je uložena na serveru a přiřazena k zařízení pro jeho jednoznačnou identifikaci. Mezi její vlastnosti patří jméno souboru a zařízení, ke kterému fotka patří.

Přesměrování portů určuje, který port na zařízení je přesměrován na port na nadřazené bráně. Mezi jeho vlastnosti patří port na zařízení, port na bráně, expirace přiřazení a zařízení, ke kterému je přiděleno.

Tokeny pro obnovu JWT jsou řetězce sloužící k obnově komunikace pomocí JWT. Jsou přiřazeny uživateli přes jeho uživatelské jméno, což umožňuje sledovat, na kolika zařízeních je uživatel současně přihlášen do systému.

Statické veřejné adresy jsou seznamem adres v databázi, které lze přidělit zařízení. Jejich vlastnosti zahrnují zařízení, ke kterému je adresa přidělena, expiraci přiřazení a samotnou veřejnou IPv4 adresu.

Notifikace uživatele slouží k zobrazení zprávy uživateli, že bylo připojeno nové zařízení. Tato zpráva se odesílá pouze administrátorům a zobrazuje se po každém přihlášení, dokud ji uživatel aktivně nezavře. Má tyto vlastnosti: text zprávy, uživatele, pro kterého je určena, a příznak, zda byla zobrazena.

U většiny těchto celků jsou také zaznamenány časy vytvoření a úpravy záznamu.

Obrázek 5.1: Adresářová struktura systému.

Všechny části jsou verzovány nástrojem Git a synchronizovány se službou GitLab[1]. Soubory Gitu jsou uloženy ve složce .git. Složka .vscode obsahuje doporučení pro rozšíření a nastavení vývojového prostředí Visual Studio Code. Složka `client` obsahuje zdrojový kód aplikace pro webový prohlížeč a Android. Složka `server` obsahuje aplikaci napsanou v jazyce PHP, která realizuje API. Složka `docker` obsahuje soubory Dockerfile pro vývoj obou částí a doporučený příklad souboru `docker-compose.yml`. Ve složce graphql je uložena definice schématu pro vzájemnou komunikaci klientů a serveru.

5.1 Část server

5.1.1 Vývojové nástroje

Každý webový projekt vyvíjený v PHP řeší problémy, jako je automatické načítání souborů a tříd nebo použití již hotového kódu, například vývojové platformy. Vzhledem k množství knihoven, které jsou pro vývoj nutné, a tomu, že knihovny bývají závislé na ostatních knihovnách, je zcela neudržitelné zdrojové kódy manuálně kopírovat do složky s projektem. Proto vznikl projekt s názvem Composer [62], který tyto problémy řeší.

Umožňuje pomocí konfigurace ve formátu JSON udržovat závislosti v požadované verzi a automaticky je stahovat z repozitáře Packagist[2]. Dále umožňuje definovat minimální verzi PHP a potřebná PHP rozšíření. Pokud nejsou tyto závislosti splněny, instalace neproběhne. Požadované verze závislostí jsou obvykle definovány tak, že je fixována hlavní verze (major) a Composer instaluje vždy poslední vedlejší verzi (minor). Hlavní verze se obvykle povyšuje, pokud software obsahuje změny narušující kompatibilitu s předchozí verzí. Definice závislostí potom vypadá takto:

```
1 ...
2 "require": {
3     "doctrine/doctrine-bundle": "^2.12",
4     ...
5 }
6 ...
```

Obrázek 5.2: Definice závislostí v composer.json

[1] https://gitlab.com
[2] https://packagist.org

Pro automatické načítání tříd v tomto projektu se využívá mechanismus PSR-4. Díky danému pojmenování souborů, složek a jmenných prostorů je potom možné v projektu vytvářet nové instance tříd, aniž by se soubory musely načítat manuálně pomocí PHP příkazů `require(...)` nebo `include(...)`.

```
...
"autoload": {
    "psr-4": {
         "App\\": "src/"
    }
},
"autoload-dev": {
    "psr-4": {
        "DataFixtures\\": "DataFixtures/"
     }
},
...
```

Obrázek 5.3: Definice automatického načítání PSR-4 v composer.json

Takto je v projektu definováno automatické načítání složky src do jmenného prostoru App a pouze v případě běhu ve vývojovém módu se načtou třídy DataFixtures, které nahrávají testovací data do databáze. Třídy v tomto jmenném prostoru dodržují velbloudí notaci (CamelCase). Pro správnou funkci mechanizmu automatického načítání se musí třída jmenovat stejně, jako soubor, ve kterém je přítomna.

Po spuštění příkazu `composer update` jsou balíčky aktualizovány na požadovanou verzi. Vytvoří se soubor `vendor/autoload_runtime.php`, který po načtení ze souboru `public/index.php` zajistí přípravu všech potřebných tříd. Dále se vytvoří soubor `composer.lock`, ten slouží pro uložení současného stavu balíčků. Toto je důležité při nasazení do produkčního prostředí, kde se spustí pouze příkaz `composer install` a je tak zajištěno, že i kdyby vyšla novější verze balíčku, tak je nainstalována stále verze, se kterou probíhal vývoj a testování.

Composer umožňuje definovat příkazy, které mohou spouštět nástroje pro kontrolu kódu, čištění mezipaměti nebo instalační skripty. V tomto projektu jsou definovány příkazy `phpstan`, `fs` a `cs`. Jejich použití je popsáno na následujících řádcích.

Dalším důležitým nástrojem pro vývoj kvalitního kódu s minimem chyb je PHPStan. Jedná se o nástroj pro statickou analýzu kódu. Instaluje se pomocí composeru přímo do projektu jako vývojová závislost. V tomto projektu je spouštěn přes příkaz `composer phpstan`, který je alias pro `phpstan analyse`. V konfiguračním souboru jsou definovány cesty k souborům a přísnost jejich kontroly.

```
parameters:
    level: 9
    paths:
        - bin/
        - config/
        - public/
        - src/
```

Obrázek 5.4: Konfigurace PHPStan v souboru `phpstan.dist.neon`

Pro tento projekt byla zvolena úroveň 9, což je nejpřísnější nastavení. Kontrolují se například volání neznámých funkcí, špatný počet argumentů při volání funkce, dokumentace PHPDocs, návratové typy funkcí, kód, který nebude nikdy spuštěn, kontrola volání a přístupů na typech, které mohou být null a mnoho dalšího [63].

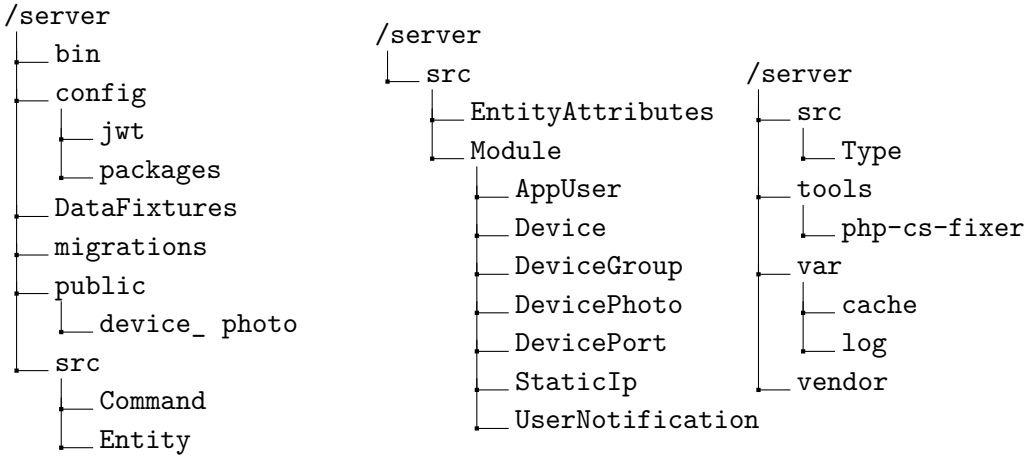
Jako poslední nástroj pro zlepšení kvality kódu je použit PHP_CodeSniffer. Kontrola se spouští příkazem `composer cs` a automatická oprava `composer fs`. Jedná se o alias pro cestu `tools/php-cs-fixer/vendor/bin/php-cs-fixer -vv check` a `fix`. Nastavení je v souboru `.php-cs-fixer.dist.php`. Jsou zde definovány složky, které se naopak nemají kontrolovat a opravovat. Soubor pravidel je doporučený vývojáři platformy Symfony. Tento nástroj tak dokáže upravovat formátování kódu, mazat přebytečně použité jmenné prostory a další. Díky tomu kód obsahuje pouze nutné znaky a je vždy formátován stejně.

```
<?php

$finder = (new PhpCsFixer\Finder())
    ->in(__DIR__)
    ->exclude([
        'vendor',
        'var',
        'tools'
    ])
;

return (new PhpCsFixer\Config())
    ->setRules([
        '@Symfony' => true
    ])
    ->setFinder($finder)
;
```

Obrázek 5.5: Konfigurace PHP_CodeSniffer v souboru `.php-cs-fixer.dist.php`

5.1.2 Struktura serveru

PHP aplikace je ve složce server.



Obrázek 5.6: Adresářová struktura části server.

Kořenová složka obsahuje soubory pro konfiguraci projektu a nástrojů pro jeho vývoj pro nástroje Composer, PHPStan, PHP_CodeSniffer a také soubor `.gitignore`.

Soubor `.gitignore` definuje cesty k souborům, které nebudou zahrnuty ve verzích vytvořených nástrojem GIT. Jedná se hlavně o generované soubory ve složce `vendor`, ve kterých jsou časté změny, zabírají hodně prostoru a hlavně nesouvisí přímo s projektem, takže je nevhodné je verzovat. Dále je důležité vynechat z verzování soubory, které mohou uživatelé dynamicky nahrávat na web jako ve složce `public/device_ photo`. Chybou by bylo zahrnout do verzování složku `var`. Ta obsahuje pouze dočasná data generovaná ORM Doctrine a Symfony. Poslední skupina souborů, které verzovací nástroj nesmí obsahovat jsou soubory s citlivými údaji, jako jsou přístupové údaje do databáze nebo vygenerované certifikáty.

Složka `bin` obsahuje pouze soubor se jménem `console`. Pomocí něj je umožněn přístup k Symfony CLI, která umožňuje spouštět příkazy doctrine, například pro vytváření migrací databáze nebo nahrání testovacích dat. Dále slouží k spouštění uživatelsky definovaných příkazů, které jsou uloženy ve složce `src/Command`.

Složka `config` obsahuje konfigurační soubory pro Symfony a v podsložce `packages` pro nainstalované balíčky. V podsložce `jwt` jsou vygenerované certifikáty pro autentizaci pomocí technologie Json Web Token.

Složka `DataFixtures` obsahuje třídy pro generování testovacích dat. Popsané v 5.1.6.

Složka `migrations` je určena pro soubory migrací, tedy skripty s popisem databázové struktury. Popsané v 5.1.3.

Do složky `public` je směřovaný webový server v případě vývoje to je integrovaný PHP DEV server. V produkčním nasazení to je Nginx. Obsahuje tedy `index.php` a soubory dostupné veřejně z internetu.

Složka `src` obsahuje třídy pro zpracování dat a Doctrine entity. Popsané v 5.1.4 a 5.1.3.

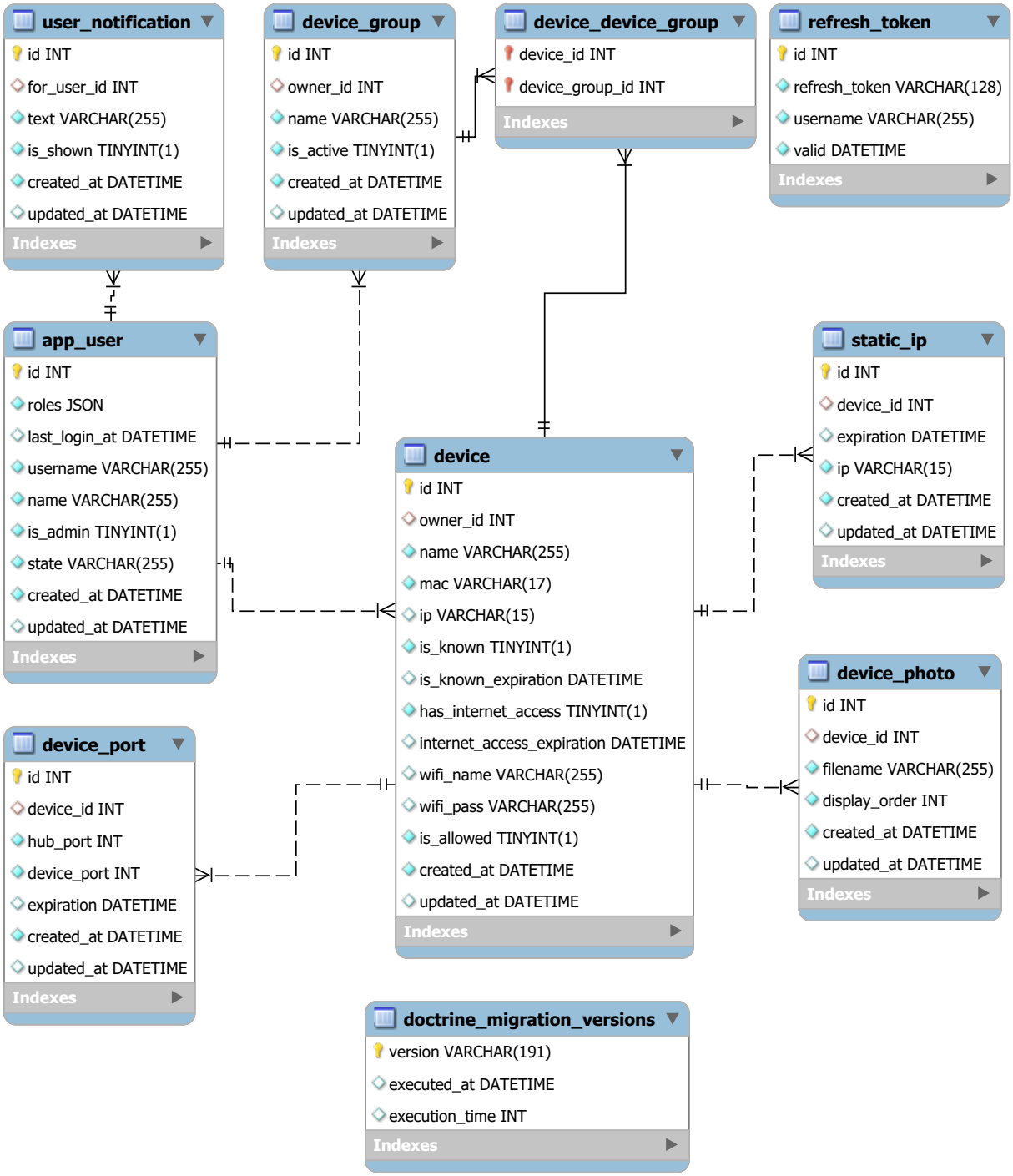
Složka `tools` obsahuje nástroj PHP_CodeSniffer nebo soubory s poznámkami, které je vhodné mít u projektu, ale ne v GITu. Obsah této složky je totiž také v .gitignore.

5.1.3 Databázová struktura

Systém využívá ORM Doctrine. Struktura dat je definována v entitách ve složce `src/Entity`. Entita je třída s veřejnými proměnnými s určeným typem. Některé proměnné, pokud je jejich zadání při vytváření povinné, jsou rovnou definované v konstruktoru (`__construct`). Pro definici databázových polí a vazeb je využito atributů (Attributes), což je moderní vlastnost PHP podporovaná od verze 8.0. Stavy uživatele jsou realizované pomocí výčtového typu, který je v databázi uložen jako řetězec, ale při načtení ORM je převeden na PHP enum.

Všechny entity mají definované id a pole s časem, kdy byl záznam v databázi vytvořen a kdy byl záznam upraven. Proto byly tyto vlastnosti definovány jako šablona (Trait) ve složce `src/EntityAttributes`. Tyto šablony jsou poté vloženy do třídy entity příkazem use.

Vygenerované databázové tabulky jsou vidět na obrázku 5.7. Mezi daty je využito více druhů vazeb. Například mezi uživateli a notifikací pro ně je vazba 1:N (OneToMany), protože jeden uživatel může mít mnoho notifikací. Zařízení mohou být přidělena do více skupin, ale zároveň skupina může obsahovat i více různých zařízení, proto jsou mezi sebou ve vazbě M:N (ManyToMany), vznikla tak pomocná tabulka s názvem device_device_group.



Obrázek 5.7: EER diagram databáze.

5.1.4 GraphQL

Ve složce `src/Module` jsou soubory děleny způsobem, který byl popsán v úvodu této kapitoly. Jednotlivé moduly potom obsahují třídy *Repository a *Resolver. Ty vždy dědí od základních BaseRepository a BaseResolver. Třídy tak implementují pouze specifické vlastnosti a všechny společné jsou implementovány v třídách Base*. Třídy *Repository slouží k práci s databází přes ORM Doctrine. Třídy *Resolver zase k formátování dat, spouštění funkcí z *Repository podle

dotazu od klienta a validaci uživatelských oprávnění.

Funkcionalita, jako validace dat podle schématu GraphQL, formátování odpovědí a požadavků, zajišťuje balíček overblog/graphql-bundle. Základní třídou realizující API je GraphQLResolverMap. Jsou v ní pomocí Dependency Injection vloženy instance tříd *Resolver. Třída pak obsahuje jedinou funkci map, která vrací pole s přiřazením Query a Mutation ze schématu GraphQL na funkce ze tříd *Resolver. Příklad je zkrácený. Pracuje pouze se třídou AppUserResolver a některými typy.

```php
<?php
[
    'Query' => [
        'me' => [$this->appUser, 'me'],
        'appUser' => [$this->appUser, 'findOneBy'],
        'appUserList' => [$this->appUser, 'all'],
    ],
    'Mutation' => [
        'createAppUser' => [$this->appUser, 'create'],
        'updateAppUser' => [$this->appUser, 'update'],
        'removeAppUser' => [$this->appUser, 'removeOneBy'],
    ],
    'DateTime' => [self::SCALAR_TYPE => new DateTimeType()],
    'APP_USER_STATE' => AppUserStateEnum::nameEnumList(),
];
```

Obrázek 5.8: Příklad mapování z GraphQLResolverMap

V případě, že klient pošle například dotaz:

```
query Me {
    me {
        appUser {
            id
            username
        }
        notificationList {
            id
            text
            createdAt
        }
    }
}
```

Obrázek 5.9: Příklad dotazu na AppUser

Bude zavolána funkce `AppUserResolver::me` s příslušnými parametry, které popisují dotaz. Dále se spustí parsování typu DateTime u pole a pokud bude platné, převede se na PHP třídu DateTime. Nakonec se odpověď odešle zpět klientovi.

Pro validaci vlastních typů jsou použity třídy DateTimeType, PortType, MacType a IPv4Type. Ty slouží k validaci přijímaných nebo odesílaných dat. Třída CursorType neprovádí vlastní validaci, slouží pouze jako pomoc při dekódování požadavků, které obsahují stránkování. Třída GraphQLUploadType je implementována v balíčku graphql-bundle a umožňuje zpracování nahraných souborů, ty jsou potom dostupné jako třída `Symfony\Component\HttpFoundation\File\UploadedFile`.

Některé seznamy podporují stránkování. Byla implementována funkcionalita známá jako stránkování Relay. Při požadavku na data je zadán parametr počet (first), který určuje, kolik položek je potřeba načíst, a parametr kurzor (cursor), což je řetězec vygenerovaný stránkovacím systémem. V odpovědi od serveru jsou vrácena data se současným kurzorem a objekt PageInfo, ten obsahuje poslední vrácený kurzor, který lze použít pro další požadavek. PageInfo také obsahuje informace o tom, jestli je dostupná další stránka. Stránkování je mnohem flexibilnější a univerzálnější než klasická implementace s parametry offset a limit.

5.1.5 Balíčky a další functionalita

Pro správnou funkcionalitu jsou využity další Symfony balíčky. Pro tvorbu JWT a jejich obnovovacích tokenů je použit balíček gesdinet/jwt-refresh-token-bundle. Při jeho instalaci je potřeba balíček správně nakonfigurovat a vygenerovat klíče pro podpis JWT.

U každého API je nutné správně nastavit pravidla CORS. K tomu slouží balíček nelmio/cors-bundle. CORS je mechanismus založený na hlavičkách HTTP. Určuje, z jakých zdrojů je povolena komunikace. V tomto případě je povolena komunikace ze všech zdrojů, hlavičky pouze Content-Type a Authorization. Metoda je omezena na POST.

Pro uložení citlivých údajů je použit balíček symfony/dotenv. Díky němu je možné do souboru .env vložit přístupové údaje k databázi a heslo k klíčům JWT. Soubor .env potom zůstává pouze na počítači, kde byl vytvořen.

Pro autentizaci uživatele je připravena třída `Saml2JsonLogin`, která dědí od `JsonLoginAuthenticator`. Třída přijímá přihlašovací údaje ve formátu JSON. Oproti původní třídě je rozšířena o funkci checkCredentials. Ta po dokončení implementace může provádět autentizaci oproti jednotnému přihlašování, přes protokol SAML2, případně jakýkoliv jiný.

5.1.6 Testovací data

Účelem testovacích dat je zaplnit databázi náhodnými daty pro co nejjednodušší otestování co největšího množství různých situací, které mohou při provozu aplikace nastat. Pro generování takových dat je využita knihovna s názvem Faker. Umožňuje generovat data ve tvaru například jmen, IPv4 adres nebo jakýkoliv jiný formát. U náhodných dat lze také určit s jakou

pravděpodobností bude vrácen například boolean nebo null. Do databáze jsou potom data nahrána pomocí nástroje Doctrine Fixtures.

Toto je úsek z třídy AppUserFixture, která slouží pro naplnění databáze náhodnými uživateli. Tato třída zároveň vytvoří i pevně dané uživatele admin a user pro účely testování.

```php
<?php

for ($i = 2; $i < self::AMOUNT; ++$i) {
    $appUser = new AppUser(
        $this->faker->unique()->userName(),
        $this->faker->name(),
        $this->faker->boolean(10),
        $this->randomState()
    );

    $manager->persist($appUser);
    $this->setReference(self::APP_USER_REFERENCE.$i, $appUser);
}
$manager->flush();
```

Obrázek 5.10: Generování testovacích uživatelů.

Podobně je naplněna celá databáze náhodnými daty. Vzhledem k závislostem dat na sobě je nutné ve třídě definovat pořadí spouštění. To zajišťuje funkce `getDependencies()`, ta vrací pole řetězců se jmény tříd, které musí být spuštěny nejdříve.

5.2 Část klient

5.2.1 Vývojové nástroje

Pro správu knihoven v prostředí jazyka JavaScript se používá nástroj NPM. Stahuje balíčky z veřejného repozitáře NPMJS[3]. Balíčky jsou velmi rozmanité kvality. Příkladem toho může být balíček is-even, který pro svou instalaci vyžaduje balíček is-odd a vrací negaci výsledku jeho funkce. Šokující na tom není samotná existence balíčku, ale jeho 154 144 stažení za týden. Požadovaná verze balíčku je rozbalena do složky `node_modules`. Princip fungování je velmi podobný jako u nástroje Composer. Konfigurace je také ve formátu JSON, umožňuje dělit závislosti na vývojové a nutné pro provoz a také zprostředkovává možnost definovat alias příkazu. Hlavní rozdíl je, že NPM neumožňuje řešit automatické načítání tříd, v JavaScriptu totiž není kód složený ze tříd, ale modulů, proto není nutné načítat třídy automaticky. Historicky

[3]https://www.npmjs.com

existovalo více variant jako AMD, UMD, CommonJs [64]. V tomto projektu se používá přístup s názvem ES Modules a tedy příkazy import nebo export.

Pro statickou analýzu kódu je využit nástroj ESLint. Je konfigurovatelný souborem `.eslintrc.cjs`, který určuje pravidla podle kterých se kód opravuje. Pro tuto aplikaci bylo potřeba nainstalovat rozšíření pro Vue 3, prettier a TypeScript. Soubor `.eslintignore` definuje, jaké soubory se nemají kontrolovat. Chová se stejně jako `.gitignore`, ale pro ESLint.

Kód je formátován nástrojem Prettier. Jeho nastavení v souboru `.prettierrc.json` je následující.

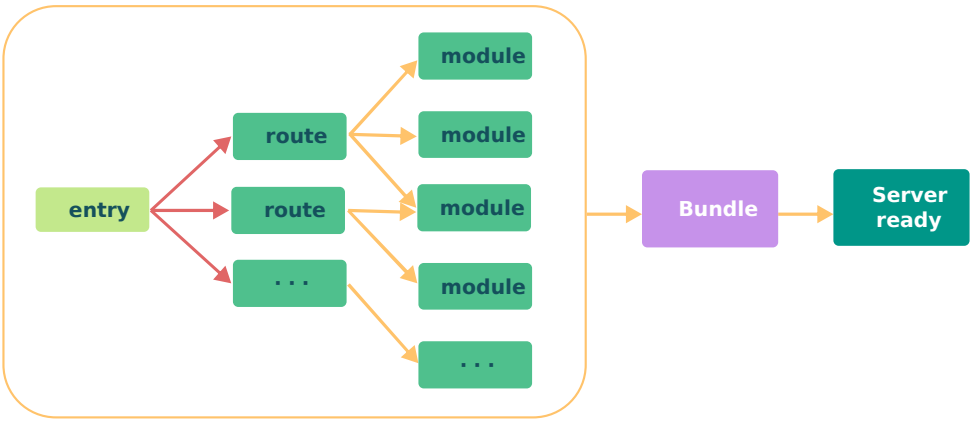
```
{
    "trailingComma": "es5",
    "tabWidth": 4,
    "semi": false,
    "singleQuote": true
}
```

Obrázek 5.11: Nastavení formátování kódu Prettier

Kód tedy neobsahuje středníky, odsazuje se mezerami v délce 4 a k definici řetězců jsou použity jednoduché uvozovky. Souborem `.prettierignore` se opět definují soubory, u kterých není žádoucí provádět formátování.

TypeScript má rovněž svůj konfigurační soubor `tsconfig.json`. Ten je dle doporučení Ionic CLI.

Díky nástroji Vite je možné celou aplikaci sestavit do minimalizovaných souborů, které jsou potom odesílány serverem Nginx do prohlížeče, nebo jsou uložené v mobilní aplikaci. Díky tomu je přeneseno/uloženo jen minimální množství dat. Pokud aplikace využívá dynamické importy pohledů, může Vite rozdělit kód na více složených souborů, které jsou načteny jen v případě potřeby, pokud uživatel přistoupí na danou stránku.



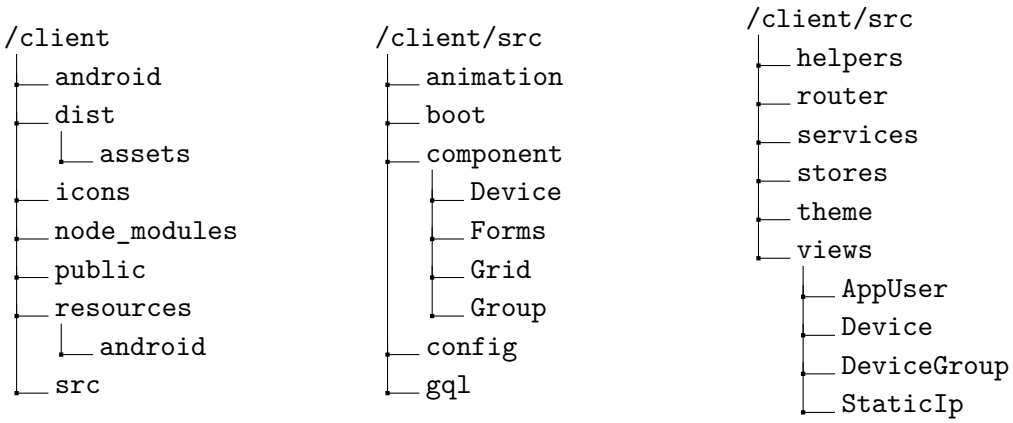
Obrázek 5.12: Sestavení aplikace nástrojem Vite

Hlavní rozdíl oproti staršímu projektu Webpack je rychlost startu vývojového serveru a

načtení stránky při změně. Díky systému výměny modulů za běhu (Hot reload) není nutné po změně kódu obnovovat stránku v prohlížeči, ale změna je ihned viditelná.

5.2.2 Struktura klienta

JavaScriptová aplikace je ve složce client.



Obrázek 5.13: Adresářová struktura části klient.

Důležitý je opět soubor `.gitignore`. Obsahuje složku `node_modules`, obsah složky `dist`, kde jsou vygenerované soubory určené k nahrání na produkční server a hlavně také soubor `src/config/config.ts`, který obsahuje absolutní adresu API, ta se liší podle stroje, na kterém je aplikace spuštěna.

Složka `android` obsahuje vygenerovaný projekt, který lze otevřít v IDE Android Studio. Ve složkách `icons` a `resources` jsou vygenerované ikony s různým rozlišením, které se zobrazují například při otevírání android aplikace. Složka `public` obsahuje soubory dostupné veřejně, které jsou při sestavení nahrány do `dist`.

Ve složce `src` je zdrojový kód samotné aplikace. Struktura aplikace začíná v souboru `index.html`. Obsahuje pouze meta tagy, zcela základní strukturu a hlavně importuje soubor `main.ts`, který nastavuje základní komponenty a vkládá CSS styly stránky pro zpracování nástrojem Vite. Vytváří se v něm instance knihovny Pinia pro správu stavů aplikace (state management), inicializuje se Vue router, který určuje, jaké stránky jsou zobrazeny podle adresy v prohlížeči. A nakonec se spouští samotná Vue aplikace, která je umístěna v souboru `App.vue`.

Po spuštění aplikace je jako první vložen soubor `boot/api.ts` a v něm je vytvořena instance třídy UserAuthService ze souboru `services/UserAuthService.ts`. Ta slouží pro ukládání tokenů pomocí @ionic/storage, které získá z API. Dále poskytuje funkci pro znovu získání JWT přes obnovovací token a nakonec funkci pro smazání úložitě, tedy odhlášení uživatele. Součástí procesu startu je i nastavení knihovny ApolloClient. Je zde implementován mechanismus pro automatické získávání JWT, pokud vyprší jeho platnost. Dále funkce pro zacházení s chybami, které mohou vzniknout buď na straně serveru, nebo při přijetí odpovědi. Nakonec je potřeba inicializovat dočasné úložiště pro data získaná z API.

Při načtení stránky je inicializováno úložiště uživatele. Zkontroluje se, jestli má uživatel v prohlížeči uložené tokeny pro autentifikaci. Pokud nemá, je přesměrován na stránku přihlášení. Pokud je má, odešle se dotaz pro načtení informací o uživateli na API. Pokud je dotaz úspěšný, je uživatel tímto přihlášen v aplikaci. Ihned následuje API dotaz na případné zprávy, které by se mohly uživateli zobrazit. Soubor spravující úložiště uživatele je `stores/userStore.ts`.

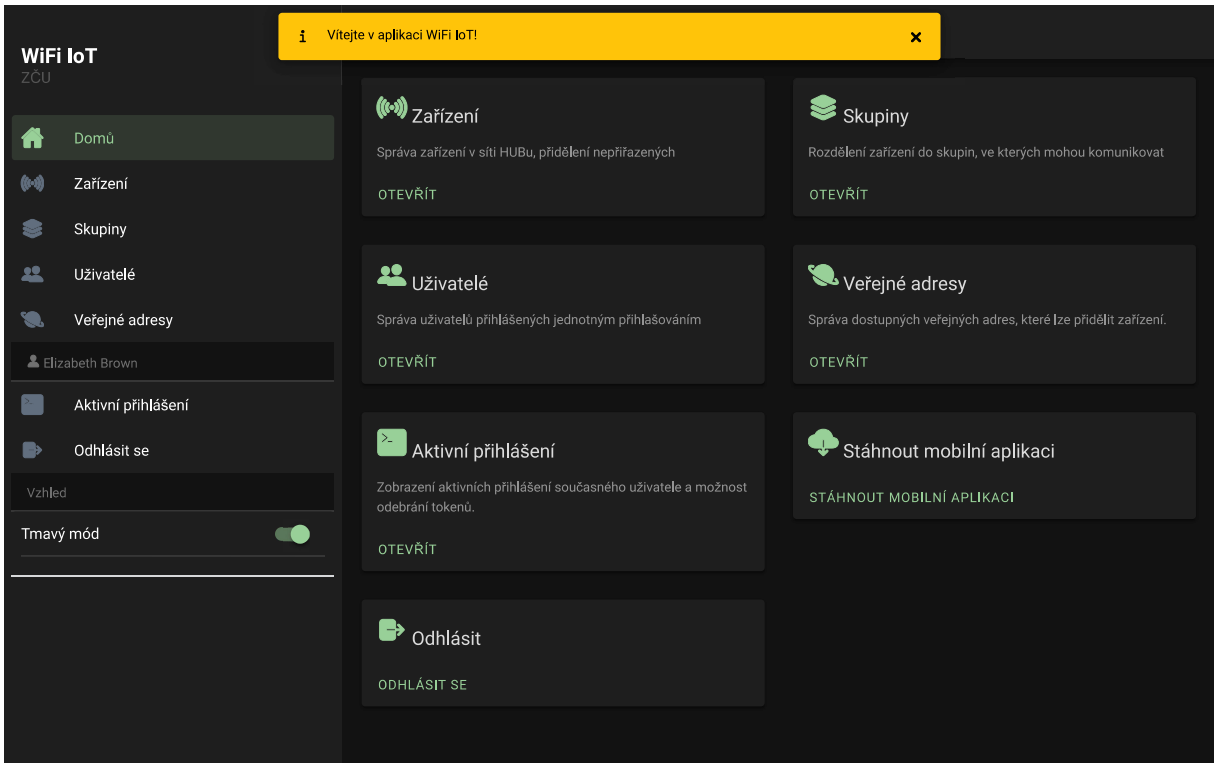
Dále se při načtení stránky inicializuje globální úložiště `stores/globalStore.ts`. To obsahuje informaci o současně zvolené položce v menu, aby mohla být zvýrazněna a informaci o tom, jestli je aktivní tmavý nebo světlý mód aplikace. Při startu je bráno v potaz nastavení prohlížeče prefers-color-scheme.

Po inicializaci úložišť Pinia následuje inicializace routeru pomocí kódu v souboru `router/index.ts`. Zde je rovněž zaregistrována funkce, která se spouští před každým přechodem mezi pohledy (beforeEach). Jejím účelem je ověřit, zda je uživatel přihlášen a má přístup do aplikace. Pokud nemá uživateli se zobrazí zpráva a je přesměrován na přihlašovací stránku. Naopak, pokud se uživatel pokouší přihlásit, ale již má přístup do aplikace povolený, je přesměrován na úvodní stránku. Dále nastavuje současně zvolenou routu do globálního úložiště. Definice jednotlivých pohledů a jejich spojení s adresami je v souboru `router/routeList.ts`. Komponenty jsou dynamicky importovány, takže jsou po sestavení uloženy jako samostatné soubory. Pro zjednodušení práce s odkazy je v souboru `router/route.ts` definován enum, který obsahuje všechny možné pohledy. Základní šablona, do které se vkládají ostatní pohledy a komponenty je App.vue.

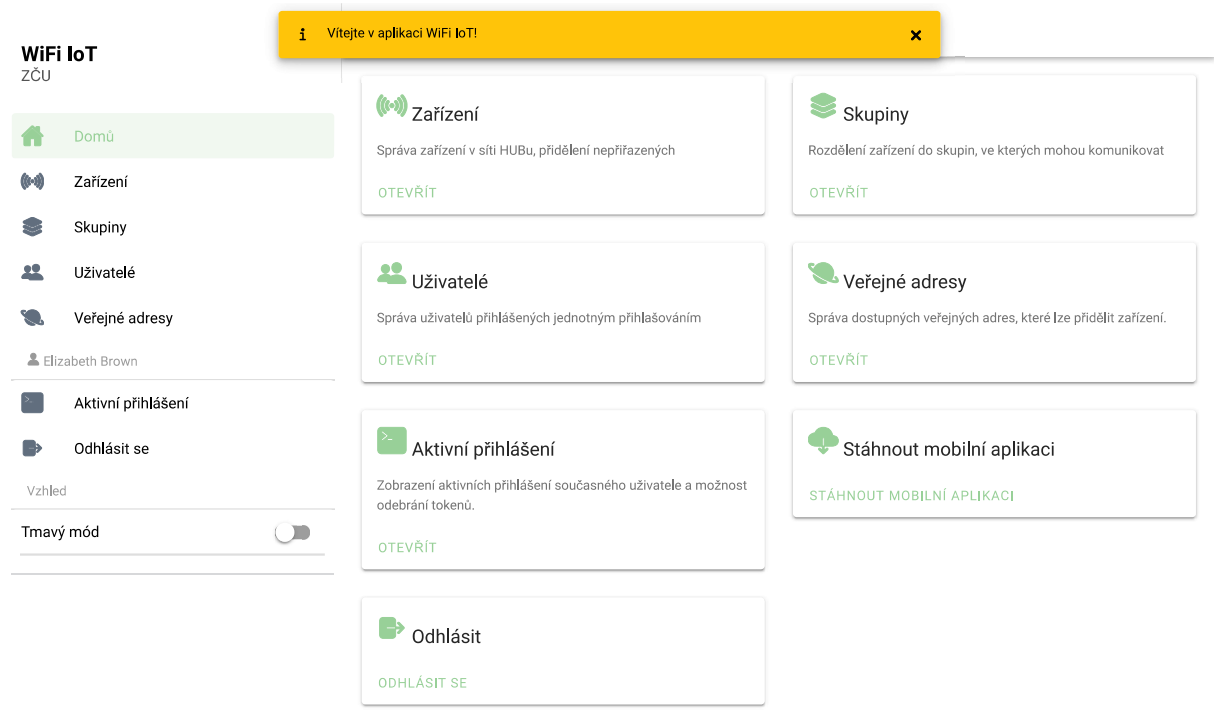
5.2.3 Webová a mobilní aplikace

Pohledy jsou uloženy ve složce `views` a komponenty ze kterých se skládají ve složce `component`.

Aplikace podporuje dva režimy zobrazení a to světlý a tmavý. Porovnání režimů na úvodní stránce webové a mobilní aplikace.



Obrázek 5.14: Úvod, tmavý mód, webová aplikace, přihlášený administrátor

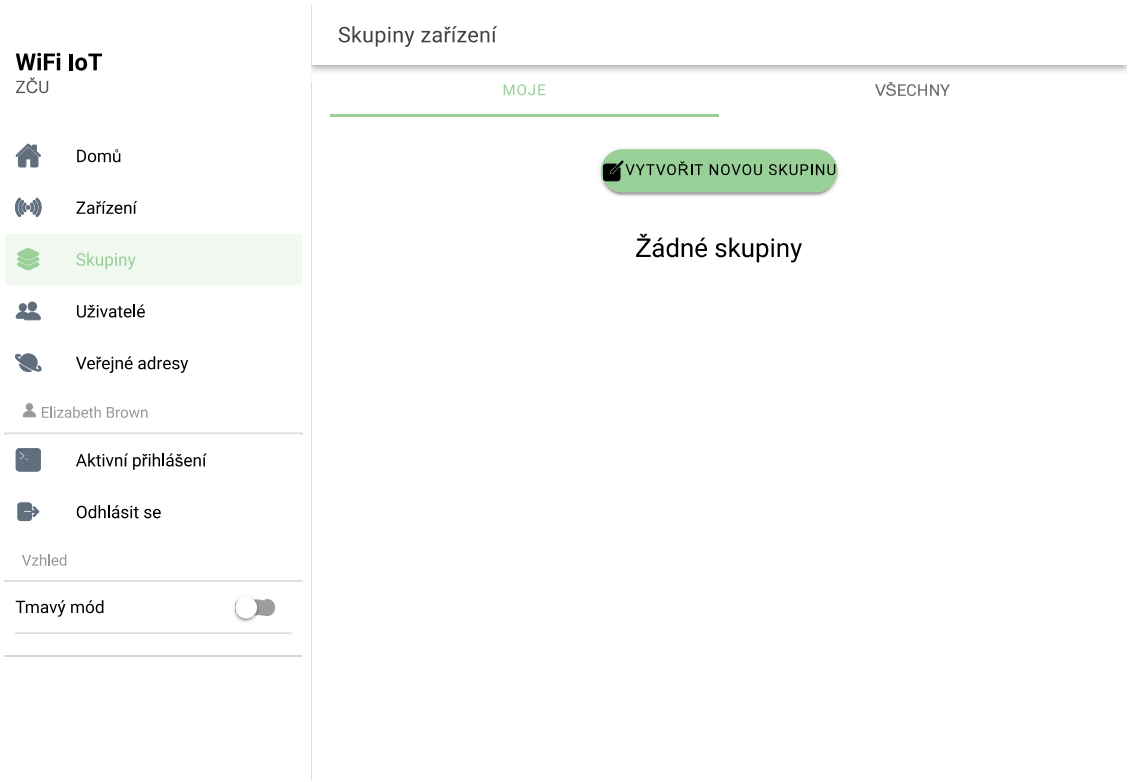


Obrázek 5.15: Úvod, světlý mód, webová aplikace, přihlášený administrátor

Obrázek 5.16: Úvod, tmavý mód, mobilní aplikace, přihlášený administrátor

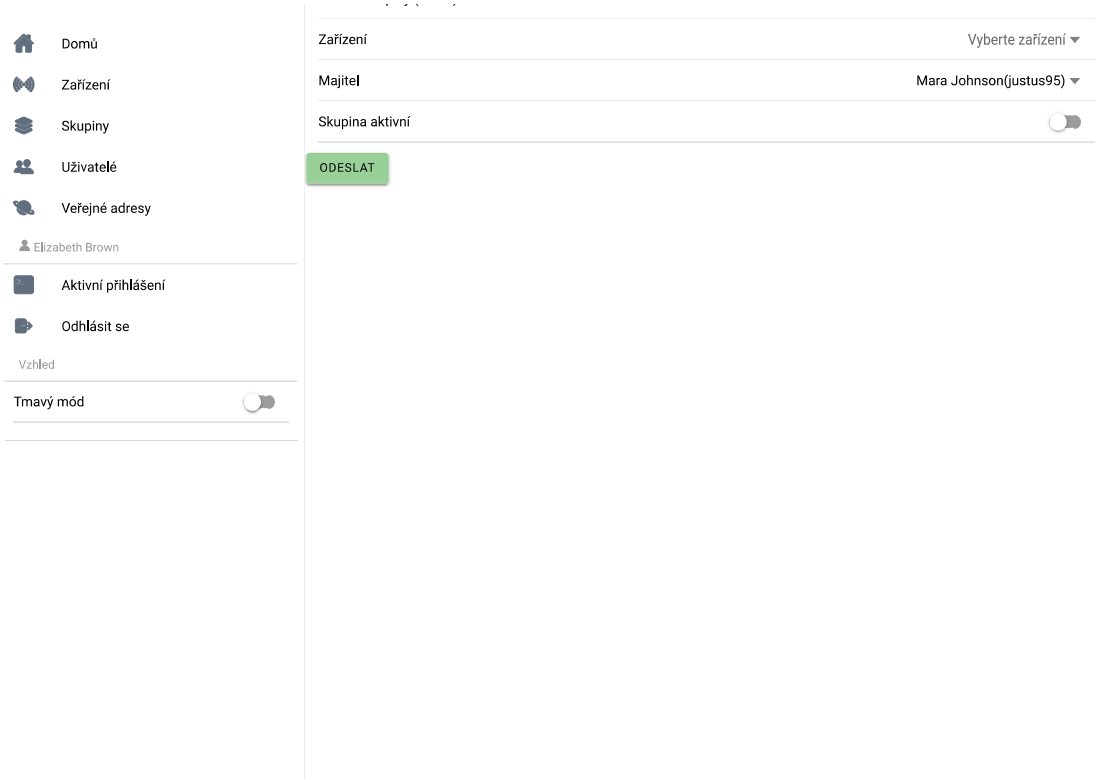
Obrázek 5.17: Úvod, světlý mód, mobilní aplikace, přihlášený administrátor

Úvodní stránka slouží jako rozcestník případně pro stažení mobilní aplikace. Jedná se o pohled UserHomepage. Seznam zobrazených položek se liší, jestli je přihlášený administrátor, nebo uživatel. Následující ukázky budou ve světlém módu. Některé další snímky obrazovky v tmavém módu jsou v příloze B. Nejsou zobrazeny všechny možné stavy uživatelského rozhraní, je doporučeno podívat se do živé aplikace.



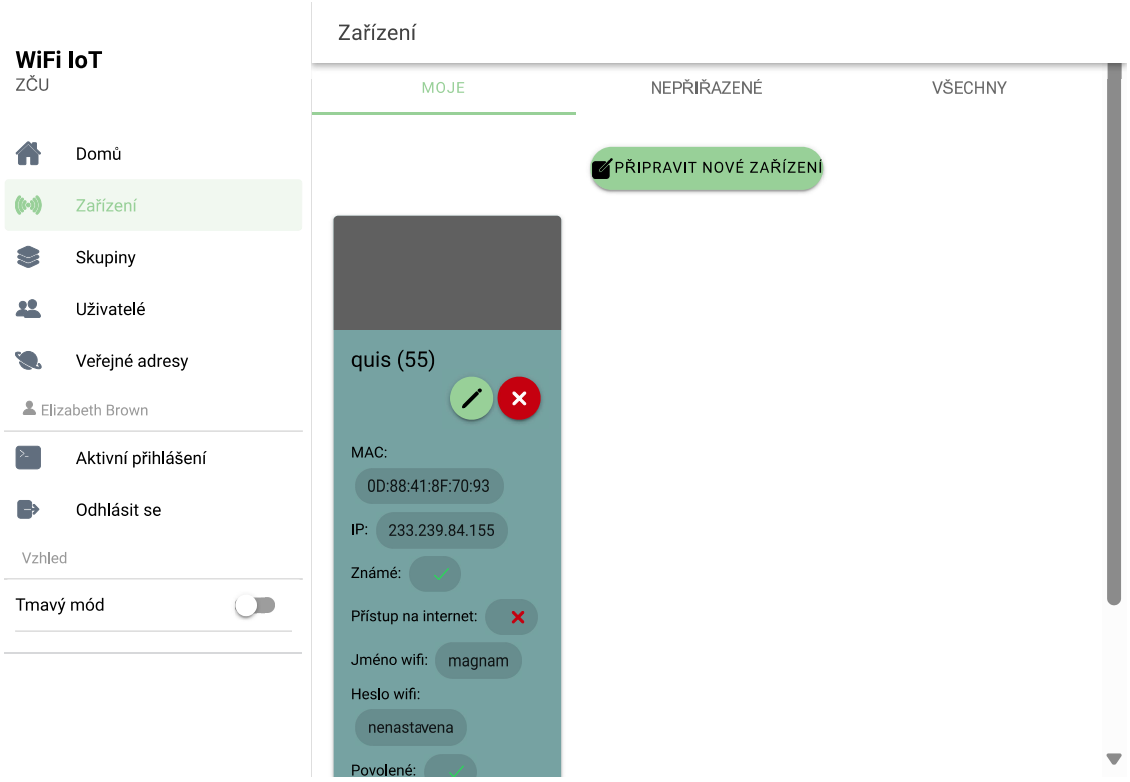
Obrázek 5.18: Moje skupiny, přihlášený administrátor

Pokud nemá administrátor žádné skupiny přiděleny, nejsou zobrazeny. Může si ji buď vytvořit, nebo přidělit ze seznamu všech. Zobrazený je pohled GroupList.

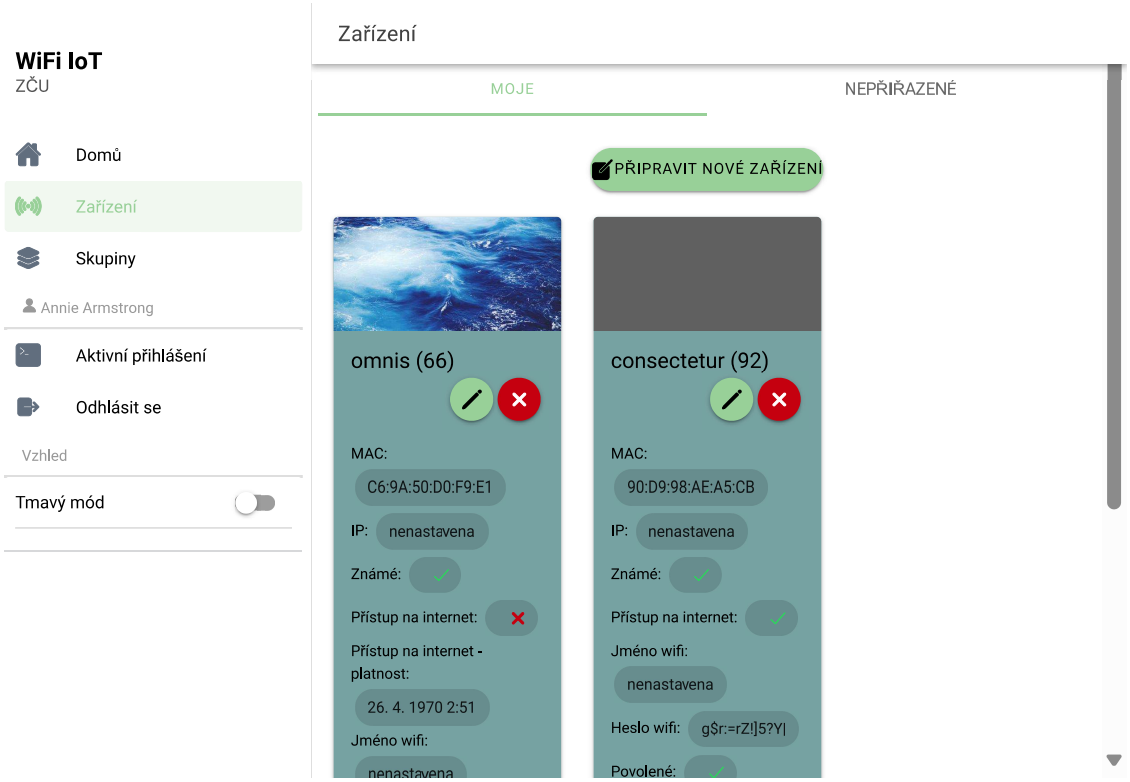


Obrázek 5.19: Úprava skupiny

Zobrazený je pohled GroupOne.

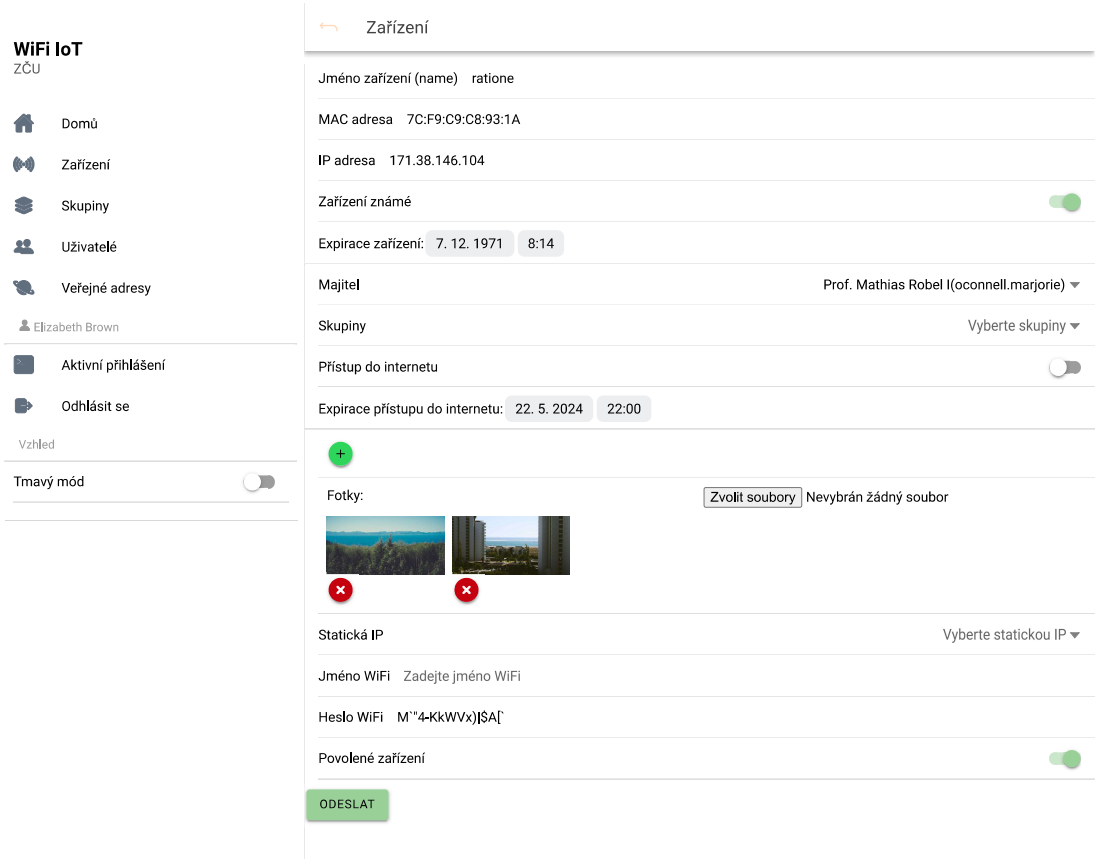


Obrázek 5.20: Moje zařízení, přihlášený administrátor



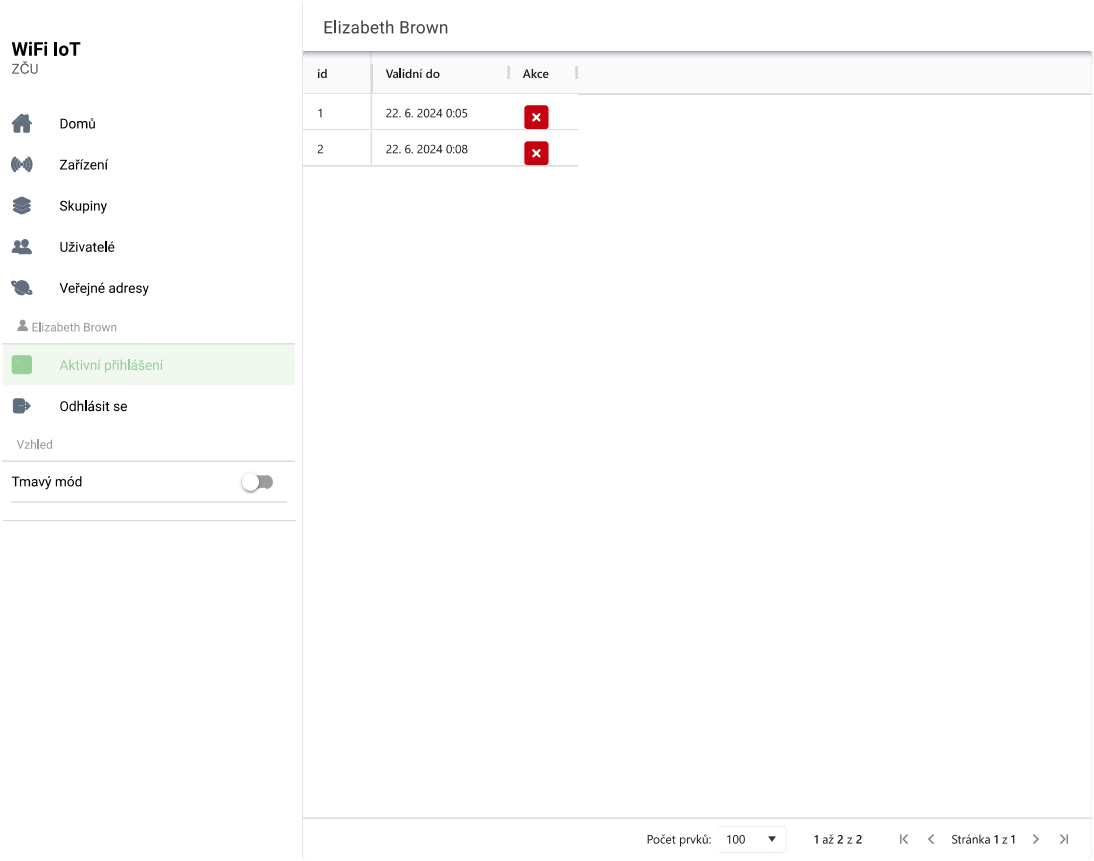
Obrázek 5.21: Moje zařízení, přihlášený uživatel

Zde je viditelný rozdíl mezi administrátorem a uživatelem. Administrátor má možnost zobrazit i všechna zařízení, uživatel pouze svoje nebo nepřiřazená. Zobrazený je pohled DeviceList. Při zobrazení dlouhých seznamů zařízení, nebo skupin je využito techniky nekonečného skrolování (infinity scroll). První požadavek vrátí pouze 6 zařízení, po dosažení spodní hranice stránky je ihned odeslán požadavek na načtení dalších položek. Tím je zajištěna plynulost procházení stránky.



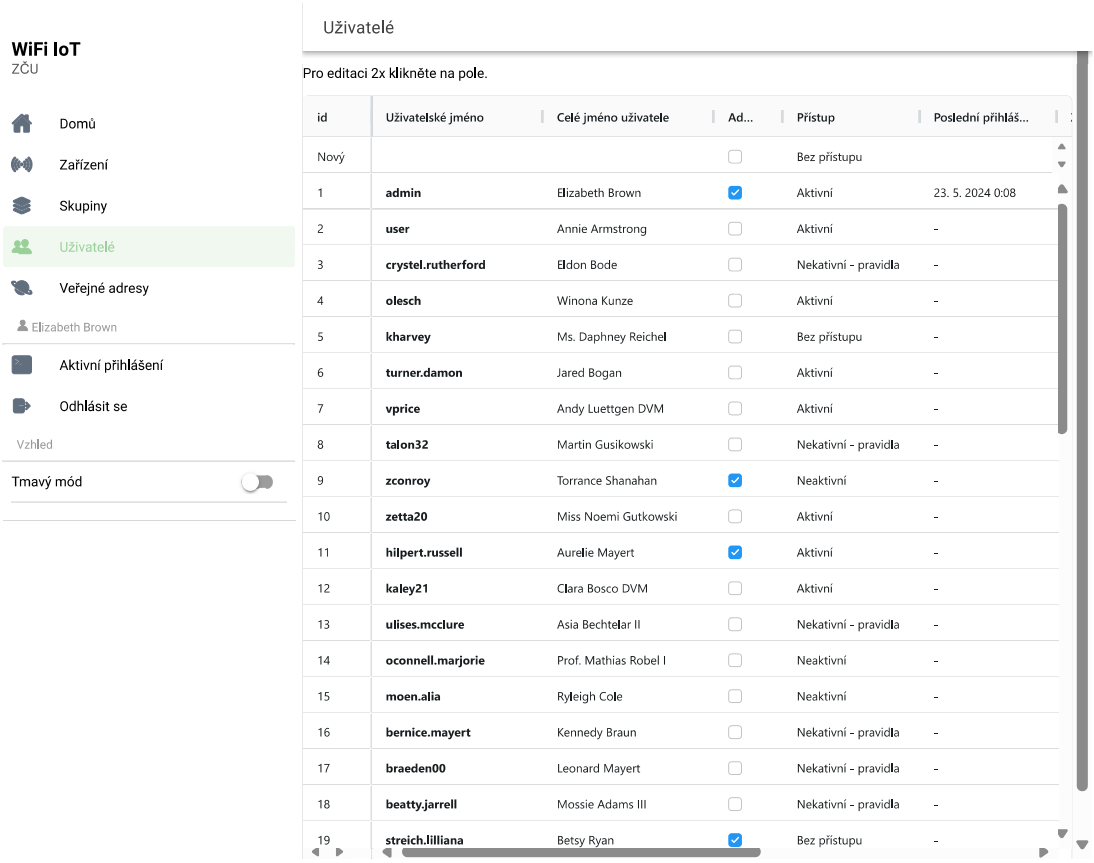
Obrázek 5.22: Úprava zařízení

Pro úpravu zařízení slouží pohled DeviceOne.



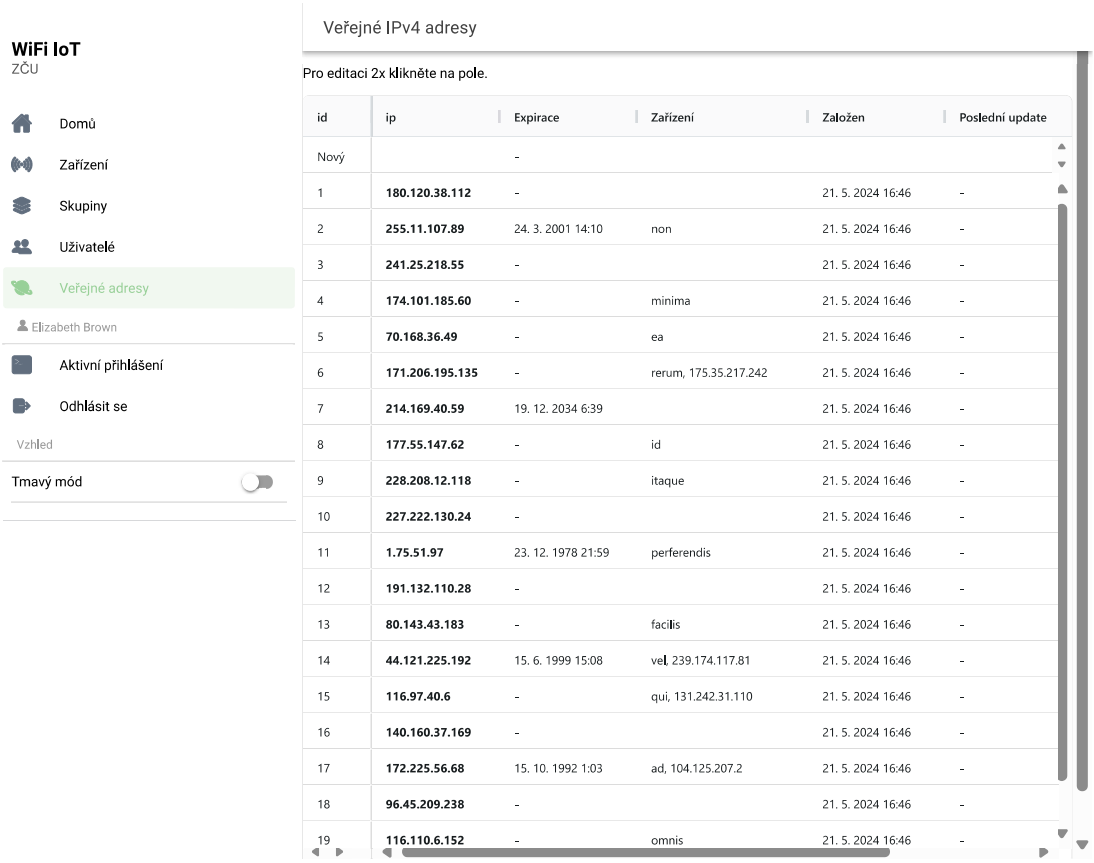
Obrázek 5.23: Aktivní přihlášení

Na stránce s aktivním přihlášením si lze smazat platné obnovovací tokeny a tak tyto zařízení odhlásit. Zobrazený pohled je UserMe.



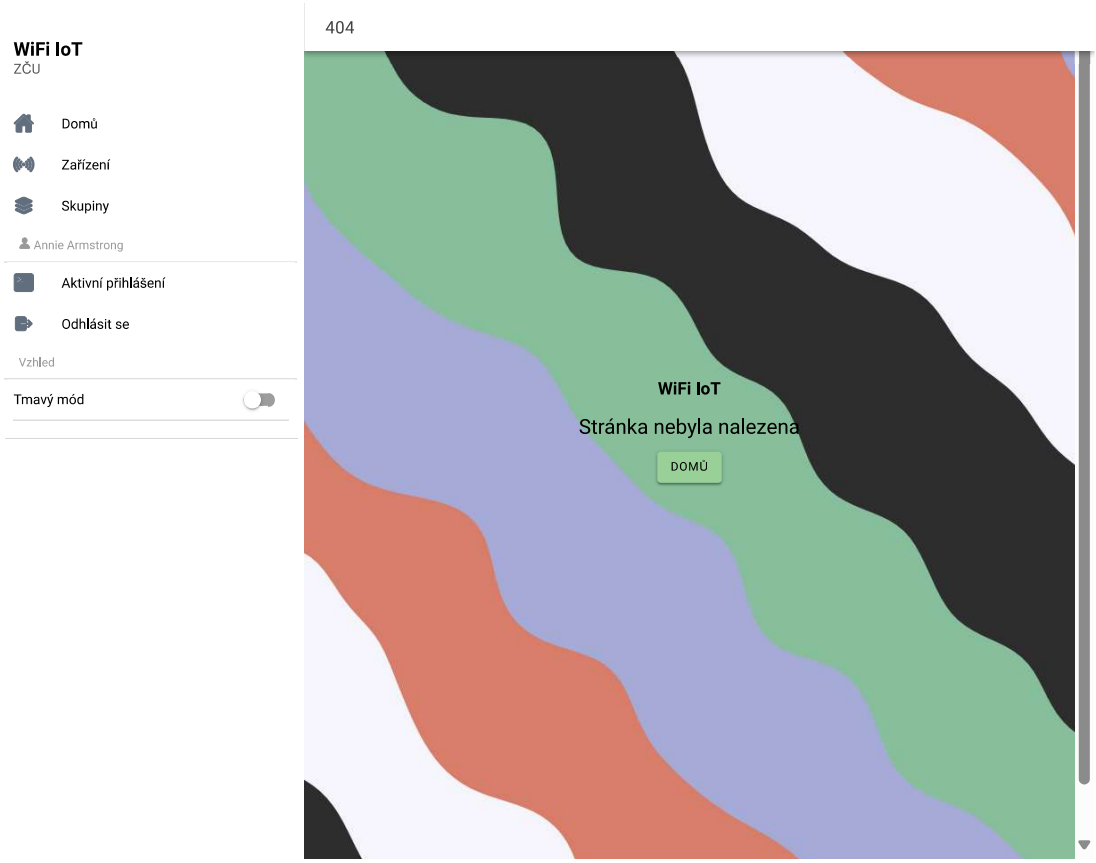
Obrázek 5.24: Administrace uživatelů

Pro zobrazení tabulky je využita knihovna ag-grid-vue3. Přidávat nové záznamy lze vyplněním prvního řádku. Upravovat existující lze dvojitým kliknutím, nebo poklepáním na telefonu. Z tohoto pohledu je umožněno zobrazit i zařízení a skupiny, která daný uživatel vlastní. Zobrazený pohled je UserList.



Obrázek 5.25: Administrace veřejných IPv4 adres

K zobrazení administrace IPv4 slouží pohled StaticIpList.



Obrázek 5.26: Chybová stránka 404

Animace na stránce 404 pochází ze stránek codepen.io [4]. Byla kompletně přepsána do TS i s knihovnou SimplexNoise, kterou využívá. Tyto soubory jsou ve složce `animation`. K jejímu zobrazení slouží pohled ErrorNotFound.

Grafický návrh loga pochází ze stránek Flaticon[5] a byl upraven. Pro vektorové ikony byla využita knihovna ionicons. Mobilní aplikace byla vytvořena pomocí knihoven @capacitor/android, @capacitor/cli příkazem `ionic capacitor build android` a zkompilována v Android Studiu.

5.2.4 GraphQL

Pro načítání dat z GraphQL API je využita knihovna @apollo/client. Tato knihovna zajišťuje odesílání dat na API s autentizačním řetězcem. Po obdržení JSON odpovědi je převedena na JS objekty. Dále zajišťuje uchovávání dat v mezipaměti. Využití této knihovny s Vue umožňuje balíček @vue/apollo-composable. Knihovna má i oficiální rozšíření do VSCode, které umí doplňovat při psaní dotazů a dokonce dotazy spouštět přímo z vývojového prostředí. Nahrávání fotek zajišťuje apollo-upload-client, který dokáže odeslat multiform data přes GraphQL dotaz.

Kód pro upload fotky může vypadat takto.

[4] https://codepen.io/trajektorijus/pen/mdeBYrX
[5] https://www.flaticon.com/free-icons/iot

```
createDevicePhoto({
    devicePhoto: {
        deviceId: route.params.id,
            file: object.target.files[0],
            displayOrder: 0,
        },
        context: {
            hasUpload: true,
         },
    })
```

Obrázek 5.27: Upload fotky při úpravě zařízení.

Vypsán je pouze obsah funkce ze souboru `DeviceOne.vue`, která je volána při změně pole pro nahrávání. V `object.target.files[0]` jsou informace o souboru a jeho binární podoba.

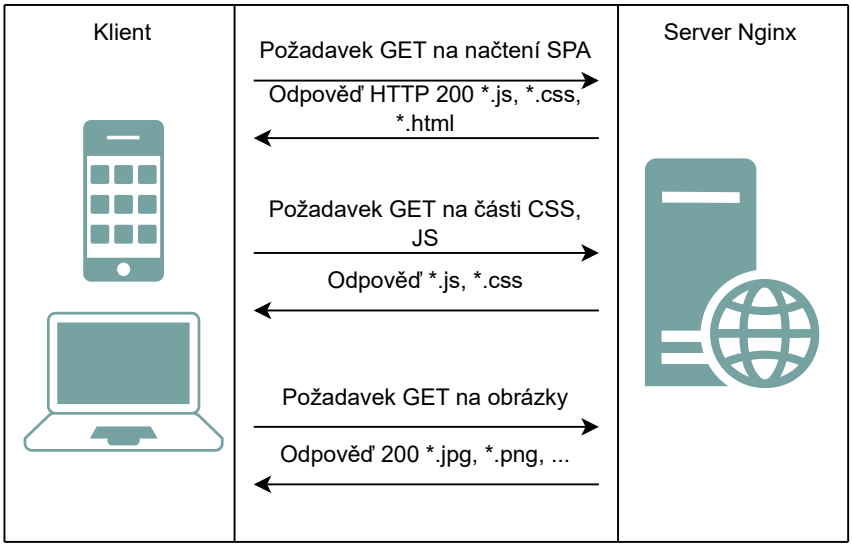
Aby bylo možné používat typy přímo definované ve schématu GraphQL je využit balíček @graphql-codegen/cli, který je dokáže dynamicky převést schéma do TS typů. Je konfigurován souborem `codegen.ts`.

5.3 Komunikace mezi serverem, klientem a zabezpečení

Při prvním přístupu na adresu stránky přes protokol HTTPS je odeslán GET požadavek na server Nginx, který vrátí index.html. Po jeho načtení jsou poté odeslány další požadavky na minifikované soubory *.css a *.js.

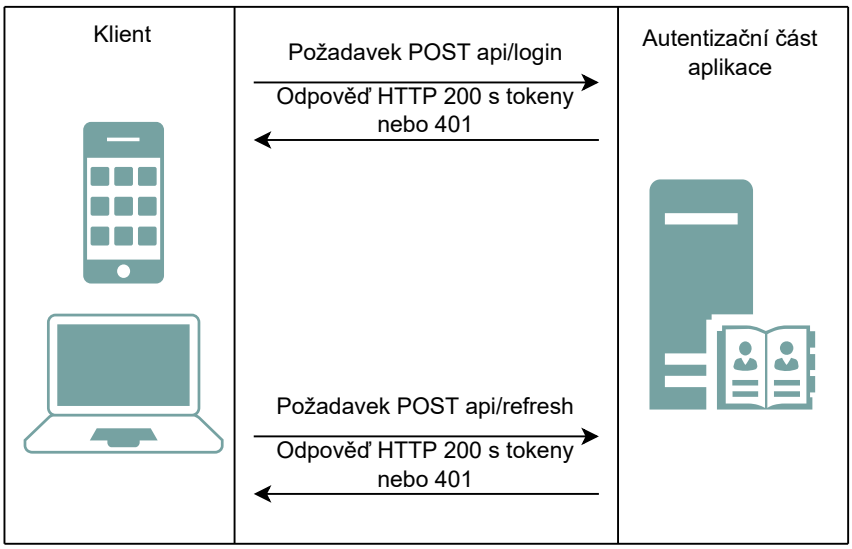
Při prohlížení aplikace mohou být načteny další samostatné soubory *.js a *.css. Toto je příznivé pro rychlost prvního načtení stránky, kdy není vše sbaleno do jednoho balíku. V případě mobilní aplikace odpadají tyto požadavky, protože soubory jsou uloženy přímo v telefonu.

Při zobrazení stránky, která obsahuje obrázky, jako například seznam zařízení, jsou tyto obrázky postupně načítány (lazy load). Přístup k obrázkům není podmíněn autorizací uživatele. Je to z toho důvodu, že po nahrání obrázku je jeho název doplněn o velmi dlouhý řetězec pomocí PHP funkce uniqid. Tím je název prakticky nemožné odhadnout. Díky tomuto přístupu je také možné vzít odkaz na fotku a například ji někomu odeslat, aniž by ji musel uživatel stahovat.



Obrázek 5.28: Požadavky klienta na webový server Nginx

Pokud chce uživatel přistupovat k obsahu, musí se autorizovat. Samotná JavaScriptová aplikace, která je nahrána v předchozím kroku, neobsahuje žádná data. Ta je potřeba načíst až z API na základě požadavků. Autentizace neprobíhá přes GraphQL API, ale přes obyčejný POST požadavek. Uživatel odešle své přihlašovací údaje. Server ověří uživatele, vygeneruje JWT a podepíše ho. Dále server vygeneruje token, který slouží pro obnovu JWT. Ten je uložen do databáze. Oba tokeny jsou odeslány uživateli.



Obrázek 5.29: Požadavky klienta na autentizační část aplikace

JWT má nastavenou mnohem kratší platnost než token pro obnovu. Nyní je například nastavena platnost na 900 sekund. Pokud platnost vyprší, server vrátí odpověď 401 a klient si může pomocí obnovovacího tokenu zažádat o oba nové tokeny.

Tento přístup má výhodu v tom, že není nutné při každém dotazu počítat složité kryptografické funkce (například při odesílání zahashovaného hesla) nebo přistupovat do

databáze. Pouze se dešifruje JWT a ověří se jeho platnost. Jednou za 900 sekund je nutné přistoupit do databáze a přegenerovat tokeny, což není velká zátěž.

Další velkou výhodou tohoto řešení je, že pokud je komunikace mezi uživatelem a serverem odposlechnuta, útočník s velkou pravděpodobností získá pouze JWT, který má krátkou platnost, nebo obnovovací token, který je v okamžiku jeho přijetí na serveru již neplatný, protože je vygenerován nový.

JWT není příliš dlouhý a nepředstavuje tak velkou datovou zátěž. Skládá se z JSON objektů hlavičky, dat a podpisu. Tyto objekty jsou zabaleny pomocí algoritmu base64.

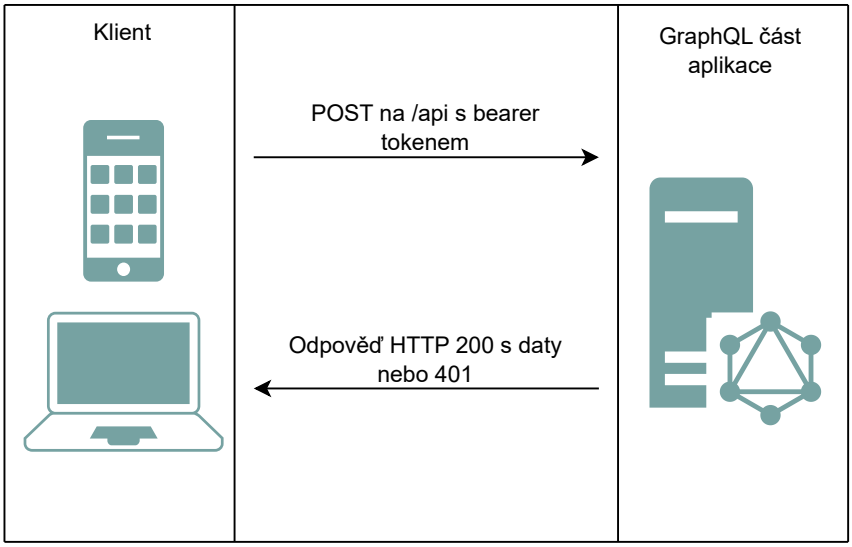
```
{
    "typ": "JWT",
    "alg": "RS256"
}
```

Obrázek 5.30: Příklad JWT header

```
{
  "iat": 1715282787,
  "exp": 1745282787,
  "username": "admin"
}
```

Obrázek 5.31: Příklad JWT data

Podpis je vytvořen takto: `HMACSHA256(base64UrlEncode(header) + "."+ base64UrlEncode(payload), PUBLIC_KEY, PRIVATE_KEY).`



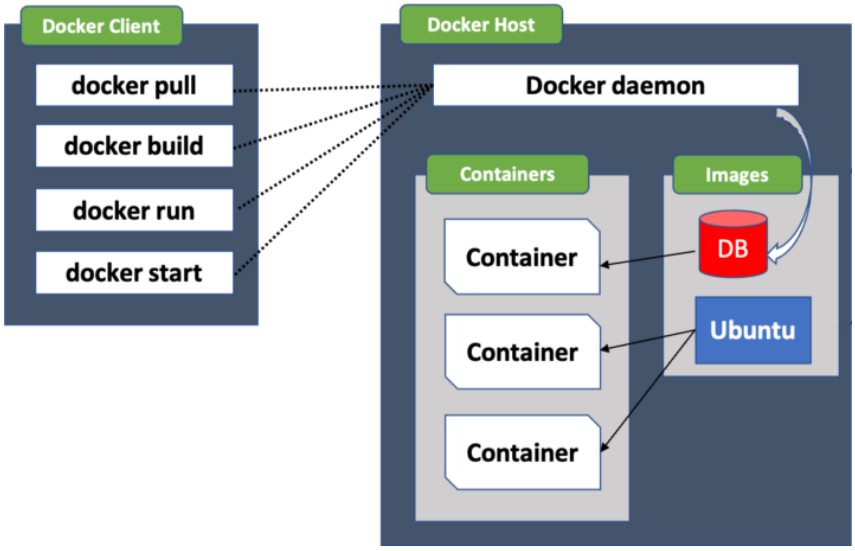
Obrázek 5.32: Požadavky klienta na GraphQL část aplikace

Pokud chce uživatel získat data z GraphQL API musí s dotazem poslat i JWT v hlavičce Authorization: Bearer. Token je nutné posílat při každém dotazu.

5.4 Prostředí pro vývoj a nasazení

Pro vývoj jsou využívána prostředí, která jsou spuštěná v kontejnerech pomocí nástroje s názvem Docker. Oddělení vývojových prostředí do kontejnerů (containers) má tu výhodu, že na počítač

není nutné instalovat veškerý požadovaný software v definovaných verzích. To je v případě potřeby práce na více rozdílných projektech nemožné, protože mohou být požadovány různé verze jednoho programu. Zároveň je opakovaná instalace prostředí mnohem méně náročná, protože veškeré instalační kroky jsou obsaženy v obrazech (image).



Obrázek 5.33: Architektura nástroje Docker. Převzato z [65], upraveno.

Popis kontejnerů je v souboru `docker-compose.yml`. Ve složce `docker` je připravený doporučený příklad. Soubor je uveden v .gitignore., protože pro každý vývojový počítač mohou být mírně odlišné požadavky na jednotlivé kontejnery. Například databáze může být spuštěna přímo na počítači. Může být žádoucí vytvořit napojení na ssh-agent, aby bylo možné používat git přímo z docker kontejneru. Dalším důvodem může být nutná změna čísla portů, protože je má vývojář obsazené jinými nástroji.

Standardně je aplikace složena z kontejnerů wifi_iot_db, wifi_iot_client_dev, wifi_iot_api_dev, wifi_iot_voyager, wifi_iot_graphi a wifi_iot_adminer.

Kontejner wifi_iot_db obsahuje databázi MariaDB ve verzi 11.3. Může být bez změny konfigurace vyměněn i za MySQL 8.4. Hesla a jména databáze mají předdefinovanou hodnotu wifiiot. Na port databáze 3306 lze přistoupit přes port 8106.

Kontejner wifi_iot_client_dev je založen na obraze node:20-slim. Dále obsahuje příkazy pro instalaci nástrojů npm, nodemon, @ionic/cli a @capacitor/assets. Tyto nástroje jsou nutné pro vývoj aplikace, instalaci závislostí a sestavení výsledné aplikace. HTTP server je spuštěn na portu 8101.

Kontejner wifi_iot_api_dev je založen na obraze php:8.3-fpm. Je doplněn o příkazy pro instalaci PHP rozšíření bcmath pro matematické výpočty, pdo_mysql pro připojení k databázi, zip pro kompresi souborů, opcache pro dočasné soubory, intl pro lokalizaci a gd pro práci s obrázky. Dále obsahuje instalaci nástroje Composer a Symfony CLI. HTTP server je spuštěn na portu 8102.

Kontejner wifi_iot_voyager slouží k vizualizaci GraphQL API a zobrazení jeho dokumentace. Je dostupný na portu 8103. Kontejner wifi_iot_graphi byl využit pro testování

dotazů na GraphQL API. Je dostupný na portu 8104. Ke stejnému účelu byl využit i nástroj Postman, který ale není třeba spouštět z kontejneru. Poslední kontejner wifi_iot_adminer slouží jako nástroj pro náhled do databáze MariaDB. Je dostupný na portu 8105.

5.5 Prostor pro vylepšení

Na straně klienta jsou nyní načteny všechny údaje v seznamu IPv4 adres a uživatelů, což nezpůsobuje žádné výkonnostní problémy. Stránkování je možné pouze na straně klienta. Sofistikovanějším přístupem by bylo implementovat stránkování na straně serveru, takže by se načítala pouze data, která jsou aktuálně zobrazena. Dále může docházet k nesprávnému zobrazení popisků stránkování na těchto stránkách, které při nižším rozlišení mohou přetéct.

Pro zlepšení uživatelského pohodlí v mobilní aplikaci by bylo možné implementovat nativní API pro fotografování. Momentálně je možné vybrat fotografii ze souborového systému. Dále by mobilní aplikace mohla kontrolovat zprávy na pozadí a zobrazovat je jako notifikace.

Vylepšení na straně serveru by se mohlo týkat hlubšího nastavení uživatelských oprávnění. V databázi je připravené pole pro přiřazení rolí uživatelům, ale není implementován systém, který by tyto role aplikoval. Dále je potřeba doimplementovat přihlášení přes ORION login, napojení na iPSK API a úpravu iptables pravidel podle nastavených oprávnění a skupin. Pokud se uživatel přihlásí, bylo by možné pro lepší identifikaci místa přihlášení zaznamenávat do tabulky s obnovovacími tokeny také prohlížeč a IP adresu, ze které požadavek proběhl.

6

Závěr

V kapitole 2 byly rozebrány požadavky kladené zařízeními internetu věcí a bezpečnostní rizika, která souvisí s provozem těchto zařízení. Třetí kapitola se zabývá obecně sítí Wi-Fi a bezpečnostními mechanismy, které je možné realizovat. Protnutím těchto vlastností a požadavků vznikl v kapitole 4 návrh systému, který zohledňuje možné využití systému a způsob jeho provozu. Jsou zde popsány realizační možnosti s jednoduchými ukázkami kódu. Nejedná se o vyčerpávající popis. Nové možnosti vznikají v této dynamické oblasti prakticky každý den.

Pro realizaci byla vybrána topologie jednostránkové aplikace. Pro tvorbu serveru byl zvolen běžně používaný jazyk PHP s vývojovou platformou Symfony. Jako nejvhodnější řešení pro uložení dat byla vybrána relační databáze, v tomto případě MariaDb. Správu databáze v PHP zajišťuje ORM Doctrine. Jako nejvhodnější technologie pro realizaci klientské strany byla zvolena oblíbená vývojová platforma Vue doplněná o Ionic. Kód je psaný v jazyce TypeScript. Díky tomu je možné vytvořit webovou i mobilní aplikaci. Komunikaci mezi klientem a serverem realizuje moderní a úsporné rozhraní GraphQL.

Vývoj systému byl podpořen mnoha nástroji, které zlepšují kvalitu kódu. Systém byl realizován za použití optimálních vybraných vývojových nástrojů. Případné návrhy k vylepšení jsou rozebrány v kapitole 5.5.

Byl navržen a implementován snadno rozšiřitelný a udržovatelný systém s využitím nejmodernějších technologií a důrazem na bezpečnost. Systém nabízí možnost snadného použití pro širokou škálu uživatelů.

Kompletní zdrojové kódy jsou přiloženy v elektronické podobě bez nainstalovaných nutných závislostí, zprovoznění projektu lze provést pomocí postupu v příloze C. Běžící testovací verze je dostupná na serveru ZČU na adrese https://sulis198.zcu.cz. Případně lze pomocí postupu v příloze spustit na libovolném OS založeném na systému Debian.

Reference, použitá literatura

1. *ITU* [online]. [cit. 2024-05-14]. Dostupné z: https://www.itu.int/en/ITU-T/gsi/iot/Pages/default.aspx.
2. WELOTEC. *IoT gateways and their central role in the Internet of Things* [online]. [cit. 2024-05-14]. Dostupné z: https://www.welotec.com/iot-gateways-and-their-central-role-in-the-internet-of-things/.
3. GILLIS, Alexander S. *IoT devices (internet of things devices)* [online]. [cit. 2024-05-14]. Dostupné z: https://www.techtarget.com/iotagenda/definition/IoT-device.
4. PRATT, Mary K. *Top 12 most commonly used IoT protocols and standards* [online]. 2023-07-12. [cit. 2024-05-14]. Dostupné z: https://www.techtarget.com/iotagenda/tip/Top-12-most-commonly-used-IoT-protocols-and-standards.
5. HAMAM, Muhammad Shafiq; Zhaoquan Gu; Omar Cheikhrouhou; Wajdi Alhakami; Habib. The Rise of "Internet of Things": Review and Open Research Issues Related to Detection and Prevention of IoT-Based Security Attacks. *Wireless Communications and Mobile Computing*. 2022, roč. 2022, č. 8669348, s. 12. Dostupné z DOI: https://doi.org/10.1155/2022/8669348.
6. *ChirpStack, open-source LoRaWAN® Network Server* [online]. [cit. 2024-05-14]. Dostupné z: https://www.chirpstack.io/.
7. *ThingsBoard Open-source IoT Platform* [online]. [cit. 2024-05-14]. Dostupné z: https://thingsboard.io/.
8. *OpenRemote v3* [online]. [cit. 2024-05-14]. Dostupné z: https://github.com/openremote/openremote.
9. BAKHSHALIYEVA, E. Firdus; R. Aghababayev; V. Aliyev; G. Mustafayeva; R. Mayilov; I. Sardarova; S. WiFi from past to today, consequences that can cause and measures of prevention from them, WiFi security protocols. *E3S Web of Conferences*. 2024, roč. 474, č. 02004, s. 12. Dostupné z DOI: https://doi.org/10.1051/e3sconf/202447402004.
10. ALLIANCE, Wi-Fi. *Generational Wi-Fi User Guide* [online]. [cit. 2024-05-14]. Dostupné z: https://www.wi-fi.org/system/files/Generational_Wi-Fi_User_Guide_202304.pdf.

11. PHILLIPS, Gavin. *The Most Common Wi-Fi Standards and Types, Explained* [online]. 2023-12-06. [cit. 2024-05-14]. Dostupné z: https://www.makeuseof.com/tag/understanding-common-wifi-standards-technology-explained/.

12. CISCO. *What Is MU-MIMO?* [online]. [cit. 2024-05-14]. Dostupné z: https://www.cisco.com/c/en/us/products/wireless/what-is-mu-mimo.html.

13. FITZPATRICK, Jason. *The Difference Between WEP, WPA, and WPA2 Wi-Fi Passwords* [online]. 2016-09-21. [cit. 2024-05-14]. Dostupné z: https://www.howtogeek.com/167783/htg-explains-the-difference-between-wep-wpa-and-wpa2-wireless-encryption-and-why-it-matters/.

14. ALLEN, Jennifer. *What Is WPS and How Does It Work?* [online]. 2020-05-14. [cit. 2024-05-14]. Dostupné z: https://www.lifewire.com/what-is-wps-4842308.

15. LORENC., D. Rohleder; V. 802.1X - autentizace v počítačových sítích. *Zpravodaj ÚVT MU*. 2008, roč. XIX, č. 1, s. 2–4.

16. *Espressif* [online]. [cit. 2024-05-14]. Dostupné z: https://www.espressif.com/en.

17. *esphome* [online]. 2022-12-27. [cit. 2024-05-14]. Dostupné z: https://github.com/esphome/issues/issues/3971.

18. *esphome* [online]. [cit. 2024-05-14]. Dostupné z: https://gist.github.com/Matheus-Garbelini/2cd780aed2eddbe17eb4adb5eca42bd6.

19. *WFI32E02UC* [online]. [cit. 2024-05-14]. Dostupné z: https://www.microchip.com/en-us/product/WFI32E02UC.

20. *Mini Smart Wi-Fi Plug, Energy Monitoring Tapo P110M* [online]. [cit. 2024-05-14]. Dostupné z: https://static.tp-link.com/upload/product-overview/2023/202308/20230807/Tapo%20P110M(EU)1.0%20datasheet.pdf.

21. *Identity PSK Feature Deployment Guide* [online]. 2021-12-16. [cit. 2024-05-14]. Dostupné z: https://www.cisco.com/c/en/us/td/docs/wireless/controller/technotes/8-5/b_Identity_PSK_Feature_Deployment_Guide.html.

22. *Identity PSK Manager for Cisco ISE* [online]. [cit. 2024-05-14]. Dostupné z: https://github.com/CiscoDevNet/iPSK-Manager.

23. *1:1 NAT* [online]. [cit. 2024-05-14]. Dostupné z: https://docs.netgate.com/pfsense/en/latest/nat/1-1.html.

24. MAKHIJA, Ravi. *11 Different Types of Web Applications* [online]. 2020-06-26. [cit. 2024-05-14]. Dostupné z: https://www.gurutechnolabs.com/types-of-web-applications/.

25. BHATT, Tuhin. *Web Applications Types — Examples of Web Apps + Benefits and Challenges* [online]. 2023-12-27. [cit. 2024-05-14]. Dostupné z: https://www.intelivita.com/blog/types-of-web-applications/.

26. PATEL, Bhaval. *9 Different Types of Web Applications with Use Cases* [online]. 2024-04-24. [cit. 2024-05-14]. Dostupné z: https://www.spaceotechnologies.com/blog/types-of-web-applications/.

27. *PHP* [online]. [cit. 2024-05-14]. Dostupné z: https://www.php.net/manual/en/intro-whatis.php.

28. *wordpress* [online]. [cit. 2024-05-14]. Dostupné z: https://wordpress.com/.

29. *prestashop* [online]. [cit. 2024-05-14]. Dostupné z: https://prestashop.com/.

30. *JavaScript* [online]. [cit. 2024-05-14]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/javascript.

31. *node.js* [online]. [cit. 2024-05-14]. Dostupné z: https://nodejs.org/en/about.

32. KANERIYA, Tejas. *12 Best Nodejs Frameworks for App Development in 2024* [online]. 2021-01-05. [cit. 2024-05-14]. Dostupné z: https://www.simform.com/blog/best-nodejs-frameworks/.

33. *The Python Tutorial* [online]. 2024-05-14. [cit. 2024-05-14]. Dostupné z: https://docs.python.org/3/tutorial/index.html.

34. *TIOBE Index for May 2024* [online]. 2024-05-01. [cit. 2024-05-14]. Dostupné z: https://www.tiobe.com/tiobe-index/.

35. ALI, Inshal. *10 Best PHP Frameworks for Web Development in 2023* [online]. [cit. 2024-05-14]. Dostupné z: https://www.cloudways.com/blog/best-php-frameworks/.

36. ŠAFAŘÍK, Daniel. *Nejpoužívanější programovací jazyky pro backend webu* [online]. 2021-09-29. [cit. 2024-05-14]. Dostupné z: https://www.zonercloud.cz/magazin/nejpouzivanejsi-programovaci-jazyky-pro-backend-webu.

37. *Spring Boot* [online]. [cit. 2024-05-14]. Dostupné z: https://spring.io/projects/spring-boot.

38. *rust* [online]. [cit. 2024-05-14]. Dostupné z: https://www.rust-lang.org/.

39. *Rocker* [online]. [cit. 2024-05-14]. Dostupné z: https://rocket.rs/.

40. *What is ASP.NET?* [online]. [cit. 2024-05-14]. Dostupné z: https://dotnet.microsoft.com/en-us/learn/aspnet/what-is-aspnet.

41. *Go* [online]. [cit. 2024-05-14]. Dostupné z: https://go.dev/doc/.

42. KAUR, Taranjit. *Mastering Database Selection: A Guide to Choosing the Right Database for Your Application.* [online]. 2023-08-02. [cit. 2024-05-14]. Dostupné z: https://medium.com/plumbersofdatascience/mastering-database-selection-a-guide-to-choosing-the-right-database-for-your-application-f0f7009daf29.

43. *MongoDB vs. MySQL Differences* [online]. [cit. 2024-05-14]. Dostupné z: https://www.mongodb.com/compare/mongodb-mysql.

44. *WebAssembly* [online]. [cit. 2024-05-14]. Dostupné z: https://developer.mozilla.org/en-US/docs/WebAssembly.

45. *HTML: HyperText Markup Language* [online]. [cit. 2024-05-14]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTML.

46. *CSS basics* [online]. [cit. 2024-05-14]. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/CSS_basics.

47. *First-class Function* [online]. [cit. 2024-05-14]. Dostupné z: https://developer.mozilla.org/en-US/docs/Glossary/First-class_Function.

48. *NPM* [online]. [cit. 2024-05-14]. Dostupné z: https://docs.npmjs.com/about-npm.

49. MICROSOFT. *typescript* [online]. [cit. 2024-05-14]. Dostupné z: https://www.typescriptlang.org.

50. *Front-end frameworks and libraries* [online]. [cit. 2024-05-14]. Dostupné z: https://2022.stateofjs.com/en-US/libraries/front-end-frameworks/.

51. JOSHI, Mohit. *Angular vs React vs Vue: Core Differences* [online]. 2023-05-11. [cit. 2024-05-14]. Dostupné z: https://www.browserstack.com/guide/angular-vs-react-vs-vue.

52. *Killed by Google* [online]. [cit. 2024-05-14]. Dostupné z: https://killedbygoogle.com/.

53. COLLINSWORTH, Josh. *React vs Vue vs Angular vs Svelte* [online]. 2022-01-18. [cit. 2024-05-14]. Dostupné z: https://dev.to/hb/react-vs-vue-vs-angular-vs-svelte-1fdm.

54. COLLINSWORTH, Josh. *Introducing Svelte, and Comparing Svelte with React and Vue* [online]. 2022-01-18. [cit. 2024-05-14]. Dostupné z: https://joshcollinsworth.com/blog/introducing-svelte-comparing-with-react-vue.

55. *The mobile SDK for the Web.* [online]. [cit. 2024-05-14]. Dostupné z: https://ionicframework.com/.

56. LYNCH, Max. *Capacitor vs Cordova: What are the Differences Between Them* [online]. [cit. 2024-05-14]. Dostupné z: https://ionic.io/resources/articles/capacitor-vs-cordova-modern-hybrid-app-development.

57. TEAM, The Postman. *A guide to the different types of APIs* [online]. 2022-07-06. [cit. 2024-05-14]. Dostupné z: https://blog.postman.com/different-types-of-apis/.

58. *Websocket vs HTTP* [online]. [cit. 2024-05-14]. Dostupné z: https://ably.com/topic/websockets-vs-http.

59. LANE, Kin. *What Is a SOAP API?* [online]. 2023-06-28. [cit. 2024-05-14]. Dostupné z: https://blog.postman.com/soap-api-definition/.

60. DATWANI, Karan. *How GraphQL is better than REST APIs.* [online]. 2023-10-13. [cit. 2024-05-14]. Dostupné z: https://backpackforlaravel.com/articles/tutorials/how-graphql-is-better-than-rest-apis.

61. VÍTEK, Rostislav. *N+1 problém a jeho řešení* [online]. 2022-05-20. [cit. 2024-05-14]. Dostupné z: https://www.shopsys.cz/n1-problem-a-jeho-reseni/.

62. *A Dependency Manager for PHP* [online]. [cit. 2024-05-14]. Dostupné z: https://getcomposer.org/.

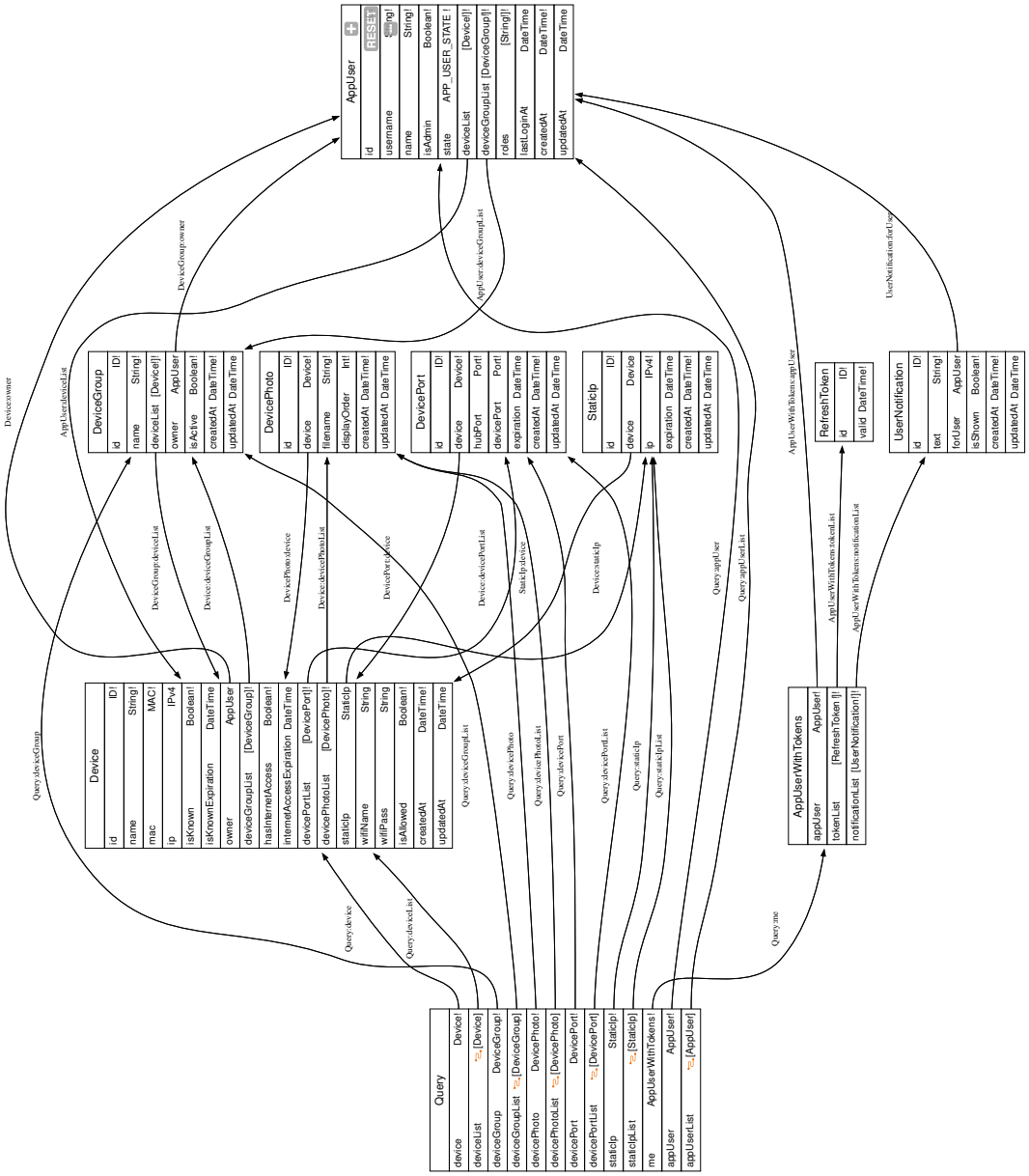
63. *Rule Levels* [online]. [cit. 2024-05-14]. Dostupné z: https://phpstan.org/user-guide/rule-levels.

64. SOFELA, Oluwatobi. *Complete Guide to ES Modules and Module Bundlers* [online]. 2022-05-11. [cit. 2024-05-14]. Dostupné z: https://www.freecodecamp.org/news/javascript-es-modules-and-module-bundlers/.

65. JAIN, Ankur. *Architecture and basic terminologies of Docker* [online]. 2021-08-26. [cit. 2024-05-14]. Dostupné z: https://www.techsupper.com/2021/08/architecture-and-basic-terminologies-of-docker/.

Příloha A

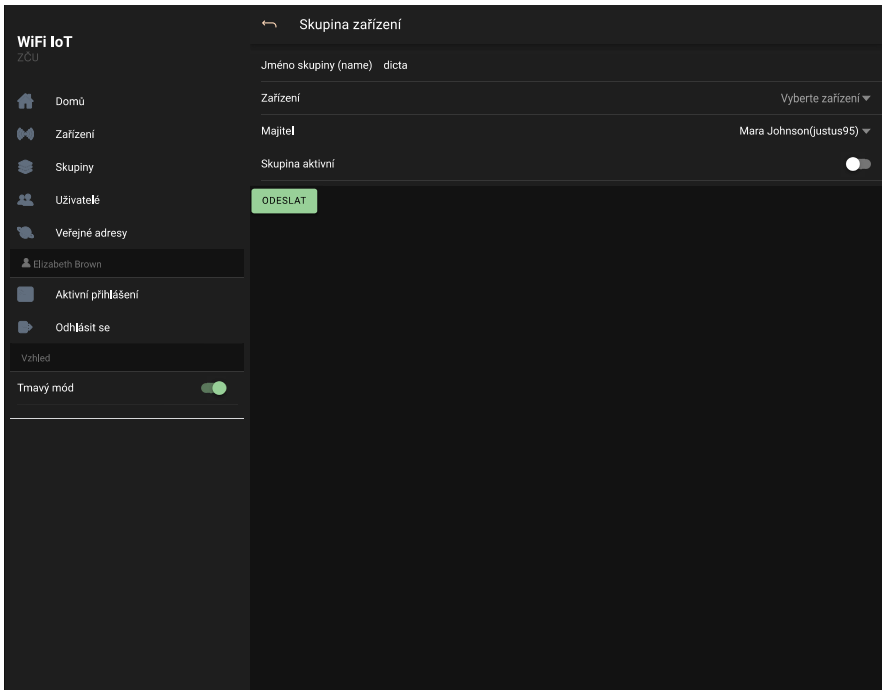
Schéma GraphQL



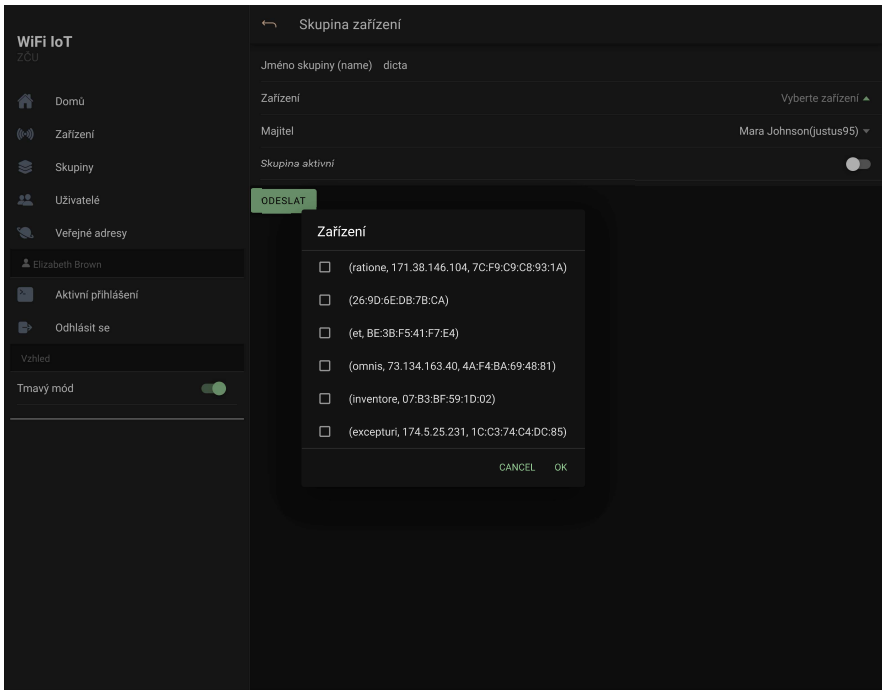
Obrázek A.1: Schéma GraphQL API vygenerované aplikací GraphQL Voyager.

Příloha B

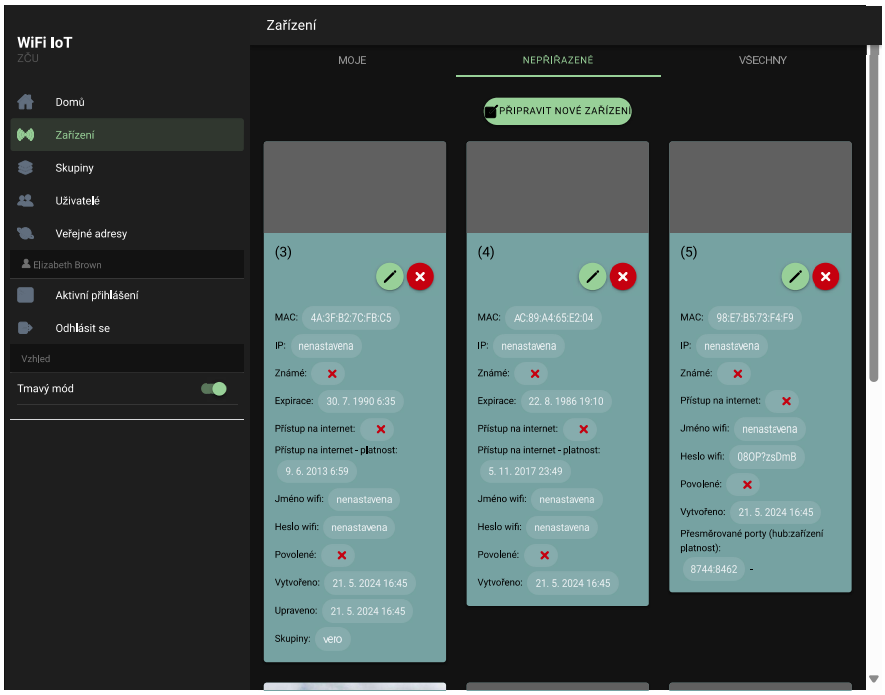
Snímky obrazovky



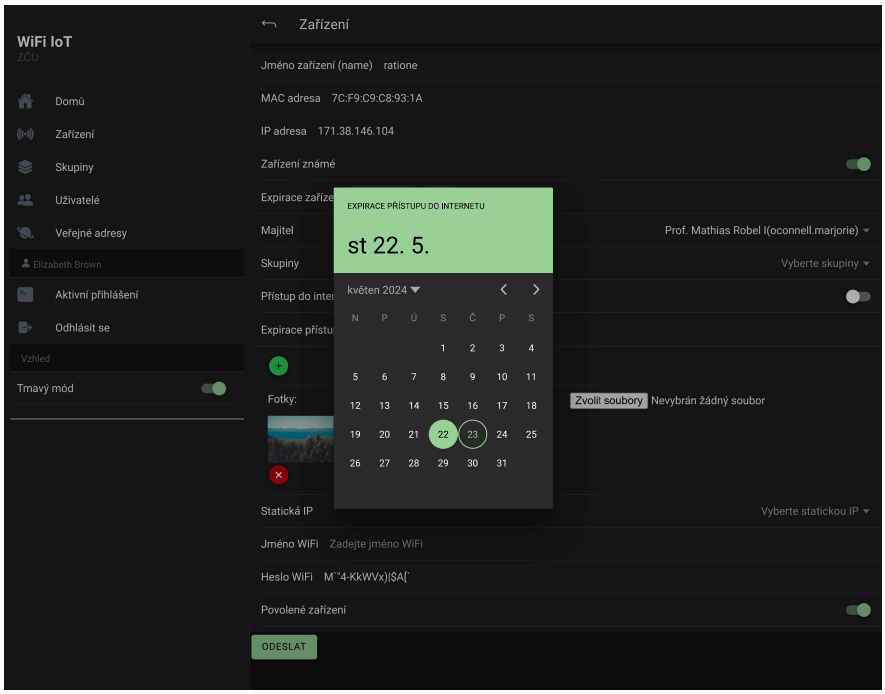
Obrázek B.1: Úprava skupiny



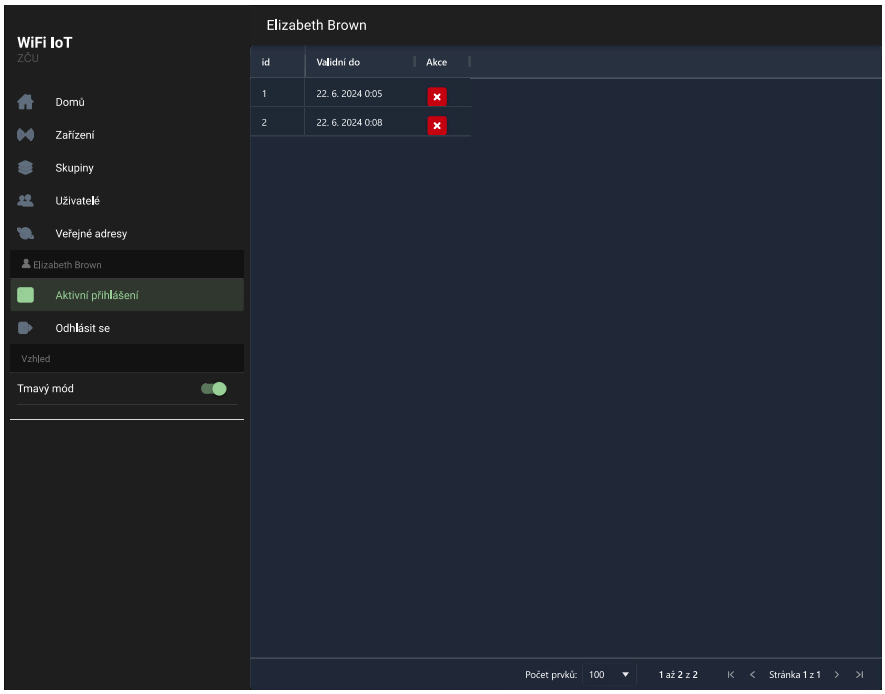
Obrázek B.2: Úprava skupiny - výběr zařízení



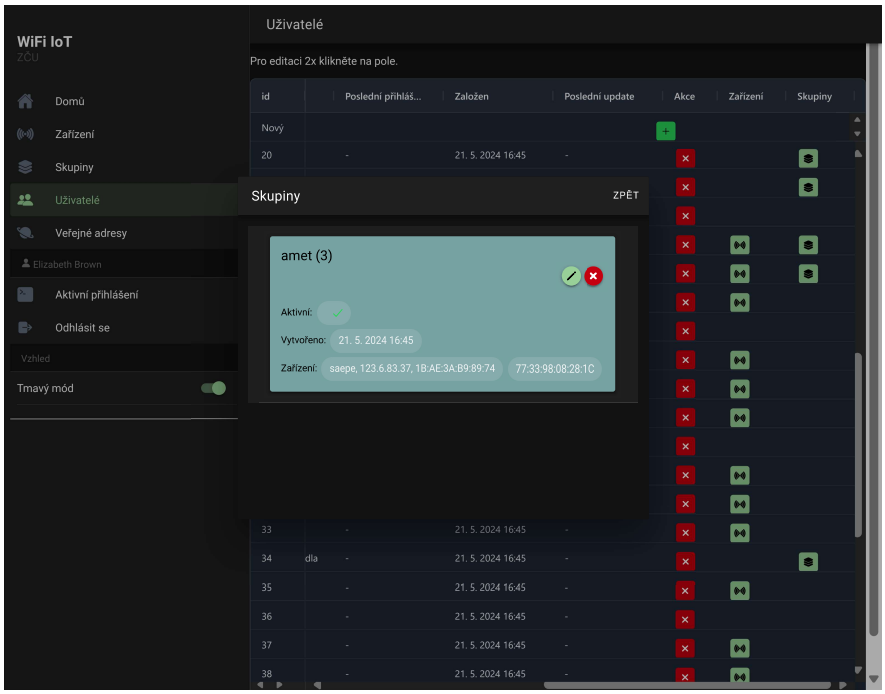
Obrázek B.3: Nepřiřazená zařízení, přihlášený je administrátor



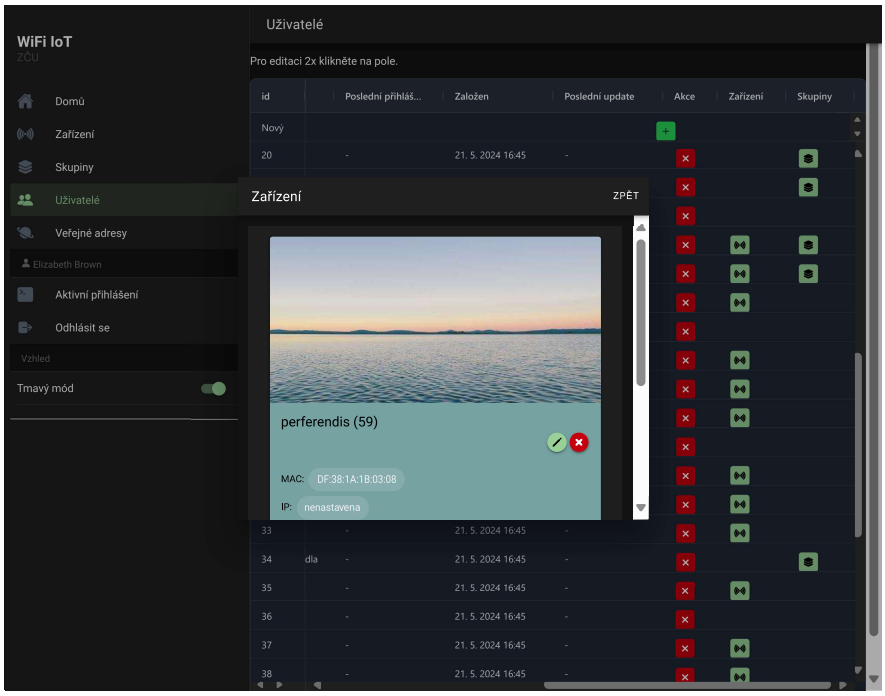
Obrázek B.4: Úprava zařízení, výběr expirace



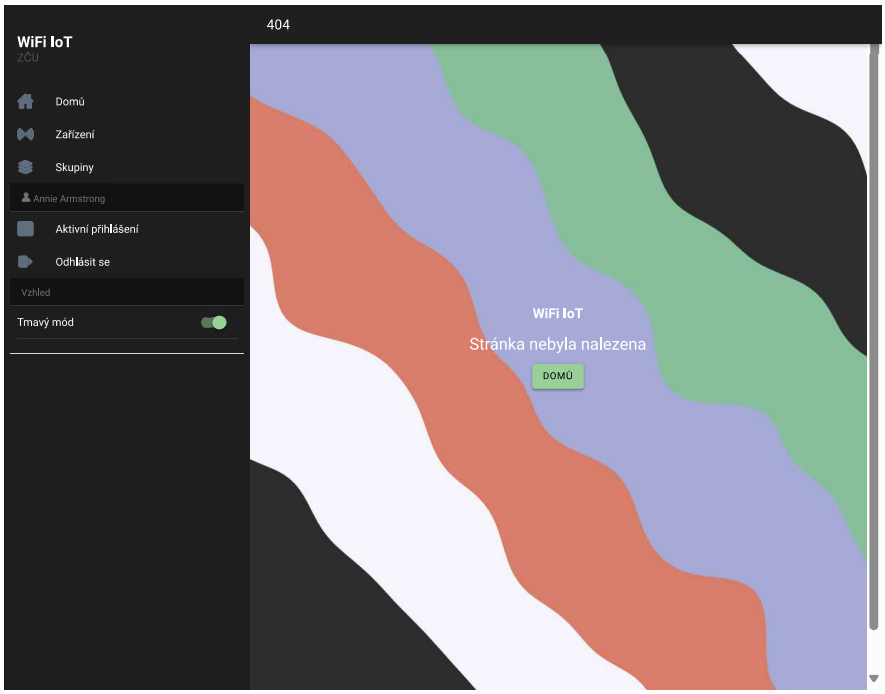
Obrázek B.5: Aktivní přihlášení



Obrázek B.6: Administrace uživatelů, zobrazení skupiny



Obrázek B.7: Administrace uživatelů, zobrazení zařízení



Obrázek B.8: Chybová stránka 404

Příloha C

Spuštění projektu, konfigurace a knihovny

Tato příloha je obsahem souboru README.md.

C.1 Client

- JavaScript
  - Ionic
  - Vue.js 3
  - Vue Router
  - Pinia
  - Apollo Client
  - graphql
  - graphql-tag
  - ionicons
  - apollo-upload-client
  - vuelidate
- Android
  - Capacitor
- Grafika
  - Iot icons created by Freepik - Flaticon
  - 404 animace
  - Simplex noise

C.2 Server - api

- PHP >=8.3
  - bcmath
  - pdo_mysql
  - zip
  - intl
  - gd –with-freetype –with-jpeg
  - Ctype, iconv, PCRE, Session, SimpleXML, Tokenizer
- Symfony
- Doctrine
- GraphQLBundle
- JWTRefreshTokenBundle
- NelmioCorsBundle

C.3 Prostředí pro nasazení

- Debian Bookworm
  - zip
  - libfreetype6-dev
  - libzip-dev
  - libjpeg-dev
  - libpng-dev
  - libicu-dev
- MariaDB 11.3

C.4 Vývoj

- Nástroje
  - Git
  - Docker

  - Adminer
  - Voyager
  - GraphQL IDE Monorepo
  - Postman
  - Visual Studio Code
    * devsense.composer-php-vscode
    * devsense.intelli-php-vscode
    * devsense.phptools-vscode
    * devsense.profiler-php-vscode
    * graphql.vscode-graphql
    * graphql.vscode-graphql-syntax
    * ionic.ionic
    * ms-azuretools.vscode-docker
    * neilbrayfield.php-docblocke
    * vue.volar

- Client
  - TypeScript
  - NPM
  - Vite
  - ESLint
  - terser
  - Vue Language Tools

- Server
  - Composer
  - FakerPHP
  - PHP Coding Standards Fixer

C.4.1 Instalace pro vývoj

Nastavení cesty k Android SDK:

```
export ANDROID_SDK_ROOT=~/Android/Sdk/
```

Naklonování projektu:

```
clone git@gitlab.com:krystof.trko/wifiiot.git
```

Konfigurační soubory dockeru:

```
cp ./docker/docker-compose.example.yml ./docker-compose.yml
```

Spuštění docker containeru a instalace závislostí:

```
docker-compose run wifi_iot_client_dev bash -c "npm update"
docker-compose run wifi_iot_api_dev bash -c "composer update"
docker-compose up
```

Po připojení do kontejneru wifi_iot_api_dev:

```
docker exec -it wifi_iot_api_dev bash
```

PHP CS

```
mkdir --parents tools/php-cs-fixer
composer require --working-dir=tools/php-cs-fixer friendsofphp/php-cs-fixer
```

Instalace databáze:

```
bin/console doctrine:migrations:migrate -n
bin/console doctrine:fixtures:load -n
```

Oprávnění, exit

```
chmod -R 777 server/temp
exit
```

C.4.2 Vývoj

Připojení do server kontejneru:

```
docker exec -it wifi_iot_api_dev bash
```

Připojení do client kontejneru:

```
docker exec -it wifi_iot_client_dev bash
```

Kontrola a oprava kódu v wifi_iot_api_dev kontejneru:

```
composer fs
composer phpstan
```

Kontrola a oprava kódu v wifi_iot_client_dev kontejneru:

```
npm run lint
```

C.4.3 Instalace pro produkční nasazení

Server Ubuntu 22.04

```
sudo apt update
sudo apt upgrade
sudo apt dist-upgrade

reboot

sudo apt update
sudo apt upgrade

sudo apt install mariadb-server git nginx

sudo apt install ca-certificates apt-transport-https software-properties-common lsb
sudo add-apt-repository ppa:ondrej/php -y
sudo apt update

sudo apt install php8.3 php8.3-fpm php8.3-cli
sudo apt install php8.3-{cli,fpm,curl,mysqlnd,gd,opcache,zip,intl,common,bcmath,ima
sudo apt install composer

curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.7/install.sh | bash

export NVM_DIR="$HOME/.nvm"
[ -s "$NVM_DIR/nvm.sh" ] && \. "$NVM_DIR/nvm.sh"
[ -s "$NVM_DIR/bash_completion" ] && \. "$NVM_DIR/bash_completion"

nvm install 20
```

Deploy key pro gitlab:

```
ssh-keygen
more /root/.ssh/id_rsa.pub
```

Konfigurace souborů /var/www/wifiiot/server/.env
Clone a instalace npm composer

```
cd /var/www
rm -rf html
git clone git@gitlab.com:krystof.trko/wifiiot.git
```

```
cd /var/www/wifiiot/server
composer install

cd /var/www/wifiiot/client
npm install
npm run build
```

Instalace databáze:

```
mysql -u root

CREATE DATABASE wifiiot;
CREATE USER wifiiot@localhost IDENTIFIED BY password;
GRANT ALL PRIVILEGES ON *.* TO wifiiot@localhost IDENTIFIED BY 'password';
FLUSH PRIVILEGES;

cd /var/www/wifiiot/server/bin

php console doctrine:migrations:migrate -n
php console doctrine:fixtures:load -n # test data
```

Konfigurace nginx

```
server {
  root  /var/www/wifiiot/client/dist;

  index index.html;
    server_name sulis198.zcu.cz; # managed by Certbot

  error_log  /var/log/nginx/wifiiot_front_error.log;
    access_log /var/log/nginx/wifiiot_front_access.log;

  gzip on;
    gzip_types text/plain text/css text/js text/xml text/javascript application/jav

  add_header X-Frame-Options "SAMEORIGIN";
    add_header X-XSS-Protection "1; mode=block";
    add_header X-Content-Type-Options "nosniff";

  charset utf-8;

  location /api {
```

```
        error_log  /var/log/nginx/wifiiot_api_error.log;
        access_log /var/log/nginx/wifiiot_api_access.log;

        alias /var/www/wifiiot/server/public;
        include fastcgi.conf;

        rewrite /api/(.*) /api/$1 break;

        fastcgi_param SCRIPT_FILENAME $document_root/index.php;
        fastcgi_param DOCUMENT_URI /index.php;
        fastcgi_param SCRIPT_NAME /index.php;
        fastcgi_param REQUEST_URI $1$is_args$query_string;
        fastcgi_pass unix:/var/run/php/php8.3-fpm.sock;
    }

    location /device_photo {
        error_log  /var/log/nginx/wifiiot_public_error.log;
        access_log /var/log/nginx/wifiiot_public_access.log;

        alias /var/www/wifiiot/server/public/device_photo;

        try_files $fastcgi_script_name =404;
    }

    location / {
        try_files $uri $uri/ /index.html;
    }

    listen [::]:443 ssl ipv6only=on; # managed by Certbot
    listen 443 ssl; # managed by Certbot
    ssl_certificate /etc/letsencrypt/live/sulis198.zcu.cz/fullchain.pem; # managed
    ssl_certificate_key /etc/letsencrypt/live/sulis198.zcu.cz/privkey.pem; # manage
    include /etc/letsencrypt/options-ssl-nginx.conf; # managed by Certbot
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem; # managed by Certbot

}
server {
    if ($host = sulis198.zcu.cz) {
        return 301 https://$host$request_uri;
    } # managed by Certbot
```

```
    listen 80 ;
    listen [::]:80 ;
      server_name sulis198.zcu.cz;
      return 404; # managed by Certbot
}
```

Let's Encrypt certifikát

```
sudo apt install snapd
sudo snap install --classic certbot
ln -s /snap/bin/certbot /usr/bin/certbot
```

```
composer install --no-dev --optimize-autoloader
npm install
npm run build
```

Klíč pro generování JWT

```
cd /var/www/wifiiot/server/bin
php server/bin/console lexik:jwt:generate-keypair
```

C.5 Struktura aplikace

- Přihlásit (login)
- 404 (notFound)
- Menu - uživatel
  - Domů (home)
  - Zařízení - filtr moje/nepřiřazená (deviceList)
    - CRUD zařízení (device)
  - Skupiny - filtr moje (groupList)
    - CRUD skupina (group)
  - Aktivní přihlášení (me)
  - Odhlásit (-)
  - Vzhled (-)
- Menu - admin

  - Domů (home)
  - Zařízení - filtr moje/nepřiřazená/všechna (deviceList)
    * CRUD zařízení (device)
  - Skupiny - filtr moje/všechny (groupList)
    * CRUD skupina (group)
  - Uživatelé (userList)
    * CRUD uživatele (user)
  - Správa statických IP (staticIpList)
    * CRUD statické IP (staticIp)
  - Aktivní přihlášení (me)
  - Odhlásit (-)
  - Vzhled

License: GPL-3.0-only