VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

KLIENT KONTROLÉRU ZEROTIER

ZEROTIER CONTROLLER CLIENT

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. VOJTĚCH KULÍŠEK**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. JIŘÍ HYNEK, Ph.D.**
SUPERVISOR

BRNO 2025

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání diplomové práce

161510

Ústav:	Ústav informačních systémů (UIFS)
Student:	**Kulíšek Vojtěch, Bc.**
Program:	Informační technologie a umělá inteligence
Specializace:	Informační systémy a databáze
Název:	**Klient kontroléru ZeroTier**
Kategorie:	Informační systémy
Akademický rok:	2024/25

Zadání:

1. Prostudujte problematiku tvorby sdílených sítí. Prostudujte dostupné nástroje. Zaměřte se na platformu ZeroTier.
2. Prostudujte principy tvorby multiplatformních a webových aplikací.
3. Analyzujte požadavky na klienta kontroléru technologie ZeroTier. Porovnejte s existujícími řešeními.
4. Dle výsledků analýzy a na základě konzultací s vedoucím navrhněte webového klienta pro ovládání kontroléru technologie ZeroTier.
5. Navržené řešení implementujte.
6. Otestujte funkčnost a použitelnost výsledné aplikace.

Literatura:

- Grigorik, I. (2013). *High Performance Browser Networking: What every web developer should know about networking and web performance*. " O'Reilly Media, Inc.".
- Johnson, J. (2020). *Designing with the mind in mind: simple guide to understanding user interface design guidelines*. Morgan Kaufmann.
- Tailscale Inc. (2024). Tailscale docs. Citováno 17. 10. 2024: https://tailscale.com/kb/.
- ZeroTier Inc. (2024). ZeroTier Documentation. Citováno 17. 10. 2024: https://docs.zerotier.com/.

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

Vedoucí práce:	**Hynek Jiří, Ing., Ph.D.**
Vedoucí ústavu:	Kolář Dušan, doc. Dr. Ing.
Datum zadání:	1.11.2024
Termín pro odevzdání:	21.5.2025
Datum schválení:	22.10.2024

Abstrakt

Cílem této práce bylo vytvořit klientskou webovou a desktopovou aplikaci pro ovládání jednoho a více kontrolérů platformy ZeroTier. Aplikace ulehčuje práci při vytváření virtuálních počítačových sítí a jejich následném spravování oproti dosavadním řešením. Nová aplikace umožňuje uživatelům bez znalostí počítačových sítí jednoduše a korektně nastavit kontrolér platformy ZeroTier. Před vznikem této práce neexistovala žádná aplikace, která by uživatelům bez znalostí počítačových sítí toto umožnila.

Abstract

The aim of this work was to create a client web application and desktop application for controlling one or more ZeroTier controllers. The application facilitates the work of creating virtual networks and their subsequent management compared to existing solutions. The new application allows users without knowledge of computer networks to easily and correctly set up a ZeroTier controller. Prior to this work, there was no application that would allow users without knowledge of computer networks to do this.

Klíčová slova

sdílení služeb, přesměrování portů, virtuální počítačové sítě, děrování překladu síťových adres, TailScale, ZeroTier, virtuální internetové stránky, multiplatformní vývoj

Keywords

sharing services, port forwarding, virtual computer networks, network address translation hole punching, TailScale, ZeroTier, virtual websites, multiplatform development

Citace

KULÍŠEK, Vojtěch. *Klient kontroléru ZeroTier*. Brno, 2025. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Jiří Hynek, Ph.D.

Klient kontroléru ZeroTier

Prohlášení

Prohlašuji, že jsem tuto semestrální práci vypracoval samostatně pod vedením pana doktora Jiřího Hynka. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Vojtěch Kulíšek
18. května 2025

Poděkování

Děkuji vedoucímu práce doktoru Jiřímu Hynkovi, za vedení této práce a pomáhání při její řešení. Dále bych chtěl poděkovat uživatelské základně, která poskytovala průběžně zpětnou vazbu k aplikaci.

Obsah

Seznam obrázků

Seznam zkratek

API	Application Programming Interface. 49
CGI	Common Gateway Interface. 3, 16, 17, 41, 47
CGN	Carrier-grade Network Address Translation. 9
CI/CD	Continuous Integration Continuous Delivery. 45
CSS	Cascading Style Sheets. 18
DDNS	Dynamic Domain Name System. 9, 23
DERP	Designated Encrypted Relay for Packets. 13
DNS	Domain Name System. 26, 34, 43
DOM	Document Object Model. 3, 18, 19
DoS	Denial of Service. 47
FPM	FastCGI Process Manager. 17
HTML	Hypertext Markup Language. 18, 19
HTTP	Hypertext Transfer Protocol. 16, 22, 24, 44
httpd	Hypertext Transfer Protocol Daemon. 16, 17
HTTPS	Hypertext Transfer Protocol Secure. 24
ICE	Interactive Connectivity Establishment. 12, 13
IP	Internet Protocol. 3, 8–14, 21–26, 34
IPv4	Internet Protocol version 4. 3, 7–9, 33, 37
IPv6	Internet Protocol version 6. 3, 7–9, 34, 37
JWT	JSON Web Token. 33, 44
LAN	Local Area Network. 10, 22–24, 28
MTU	Maximum Transmission Unit. 33, 37
NAS	Network Attached Storage. 44
NAT	Network Address Translation (překlad síťových adres). 8, 9
NAT-PMP	Network Address Translation Port Mapping Protocol. 14
NCSA	National Center for Supercomputing Applications. 16, 17
PHP	Hypertext Preprocessor. 18, 20, 41, 44, 46, 47, 49
REST	Representational State Transfer. 30, 33, 40, 44

SaaS	Software as a Service. 25
SFTP	SSH File Transfer Protocol. 21
SMB	Server Message Block. 21
SSH	Secure Shell. 10, 21, 22
STUN	Session Traversal Utilities for NAT. 11, 13
TCP	Transmission Control Protocol. 10, 24
TURN	Traversal Using Relays around NAT. 11, 12
UDP	User Datagram Protocol. 11
UPnP	Universal Plug and Play. 14
USB	Universal Serial Bus. 44
VPN	Virtual Private Network (Virtuální privátní síť). 3, 10, 11, 22–24, 28
WebRTC	Web Real-Time Communication. 3, 11, 12
WoL	Wake on LAN. 22
XML	Extensible Markup Language. 19

Kapitola 1

Úvod

V dnešní době nedostatku IPv4 adres je problematické sdílet služby s ostatními přes internet. Doposud vzniklo mnoho aplikací, které se snaží tento problém vyřešit. Bohužel tyto aplikace přinášejí jisté limitace popsané v kapitole 4. Před pár lety vzniklo nové řešení, které tyto problémy řeší. Bohužel k tomuto řešení neexistuje žádná aplikace, která by ho umožnila používat uživatelům bez pokročilých znalostí počítačových sítí. Naproti tomu existují dvě aplikace (ztncui, ZeroUI), které toto řešení umožňují používat pokročilým uživatelům. Tyto aplikace, ale neimplementují některé užitečné funkce a nejsou moc uživatelsky přívětivé. Tyto aplikace jsou navíc náročné na zdroje zařízení, díky čemuž se nedají bez obtíží spouštět například na WiFi routerech, nebo chytrých úložištích. Na nich by je bylo vhodné provozovat v domácích podmínkách, jelikož by pak odpadala nutnost mít doma spuštěný server.

Cílem této práce bylo vytvořit novou aplikaci, která bude řešit tyto dosavadní problémy současných řešení. Novou aplikaci lze spouštět na síťových prvcích a síťových úložištích a lze ji využít k velkému množství účelů, jako je například přístup k souborovému serveru, který má uživatel aplikace doma. Zároveň je nová aplikace více uživatelsky přívětivá, odděluje servery od klientů ve virtuální počítačové síti, umožňuje propojit více kontrolérů dohromady a lze ji nainstalovat i jako desktopovou aplikaci.

Kapitola 2 se zabývá problematikou sdílení služeb přes internet. Nejdříve popisuje IPv4 a IPv6 adresy a architektury koncových sítí, které se u nich často používají. Následně se věnuje problematice sdílení služeb v těchto sítích. Popisuje přesměrování portů přes třetí zařízení, komunikaci mezi počítači ve virtuální privátní síti a ke konci se věnuje architekturám, které využívají platformy TailScale a ZeroTier. Následující kapitola 3 se zabývá principy vývoje webových aplikací a možnostmi tvorby multiplatformních aplikací.

Analýzou požadavku na novou desktopovou a webovou aplikaci pro sdílení služeb se zabývám v kapitole 4. Definuji zde jednotlivé uživatele, pro které bude aplikace určena a nejčastější účely, ke kterým bude aplikace používána. Na konci popisuji požadavky na nové řešení. Kapitola 5 se zabývá samotným návrhem řešení nové aplikace. Popisuje změny, které byly v návrhu prováděny na základě zpětné vazby od uživatelů. Po návrhu aplikace následuje samotná implementace aplikace v kapitole 6. Po implementaci celé aplikace provádím její samotné testování v kapitole 7 různými metodami. V poslední kapitole 8 zhodnocuji výslednou aplikaci, která byla tvořena s touto prací.

Kapitola 2

Sdílení služeb přes internet

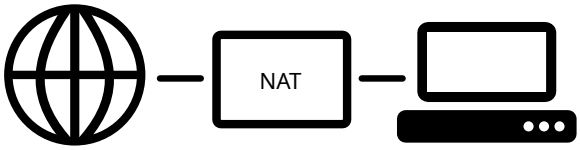
Jak je popsáno v [31], vzhledem k rychlému nárůstu počtu zařízení, která jsou připojena k internetu, začaly rychle docházet původně navržené IPv4 adresy. Proto vznikly IPv6 adresy, které se nedostatečně rychle nasazují, díky tomu se začaly používat překlady síťových adres zkráceně NAT.

Překlady síťových adres tento problém pomohly vyřešit tím, že jedna IPv4 adresa může být sdílena mezi více zařízeními. Sdílení adres mezi více zařízeními, ale vytvořilo další problém a to je koncová konektivita mezi zařízeními v síti. Díky tomu je v některých případech složitější sdílet některé služby přes internet.

V této kapitole jsou popsány možnosti sdílení služeb přes dnešní internet. Podkapitola 2.1 se věnuje sdílení služeb v situaci, kdy účastník využívá pro připojení k internetu jeden a více překladů síťových adres, nebo nevyužívá žádný překlad síťových adres. Následně jsou v podkapitolách 2.2 a 2.3 popsány možnosti, které mají účastníci v případě, kdy jsou připojeni k internetu přes překlad síťových adres a nemají možnost tyto překlady nastavit. Na konci se kapitola podrobně věnuje architektuře, kterou využívá platforma ZeroTier[1] a TailScale[2] pro vytváření privátních sítí.

2.1 IPv4 a IPv6

V dnešní době, kdy je velmi málo IPv4 adres, se s nimi snaží všichni co nejvíce šetřit. K šetření IPv4 adres se využívá překlad síťových adres, zkráceně NAT. Díky tomu je bohužel obtížnější dosáhnout hostování služby přes internet.

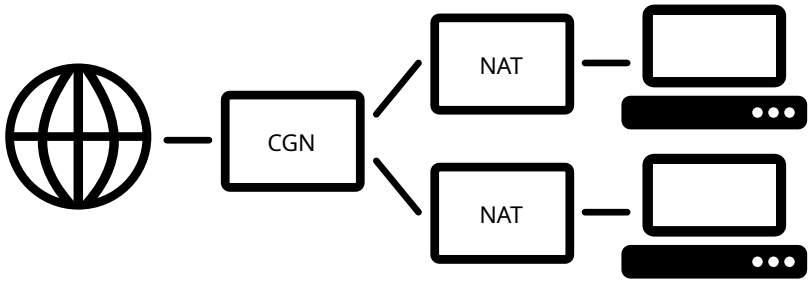


Obrázek 2.1: IPv4 koncová síť s přiřazenou veřejnou IPv4 adresou

Jestliže má odběratel přiřazenou veřejnou IPv4 adresu, nebývá hostování služeb přes internet problém. Nejčastěji je připojený přes architekturu uvedenou na obrázku 2.1 k internetu. K hostování služeb potom odběrateli stačí nastavit na zařízení, provozující překlad IP adres, přesměrování portů.

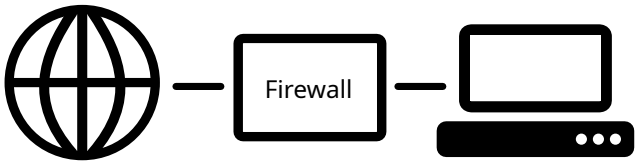
[1] https://www.zerotier.com/
[2] https://tailscale.com/

Někdy to může mít odběratel o trochu obtížnější a to v případě, kdy se přiřazená veřejná IPv4 adresa může měnit. Tento problém jde vyřešit pomocí dynamického DNS (DDNS) serveru a následně je možné přistupovat ke službě přes definované doménové jméno [70].



Obrázek 2.2: IPv4 koncová síť bez přiřazené veřejné IPv4 adresy[3]

Odběratel internetu může být připojený k internetu přes veřejnou IPv4 adresu, která je sdílena mezi více klienty poskytovatele internetu. Architektura takového připojení je znázorněna na obrázku 2.2. Jak popisuje [10], pokud klient provede přesměrování portů na svém překladu síťových adres na této síťové architektuře, tak přesměrovaný port nebude stále přístupný z internetu. Veškerá komunikace bude váznout na CGN (*Carrier-grade NAT*), který provozuje poskytovatel internetu. CGN je varianta překladu síťových adres, kterou používají poskytovatelé internetu, pro tento typ překladu síťových adres je vyhrazený adresový prostor 100.64.0.0/10 [83]. Poskytovatel internetu může i nemusí na svém překladu síťových adres umožnit vytvořit přesměrování portů odběrateli [10].



Obrázek 2.3: IPv6 koncová síť[4]

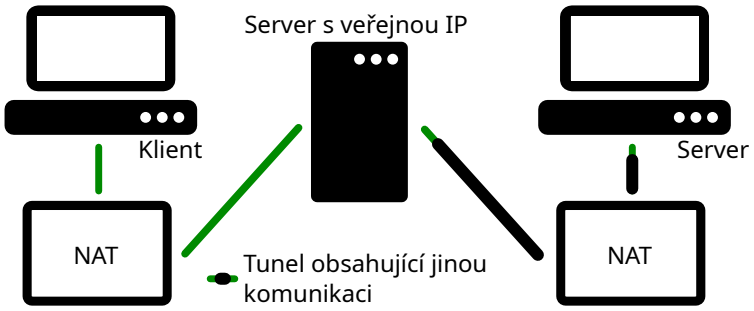
Jak je uvedeno v knize [65], v případě IPv6 adres, kterých je dostatek, mají výrobci koncových zařízení, doporučené využívat místo překladu síťových adres firewall (Firewall slouží k filtrování síťové komunikace na základě definovaných pravidel.). Ten by měl být nastavený tak, aby blokoval nová příchozí spojení z internetu. Stejně jak to doposud dělal jako vedlejší funkci překlad síťových adres. Pokud bude někdo sdílet službu, bude díky tomu k dispozici jen na lokální síti. Pro zveřejnění služby potom firewall, umožňuje vytvořit pravidlo s výjimkou.

2.2 Přesměrování portů přes server

Jak je popsáno v [12, 18, 50], k hostování služby přes internet lze využít další server, na který se lze připojit přes veřejnou IP adresu. Zařízení, které může být za překladem síťových adres nebo firewallem, vytvoří nové spojení, takzvaný tunel k serveru. Pokud chce klient následně využívat službu hostovanou na zařízení, které vytvořilo tunel, tak se připojí k vzdálenému

[3]Obrázek vychází z informací z dokumentu [10].
[4]Obrázek vychází z informací z knihy [65].

serveru, který je přístupný z internetu. Ten následně přijatou komunikaci přeposílá přes vytvořený tunel.



Obrázek 2.4: Tunel k serveru s přiřazenou veřejnou IP adresou[5]

Tímto způsobem lze docílit konektivity k zařízení poskytujícímu službu. Bez nutnosti provádět přesměrování portů na překladu síťových adres, popřípadě není nutné povolovat příchozí spojení na firewallu. Komunikace je znázorněna na obrázku 2.4.

Jak vychází z [12, 13], šifrovaný TCP (*Transmission Control Protocol*) tunel k tomuto účelu lze jednoduše vytvořit například pomocí SSH (*Secure Shell*) klienta a serveru, kde na serveru SSH musí být povoleno přesměrování portů, na klientovi potom stačí spustit příkaz:

```
ssh -R <port na vzdáleném serveru>:<adresa zařízení>:<port služby>
<uživatelské jméno>@<adresa serveru>
```

2.3 Virtuální privátní sítě

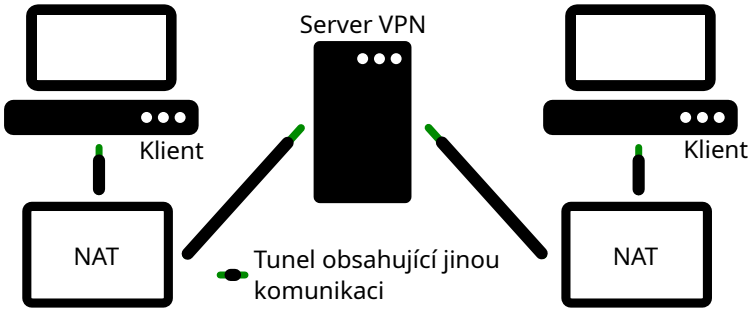
Ne vždy je možné sdílet některé služby přes přesměrování portu na překladu síťových adres, nebo povolením komunikace ve firewallu přes internet. Příkladem jsou služby, které ke své funkci vyžadují *broadcast*, nebo *multicast*. Patří sem například LAN hry, které k nalezení druhé instance hry, mohou využívat již dříve zmíněný *broadcast*. Dalším problémem může být, že některé služby nevyžadují autentizaci a nepoužívají šifrování. Tyto vlastnosti, ale mohou být vyžadovány.

Všechny tyto problémy lze vyřešit za pomocí virtuální privátní sítě (VPN). Jak je uvedeno v dokumentu [39], virtuální privátní síť emuluje privátní počítačovou síť využívající IP adresy. Ve virtuální privátní síti může být *broadcast* a *multicast* povolen, pokud to daná implementace virtuální sítě umožňuje a může poskytovat šifrování dat přes internet.

2.3.1 Přeposílání přes centrální prvek

Jak vychází z manuálu [56], VPN může umožnit komunikaci mezi dvěma zařízeními, z nichž každé se může nacházet za vlastním překladem síťových adres nebo firewallem. V případě, kdy má server VPN přiřazenou veřejnou IP adresu a lze se k němu přes tuto adresu připojit.

[5] Obrázek vychází z informací uvedených v [12, 18, 50].



Obrázek 2.5: VPN s centrálním serverem[6]

Zjednodušená architektura této virtuální privátní sítě je znázorněna na obrázku 2.5. Vzhledem k tomu, že architektura využívá třetí zařízení k přeposílání komunikace, je nutné počítat s vyšší latencí mezi klienty, pokud komunikují mezi sebou navzájem.

2.3.2 Děrování překladu síťových adres

Ideálnějším řešením je nevyužívat další zařízení k přeposílání komunikace mezi klienty virtuální privátní sítě a komunikovat spolu přímou cestou, aby byla latence nižší. K tomu, aby dva klienti za překladem síťových adres, nebo firewallem byli schopni spolu komunikovat přímo mezi sebou, se využívají techniky děrování překladu síťových adres.

Nejvíce podporovaná technika z děrování překladu síťových adres, která se používá, je UDP děrování překladu síťových adres [22]. V této technice se vhodně využívá bezstavovost UDP komunikace [21]. UDP děrování překladu síťových adres mohou různé aplikace implementovat trochu odlišně. Tato technika se používá i v technologii WebRTC, kde je dobře standardizovaná a využívá již existující protokoly, které jsou dobře zdokumentovány [21, 26].

Jak je popsáno v [26] síťová architektura WebRTC se skládá z následujících zařízení:

1. **Signalizační server**: slouží k počáteční komunikaci mezi zařízeními, které chtějí vytvořit mezi sebou přímé spojení.
2. **Server STUN (*Simple Traversal Under under NAT*)**: umožňuje klientovy zjistit namapovanou veřejnou IP adresu a port na překladu síťových adres, pokud je překlad síťových adres přítomen [41, 26].
3. **Server TURN (*Traversal Using Relays around NAT*)**: slouží pro přeposílání komunikace v případě, kdy techniky děrování překladu síťových adres nejsou úspěšné.
4. **Klienti**: zařízení, které se snaží mezi sebou vytvořit přímé spojení.

Jak je dále popsáno v [26], k tomu aby se mohla dvě zařízení přímo spojit mezi sebou, potřebují nejdříve komunikační kanál, přes který by se mohli nejdříve domluvit, jak přímé spojení vytvořit. K tomuto komunikačnímu kanálu se používá signalizační server. Signalizační server musí být přístupný klientům přes veřejnou IP adresu.

Aby bylo WebRTC schopné provést přímé propojení mezi koncovými zařízeními skrze překlad síťových adres, potřebuje nejdříve otevřít port na překladu síťových adres. K otevření portu odešle zprávu na server STUN, který odpoví zprávou obsahující veřejnou IP adresu a port, ze kterého zpráva od klienta přišla [26].

[6] Obrázek vychází z informací z dokumentace [56].

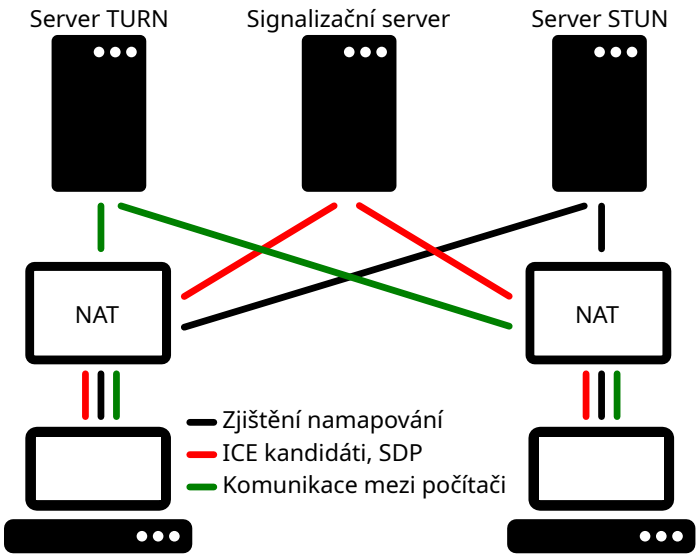
Jak popisuje dokument [63], v případě, kdy jsou zařízení za otevřeným překladem síťových adres nebo striktním překladem síťových adres nebo firewallem, je WebRTC schopné znovu použít stejný namapovaný port tím, že odešle novou zprávu ze stejného portu na lokální adrese. Symetrický překlad síťových adres tuto funkci neumožňuje, vždy vytvoří nové namapování portů na překladu síťových adres.

Jak je popsáno v [26], zjištěné namapování se odešle přes signalizační server v ICE kandidátech (*Interactive Connectivity Establishment* kandidáti obsahují možnosti, jak se lze připojit na daného klienta). V ICE kandidátech bývá typicky uveden jako poslední navíc server TURN pro případ, kdyby se zařízení nebyla schopna přímo propojit.

Koncová zařízení se navzájem následně snaží propojit pomocí informací z ICE kandidátů [26]. Nejdříve se každý snaží připojit pomocí privátní adresy cílového zařízení [26]. Jestliže se to nezdaří, pokusí se následně připojit na namapovaný port na veřejné IP adrese cílového zařízení [26]. V případě striktního překladu síťových adres se může ztratit pár paketů, než druhá strana stihne také odeslat paket. Jelikož pro striktní překlad síťových adres se tato komunikace bude jevit jako příchozí z internetu, proto ji zablokuje [63].

Úspěšně provedená technika děrování překladu síťových adres může například snížit latenci o 6.58 ms, pokud spolu dvě zařízení komunikují v rámci České republiky, namísto přeposílání komunikace přes server v německém datacentru [36].

Děrování překladu síťových adres se nepovede v případě použití symetrických překladů síťových adres [36]. Tento problém by šlo částečně vyřešit predikováním překladu síťových adres, některé symetrické překlady síťových adres jdou velmi dobře predikovat [82]. Bohužel WebRTC predikci neimplementuje [36].



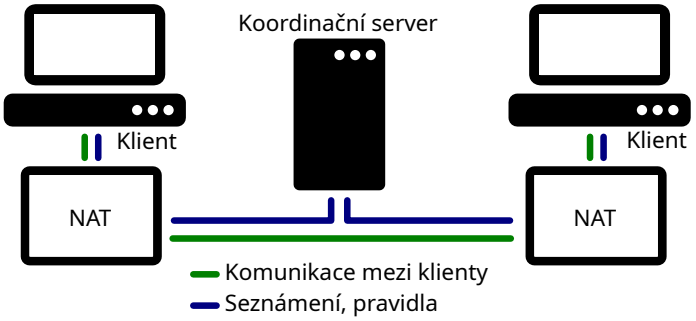
Obrázek 2.6: WebRTC síťová architektura [36]

Na obrázku 2.6 je znázorněna síťová architektura WebRTC v okamžiku, kdy není možné vytvořit přímé spojení. V případě, kdyby WebRTC bylo schopné provést přímé spojení, nebyl by na obrázku server TURN. Komunikace mezi počítači by šla přímo mezi překlady síťových adres.

TailScale

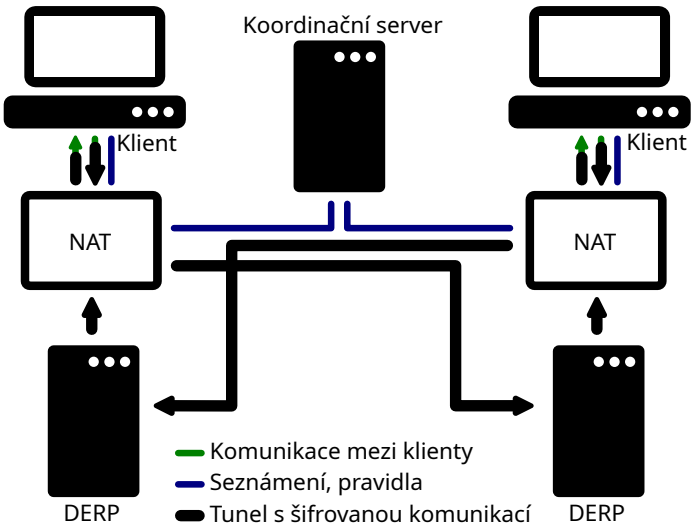
Tato podkapitola vychází z dokumentace [58]. Architektura TailScale se skládá ze zařízení označených koordinační server, DERP (*Designated Encrypted Relay for Packets*) a klient:

1. **Koordinační server**: zajišťuje výměnu klíčů mezi zařízeními připojenými do virtuální sítě. Klíče slouží pro následné šifrování tunelů mezi nimi. Dále sděluje pravidla pro komunikaci v síti. Ke své funkci potřebuje přiřazenou veřejnou IP adresu.
2. **DERP**: slouží k přeposílání komunikace v případě, kdy konfigurace sítě neumožňuje aplikaci vytvořit přímé propojení mezi dvěma zařízeními, které chtějí mezi sebou komunikovat.
3. **Klient**: je koncové zařízení, které buď využívá službu, nebo hostuje nějakou službu ostatním.



Obrázek 2.7: Varianta, kdy TailScale je schopný vytvořit přímé propojení[7]

Pro tunelování překladu síťových adres a firewallu TailScale využívá ICE a server STUN. V případě, kdy jsou techniky děrování úspěšné. Klienti komunikují mezi sebou přímo, jak je vyobrazeno na obrázku 2.7.



Obrázek 2.8: TailScale v případě, kdy není schopný vytvořit přímé propojení[7]

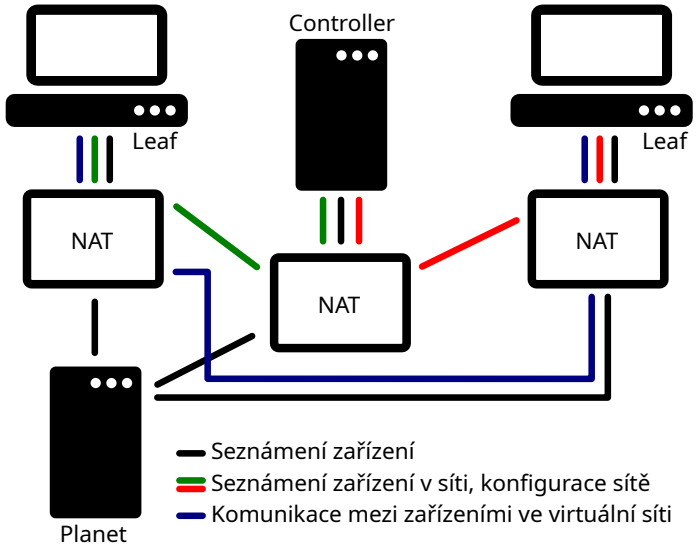
[7] Obrázek vychází z dokumentace [58].

Pokud koncová zařízení nejsou schopna vytvořit přímé propojení mezi sebou, tak přeposílají svou komunikaci přes nejbližší DERP k příjemci zprávy. Tím se snaží co nejvíce redukovat latenci. DERP není schopný samotnou komunikaci rozšifrovat, jelikož samotná komunikace je zašifrována mezi koncovými body pomocí WireGuard tunelu. Jak je znázorněno na obrázku 2.8.

ZeroTier

Tato podkapitola vychází z dokumentace [85]. Architektura ZeroTier se skládá ze zařízení označených planet, controller, leaf[8] a moon:

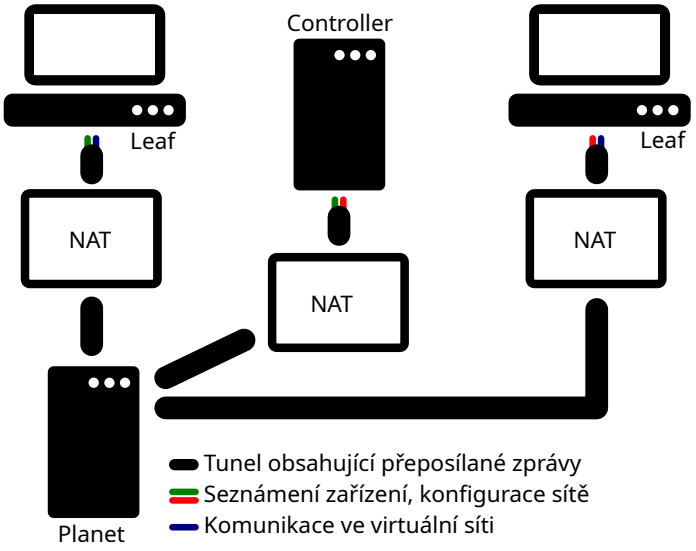
1. **Planet/moon**: musí mít přiřazenou veřejnou IP adresu. Tyto zařízení plní stejnou funkci. Slouží k přeposílání dat mezi zařízeními, které nevytvořili přímé spojení mezi sebou. Jediný rozdíl mezi nimi je ten, že planet servery jsou hostované firmou ZeroTier, Inc. jako bezplatná služba a moon servery může hostovat úplně kdokoliv. Moon servery je vhodné použít k snížení latence, pokud se nenachází nikde poblíž planet server.
2. **Controller**: slouží k vytváření a spravování virtuální sítě. Vytvořená virtuální síť může být privátní nebo veřejná. Nevyžaduje mít přiřazenou veřejnou IP adresu. Zařízení, které se chtějí připojit do virtuální sítě, s ním nejdříve komunikují přes planet/moon server.
3. **Leaf**: je koncové zařízení, které se připojuje do virtuální sítě. Může hostovat služby, nebo využívat služby, které hostují ostatní zařízení.



Obrázek 2.9: ZeroTier v případě, kdy je schopný vytvořit přímá propojení[9]

ZeroTier k děrování překladu síťových adres a firewallu používá UDP, TCP děrování překladu síťových adres, predikci portů u symetrického překladu síťových adres, NAT-PMP a UPnP. Pokud jsou tyto techniky úspěšné, tak klienti a kontrolér mezi sebou následně komunikují přímo. Přímá komunikace je znázorněná na obrázku 2.9.

[8] Leaf sice není zmíněný nikde v dokumentaci, ale implementace takto označuje koncové zařízení.
[9] Obrázek vychází z dokumentace [85].



Obrázek 2.10: Varianta, kdy ZeroTier není schopný vytvořit přímá propojení[10]

Jestliže nejsou schopny se dvě zařízení propojit přímo mezi sebou. Přeposílají komunikaci přes planet/moon zařízení. Samotný planet/moon není schopný přeposílanou komunikaci rozšifrovat. Jak je znázorněno na obrázku 2.10.

[10]Obrázek vychází z dokumentace [85].

Kapitola 3

Tvorba webových a multiplatformních aplikací

Před samotným vytvářením nové aplikace je vhodné poznat samotné principy tvorby. Tím se dosáhne efektivnějšího vývoje. V následující kapitole se věnuji principům tvorby webových aplikací (3.1) a různým možnostem vývoje multiplatformních aplikací na základě zvoleného programovacího jazyka (3.2).

3.1 Principy tvorby webových aplikací

Webové aplikace se skládají ze dvou hlavních částí klientské aplikace, která běží ve webovém prohlížeči a serverové části, která běží většinou na jiném zařízení. Serverová část poskytuje přes HTTP (*Hypertext Transfer Protocol*) webovému prohlížeči zdrojové kódy klientské části aplikace. Webový prohlížeč následně zobrazuje uživateli klientskou část aplikace na základě získaného zdrojového kódu.

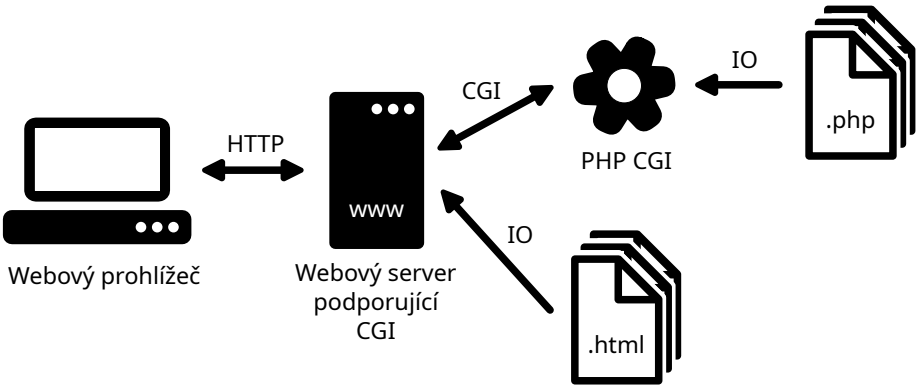
3.1.1 Serverová část

Serverová část vždy obsahuje nějaký webový server, který implementuje určitou verzi protokolu HTTP. Tento protokol slouží pro komunikaci s webovým serverem. Protokol HTTP byl využíván už od samotného počátku webu [7].

Jak popisuje dokument [14], na počátku webu prvotní univerzální webový server, který byl vyvíjen v CERNu, byl schopný pouze poskytovat statické soubory klientské části aplikace. Prvotní verze univerzálního webového serveru nebyly schopné nijak reagovat na uživatelské vstupy, které by například umožnily vyhledávání. Později byl tento problém vyřešen přidáním rozhraní *index search interface*, které umožňovalo doprogramovat externí skripty pro vyhledávání. Toto rozhraní se moc neuchytilo a dnes se již nevyužívá v moderních webových serverech.

Mezitím tým z NCSA (National Center for Supercomputing Applications) vyvíjel vlastní webový server (NCSA Httpd), pro který vytvořil rozhraní *Common Gateway Interface* zkráceně CGI na základě komunikace s ostatními vývojáři webových serverů [17]. Toto rozhraní umožnilo spouštění externích programů, které sloužily pro generování virtuálních stránek [17]. Samotné virtuální stránky mohly být vygenerovány na základě informací od uživatelů. CGI bylo později implementováno i do webového serveru od CERNu [14]. CGI bylo první rozhraní, které se více používalo a umožnilo uživateli klientské aplikace interaktivně

pracovat se serverovou částí webové aplikace. CGI je podporováno i některými dnešními webovými servery pomocí modulu [23, 78].



Obrázek 3.1: Komunikace web serveru s CGI[1]

Jak je popsáno v [16], CGI funguje následovně: pokud je na server odeslán požadavek na zdroj, který je generován externím programem. Webový server převezme parametry z dotazu a nastaví je jako proměnné prostředí pro externí program, který následně spustí. Pokud požadavek na server obsahuje dodatečná data zadaná uživatelem, která nejsou vidět v URL adrese, jsou tato data zapsána na standardní vstup externího programu. Externí program následně informace zpracuje, zpracovaná data vypíše na standardní výstup a ukončí se. Standardní výstup externího programu přečte webový server a odešle přečtená data klientovi. Komunikace je znázorněná na obrázku 3.1.

Jak naznačuje dokumentace [11], postupem času se ukázalo rozhraní CGI jako neefektivní, jelikož bylo při každém dotazu potřeba spouštět znovu externí program. Proto vzniklo nové rozhraní FastCGI. FastCGI se snaží zrychlit zpracovávání tím způsobem, že umožňuje nechávat spuštěný externí program mezi požadavky a dovoluje zpracovávat více požadavků paralelně v rámci jednoho spuštěného externího programu [11]. Díky tomu, že externí program nemusel být mezi požadavky ukončován, byl schopný mezi dotazy používat mezipaměť, nebo udržovat spojení s databázovými servery.

Jak je dále popsáno v [11], vzhledem k tomu, že u FastCGI bylo možné nechat externí program spuštěný mezi více různými požadavky. Nebylo možné předávat parametry dotazů přes proměnné prostředí jako u CGI. Proto FastCGI používá pro předávání parametrů s webovým serverem soket, pro který definuje svůj vlastní komunikační protokol.

Později k FastCGI vzniklo FPM (*FastCGI Process Manager*), které implementuje dodatečnou funkcionalitu pro těžce zatěžované webové servery [59]. FPM se využívá dodnes u webového serveru Nginx[2] [42]. U tohoto webového serveru slouží pro generování virtuálních stránek a jiných virtuálních souborů.

Na druhou stranu webový server Apache, který byl odvozen od webového serveru NCSA Httpd [33]. Byl podle [57] navržen s hlavní myšlenkou modularity, která umožňovala vylepšit rychlost webového serveru. Pokud někdo nepotřeboval některý modul, mohl tento modul zakázat. Navíc moduly umožňovaly komukoliv implementovat funkcionalitu do webového serveru, kterou potřeboval a nebyla ještě naprogramována. Jak popisuje [33] později vznikl modul pro Perl, který umožňoval vytvářet virtuální stránky. Tento modul v tehdejší době

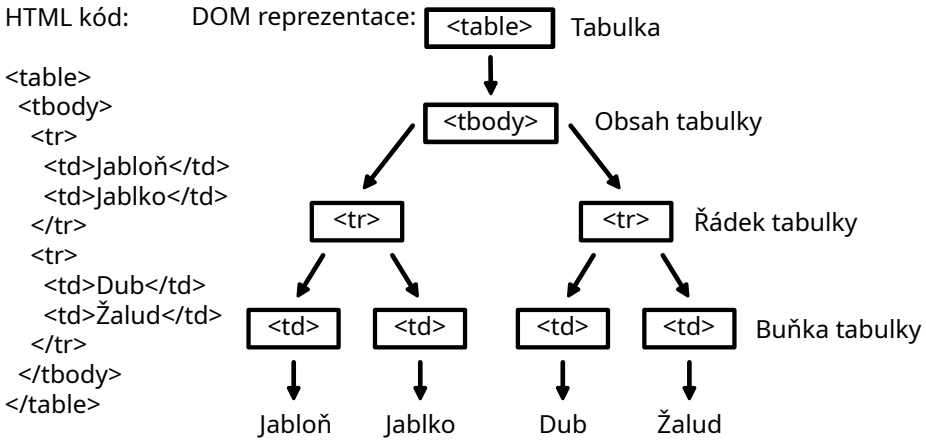
[1] Obrázek vychází z informací ze standardu [16].
[2] https://nginx.org/en/

využívala většina vývojářů. Později následovaly moduly pro další programovací jazyky. Tyto moduly pro jiné jazyky, jako je například PHP se používají dodnes.

Pokud někdo dnes vyvíjí nějakou webovou aplikaci, využívá webový server. Zároveň využitý webový server s vysokou pravděpodobností používá jedno z dříve zmíněných rozhraní pro generování virtuálních webových stránek a virtuálních souborů. To umožňuje serverové části webové aplikace interaktivně reagovat s uživatelem. Znalosti těchto rozhraní můžou pomoci při navrhování architektury webové aplikace.

3.1.2 Klientská část

Už od samého počátku webu prohlížeče pracují se značkovacím jazykem HTML (*Hypertext Markup Language*), jak je zmíněno v samotném dokumentu od autorů webu [7]. Tento jazyk slouží dodnes pro tvorbu klientských částí webových aplikací. Jak popisuje dokument [44], samotné soubory napsané za pomoci tohoto jazyka se skládají z tagů, které definují jednotlivé prvky webové stránky. Mezi těmito prvky může být například: text, odkazy, nadpisy a tabulky.



Obrázek 3.2: Grafická reprezentace DOM[3]

Jak popisuje standard [27], soubory napsané v tomto značkovacím jazyce lze reprezentovat pomocí DOM (*Document Object Model*). DOM umožňuje reprezentovat obsah souboru HTML jako objekty a tím zjednodušuje práci s těmito soubory v objektově orientovaných programovacích jazycích. Programy napsané pomocí těchto jazyků mohou například sloužit pro generování virtuálních stránek. Virtuální stránky jsou zmíněny v podkapitole 3.1.1. Grafická reprezentace DOM je znázorněna na obrázku 3.2.

Samotné HTML, ale začalo být nedostatečné s postupnými vyššími nároky. Jak je uvedeno v [9], ze začátku nebyl žádný způsob, jak vytvořit globálně styly pro webové stránky. Nebylo možné například vytvořit rozložení stránky pomocí stylů. Samotný vzhled všech stránek se musel definovat v každém souboru HTML zvlášť. Jako řešení tohoto problému vznikl například prohlížeč Viola, který používal svůj vlastní stylovací jazyk. Později se začala vytvářet specifikace kaskádových stylů (CSS), která byla velmi diskutována. Kaskádové styly byly později implementovány do internetového prohlížeče Internet Explorer a byly následně standardizovány. Kaskádové styly do dnešního dne slouží pro vytváření stylů klientských částí webových aplikací.

[3] Jedná se o upravený obrázek z dokumentu [27].

Jak popisuje kniha [62], na počátku webových prohlížečů neexistovaly stránky, které by se mohly měnit dynamicky bez interakce se serverem. Například pokud uživatel vyplňoval formulář a měl v něm chybu, tak na tuto chybu mohl upozornit až webový server. Zpočátku byl nejvíce populární prohlížeč Netscape. Vývojáři Netscape si tento problém uvědomili a snažili se tento problém vyřešit. Nakonec přišli se skriptovacím jazykem JavaScript, který byl následně využíván se značkovacím jazykem HTML.

Jak je v knize [62] dál popsáno, z počátku JavaScript umožňoval jen dynamicky měnit obsah stránky bez komunikace se serverem. Byl schopný kontrolovat částečně obsah formulářů. Později Microsoft ve svém prohlížeči Internet Explorer 5 představil první rozhraní, pomocí kterého byl JavaScript schopný komunikovat se serverem. Toto rozhraní bylo dlouho přehlíženo, dokud společnost Google nepřišla s interaktivními mapami, které využívaly toto rozhraní. Jesse James Garrett následně zjistil, že Google mapy sdílejí určité rysy s jinými interaktivními webovými stránkami. Tyto rysy pojmenoval *Asynchronous JavaScript and XML* (Ajax). Jak plynul čas, vznikaly stále složitější interaktivní webové stránky. Při jejich tvorbě ovšem byl největší problém práce s modelem DOM. Proto vznikla knihovna jQuery, která usnadňovala tuto práci.

S postupem času dále vznikají nové knihovny, které se snaží zjednodušit vývoj. Dnes mezi nejvíce používané knihovny v době psaní této práce (květen 2025) patří například jQuery, Bootstrap, Underscore, Modernizr a React [80].

3.2 Multiplatformní vývoj

Multiplatformní aplikace je taková aplikace, která má jednotný základní kód pro více operačních systémů [71]. K vývoji multiplatformních aplikací lze využít překládané jazyky 3.2.1, jazyky překládané do bajt kódu 3.2.2 a plně interpretované jazyky 3.2.3.

3.2.1 Strojový kód

Jak je uvedeno v [84], pro vývoj multiplatformních aplikací lze využít překládané programovací jazyky, jako je například C a C++. Tyto jazyky se překládají do strojového kódu pro každou platformu a architekturu procesoru zvlášť. Při použití těchto programovacích jazyků má hlavní část aplikace společný kód pro všechny platformy. Tento společný kód je doplněn o části kódů, které jsou jen pro dané platformy. Tyto části kódu často slouží k ovládání hardwaru dané platformy, jelikož některé specifické akce nelze zajistit na různých platformách stejným kódem.

Tyto jazyky vyžadují zvlášť překlad pro různé platformy i pro různé architektury procesorů [38]. Jelikož různé platformy mohou mít různá systémová volání implementovaná v jádře operačního systému [32, 30]. Zároveň různé architektury procesorů mohou implementovat odlišné strojové instrukce [15, 3]. Proto je vhodné pro multiplatformní vývoj s těmito jazyky využít pro správu překladu CMake [84]. CMake umožňuje jednodušší překlad aplikace mezi různými platformami a umožňuje tím celkově ulehčit vývoj multiplatformních aplikací [35].

3.2.2 Bajt kód

Další možností pro vývoj multiplatformních aplikací jsou programovací jazyky, které se překládají do bajtkódu. Nejspíše každý zná programovací jazyk Java. Tento jazyk se překládá do bajtkódu, jak je uvedeno v [72].

Dále zdroj [72] popisuje, jak vznikl tento programovací jazyk. Programovací jazyk Java vznikl jako část velkého projektu, který byl původně psaný v C++. Jazyk vznikl z důvodu problémů, se kterými se vývojáři potýkali. Původně měli problémy s překladačem programovacího jazyka. Postupem času i s jazykem C++ samotným. S narůstajícími problémy naznali, že změna jazyka bude jednodušší. Při návrhu tohoto nového jazyka se snažili co nejvíce znovu použít syntaxi z jazyka C++, aby byl přechod pro programátory co nejjednodušší. V době vzniku programovacího jazyka Java byly nejvíce používané jazyky C a C++. Zároveň se snažili vyhnout funkcím z jazyka C++, které většina programátorů používala špatně. Dále se v návrhu snažili programátorům ulehčit práci přidáním automatické správy paměti oproti jazyku C++. Návrh také zahrnoval interpret bajtkódů, díky kterému mohly programy napsané v Javě běžet po jednom zkompilování do bajtkódu na různých platformách. Zároveň bajtkód byl navržen tak, aby ho šlo velmi jednoduše zkompilovat do spustitelného binárního kódu na dané platformě, kde běžel interpret bajtkódu. Bajtkód díky tomu může interpret za běhu přeložit do spustitelného binárního kódu a tím se může zvýšit rychlost vykonávání programu. Tím se snažili autoři zajistit co největší rychlost zpracovávání v náročných programech na výpočty.

3.2.3 Zdrojový kód

Pro vytváření multiplatformních aplikací je možnost využít i interpretované jazyky. V případě použití interpretovaných jazyků se výsledný program distribuuje s čitelným zdrojovým kódem, kde interpret na cílovém zařízení provádí překlad při běhu aplikace [38]. Mezi interpretované jazyky patří například Python, PHP, JavaScript [64, 1, 43].

3.2.4 Webové technologie

Postupem času se začaly používat i webové technologie k vytváření multiplatformních aplikací. Tyto aplikace lze distribuovat spolu s upraveným prohlížečem, jak se to například využívá v Electronu [54]. Každá aplikace ale nemusí obsahovat v sobě tento upravený prohlížeč. Například Arch Linux má v repozitáři aplikaci *code*, která používá prohlížeč z jiného balíčku pro interpretaci [20].

Podle [62] ze začátku vznikl projekt PhoneGap (open-source verze se nazývá Apache Cordova), který umožnil psát aplikace na mobilní telefony pomocí webových technologií. V PhoneGap byla doprogramována aplikační rozhraní pro přístup k nativním funkcím, jako byla například kamera nebo kontakty. Ve stejném roce vznikl ChromeOS, který je plně postavený na prohlížeči Google Chrome. O tři roky později vyšel Windows 8 s některými základními aplikacemi napsanými pomocí webových technologií.

O dva roky později vyšel framework Electron pro psaní webových aplikací pro stolní počítače [90]. Tento framework se stal s postupem času velmi populární [61]. V době psaní této práce jsou v něm vytvořeny například aplikace Visual Studio Code, Skype, Discord, Slack, Figma, Balena Etcher [55, 5]. Electron a Apache Cordova lze spojit dohromady a lze vytvářet za pomocí těchto frameworků multiplatformní aplikace, které fungují, jak na počítačích, tak i na telefonech [79]. Což je hlavní výhoda oproti programovacímu jazyku Java, s kterým by se vývoj pro různé mobilní operační systémy dělal jen velmi obtížně.

Jak je uvedeno v dokumentu z konference [60] tyto frameworky umožňují snížit náklady spojené s vývojem aplikací na více různých systémů. Dříve než byly tyto frameworky vytvořeny, bylo nutné pro každý systém vyvíjet aplikaci zvlášť s knihovnami a programovacími jazyky, které byly na daném zařízení podporovány.

Kapitola 4

Analýza požadavků

Předtím, než vznikne nové řešení, je vhodné udělat analýzu požadavků uživatelů a analýzu existujících řešení, aby se předešlo vzniku nové aplikace, která nebude nakonec používána. Proto se v této kapitole věnuji požadavkům uživatelů a současnými řešeními, které uživatelé mají k řešení těchto problémů. Věnuji se nedostatkům těchto řešení a předkládám návrh požadavků na nové řešení, které by lépe splňovalo požadavky uživatelů.

4.1 Analýza cílových uživatelů a jejich potřeb

Aplikace se bude orientovat na sdílení služeb přes internet. Je cílená na kohokoliv, kdo bude potřebovat sdílet nějakou službu přes internet a nebude si chtít připlácet za možnost otevřít port k naslouchání na veřejné IP adrese. Bude tedy cílená na dvě skupiny uživatelů:

1. **Uživatelé bez znalostí počítačových sítí** vyžadují jednoduchý způsob, jak vše nastavit, bez nutnosti číst odbornou literaturu k počítačovým sítím. Současně musí být výsledné nastavení co nejvíce bezpečné. Dále vyžadují jednoduchý způsob instalace aplikace.

2. **Uživatelé se znalostmi počítačových sítí** vyžadují možnost vše si sami podrobněji nastavit, popřípadě doladit nastavení tak, aby co nejlépe vyhovovalo jejich účelu. Pokročilí uživatelé uvítají plnou kontrolu nad serverem, který bude virtuální síť kontrolovat. Díky tomu nebudou muset spoléhat na zabezpečení poskytovatele serveru a budou schopni server provozovat bez nutnosti mít přístupnou administraci mimo virtuální síť.

Během vytváření této práce jsem komunikoval s menší uživatelskou základnou o počtu 5 lidí, kteří aktivně používají jedno z dosavadních řešení. Z toho 3 uživatelé měli pokročilé znalosti počítačových sítí. Měl jsem s nimi vytvořenou chatovací skupinu, kde jsem s nimi komunikoval. Menší skupina mi umožnila nahlédnout hluboko do detailů potřeb samotných uživatelů. Což by se s velikým množstvím uživatelů dělalo jen velmi obtížně. Na základě komunikace s uživatelskou základnou lze předpokládat, že uživatelé budou chtít aplikaci používat nejčastěji k těmto účelům:

1. **Sdílení služby serveru**: mezi první kategorii služeb, které uživatelé sdílí, lze zařadit služby, které nevyžadují *broadcast*. Patří sem sdílení síťových úložišť, kde jsou podle rozhovorů nejvíce využívané protokoly SMB (*Server Message Block*), SFTP (*SSH file*

transfer protocol) a HTTP pro Nextcloud[1]). Uživatelé navíc, kromě sdílení souborů, využívají server SSH pro vzdálenou správu serverů, které mají doma, pro vzdálený vývoj v programu Visual Studio Code[2] a pro správu virtuálních strojů v aplikaci Virtual Machine Manager[3]. Dále někteří uživatelé přistupují vzdáleně domů k aplikaci Home Assistant[4] přes protokol HTTP.

2. **Hraní LAN her s přáteli**: další kategorii je hraní LAN her s přáteli, které vyžadují *broadcast* pro nalezení druhé instance. Díky tomu uživatelé nemusí při vytváření nové hry zadávat IP adresy.

3. **Přeposílání komunikace přes třetí prvek do internetu**: tuto kategorii si představuje mnoho uživatelů, když se zmiňuje VPN. V podstatě se jedná o posílání dodatečně zašifrované komunikace k jinému zařízení připojenému k internetu, které následně rozšifruje dodatečné šifrování a přepošle komunikaci pod svoji IP adresou do internetu. Uživatelé tuto funkci využívají, když jsou na dovolené a v dané zemi jsou zablokované jisté služby na internetu, které chtějí využívat. Pokud uživatelé chtějí k těmto službám přistoupit, přeposílají svoji komunikaci přes zařízení, které mají doma.

4. **Vzdálený přístup k vypnutému počítači**: toto je docela zajímavá kategorie, která je naprosto reálná, nejedná se o vtip. V případě, kdy se uživatel chce vzdáleně připojit k počítači, který je doma vypnutý. Může využít domácí WiFi router, na který se připojí a zapne vypnutý počítač pomocí WoL (*Wake on LAN*) paketu. Po zapnutí počítače, se uživatel může na daní počítač vzdáleně připojit a pracovat s ním. Po dokončení práce může zase počítač vypnout pro šetření energie.

5. **Sdílení herních serverů pro širokou veřejnost (veřejná síť)**: další kategorií je sdílení herních serverů pro širokou veřejnost bez nutnosti mít otevřený port k naslouchání na veřejné IP adrese, kterou má uživatel přiřazenou od poskytovatele internetu. Jak je popsáno v 2, ne vždy mají uživatelé možnost otevřít port k naslouchání.

6. **Vývoj aplikací s mikroslužbami**: poslední kategorií je vývoj aplikací využívající mikroslužby. Mikroslužby je vhodné rozdělit mezi více serverů, které následně komunikují mezi sebou. Zároveň by komunikace mezi servery měla být zašifrovaná dodatečnou vrstvou, pokud mikroslužby mezi sebou komunikují bez šifrování.

4.2 Současný stav

Během let vzniklo mnoho aplikací, které částečně řeší některé problémy. Tyto aplikace, ale bývají často různě omezeny v neplacené verzi, nebo vyžadují nutnost mít minimálně jednu veřejnou IP adresu a na ni mít otevřený jeden či více portů k naslouchání v případě vlastního hostování aplikace.

[1] https://nextcloud.com/
[2] https://code.visualstudio.com/
[3] https://virt-manager.org/
[4] https://www.home-assistant.io/

Hamachi

První možnost, která se nabízí uživatelům, je Hamachi[5]. Hamachi umožňuje v neplacené verzi připojit do sítě jen 5 zařízení v době psaní této práce (leden 2024) [40]. Díky tomu je Hamachi vhodné v neplacené verzi jen pro malé sítě. Hamachi sice umožňuje oddělit servery od klientů ve virtuální síti [25]. Ale neumožňuje samotné na serveru vybrat konkrétní služby, které mají být sdílené. Pro tento účel je nutné doinstalovat a nastavit dodatečný firewall na serveru.

V případě, kdy uživatel nechce platit, je též Hamachi nevhodné pro vývoj mikroslužeb. Vzhledem k tomu, že v síti při vývoji bude s velkou pravděpodobností více serverů a každý, kdo bude na projektu pracovat, bude i s velkou pravděpodobností používat více než jedno zařízení. Limit 5 zařízení se velmi rychle vyčerpá. Na druhou stranu Hamachi lze jednoduše použít ke hraní LAN her s méně kamarády v neplacené verzi.

GameRanger

GameRanger[6] je určený jen pro hraní her. Implementuje přenos hlasu, okamžité zasílání zpráv, list přátel a žebříčky s hodnocením hráčů [24]. Nelze jej využít k ostatním účelům, jako je například sdílení souborů. Neumožňuje hrát jiné hry než ty, které jsou v předem definovaném seznamu her[7].

OpenVPN / WireGuard

Jak vyhází z manuálů [52, 4], OpenVPN a WireGuard jsou protokoly, které lze k vytváření virtuálních sítí též použít. Pokud si chce uživatel vytvořit přes tyto protokoly virtuální privátní síť, potřebuje veřejnou IP adresu minimálně na jednom zařízení s otevřeným portem k naslouchání. Sami o sobě neumožňují přímo propojit dvě zařízení, která jsou obě za překlady síťových adres, bez nastaveného přesměrování portů. Díky tomu je nutné, pro možnost komunikace mezi těmito zařízeními, mít třetí zařízení přeposílající komunikaci a zároveň musí být toto třetí zařízení viditelné z internetu. Díky otevřenosti těchto řešení existují klienti na velké množství zařízení.

Bohužel řešení samy o sobě nijak neimplementují možnost filtrovat komunikaci. Pokud chce uživatel filtrovat komunikaci ve virtuální privátní síti, musí do sítě přidat firewall mezi koncová zařízení. Řešení spolu s firewallem lze využít ke všem účelům, které uživatelé vyžadují. Nicméně pro jejich správné použití je potřeba mít odborné znalosti, což zahrnuje jen část uživatelů. Dále je vyžadováno u těchto řešení mít otevřený port k naslouchání na veřejné IP adrese, což je v rozporu s potřebami uživatelů.

SoftEther VPN

Usnadnit práci oproti OpenVPN/WireGuard se snaží SoftEther VPN[8]. Řešení integruje více nástrojů dohromady, které je jinak nutné instalovat a konfigurovat separátně. Patří sem například filtrování paketů ve virtuální síti [66].

Pro optimální funkci SoftEther VPN je vhodné mít server dostupný z veřejné IP adresy. Adresa nemusí být staticky přidělená. Tato aplikace integruje DDNS [67]. DDNS je

[5] https://vpn.net/
[6] https://www.gameranger.com/
[7] Předdefinovaný seznam her je k dispozici zde: https://www.gameranger.com/games/
[8] https://www.softether.org/

zmíněno v kapitole 2.1. SoftEther VPN pro lepší konektivitu implementuje techniky děrování překladu síťových adres [67]. Tato implementace se hodí v případě, kdy server je za překladem síťových adres, nebo firewallem, které v daném nastavení neumožňují serveru naslouchat na veřejné IP adrese. V případě, kdy server není schopen naslouchat na veřejné IP adrese a implementované techniky děrování překladu síťových adres nejsou úspěšné, lze využít VPN Azure Service[9]. Tuto službu SoftEther VPN nabízí spolu s univerzitou Tsukuba, jako akademický experiment svým uživatelům zdarma [69] (v době psaní této práce v lednu 2025).

Jak je popsáno v [68] SoftEther VPN implementuje protokoly L2TP/IPsec, OpenVPN, MS-SSTP, L2TPv3 a EtherIP. Díky tomu lze použít jakéhokoliv klienta, který podporuje alespoň jeden z těchto protokolů. Lepší je každopádně použít přímo klienta SoftEther VPN, jelikož některé techniky děrování překladu síťových adres vyžadují dodatečnou interakci i od klienta. Příkladem může být UDP děrování, jak je popsáno v podkapitole 2.3.2.

SoftEther VPN je velmi komplexní řešení, které je spíše určeno pro uživatele s pokročilými znalostmi počítačových sítí. Je obtížné vše správně nastavit pro uživatele bez pokročilých znalostí. Zároveň SoftEther VPN je poměrně velká aplikace, díky tomu se moc nehodí na síťové prvky a nemá webovou aplikaci.

Ngrok

Ngrok[10] je určený pro sdílení služeb, které využívají TCP protokol. Využívá přesměrování portů přes další zařízení [50]. Tato metoda je popsána v kapitole 2.2. Všechny služby, které jsou přes Ngrok sdílené, jsou veřejně viditelné z internetu. Ngrok se převážně zaměřuje na webové stránky. Snaží se zjednodušit jejich provozování. Umožňuje automaticky transformovat komunikaci z HTTP protokolu na HTTPS protokol [48].

Jak je popsáno v [51, 49], Ngrok v době psaní této práce (leden 2025) umožňuje v neplacené verzi po založení účtu jen sdílet služby využívající protokol HTTP. Po přidání platební karty je možné bez placení sdílet i služby využívající protokol TCP. Neplacené verze jsou určeny jen pro testování a vývoj aplikací.

Samotný Ngrok nelze využít k hraní LAN her přes internet. Jelikož LAN hry mohou využívat *broadcast* k nalezení druhé instance hry. Navíc nemusí počítat s přenosem dat přes veřejnou síť, a tudíž nemusí například implementovat šifrování dat.

Serveo

Podobné funkce nabízí Serveo[11]. Serveo sice nemá tolik funkcí jako Ngrok, ale nabízí některé funkce, které jsou u Ngrok placené. Dále nevyžaduje registraci pro používání a nemá žádný placený plán [18].

Jak je popsáno v [18], Serveo umožňuje také sdílet služby využívající protokol TCP. Pro sdílení služeb využívající protokol TCP není potřeba žádné vytváření a ověřování účtu. Snaží se stejně, jako Ngrok zjednodušit sdílení webových serverů. Umožňuje automaticky transformovat komunikaci HTTP na HTTPS.

Serveo nelze využít k hraní LAN her s přáteli stejně jako Ngrok. Vzhledem k velmi podobné funkcionalitě se potýká se stejnými problémy jako Ngrok. Jak bylo zmíněno u Ngrok, LAN hry byly navrženy pro hraní na LAN síti. Nemusí počítat s přenosem dat přes internet.

[9] https://www.vpnazure.net/en/
[10] https://ngrok.com/
[11] https://serveo.net/

Tailscale

TailScale[12] je určen pro uživatele s pokročilými znalostmi počítačových sítí, které jsou nutné pro jeho správné nastavení. Je podporován na stolních počítačích, telefonech a některých síťových prvcích a chytrých úložištích [75, 6, 76]. TailScale lze využít ke všem nejčastějším účelům, které uživatelská základna uvedla.

TailScale umožňuje v neplacené verzi bez hostování vlastního koordinačního serveru připojit do virtuální privátní sítě až 100 zařízení (v době psaní této práce leden 2025) [74]. Koordinační server sice nemá otevřený zdrojový kód, ale vznikl koordinační server Headscale[13], který má otevřený zdrojový kód a jde používat k vlastnímu hostování [77]. Bohužel, jak vyplývá z architektury technologie uvedené v 2.3.2, tak tento server musí být přístupný skrze nějakou veřejnou IP adresu, jinak nebude fungovat správně.

V posledních letech vyšly další alternativy k TailScale, které vycházejí ze stejné hlavní myšlenky. Mají mezi sebou menší odlišnosti. Stejně jako TailScale ale vyžadují, aby server, který spravuje virtuální sítě, byl přístupný skrze nějakou veřejnou IP adresu [47, 46]. Proto jsou nevhodné, stejně jako TailScale, pro některé cílové uživatele. Někteří uživatelé chtějí provozovat vlastní server pro správu virtuální sítě, ale nemají možnost otevřít serveru port k naslouchání na některé veřejné IP adrese. K těmto alternativám například patří Netmaker[14] a NetBird[15].

ZeroTier

Nejlepší možností je ZeroTier[16]. Toto řešení má klientskou aplikaci pro počítače, mobilní zařízení a lze jej spustit dokonce i na některých síťových prvcích a síťových úložištích [88]. Zároveň jde využít stejně jako TailScale ke všem nejčastějším účelům, které zmínila uživatelská základna.

ZeroTier, v neplacené verzi bez hostování vlastního kontroléru sítě, umožňuje připojit do virtuální sítě 10 zařízení v době psaní této práce (únor 2025) [87]. ZeroTier dále nabízí možnost hostovat vlastní kontrolér, který spravuje vytvořené sítě [89, 85]. Tento kontrolér je možné podle licence [28] provozovat v případě, kdy není prodávaný jako SaaS, není provozovaný v některých státních organizacích a není provozovaný jako výdělečná služba v době psaní této práce (leden 2025).

Jak je uvedeno v podkapitole 2.3.2, kontrolér platformy ZeroTier jde provozovat díky chytré architektuře bez nutnosti mít otevřený port k naslouchání na veřejné IP adrese. Kontrolér, který je takto provozovaný, není nijak omezený a může zároveň poskytovat větší bezpečnost, pokud jeho rozhraní k nastavování sítě není přístupné z internetu. Jako bonus navíc, díky vlastnímu hostování, odpadá prostředník, který potenciálně může přidávat zařízení do virtuální počítačové sítě. Nicméně oficiální aplikace, která je určena pro ovládání kontroléru, není veřejnosti přístupná. Tato webová aplikace má uzavřený zdrojový kód a je určena jen pro uživatele s pokročilými znalostmi počítačových sítí.

Jako řešení tohoto problému vznikly dvě webové aplikace (ztncui[17] a ZeroUI[18]), které cílí bohužel opět jen na uživatele s pokročilými znalostmi počítačových sítí, Linuxu a popřípadě

[12] https://tailscale.com/
[13] https://headscale.net/
[14] https://www.netmaker.io/
[15] https://netbird.io/
[16] https://www.zerotier.com/
[17] https://key-networks.com/ztncui/
[18] https://github.com/dec0dOS/zero-ui

kontejnerů. Navíc tyto aplikace nejsou kompletní a neimplementují veškerou funkcionalitu nastavení kontroléru. Zároveň obě aplikace jsou napsány s použitím Node.js[19] [34, 2]. Díky tomu jsou aplikace náročné na zdroje a nejsou vhodné pro spouštění na síťových prvcích, popřípadě na síťových úložištích. Na těchto zařízeních by bylo vhodné webové aplikace v domácích podmínkách provozovat, jelikož pak není nutné mít někde navíc spuštěný server. Dále ani jednu z aplikací nelze nainstalovat jako desktopovou aplikaci, což by bylo vhodné pro méně znalé uživatele, nemuseli by otevírat prohlížeč internetových stránek a zadávat do něj adresu lokálního serveru.

Zmíněné aplikace navíc neoddělují v nastavení virtuální sítě servery od klientů, což zhoršuje celkovou přehlednost. Pokud někdo chce upravovat nastavení serverů, nebo zkontrolovat jejich stav, musí je hledat v seznamu se všemi zařízeními ve virtuální síti. Obě aplikace dále neumožňují spravovat více instancí kontroléru platformy ZeroTier a nemají žádný přehledný *dashboard*, který by zobrazil stav celé infrastruktury, případně hned přehledně upozornil na možné problémy. Dále neimplementují možnost připojit se k více kontrolérům platformy ZeroTier. Využití více ZeroTier kontrolérů by umožnilo vytvářet větší virtuální sítě i v případě, kdyby všechny kontroléry byly spuštěny jen na síťových prvcích a síťových úložištích.

Jen jedno z dosavadních řešení umožňuje vytvářet více uživatelů, a to ztncui, ale neimplementuje možnost uživatelům nastavovat oprávnění. Což by se v některých situacích mohlo hodit. Například pokud studenti spolupracují na projektu, bylo by vhodné spolupracujícím studentům umožnit přidávat nová zařízení do virtuální privátní sítě, ale neměli by mít možnost vidět ostatní virtuální sítě.

Jen ZeroUI implementuje možnost nastavovat pravidla komunikace ve virtuální síti, která jsou důležitá pro zvýšení celkové bezpečnosti. Pravidla komunikace jsou velmi důležitá, například pokud někdo chce vytvářet veřejnou virtuální síť. Jelikož umožňují zablokovat veškerou komunikaci, která je mířená na jiné zařízení, než jsou servery. Na druhou stranu ZeroUI zase neimplementuje možnost nastavovat server DNS pro virtuální síť. Všechna zařízení se musí adresovat přes IP adresy. Možnost nastavit server DNS implementuje jen ztncui.

[19] https://nodejs.org/

Tabulka 4.1: Porovnání řešení v neplacené verzi

	Hamachi	GameRanger	OpenVPN / WireGueard	SoftEther VPN	Ngrok	Serveo	Tailscale / Headscale	ztncui / ZeroUI	ZeroTier Central
nastavení pro neznalé uživatele	✓	✓	✗	✗	✓	✓	✗	✗	✗
jednoduchá instalace klienta pro neznalé	✓	✓	✓	✓	✓	✓	✓	✓	✓
jednoduchá instalace aplikace pro správu	–	–	–	✓	–	–	✗	✗	–
podrobné nastavení virtuální sítě	✗	–	✗	✓	–	–	✓	✓	✗
možnost provozování vlastní instance	✗	✗	✓	✓	✗	✓	Headscale	✓	✗
vlastní instance bez veřejné IP	–	–	✗	–	–	✗	✗	✓	–
sdílení služby na severu	✓	✗	✓	✓	✓	✓	✓	✓	✓
přístup k internetu	✗	✗	✓	✓	✗	✗	✓	✓	✗
veřejné herní servery	✗	✓	✗	✗	✓	✓	✓	✓	✓
vhodné pro mikroslužby	✗	✗	✓	✓	✗	✗	✓	✓	✗
webová aplikace pro správu	✓	✗	✗	✗	✓	✗	✓	✓	✓
nastavení desktopovou aplikací	✗	✓	✗	✓	✗	✗	✗	✗	✗
hraní LAN her	✓	✓	✓	✓	✗	✗	✓	✓	✓

4.3 Požadavky na nové řešení

Tabulka 4.1 porovná dostupná řešení z hlediska uživatelských požadavků. Nejméně uživatelských požadavků ze všech řešení splňuje GameRanger, který slouží jen pro hraní her a nelze jej použít k jiným účelům. Dále nejméně požadavků splňují řešení, která neimplementují virtuální počítačové sítě (Ngrok, Serveo). Tato řešení lze vhodně použít jen pro sdílení konkrétní služby ze serveru. Současně samotné zabezpečení následně řeší daná konkrétní služba za sebe. Zároveň tato řešení vytvářejí vyšší latenci, jak je popsáno v 2.3.1.

Ze současných možností nejlépe splňují požadavky uživatelů řešení implementující virtuální počítačové sítě. Řešení, která neimplementují virtuální počítačové sítě, nedokážou sdílet LAN hry, které k nalezení druhé instance používají *broadcast*. Zároveň řešení neimplementující virtuální počítačové sítě jsou nevhodná pro služby, které nepoužívají šifrování ke komunikaci a u kterých je šifrování vyžadováno. Dále virtuální počítačové sítě jsou schopné povolit přístup k dané službě jen daným oprávněným uživatelům. Mezi řešení implementující virtuální sítě patří Hamachi, OpenVPN/WireGuard, SoftEther VPN, TailScale a ZeroTier.

V případě, kdy uživatel chce mít celou virtuální síť pod kontrolou, jen jedno z řešení umožňuje kontrolér virtuální sítě provozovat na zařízení bez otevřeného portu k naslouchání na veřejné IP adrese a zároveň implementuje techniky děrování překladu síťových adres. Tímto řešením je platforma ZeroTier. K vlastnímu hostování kontroléru platformy ZeroTier lze využít aplikace ztncui a ZeroUI, bohužel tyto aplikace mají mnoho nedostatků, jak bylo zmíněno v podkapitole 4.2. Nové řešení by mělo tyto nedostatky řešit:

1. **Jednoduché vytváření privátních sítí**: dosavadní řešení neimplementují žádného průvodce, který by umožňoval jednoduše vytvářet nové virtuální privátní sítě. Tato funkce by se obzvláště hodila uživatelům bez pokročilých znalostí počítačových síti a navíc by ulehčovala vytváření nových virtuálních sítí pokročilým uživatelům. Nemuseli by pokaždé vytvářet konfiguraci plně od začátku.

2. **Desktopová a webová aplikace**: dosavadní řešení implementují pouze webovou aplikaci, která je vhodnější pro pokročilejší uživatele. Pro ostatní uživatele je vhodnější desktopová aplikace u které není potřeba do webového prohlížeče ručně zadávat adresu serveru. Na druhou stranu webová aplikace je vhodná pro uživatele, kteří budou aplikaci provozovat na domácích serverech, chytrých úložištích, nebo síťových prvcích.

3. **Odlišení serverů od klientů**: stávající řešení neoddělují servery od klientů ve virtuálních sítích. Díky tomu jsou sítě méně přehledné. Kdyby, byly oddělené výrazně by se tím zrychlilo hledání potencionálně nefunkčního severu a jeho následná oprava. Což by uživatele ocenili hlavně při vývoji aplikací využívající mikroslužby.

4. **Více uživatelů**: jen v jednom dosavadním řešením (ztncui) je implementovávaná možnost mít v aplikaci více uživatelů. Bohužel v této aplikaci má každý uživatel stejná práva a nelze to nijak změnit. Vhodnější by bylo mít možnost uživatelům nastavovat různá práva a tím tak zvýšit bezpečnost.

5. **Více instancí kontrolérů**: dosavadní aplikace neumožňují propojit více instancí kontroléru a spravovat je z jednoho místa. Tato funkce by se hodila v případě provozování kontrolérů na síťových prvcích a síťových úložištích, které nemají příliš veliký výkon. Šlo by tak jednodušeji rozprostřít virtuální síť mezi více zařízení.

6. **Dashboard**: stávající řešení neimplementují *dashboard*, který by přehledně informoval o stavu všech zařízení ve virtuální síti včetně stavu samotného kontroléru. Informace o stavu kontroléru by umožnily uživateli rozhodnou se, zda není potřeba přidat další instanci kontroléru a rozložit tak zátěž. Informace o stavu všech zařízení ve virtuální počítačové síti, by zase umožnily najít rychle všechny virtuální sítě, ve kterých nejede nějaký server. Informace o úspěšnosti děrování překladu síťových adres všech zařízení s latenci všech zařízení by usnadnily rozhodování, zda je potřeba někde poblíž začít hostovat server moon, který by sloužil k snížení latence, jak je popsáno v 2.3.2.

7. **Menší náročnost na zdroje**: dosavadní aplikace jsou náročné na výpočetní zdroje zařízeních na kterých jsou provozovány. Nové řešení by mělo být méně náročné, aby jej šlo provozovat na síťových prvcích a síťových úložištích.

Kapitola 5

Návrh řešení

Dalším vhodným krokem ve vytváření nového řešení je návrh, který bude odpovídat požadavkům uživatelů z kapitoly 4. Nový návrh zahrnuje architekturu, datový model, serverovou část a klientskou část aplikace. V architektuře se zabývám částmi, ze kterých se bude nová aplikace skládat a jejich vzájemnou komunikací. V navazující kapitole vytvářím návrh datového modelu pro databázi, který bude sloužit pro ukládání dodatečných dat, která nelze uložit na kontrolér platformy ZeroTier. Následně se zabývám serverovou částí aplikace a klientskou částí aplikace. U klientské části aplikace vytvářím návrh uživatelského rozhraní na základě zpětné vazby uživatelské základny z kapitoly 4.

5.1 Architektura

Výsledné řešení, jak vyplývá z požadavků popsaných v kapitole 4, je implementováno jako webová a desktopová aplikace. Webová aplikace je určena pro znalejší uživatele a pro provoz na síťových prvcích. Desktopová aplikace naopak je určena pro méně znalé uživatele. Aplikace se skládá z následujících komponent:

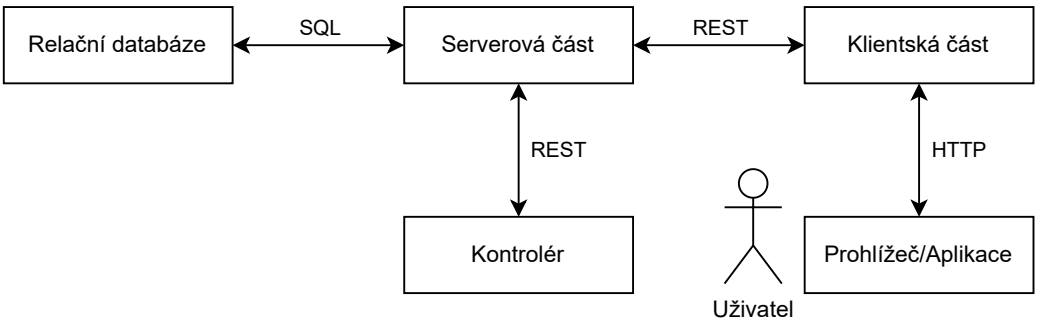
1. **Kontrolér platformy ZeroTier** je hlavní komponentou aplikace, která slouží pro spravování vytvořené virtuální sítě. Zároveň aplikace umožňuje do budoucna implementovat komunikaci i s jiným řešením než je kontrolér platformy ZeroTier v případě, kdyby se licenční podmínky změnili, nebo by vzniklo lepší řešení. Pro komunikaci s kontrolérem platformy ZeroTier se využívá rozhraní REST [86].

2. **Relační databáze** slouží pro ukládání dodatečných dat pro funkce, které nejsou implementovány na kontroléru platformy ZeroTier. Více v podkapitole 5.2.

3. **Serverová část** umožňuje hostování klientské části aplikace a zpracovává dotazy z této aplikace přes rozhraní REST. Serverová část dále komunikuje s kontrolérem platformy ZeroTier přes rozhraní REST. Implementuje dodatečnou funkcionalitu, která chybí i v dosavadních řešeních, jako je monitorování serveru, oddělení klientů od serverů a správa uživatelů s různými právy. Serverová část je více popsána v podkapitole 5.3.

4. **Klientská část** umožňuje ovládání celé aplikace ze strany uživatele a bude implementovat funkce, které dosavadní řešení nemají a jsou vyžadovány uživatelskou základnou. Návrh uživatelského rozhraní klientské aplikace je v podkapitole 5.4.

5. **Webový prohlížeč a desktopová aplikace** slouží pro interpretaci klientské části aplikace. Klientskou část aplikace získávají ze serverové části aplikace. Ulehčuje se tak celková implementace, obě verze mají jednu společnou serverovou část.



Obrázek 5.1: Architektura komunikace

5.2 Návrh datového modelu

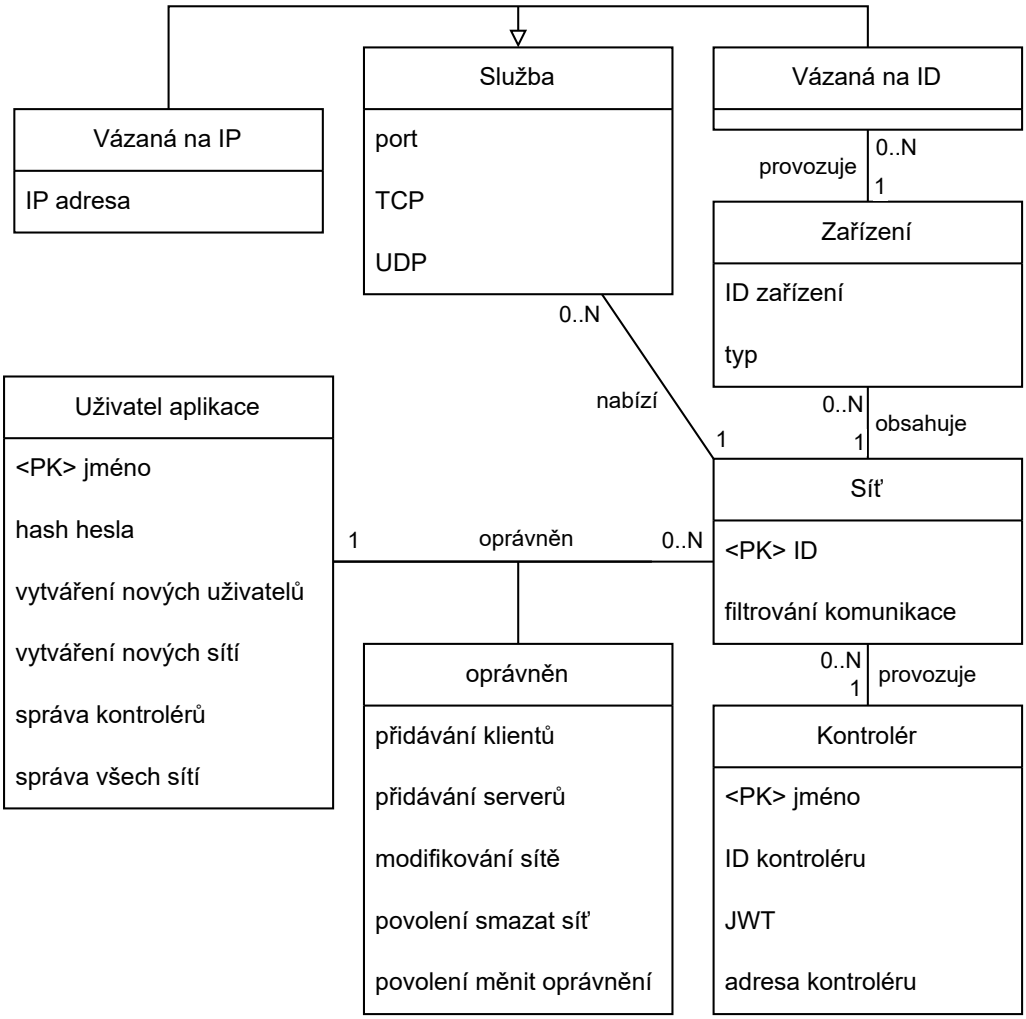
Vzhledem k tomu, že aplikace implementuje funkcionalitu navíc, kterou kontrolér platformy ZeroTier neimplementuje, vyžaduje pro tuto funkcionalitu navíc úložiště ve formě databáze. Mezi dodatečnou funkcionalitu navíc, která ulehčuje práci s novou aplikací a zároveň vyžaduje ukládání dodatečných dat patřit:

1. **Správa uživatelů a jejich oprávnění**: tato funkce umožňuje používat aplikaci více uživatelům s různými oprávněními v rámci aplikace a v rámci separátních virtuálních sítí. Toto v dosavadních aplikacích nejde. V rámci aplikace lze uživatelům nastavit oprávnění vytvářet nové uživatele, vytvářet nové virtuální počítačové sítě, spravovat kontroléry připojené do aplikace a spravovat všechny vytvořené virtuální počítačové sítě v aplikaci. V rámci virtuální počítačové sítě lze každému uživateli nastavit oprávnění přidávání klientů, přidávání serverů, upravovat nastavení virtuální počítačové sítě, oprávnění smazat síť a oprávnění měnit oprávnění ostatním uživatelům v rámci virtuální počítačové sítě.

2. **Rozdělení na servery a klienty**: aplikace dále ukládá informaci, zda je dané zařízení v rámci virtuální počítačové sítě server, nebo klient. Díky tomu je efektivnější práce s aplikací oproti dosavadním řešením, které zařízení nerozdělují.

3. **Provozované služby ve virtuální síti**: dosavadní aplikace neumožňují specifikovat porty služeb, které jsou provozovány na serverech ve virtuální síti. Jen jedno z dosavadních řešení umožňuje definovat *flow pravidla* a dovoluje tak komunikaci jen k serverům na dané služby. Nová aplikace umožňuje specifikovat porty služeb na serverech a automaticky na základě uložených portů služeb vygenerovat *flow pravidla*, která aplikuje.

4. **Možnost spravovat více kontrolérů**: nové řešení dále umožňuje propojit více kontroléru dohromady. K tomu, aby nové řešení bylo schopno komunikovat s jinými kontroléry, potřebuje ukládat informace o tom s kým a jak může komunikovat.

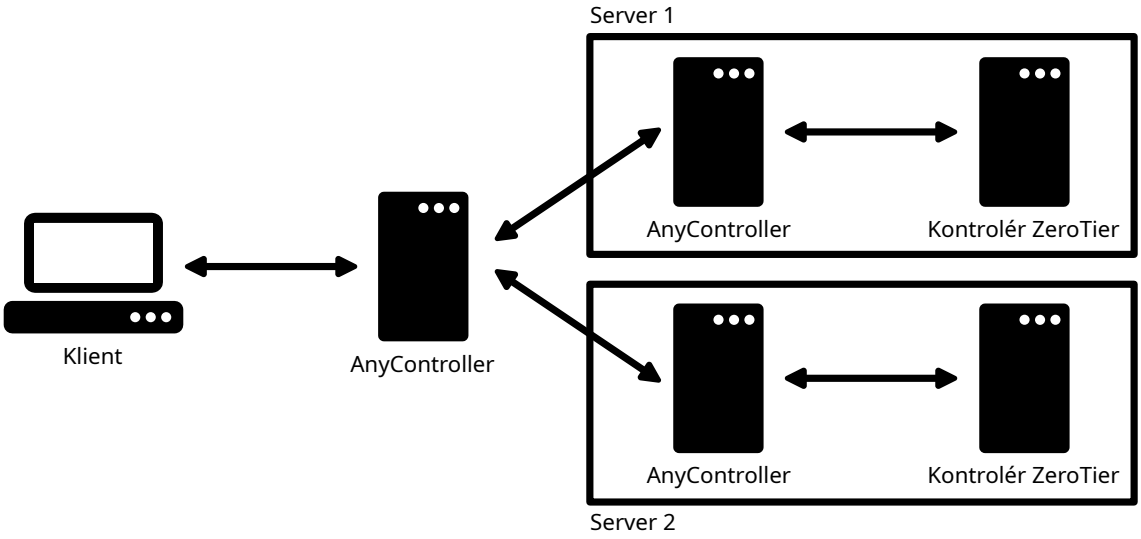


Obrázek 5.2: Entitně-vztahový model databáze

Všechna tato dodatečná data jsou zahrnuta v entitně-vztahovém modelu vyobrazeném na obrázku 5.2 včetně jejích závislostí mezi sebou.

5.3 Serverová část aplikace

Oproti dosavadním řešením je serverová část o dost méně náročná na výpočetní zdroje zařízení a zároveň je co možná nejmenší. Je to z toho důvodu, aby ji byli schopní uživatelé provozovat na síťových prvcích a síťových úložištích. Serverová část nového řešení navíc implementuje, oproti dosavadním řešením, monitorování zdrojů serveru, na kterém běží. To uživatelům umožňuje lépe rozložit zátěž mezi více kontroléry, pokud budou více kontrolérů provozovat.



Obrázek 5.3: Architektura komunikace s více instancemi aplikace

Serverová část aplikace je schopná komunikovat s jinými instancemi aplikace na jiných serverech. Podle architektury znázorněné na obrázku 5.3. Je to z toho důvodu, aby šlo monitorovat stav zdrojů jiných serverů. Dosavadní řešení neumožňují sledovat prostředky serverů a dokonce neumožňují ani propojit více instancí aplikace dohromady.

Serverová část aplikace hostuje klientskou část aplikace pro webové prohlížeče a desktopovou aplikaci. Klientská část aplikace komunikuje se serverovou částí aplikace pomocí rozhraní REST s autorizací JWT. Kompletní návrh všech koncových bodů rozhraní REST jsem vytvořil za pomoci aplikace Swagger[1]. Vzhledem k velkému množství koncových bodů (celkem 67 v době psaní této práce, květen 2025) je kompletní specifikace přesunuta na GitLab[2]. Specifikace se může s novějšími verzemi aplikace měnit. Koncové body rozhraní REST lze rozdělit na tyto hlavní části:

1. **/user**: první část implementuje operace se systémovými uživateli, umožňuje klientské části aplikace zjistit, zda je již zaregistrovaný první uživatel aplikace, nebo není. Pokud není první uživatel zaregistrovaný, umožňuje registraci prvního uživatele, který má všechna práva. Již vytvořeným uživatelům, kteří mají dostatečná práva, umožňuje vytvářet, editovat a mazat jiné uživatele aplikace. Nakonec slouží i pro přihlašování uživatelů do aplikace.

2. **/network**: největší část implementuje operace s virtuálními počítačovými sítěmi. Umožňuje vytváření nových virtuálních počítačových sítí. Dovoluje, do již vytvořených virtuálních počítačových sítí, přidávat nové servery a klienty. Umožňuje u každé sítě nastavovat přístup na privátní (vyžadující dodatečné potvrzení, při připojování nového zařízení do sítě), nebo veřejný. Nabízí možnost připojit samotný kontrolér do virtuální sítě, aby se dalo aplikaci vzdáleně ovládat přes samotnou virtuální síť a nakonec umožňuje smazat jednotlivé virtuální počítačové sítě.

   Pokročilým uživatelům dovoluje nastavit ve virtuální počítačové síti *broadcast*, *multicast* a MTU (maximální velikosti IP datagramu). Pro IPv4 adresy umožňuje nastavit

[1] https://swagger.io/
[2] https://gitlab.com/anycontroller/anycontroller/-/blob/main/backend/swagger.json

automatické přiřazování adres novým zařízením v síti z daného rozsahu, který též umožňuje nastavit. Dále nabízí možnost nastavit směrování komunikace přes jiná zařízení, čímž umožňuje virtuální počítačovou síť propojit s lokální. Stejné funkce umožňuje uplatnit i u IPv6 adres.

Pro jednodušší adresování ve virtuální počítačové síti, dovoluje přidat externí servery DNS pro překlad doménových adres na IP adresy. Pro lepší bezpečnost, dovoluje nastavit filtrování komunikace ve virtuální počítačové síti a nabízí možnost dalším uživatelům nastavit práva na úpravu konkrétních nastaveních v dané virtuální síti. Nakonec nabízí možnost získat stav virtuálních počítačových síti.

3. **/controller**: poslední část implementuje operace s kontroléry. Dovoluje získat token pro propojení se s jinou instancí aplikace (pro propojení více kontrolérů). Umožňuje připojit jiné instance do lokální za pomocí dříve zmíněného tokenu. Při připojování jiné instance, nabízí možnost ověřit nově vytvářené spojení. Tato možnost tu je z důvodu, kdyby uživatel chtěl k aplikaci připojit jinou instanci, která by aktuálně nebyla dostupná. Nakonec také dovoluje monitorovat stav jednotlivých kontrolérů, které jsou připojeny do aplikace.

5.4 Klientská část aplikace

Během vytváření návrhu uživatelského rozhraní aplikace jsem komunikoval s uživatelskou základnou z kapitoly 4 a na základě její zpětné vazby jsem návrh postupně vylepšoval. S každým uživatelem jsem různé verze návrhu procházel separátně. Kompletní návrh uživatelského rozhraní je v příloze A. Návrh uživatelského rozhraní byl vytvářen v nástroji Figma[3]. Celý návrh byl z důvodů velikosti přesunut do přílohy 5.4.

První věc, kterou jsem řešil s uživatelskou základnou, byl vzhled komponent. Nakonec jsem po všech rozhovorech došel k závěru, že se nejvíce uživatelům líbí vzhled, se kterým se každodenně nejvíce setkávají v dnešních aplikacích a jsou na něj zvyklí. Tedy pro návrh aplikace jsem se rozhodl využít komponenty z knihovny PrimeReact[4], které se všem líbily.

5.4.1 Dashboard

První stránku, kterou jsem začal navrhovat, byl přehledný *dashboard* se stavem virtuálních sítí. Tento *dashboard* se nenachází v žádné dosavadní aplikaci. Přitom by uživatelům usnadnil vyhledání virtuálních sítí, ve kterých není nějaký server online a ulehčil by uživatelům rozhodnutí, zda by bylo vhodné podniknout opatření pro snížení latence. Ke každé virtuální síti budou uvedeny následující informace:

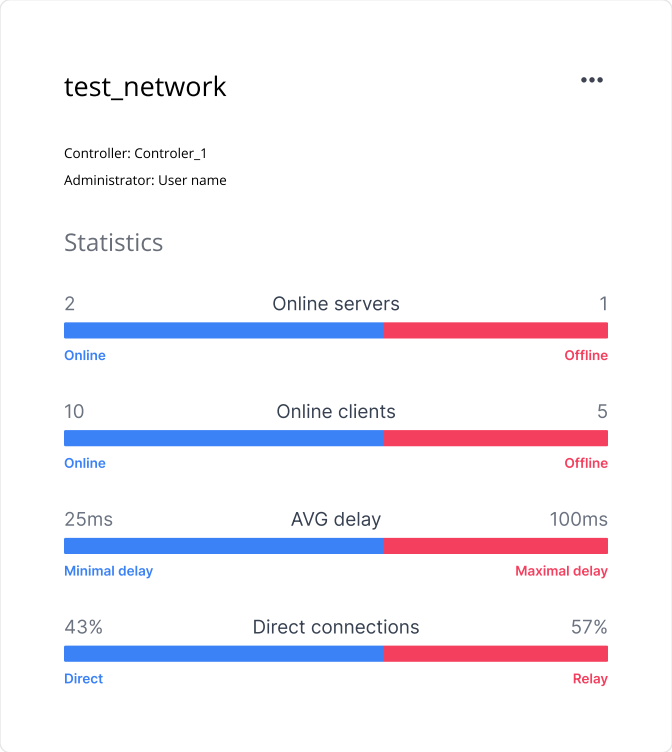
1. **Počet online a offline serverů**: pokud nějaký server v síti neběží, což by se za normálních podmínek nemělo stávat, naznačuje to problém v síti, který by se měl hned řešit.

2. **Počet online a offline klientů**: tato statistika naznačuje celkové vytížení sítě. Lze se na základě této statistiky rozhodovat, zda přidat další servery do sítě pro rozložení zátěže.

[3] https://www.figma.com/
[4] https://primereact.org/

3. **Úspěšnost technik děrování překladu síťových adres**: nastiňuje uživateli informace o zvýšené latenci, kvůli nemožnosti provést úspěšné techniky děrování překladů síťových adres.

4. **Latence ve virtuální počítačové síti**: umožňuje se uživateli lépe rozhodnou spolu s úspěšností technik děrování překladu síťových adres, zda někde poblíž nezačít hostovat server moon, který slouží pro snížení latence, jak je popsáno v 2.3.2.



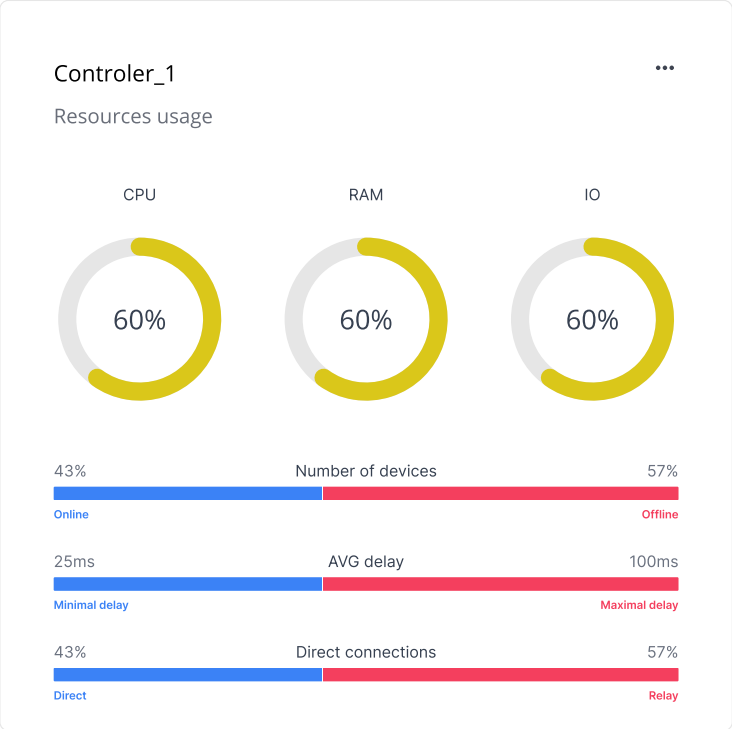
Obrázek 5.4: Statistika virtuální sítě

Výsledný návrh stavu jedné virtuální sítě je vyobrazen na obrázku 5.4. Nadpis karty reprezentuje jméno virtuální sítě. Karta navíc obsahuje jméno kontroléru a uživatele, který virtuální počítačovou síť spravuje. Karta se stavem virtuální počítačové sítě navíc obsahuje kontextovou nabídku, která se dá otevřít kliknutím na tři tečky vpravo nahoře. Kontextová nabídka umožňuje přejít rychle do nastavení konkrétní virtuální počítačové sítě. Návrh stránky zobrazující více virtuálních počítačových sítí je v příloze A.1.

Dál jsem navrhl stránku se statistikami kontrolérů platformy ZeroTier. Tyto statistiky nejsou v dosavadních řešeních. Dokonce dosavadní řešení neumožňují využívat více kontrolérů platformy ZeroTier. Tyto statistiky jsou vhodné pro uživatele, kteří budou chtít dělat rozsáhlé sítě. Jsou převážně určeny pro výpis celkového zatížení serverů, na kterých kontroléry poběží. Díky nim budou pokročilí uživatelé schopni rozprostřít lépe zátěž mezi více kontroléry. Samotná karta bude obsahovat tyto informace:

1. **Vytížení procesoru, vytížení hlavní paměti a využití disků**: umožňuje uživateli získat přehled o vytížení zdrojů serveru, na kterém aplikace běží. Pokud bude server přetížený, bude to uživateli naznačovat, že by měl zátěž rozložit mezi více kontroléry.

2. **Poměr připojených a nepřipojených zařízení**: umožňuje uživateli předvídat, jak moc lze očekávat nárůst využití zdrojů v případě, kdy všechny zařízení budou připojeny ke kontroléru.
3. **Průměrná latence zařízení**: informuje uživatele o průměrné latenci ke kontroléru virtuální počítačové sítě. Naznačuje kvalitu připojení kontroléru do internetu, spolu s úspěšností technik děrování překladu síťových adres.
4. **Počet přímých spojení ke kontroléru**: pokud některý z kontrolérů bude mít menší úspěšnost přímých spojení oproti ostatním, bude to naznačovat problém s technikami děrování překladu síťových adres v síti, přes kterou je kontrolér připojený do internetu.



Obrázek 5.5: Statistika kontroléru

Pro představu je návrh karty s informacemi vyobrazen na obrázku 5.5. Karta obsahuje navíc v nadpisu název kontroléru. Podobně jako karta virtuální počítačové sítě, umožňuje po kliknutí na tři tečky vpravo nahoře zobrazit kontextovou nabídku, která umožňuje uživateli přejít do nastavení kontrolérů. Návrh stránky zobrazující více kontrolérů je v příloze A.2.

5.4.2 Menu a rozložení

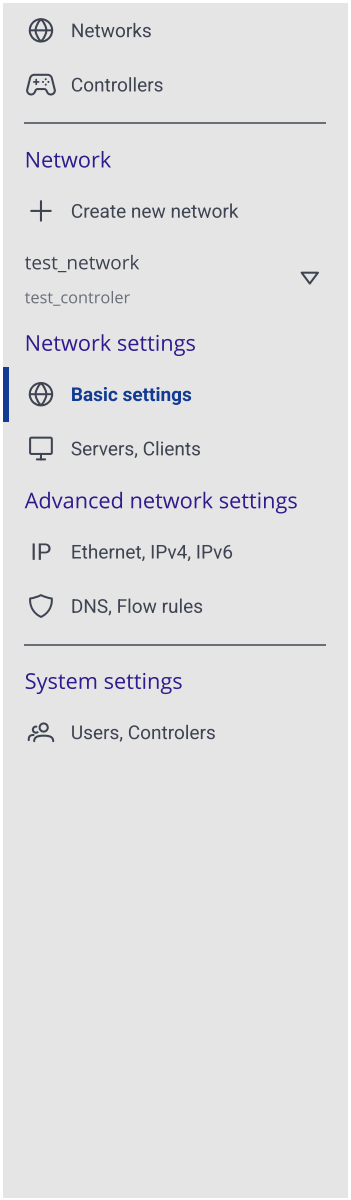
Při vytváření návrhu byl kladen větší důraz na menu a navigaci v aplikaci, než na předchozí komponenty. Navržené menu aplikace je rozděleno do 5 hlavních kategorií:

1. **Monitorování** informuje o stavu virtuálních síti a kontrolérů.
2. **Vytváření a výběr virtuální počítačové sítě** zahrnuje průvodce pro vytváření nové virtuální počítačové sítě a výběr či hledání již existujících virtuálních počítačových sítí.

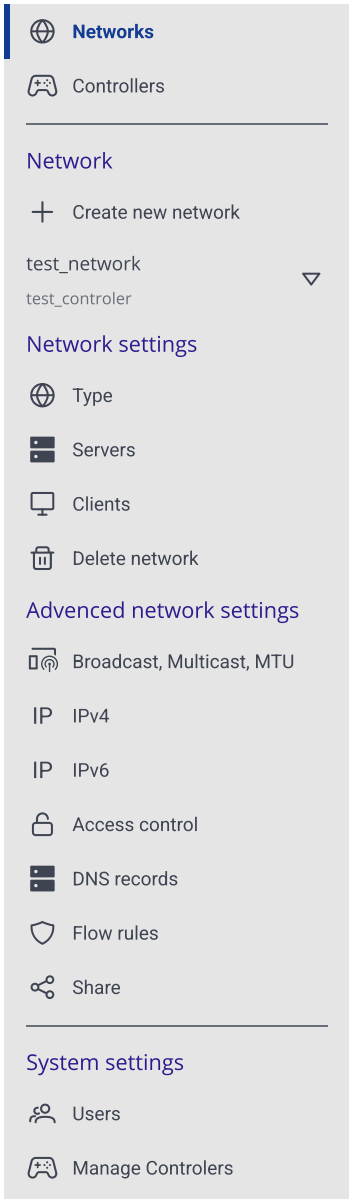
3. **Základní nastavení virtuální počítačové sítě** zahrnuje nastavení přístupu do virtuální počítačové sítě (veřejné/privátní), vzdáleného přístupu do aplikace z virtuální počítačové sítě, mazání virtuální počítačové sítě, přidávání serverů a klientů.

4. **Rozšířená nastavení virtuální počítačové sítě** obsahuje nastavení *broadcastu*, *multicastu*, nastavení MTU (maximální velikost datagramu), nastavení IPv4/IPv6 adres, nastavení směrovacích záznamů, nastavení serverů DNS, filtrování komunikace a nastavení práv ostatním uživatelům aplikace pro danou virtuální počítačovou síť.

5. **Systémová nastavení** umožňují přidávat jiné instance aplikace, vytvářet a upravovat práva uživatelům aplikace.



Obrázek 5.6: Obecné položky menu



Obrázek 5.7: Detailní položky menu

Následně jsem řešil celkové rozložení menu aplikace. První možností bylo mít v menu položky, které seskupovaly vícero nastavení dohromady. Toto menu je zobrazeno na obrázku 5.6. Seskupování položek se nakonec ukázalo jako nevhodné, uživatelé někdy hledali nastavení, které chtěli změnit. Druhé rozložení, které je na obrázku 5.7, se ukázalo jako efektivnější, uživatelé daleko rychleji nalezli položky nastavení, které chtěli změnit.

Dále jsem porovnával dvě varianty možností, jak jednotlivé odkazy budou fungovat. První možností bylo, že jednotlivé odkazy by odkazovaly na samostatné stránky s daným nastavením. Toto se ukázalo jako nevhodné, na stránkách bylo málo položek a uživatelé by navíc museli více proklikávat mezi jednotlivými položkami. Druhou možností bylo, že samostatné odkazy by odkazovaly na části v rámci jednotlivých stránek zahrnující základní nastavení sítě, pokročilé nastavení sítě a systémová nastavení aplikace. Výsledně se ukázalo, že druhé řešení je vhodnější, protože uživatelé často upravovali spolu více nastavení, které byly na stejné stránce a souvisely spolu.

Následně jsem zjišťoval, zda je vhodné v aplikaci skrýt pokročilá nastavení, pokud nejsou povolena v systémových nastaveních, s cílem udělat aplikaci více uživatelsky přívětivou pro méně znalé uživatele. Nakonec se ukázalo, že je lepší pokročilá nastavení rovnou zobrazit. Jelikož uživatelé bez pokročilých znalostí počítačových sítí se stejně pokročilým nastavením vyhýbali a uživatele s pokročilými znalostmi to zpomalovalo. Dokonce někteří uživatelé nezjistili, že je možné pokročilá nastavení v systémových nastaveních povolit.

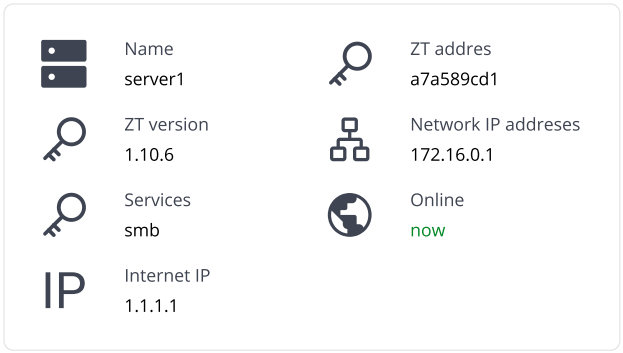
Jak je popsáno v knize [29], uživatelé vždy využívají své minulé zkušenosti. Pokud aplikace bude mít podobné rozložení ovládacích prvků, jako mají jiné aplikace, tak to umožní uživatelům daleko rychleji se zorientovat. Proto jsem zvolil rozložení, které má menu na levé straně aplikace a uživatelské jméno v pravém horním rohu aplikace. Celkové rozložení aplikace je vidět v příloze A.1.

O proti dosavadním řešením je toto navržené řešení přehlednější a rychleji se zde uživatelé orientují. Z dosavadních řešení ZeroUI umožňuje pouze vybrat virtuální počítačovou síť a následně zobrazuje všechna nastavení na jedné stránce, což je dost nepřehledné. Oproti tomu ztncui implementuje menu, kde jsou na výběr položky. Samotné stránky pod položkami v ztncui jsou dost prázdné a je zde mnoho nevyužité plochy. Navíc pokud někdo chce v této aplikaci měnit nastavení virtuální počítačové sítě, musí poměrně hodně klikat. Nejdříve musí v menu vybrat položku sítě, potom musí vybrat konkrétní virtuální počítačovou síť a následně musí kliknout na tlačítko odkazující na konkrétní nastavení, které chce měnit. V novém mnou navrženém řešení stačí v menu vybrat virtuální počítačovou síť (pokud je v aplikaci jen jedna, je hned automaticky vybrána a není potřeba ji vybírat) a potom kliknout přímo na nastavení, které chce uživatel měnit. Pokud bude chtít uživatel měnit i související nastavení, nemusí dále klikat v navigaci, jelikož související nastavení má všechny u sebe.

5.4.3 Rozložení nastavení s více položkami

Důraz byl dále kladen na vytváření návrhu rozložení nastavení s více položkami. Můj prvotní návrh nepočítal s vyhledáváním mezi položkami nastavení. Nicméně při procházení návrhu s uživatelskou základnou, navrhl jeden uživatel, dát vyhledávání ke všem nastavením, které mohou mít více položek. Tento nápad se ukázal jako velmi užitečný a všem ostatním z uživatelské základny se líbil. Tento nápad jsem zahrnul do návrhu.

Obrázek 5.8: Tabulky s rozbalováním sloupců



Obrázek 5.9: Karta

Dál jsem zjišťoval, zda je lepší použít karty, nebo tabulky s kombinací tabulky s rozbalováním sloupců. Mezi tabulkami s rozbalováním sloupců (na obrázku 5.8) a kartami (na obrázku 5.9) nakonec byly zvoleny po dlouhé diskuzi karty. Jelikož většina uživatelů nejdříve využije vyhledávání, které umístí položku, kterou hledají, hned na začátek. Následně se díky velkým obrázkům v kartě rychleji zorientují a není potřeba tabulku expandovat na malých zobrazovacích zařízeních. Bez delší debaty by se k tomuto závěru nikdy nedospělo a většina uživatelů by stále tvrdila, že jsou pro ně tabulky s kombinací tabulky s rozbalováním sloupců lepší.

5.4.4 Ikona aplikace

Na základě zpětné vazby uživatelské základny jsem i vybral ikonu nové aplikace. Nejdříve jsem vytvořil různé návrhy ikon aplikace. Mezi těmito ikonami si následně uživatelé z uživatelské základny vybírali ikonu, která se jim nejvíce líbila. Všechny ikony, mezi kterými si uživatelé vybírali, jsou v příloze A.

Obrázek 5.10: Ikona aplikace

Nejvíce hlasů obdržela ikona vyobrazená na obrázku 5.10. Tato ikona se líbila uživatelům ze všech nejvíce, zároveň se shodli na tom, že nejlépe vyjadřuje samotnou aplikaci. Osm koleček okolo vyjadřuje zařízení ve virtuální počítačové síti. Kružnice vyjadřuje propojení zařízení a Z vyjadřuje první písmeno názvu aplikace. Aplikace se původně měla jmenovat ZeroController, ale později byla přejmenována na AnyController. Nový název lépe reprezentuje potenciál rozšíření aplikace o komunikaci s dalšími řešeními, než je jen kontrolér platformy ZeroTier. Všechny vytvořené ikony jsou v příloze A.12.

Kapitola 6

Implementace

Dalším bodem je samotná implementace aplikace, která je vytvořena na základě návrhu z kapitoly 5. V této kapitole popisuji možnost, jak provozovat aplikaci na síťových prvcích a síťových úložištích, kterou využívám při implementaci aplikace. Dále vybírám a používám technologie pro realizaci architektury aplikace, která byla popsána v podkapitole 5.1. Jako poslední popisuji obtíže automatické kompilace obrazů kontejnerů, se kterými jsem se setkal.

6.1 Serverová část

První a nejdůležitější požadavek, který bylo nutné zajistit, byla komunikace s kontrolérem platformy ZeroTier. Pro komunikaci s kontrolérem platformy ZeroTier se využívá rozhraní REST, jak bylo zmíněno v 5.1. Podle dokumentace [86] je toto rozhraní REST chráněno pomocí tokenu pro autorizaci operací. Tento token pro autorizaci je uložen v textovém souboru mezi konfiguračními soubory kontroléru platformy ZeroTier. Podle mých pokusů na operačním systému Linux je tento soubor vlastněn uživatelem `zerotier-one`. Tento uživatel může jako jediný spolu s uživatelem `root` číst tento soubor. Zároveň podle pokusů lze tento soubor číst na operačním systému Windows jen s administrátorskými právy. Díky tomu bude muset serverová část aplikace tento token přečíst s vyššími právy. Kvůli tomu bude serverová část aplikace při startu vyžadovat vyšší práva, aby mohla získat tento token, který bude potřebovat pro komunikaci s kontrolérem platformy ZeroTier.

6.1.1 Webový server

Současná řešení webových serverů nešlo efektivně použít k tomuto účelu. Nebyly navrženy pro spouštění s vyššími právy, provedením několika operací s těmito právy a následném snížení práv z důvodů bezpečnosti. Dále zbytečně implementují mnoho funkcionalit i bez modulů, které aplikace nevyužívá a zvyšovaly by se tím nároky na síťové prvky a síťová úložiště. Nejlepším řešením bylo tedy napsat vlastní zjednodušený webový server, který by toto umožňoval. Zároveň tento webový server jsem vytvořil tak, aby byl co možná nejmenší, neimplementuje mnoho funkcionalit, které jsou k dispozici v jiných webových serverech. Můj webový server například podporuje jen metody GET a POST. Na druhou stranu můj vytvořený webový server implementuje *multithreading* pro lepší rychlost zpracování souběžných dotazů. Dále implementuje získávání dříve zmíněného tokenu a monitorování vytížení zařízení, na kterém běží (slouží pro kartu kontroléru na klientské části aplikace).

Celý webový server jsem napsal v jazyce C s podporou kompilace pro Linux a Windows. Vzhledem k tomu, že by bylo velmi náročné psát celou serverovou část v tomto jazyce,

implementoval jsem do webového serveru rozhraní CGI, které je popsáno v podkapitole 3.1.1. Toto rozhraní umožňuje webovému serveru používat externí programy. Mezi externí programy lze zařadit i interprety programovacích jazyků. Mnou implementovaný webový server přes toto rozhraní předává externímu programu i dodatečné informace, jako je token pro komunikaci s kontrolérem platformy ZeroTier a vytížení serveru, na kterém běží.

Rozhraní CGI jsem upřednostnil, jelikož externí program u CGI běží jen po dobu vykonávání daného požadavku a následně se ukončí oproti fastCGI, kde externí program může běžet po celou dobu, jak je popsáno v podkapitole 3.1.1. Tuto funkci bych nevyužil na síťových prvcích. Zde je vhodnější interpret hned ukončit, aby síťový prvek, který má většinou málo zdrojů, mohl své zdroje využít efektivněji než čekáním na požadavek pro interpret. Zároveň jsem předpokládal, že dotazy pro interpret budou generovány jen velmi málo (dotazy pro interpret se generují jen v době, kdy je uživatel přihlášený v aplikaci). Navíc rozhraní CGI bylo jednodušší než fastCGI na implementaci.

6.1.2 Interpret programovacího jazyka

Jako externí program jsem zvolil interpret jazyka PHP. Interpret mi značně umožnil zjednodušit vývoj aplikace. Bez použití interpretu PHP bych nebyl schopen dokončit práci včas. Interpret PHP jsem zvolil, jelikož mi přišel nejvíce vyvážený vzhledem k velikosti a náročnosti na zdroje zařízení. Dokonce se mi povedlo po větších obtížích zkompilovat interpret PHP dohromady s databází SQLite[1] do jednoho binárního souboru o velikosti 8 MiB bez závislostí na knihovny.

Ke kompilaci interpretu PHP je nejdříve nutné nastavit konfiguraci kompilace. Nastavení konfigurace kompilace se provádí pomocí skriptu *./configure* pro příkazový řádek, který se nachází u zdrojového kódu interpretu. Tento skript obsahuje i nápovědu, kterou lze vypsat pomocí parametru *–help*. Tento skript následně vytvoří podle konfigurace soubor *makefile*, který slouží k samotné kompilaci příkazem *make*.

Pro kompilaci bez závislostí je nutné při konfiguraci použít parametry *–disable-shared –enable-static CFLAGS="-static"*. Po vytvoření souboru *makefile* je dále nutné v souboru změnit všechny výskyty parametru *-export-dynamic* na *-all-static*. Jako poslední se musí interpret jazyka PHP kompilovat s knihovnou určenou pro statickou kompilaci bez závislostí (s knihovnou GNU C bude výsledný program závislý na externí knihovně, podle mých pokusů). K tomuto účelu jsem využil knihovnu musl[2]. Celkově jsem byl schopen při experimentech snížit velikost binárního souboru interpretu PHP na 1,7 MiB při použití komprese pomocí programu UPX[3].

Protože lze interpret PHP a databázi SQLite zkompilovat do jednoho spustitelného binárního souboru a databáze SQLite vyhovuje všem požadavkům aplikace. Využil jsem tuto databázi v implementaci. Serverová část databázi automaticky vytváří, pokud detekuje nepřítomnost databázového souboru. Databáze je vytvářena na základě entitně-vztahového modelu popsaného v podkapitole 5.2 samotnou aplikací.

6.2 Klientská část aplikace

Klientská aplikace je vytvořena na základě návrhu uvedeném v podkapitole 5.4. Jelikož byly v návrhu uživatelského rozhraní využity komponenty z knihovny PrimeReact, využil jsem

[1] https://www.sqlite.org/
[2] https://musl.libc.org/
[3] https://upx.github.io/

pro implementaci klientské části aplikace knihovnu React[4] a PrimeReact. Během implementace klientské aplikace jsem ještě využil knihovnu material-icons[5] pro obrázky použité v aplikaci.

Klientská část aplikace je uložená na serverové části aplikace v komprimované podobě. Díky tomu je menší celková velikost aplikace a zároveň je vyšší rychlost komunikace. Server posílá zkomprimované soubory klientské části do webového prohlížeče nebo desktopové aplikace.

Při implementaci klientské části aplikace jsem průběžně komunikoval s uživatelskou základnou, která mi dávala dodatečnou zpětnou vazbu. Proto se implementace striktně nedrží návrhu a obsahuje několik změn, které aplikaci vylepšují. Zároveň některé funkce jsem nebyl schopen efektivně realizovat a některé jsem nestihl kvůli časovému omezení neimplementovat. Snímky z implementované aplikace jsou v příloze B. Mezi změny v implementované aplikaci oproti návrhu patří:

1. **Tmavý vzhled**: návrh počítal se světlým vzhledem aplikace, ale po tom, co všichni viděli tmavý vzhled, tak většina byla pro tmavý vzhled aplikace. Aplikace neimplementuje možnost výběru mezi tmavým a světlým režimem, jelikož by bylo nutné do aplikace přidat dodatečné soubory s kaskádovými styly, které by zvětšily velikost aplikace.

2. **Karta sítě**: karta sítě obsahuje místo administrátora adresu virtuální počítačové sítě pro jednoduší identifikaci v případě, kdy uživatel vytvoří dvě virtuální počítačové sítě se stejným názvem. Administrátora sítě lze stále zjistit z pokročilých nastavení dané virtuální počítačové sítě, může jich být i více. Karta sítě u počtu přímých propojeních udává počet zařízení místo procent, jak bylo v návrhu. Procenta lze odhadnout z doprovodného grafu a samotný počet zařízení dává uživateli více informací.

3. **Karta kontroléru**: stejně jak u karty sítě, byly nahrazeny procenta za počty zařízení. Zároveň byl text popisující počet online a offline zařízení ujednocen s kartou sítě.

4. **Zprávy upozorňující na různé informace**: zprávy, které měly přehledně uživateli hlásit chybové informace, jako je například, že byl některý server offline v některou dobu v minulosti. Nebyly implementovány. Rozhraní, které využívá kontrolér platformy ZeroTier nebylo vhodně navrženo a bylo by potřeba provádět velké množství dotazů na toto rozhraní každých pár sekund, což není z výkonnostního hlediska vhodné pro síťové prvky a síťová úložiště.

5. **Uspořádání filtrovaných položek**: nebylo implementováno. Z časového hlediska se to nestihlo. Zároveň tato funkce by byla málo využívána uživateli. Je daleko rychlejší napsat další písmeno, než změnit uspořádaní a následně hledat mezi kartami. Všechny položky jsou v aplikaci seřazeny vzestupně.

6. **Vytváření nové virtuální sítě**: vytváření nové virtuální počítačové sítě se snaží být více univerzální oproti návrhu a obsahuje jiné položky. Zároveň z vytváření nové virtuální počítačové sítě, bylo odebráno přidávání serveru a klientů. Jelikož hned po vytvoření nové virtuální sítě je uživatel přesměrován na stránku základního nastavení, kde lze servery a klienty přidávat.

[4] https://react.dev/
[5] https://www.npmjs.com/package/material-icons

7. **Flow pravidla**: místo *flow pravidel* byli implementovány karty, přes které lze nastavovat otevřené porty na zařízení pro komunikaci. Díky tomu není nutné, aby uživatel znal jazyk *flow pravidel*.

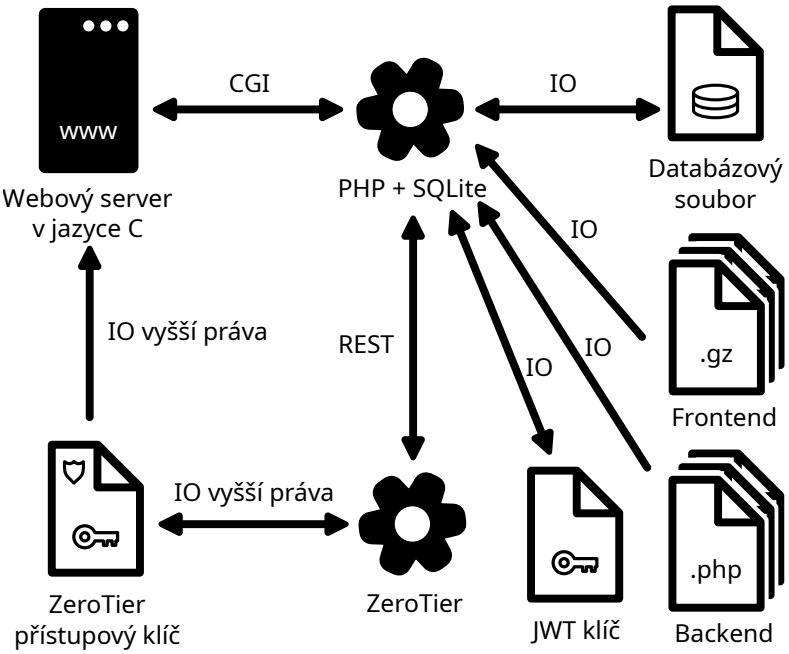
8. **DNS záznamy**: byly nahrazeny za možnost přidávat servery DNS. ZeroTier v sobě neimplementuje server DNS a aplikace by jinak musela obsahovat svůj vlastní server DNS. Implementace serveru DNS do aplikace se nestihla.

9. **Token pro připojení kontroléru**: v návrhu aplikace chyběla možnost zobrazit token pro připojení do jiné instance aplikace. Implementace je o tuto chybějící komponentu doplněna.

10. **Typ sítě**: v návrhu byl typ virtuální počítačové sítě v základních nastaveních, který by umožnil nastavit síti pro co je přesně určena. Pokud by potom chtěl uživatel změnit nastavení, které by bylo v rozporu s tímto typem. Aplikace by změnu nastavení nepovolila. Toto se ukázalo jako nevhodné. V implementované aplikaci lze typ sítě vybrat jen pří vytváření nové virtuální počítačové sítě, následně lze libovolně všechna nastavení upravovat. Typ sítě je použit jen pro vygenerování počáteční konfigurace.

6.3 Celková architektura

Implementovaná aplikace využívá architekturu komunikace, která byla představena v rámci podkapitoly 5.1. Tato architektura se skládala ze serverové a klientské části aplikace, relační databáze, kontroléru platformy ZeroTier.



Obrázek 6.1: Podrobnější architektura

Pokud tuto architekturu doplním o použité technologie pro lepší představu, vznikne architektura komunikace, která je vyobrazena na obrázku 6.1. Implementovaná architektura

obsahuje vlastní webový server, který je nezávislý na knihovnách operačního systému, aby se dosáhlo jednodušší instalace a větší stability. Webový server navíc získává přístupový klíč pro rozhraní REST kontroléru platformy ZeroTier, který následně předává interpretu PHP spolu s informacemi o zatížení zařízení, na kterém běží. Server implementuje pro komunikaci s webovým prohlížečem a desktopovou aplikací protokol HTTP 1.0.

Interpret PHP při interpretaci souborů PHP může komunikovat s kontrolérem platformy *ZeroTier* přes rozhraní REST, pomocí tokenu, který obdržel od webového serveru. Zároveň interpret PHP v sobě obsahuje databázi SQLite, která pracuje s databázovým souborem uloženým na serveru. Klientská část je uložena v komprimované podobě, jak bylo zmíněno v podkapitole 6.2. Klíč JWT, který slouží pro vytváření tokenů JWT, se ukládá bokem, stejně jako ZeroTier ukládá klíč pro rozhraní REST bokem. Umožňuje to zálohovat data aplikace bez tohoto klíče a zároveň to dovoluje pokročilým uživatelům kdykoliv přegenerovat klíč smazáním souboru.

6.4 Desktopová aplikace

Pro implementaci desktopové aplikace jsem využil Tauri[6]. Samotná implementovaná desktopová aplikace obsahuje celou webovou aplikaci včetně webového serveru, interpretu PHP s databází SQLite. Jediné, co desktopová aplikace neobsahuje, je kontrolér platformy ZeroTier, který je nutné separátně doinstalovat. Při spuštění desktopové aplikace se spustí webový server na portu `9994`, se kterým samotná aplikace komunikuje přes protokol HTTP. Aplikace získá klientskou část, kterou zobrazí uživateli. Spouštění webového serveru zajišťuje kód napsaný v programovacím jazyce Rust[7].

Díky tomu, že desktopová aplikace obsahuje webový server, lze k aplikaci přistoupit z webového prohlížeče vzdáleně. To se velmi hodí, když někdo chce používat aplikaci vzdáleně přes virtuální počítačovou síť. Navíc to může umožnit jiným uživatelům v rámci virtuální počítačové sítě využívat aplikaci, například pro přidání klientů do stávající virtuální počítačové sítě.

6.5 Provoz na síťových prvcích a úložištích

Vznik kontejnerů přinesl i nové možnosti provozovat externí aplikace na síťových prvcích a síťových úložištích. Například spouštění kontejneru podporují všechny routery od společnosti Mikrotik, které mají procesory založené na architekturách arm, arm64 a x86 v době psaní této práce (květen 2025) [53]. OpenWrt také podporuje kontejnery [8]. Ze síťových úložišť kontejnery podporuje například Synology NAS [73]. Vzhledem k tomu, že kontejnery lze provozovat i na normálních serverech, jsem se rozhodl využít obrazy kontejneru pro provoz na síťových prvcích, síťových úložištích i klasických serverech.

Proto, abych zajistil možnost provozovat aplikaci na co nejvíce zařízeních, je důležité, aby aplikace byla co možná nejmenší. Některá zařízení nemají moc velikou interní paměť. Například Mikrotik hAP ax Lite má interní úložiště jen 128 MB, které je sdílené se systémem a neumožňuje úložiště rozšířit o disk využívající USB rozhraní, jelikož nemá USB port [45]. Čím více bude aplikace menší, tím potenciálně bude moci běžet na více zařízeních.

K tomu abych dosáhl co nejmenší velikosti obrazu kontejneru, jsem nejdříve musel důkladně pochopit, z jakých částí se skládá standardní obraz kontejneru. Velká většina obrazů

[6] https://tauri.app/
[7] https://www.rust-lang.org/

kontejnerů, se kterými jsem se setkal, je postavená na základě již některých existujících kontejnerů. Mezi tyto základní obrazy patří například Debian[8], Ubuntu[9], Alpine Linux[10] a další. Do těchto základních obrazů je následně přidána konkrétní aplikace s knihovnami, které využívá pro svůj provoz. Takto vytvořený obraz je následně distribuován dál.

Samotné základní obrazy jdou stáhnout a lze se podívat, z čeho jsou vytvořeny. Tyto základní obrazy obsahují základní knihovny, shellovou aplikaci a základní programy, jako jsou například cp, mv, cd. Pokud cílová aplikace, která je přidávána do kontejnerů, nebude využívat tyto knihovny a aplikace, není důvod tyto základní obrazy využívat, pokud tvůrce nemusí šetřit s časem při vytváření obrazu. Místo toho může aplikaci vložit do prázdného obrazu kontejneru se všemi závislostmi, které bude aplikace potřebovat. Tím může dosáhnout menší velikosti obrazu. Pro vytváření prázdného obrazu kontejneru lze využít klíčové slovo *scratch* na místě zdrojového obrazu [19]. V implementaci toho využívám a vytvářím obraz aplikace úplně od začátku, abych dosáhl menší velikosti.

6.6 Automatická kompilace

Jelikož je tento projekt velký a velmi špatně by se udržovala veškerá funkční architektura projektu manuálně, využil jsem CI/CD (*continuous integration continuous delivery*) z aplikace GitLab. CI/CD při každém komunitu v tomto projektu spouští automatickou kompilaci celé aplikace, následně se spustí testy serverové části aplikace separátně pro různé operační systémy a různé architektury procesorů. Pokud všechny testy úspěšně proběhnou, jsou automaticky vytvořeny soubory, které slouží pro instalaci samotné aplikace. Automatické testy jsou podrobněji popsány v podkapitole 7.1. Pokud se navíc jedná o *merge* do hlavní větve, je automaticky vytvořena webová stránka, která tyto soubory pro instalaci aplikace obsahuje a je automaticky nasazena na web projektu[11]. Pro vykonávání všech příkazů pro CI/CD jsem využíval svůj vlastní GitLab Runner[12], který jsem hostoval na svém vlastním serveru. Díky tomu jsem nebyl nijak časově omezený.

Pro vytváření obrazu kontejneru aplikace využívám místo aplikace Docker aplikaci Kaniko[13]. Je to z toho důvodu, že při provádění CI/CD využívám Docker executor[14], který jak vyplývá z názvu využívá pro provádění příkazů Docker kontejnery. Při spouštění aplikace Docker pro kompilaci obrazu kontejneru uvnitř jiného Docker kontejneru nastává chyba, díky které není schopný vytvořit obraz kontejneru. Docker nebyl k tomuto účelu navržen. K tomuto účelu byla vytvořena aplikace Kaniko, jak vyplývá z její dokumentace [81].

Bohužel obrazy kontejnerů vytvořené za pomoci aplikace Kaniko mi nešly spustit na zařízeních od společnosti Mikrotik. Při porovnávání obsahu obrazů kontejnerů z aplikace Docker a aplikace Kaniko jsem zjistil, že Kaniko používá kompresi na každou vrstvu obrazu zvlášť, kdežto Docker používá kompresi až na celý obraz kontejneru. Když jsem vzal obraz vytvořený v aplikaci Kaniko a odebral jsem ze všech jeho vrstev kompresi a použil jsem ji na celý obraz, tak tento obraz již šel spustit na zařízeních od společnosti Mikrotik. Abych nemusel tuto úpravu dělat ručně pro každý vytvořený obraz, přidal jsem do CI/CD posloupnost příkazů, které přesně tento postup úpravy replikují.

[8] https://hub.docker.com/_/debian
[9] https://hub.docker.com/_/ubuntu
[10] https://hub.docker.com/_/alpine
[11] https://anycontroller.gitlab.io/anycontroller/
[12] https://docs.gitlab.com/runner/
[13] https://github.com/GoogleContainerTools/kaniko
[14] https://docs.gitlab.com/runner/executors/docker/

Kapitola 7

Testování

Posledním bodem je testování aplikace, které by mělo předcházet samotnému zveřejnění aplikace. Testování umožní z aplikace odstranit chyby, které by se jinak dostaly ke koncovým uživatelům. Tím umožní zlepšit celkovou uživatelskou zkušenost. Při vytváření aplikace jsem využíval automatické testy (7.1), zároveň jsem při vývoji prováděl i manuální testy (7.2). Ke konci vývoje jsem prováděl vlastní testování v ostrém provozu (7.3) a nakonec jsem prováděl testy s uživateli z uživatelské základny (7.4).

7.1 Automatické testy

Během implementace aplikace jsem implementoval integrační test pro jednotlivé koncové body serverové části aplikace. Všechny testy se spouštěly pro každou platformu zvlášť. Integrační testy v průběhu implementace odhalily mnoho chyb. Velmi mi pomohly při refaktorizaci kódu aplikace, při které odhalily chyby, které jsem neočekával. Při implementaci aplikace byly testy dokonce schopné odhalit chyby ve starší verzi kontroléru platformy ZeroTier. Při vytváření aplikace jsem toto neočekával a zpomalilo mě to, jelikož jsem si myslel, že jsou chyby v aplikaci, kterou jsem implementoval.

Pokud automatické testy neproběhly úspěšně, zablokovaly *merge* do hlavní větve. Což umožnilo mít jen stabilní verzi aplikace v hlavní větvi. Z hlavní větve se vytváří webová stránka se zkompilovanou aplikací ke stažení pro různé operační systémy a různé architektury procesoru.

7.2 Testy při vývoji

Během vývoje aplikace jsem prováděl i manuální testy aplikace. Měl jsem na lokálním stroji nainstalovaný ZeroTier a spuštěný webový server spolu s vývojovým serverem pro klientskou část aplikace. Výhodou bylo, že jsem měl klientskou část aplikace vytvořenou pomocí knihovny React a serverovou část napsanou v programovacím jazyce PHP, viděl jsem změny kódu aplikace v reálném čase, nebylo nutné nic kompilovat při změně. Pokud se nejednalo o samotný webový server.

Manuální testy mi nejvíce pomohly opravit chyby týkající se klientské části aplikace. Během testů jsem narazil i na chybu, která byla v samotné knihovně PrimeReact. Chyba se týkala komponenty pro zobrazování zpráv. Pokud první vytvořená zpráva měla stejný text, jako ostatní zprávy, byly všechny tyto zprávy automaticky zavřeny spolu s první zprávou.

Tak se zprávy neměly chovat a každá zpráva se měla zavírat zvlášť. Chyba byla později bez mojí interakce v knihovně opravena, nebyl jsem jediný, kdo na ni narazil.

Při testování jsem prováděl i pár základních penetračních testů. Během testu jsem odhalil bezpečnostní chybu na přihlašovací obrazovce aplikace. Tato chyba umožnila využít *timing attack*. Když se někdo pokusil přihlásit pomocí účtu, který nebyl v aplikaci zaregistrován, aplikace dříve informovala o neplatných přihlašovacích údajích než v případě, kdy byl uživatel zaregistrován. Díky tomu útočník mohl potenciálně zjistit, zda je daný uživatel v aplikaci zaregistrován i bez znalosti hesla. Tuto chybu jsem následně opravil.

Na svůj vlastní webový server jsem prováděl i pár útoků typu DoS. Při těchto testech jsem odhalil, že můj webový server je lépe odolný proti útoku *Slow loris* oproti serveru Apache2[1] ve výchozí konfiguraci. Dále jsem zjistil, že můj webový server zase díky použití rozhraní CGI hůře zvládal obsluhovat více dotazů oproti serveru Apache2, který jsem měl propojený s PHP pomocí modulu. To, ale ničemu nevadí, jelikož můj webový server nebyl navržen k tomuto účelu. Byl navržen pro lokální sítě pro lokální služby na výkonnostně slabých zařízeních.

7.3 Vlastní testování v ostrém provozu

První testy aplikace mimo vývojové prostředí jsem prováděl na svém vlastním WiFi routeru (Mikrotik ac^3). Pro přístup k mému vlastnímu serveru, který jsem měl doma. Během testů jsem odhalil, že využití komprese u interpretu PHP nebylo vhodné. Komprese interpretu PHP je zmíněna v sekci 6.1.2. Dekomprese interpretu PHP při každém dotazu byla daleko náročnější než samotné interpretování PHP kódu a způsobovala, že každý dotaz trval 0.9 sekundy. V případě, kdy se nevyužila komprese, dotaz trval jen 0.2 sekundy. Proto jsem kompresi interpretu PHP z aplikace odstranil.

Nejzávažnější chyba, na kterou jsem narazil při reálném provozu na svém WiFi routeru, byl kompletní pád klientské části aplikace. Tento pád nastával v případě, kdy zařízení ve virtuální počítačové síti vytvořilo přímé spojení. Tato informace se měla zobrazit v samotné kartě daného zařízení v klientské části aplikace, místo toho ale klientská aplikace přestala fungovat. Klientská aplikace nešla používat do doby, než bylo dané zařízení odpojeno ze sítě. Tuto chybu se mi podařilo opravit.

Ostatní odhalené chyby, které jsem opravil, se týkaly převážně karet stavů kontrolérů a stavů virtuálních privátních sítí. Tyto chyby jsem při testování vývoje aplikace neodhalil, jelikož vyžadovaly mít připojeno více zařízení do virtuální počítačové sítě nejlépe delší dobu (nějakou dobu trvá, než se vytvoří přímá spojení mezi zařízeními).

7.4 Testování s uživateli

Při prvním testu s uživateli jsem se snažil otestovat co nejvíce funkcí. Měli jsme mezi sebou propojeno více instancí aplikace. Hlavní instance, která kontrolovala ostatní, běžela na WiFi routeru Mikrotik hAP ac^3 na architektuře arm. K této hlavní instanci byly propojeny následující instance:

1. desktopová aplikace na Windows na architektuře x86_64,
2. instance běžící na WiFi routeru Mikrotik hAP ax^3 na architektuře arm64,

[1] https://httpd.apache.org/

3. instance běžící v kontejneru na architektuře x86_64.

Při testování se odhalilo mnoho chyb spojených s používáním aplikace více uživateli. Například pokud první uživatel editoval virtuální počítačovou síť a druhý uživatel mezitím tuto síť smazal, tak první uživatel byl odhlášen z aplikace. Tuto chybu jsem opravil. Dále se například odhalilo, že uživatel nemůže změnit heslo sám sobě.

Při komunikaci s více instancemi aplikace se odhalila nepříjemná chyba, která způsobovala nepoužitelnost aplikace v případě, kdy jeden z kontrolérů přestal fungovat. Aplikace čekala na jeho odpověď jednu minutu v každém dotazu na vzdálenou instanci, to způsobilo nepoužitelnost aplikace. Čekání na odpověď se mi povedlo razantně zkrátit, aby aplikace byla použitelná i při výpadku některé z instancí.

Další nepříjemnou chybou, kterou se povedlo odhalit, bylo padání aplikace v kontejneru na jednom konkrétním zařízení, při získávání informací o zatížení zařízení. Chyba byla uvnitř webového serveru, který padal na chybu neznámé instrukce. Po delším debugování jsem zjistil, že se jedná o chybu spojenou s knihovnou musl. Při provádění jedné z operací v knihovně narazil procesor zařízení na instrukci, kterou neměl ve své instrukční sadě. Tuto chybu se mi povedlo opravit změnou funkce z knihovny za jinou, která šla použít ke stejnému účelu a byla více doporučená v dokumentaci ke knihovně. Tato nová funkce tuto instrukci neobsahovala a díky tomu začal webový server na zařízení fungovat normálně.

Poslední testy, které jsem prováděl s uživateli, obsahovaly jednu větší síť s více zařízeními. Tato síť byla použita pro hraní počítačové hry a fungovala delší dobu. Tyto testy odhalily specifické chyby, které se doposud neprojevovaly. Všechny chyby, které se projevily, jsem opravil. Mezi chyby, které se projevily, patřilo:

1. latence zařízení v síti někdy obsahovala nepravdivé údaje,
2. pokud byl kontrolér připojen do virtuální sítě, byl vždy podle aplikace offline,
3. překlepy v popiscích nastavení.

Aplikaci při testování šlo využít k účelům, které byly specifikovány v rámci kapitoly 5 a šlo ji provozovat i na síťových prvcích. Aplikace byla zároveň více uživatelsky přívětivá oproti dosavadním aplikacím pro ovládání kontroléru platformy ZeroTier, které může kdokoliv používat.

Novou aplikaci by šlo ještě více vylepšit. Při používání aplikace uživateli jsem si všiml, že by bylo lepší pro větší uživatelský komfort přidat možnost změnit si heslo do kontextového menu, které se zobrazuje po kliknutí na své uživatelské jméno. V aplikaci si lze změnit heslo v systémových nastaveních, kam se v případě, kdy si chtěli uživatelé změnit heslo, hned nepodívali. Bohužel z časového hlediska jsem tuto funkci nestihl implementovat.

Další vylepšení, které by bylo vhodné implementovat, by bylo zvýraznění nastavení podle kliknutí na nastavení v menu aplikace. Při kliknutí na nastavení v menu aplikace je uživatel přesměrován na stránku s nastaveními, kde se jeho konkrétní nastavení nachází. Následně by se automaticky měla stránka na to nastavení posunout, aby bylo na obrazovce zařízení. Tato funkce funguje jen občas. Zároveň uživatel pak nastavení na stránce hledá mezi ostatními nastaveními. Kdyby se nastavení zvýraznilo, nemusel by ho uživatel hledat. Bohužel jsem opět toto nestihl implementovat z časových důvodů.

Kapitola 8

Závěr

Cílem práce bylo vytvořit aplikaci, která by jednoduše umožňovala uživatelům sdílet služby přes internet. K tomuto problému jsem v rámci teoretické části zpracoval problematiku sdílení služeb skrze internet. Následně jsem se věnoval principům tvorby multiplatformních a webových aplikací. Analyzoval jsem současná řešení, která uživatelé mají pro sdílení služeb přes internet a jejich problémům. Na základě těchto problémů a zpětné vazby od uživatelské základny jsem navrhl novou aplikaci, která lépe řeší jejich problémy. Navrženou aplikaci jsem na základě návrhu spolu s dodatečnou průběžnou zpětnou vazbou od uživatelské základny implementoval.

Cíle se povedlo dosáhnout s jistými kompromisy. Nová aplikace je více uživatelsky přívětivá oproti dosavadním řešením (ztncui a ZeroUI). Umožňuje rychleji a jednodušeji vytvářet nové virtuální počítačové sítě. Implementuje funkcionalitu navíc, která v těchto dosavadních řešeních není, jako jsou statistiky virtuálních počítačových sítí, statistiky kontrolérů, možnost propojit více instancí aplikace dohromady a možnost přidat do aplikace více uživatelů s různými právy.

Na druhou stranu bohužel API kontroléru platformy ZeroTier neumožnilo efektivně implementovat užitečnou funkci, která by uživatelům hlásila například informaci, že nějaký server ve virtuální síti byl nefunkční určitou dobu. Muselo by být prováděno extrémní množství požadavků na kontrolér platformy ZeroTier každých pár sekund. To by bylo obzvláště nevhodné pro síťové prvky, pokud by na nich aplikace běžela. Zároveň jsem nestihl implementovat některé užitečné funkce na základě informací z testování s uživateli, které by zefektivnily práci uživatelům. Je ale nutno dodat, že tento projekt nekončí touto diplomovou prací a bude rozvíjen dál. Je plánováno tuto aplikaci v budoucnu o tyto funkce doplnit. Sám aplikaci aktivně používám a mám ji nainstalovanou na svém WiFi routeru. Aplikaci používám k přístupu k serveru, který osobně vlastním. Celá aplikace je plně open-source pod licencí MIT[1] a je zveřejněná na platformě GitLab[2].

Během práce navíc vznikl separátní projekt[3] který obsahuje webový server s interpretem PHP spolu s databází SQLite, ze které lze vytvořit obraz kontejneru, který zabírá pouhé 2 MB. Původně jsem tento projekt vůbec neměl v plánu vytvářet, ale hodně lidí o toto řešení projevilo velký zájem, a tak jsem tento projekt vytvořil. Tato práce byla navíc vybrána a následně prezentována v rámci studentské konference Excel@FIT 2025 [37].

[1] https://gitlab.com/anycontroller/anycontroller/-/blob/main/LICENSE
[3] https://gitlab.com/anycontroller

Literatura

[1] Achour, M.; Betz, F.; Dovgal, A.; Lopes, N.; Magnusson, H. et al. *Introduction* online. únor 2024. Dostupné z: https://www.php.net/manual/en/security.intro.php. [cit. 2024-02-15].

[2] Aliaksei a Nieto, A. R. ZeroUI. *GitHub* online. 2023. Dostupné z: https://github.com/dec0dOS/zero-ui/blob/main/README.md. [cit. 2025-02-3].

[3] Arm Limited (or its affiliates). *A64 – Base Instructions (alphabetic order)* online. Prosinec 2021. Dostupné z: https://developer.arm.com/documentation/ddi0596/2021-12/Base-Instructions. [cit. 2024-02-14].

[4] Artūrs; Serhii; Oskars; Matīss a Māris. *RouterOS: OpenVPN* online. Riga: MikroTik, prosinec 2024. Dostupné z: https://help.mikrotik.com/docs/spaces/ROS/pages/2031655/OpenVPN. [cit. 2025-05-15].

[5] Balena. *BalenaEtcher* online. 2023. Dostupné z: https://etcher.balena.io/. [cit. 2023-12-21].

[6] beetlrokr; n0cn0c; joshenders; daztucker; supersandro2000 et al. *OpenWrt Project: Tailscale* online. 2025. Dostupné z: https://openwrt.org/docs/guide-user/services/vpn/tailscale/start. [cit. 2025-2-2].

[7] Berners Lee, T.; Cailliau, R.; Groff, J.-F. a Pollermann, B. World-Wide Web: the information universe. *Internet Research*. MCB UP Ltd, 1992, sv. 2, č. 1, s. 52–58.

[8] bobafetthotmail; dvn; stintel; palebloodsky; stokito et al. *OpenWrt Project: OpenWrt as Docker container host* online. 2025. Dostupné z: https://openwrt.org/docs/guide-user/virtualization/docker_host. [cit. 2025-5-4].

[9] Bos, B. *A brief history of CSS until 2016* online. 2016. Dostupné z: https://www.w3.org/Style/CSS20/history.html. [cit. 2023-12-17].

[10] Boucadair, M.; Ford, M.; Roberts, P.; Durand, A. a Levis, P. *Issues with IP Address Sharing* RFC 6269. RFC Editor, jun 2011. Dostupné z: https://doi.org/10.17487/RFC6269.

[11] Brown, M. R. *FastCGI Specification*. 1996. Dostupné z: https://web.archive.org/web/20160119141816/http://www.fastcgi.com/drupal/node/6?q=node%2F22.

[12] Campbell, T. Y. A.; Beck, B.; Friedl, M.; Provos, N.; Raadt, T. de et al. *Ssh — OpenSSH remote login client* online. červenec 2023. Dostupné z: https://man.openbsd.org/ssh. [cit. 2023-09-26].

[13] Campbell, T. Y. A.; Beck, B.; Friedl, M.; Provos, N.; Raadt, T. de et al. *Sshd — OpenSSH daemon* online. Září 2023. Dostupné z: https://man.openbsd.org/sshd. [cit. 2023-09-27].

[14] CERN. Change History for httpd. *W3C* online. 2018. Dostupné z: https://www.w3.org/Daemon/Features.html. [cit. 2023-12-16].

[15] Cloutier, F. *X86 and amd64 instruction reference* online. 2024. Dostupné z: https://www.felixcloutier.com/x86/. [cit. 2024-2-14].

[16] Coar, K. A. a Robinson, D. *The Common Gateway Interface (CGI) Version 1.1* RFC 3875. RFC Editor, říjen 2004. Dostupné z: https://doi.org/10.17487/RFC3875.

[17] Development Team, N. httpd. Introduction to the CGI. *NCSA httpd* online. 1998. Dostupné z: https://www6.uniovi.es/~antonio/ncsa_httpd/cgi/intro.html. [cit. 2023-12-16].

[18] Dixon, T. *Serveo: expose local servers to the internet using SSH* online. 2024. Dostupné z: https://serveo.net/. [cit. 2024-12-29].

[19] Docker, Inc. *Dockerhub: scratch* online. 2025. Dostupné z: https://hub.docker.com/_/scratch. [cit. 2025-5-5].

[20] Filipe Laíns, M. T. *Code 1.86.0-1* online. 2024. Dostupné z: https://archlinux.org/packages/extra/x86_64/code/. [cit. 2024-2-14].

[21] Ford, B.; Kegel, D. a Srisuresh, P. *State of Peer-to-Peer (P2P) Communication across Network Address Translators (NATs)* RFC 5128. RFC Editor, březen 2008. Dostupné z: https://doi.org/10.17487/RFC5128.

[22] Ford, B.; Srisuresh, P. a Kegel, D. *Peer-to-Peer Communication Across Network Address Translators*. 2006. Dostupné z: https://arxiv.org/abs/cs/0603074.

[23] Foundation, T. A. S. *Apache Tutorial: Dynamic Content with CGI* online. 2023. Dostupné z: https://httpd.apache.org/docs/2.4/howto/cgi.html. [cit. 2023-12-16].

[24] GameRanger Pty Ltd.. About. *GameRanger* online. 2025. Dostupné z: https://www.gameranger.com/about/. [cit. 2025-01-28].

[25] GoTo Group, Inc. *Hamachi network types: About hub-and-spoke networks* online. 2025. Dostupné z: https://support.goto.com/hamachi/help/logmein-hamachi-network-types-hamachi-c-hamachi-networktypes. [cit. 2025-02-26].

[26] Grigorik, I. *High Performance Browser Networking: What every web developer should know about networking and web performance*. 3. vyd. O'Reilly Media, Inc., 2013. ISBN 978-1-449-34476-4.

[27] Hors, A. L.; Hégaret, P. L.; Wood, L.; Nicol, G.; Robie, J. et al. *Document Object Model (DOM) Level 2 Core Specification* online. 2020. Dostupné z: https://www.w3.org/TR/2020/SPSD-DOM-Level-2-Core-20201103/. [cit. 2025-05-11].

[28] Ierymenko, A.; Finney, B. a Smith, T. ZeroTier Network Virtualization Engine 1.4.4. *GitHub* online. Leden 2025. Dostupné z: https://raw.githubusercontent.com/zerotier/ZeroTierOne/refs/heads/dev/LICENSE.txt. [cit. 2025-01-20].

[29] Johnson, J. *Designing with the mind in mind: simple guide to understanding user interface design guidelines*. 3. vyd. Cambridge United States: Morgan Kaufmann, srpen 2020. ISBN 9780128182024.

[30] Jurczyk, M. *Windows X86-64 System Call Table (XP/2003/Vista/7/8/10/2022/11)* online. 2024. Dostupné z: https://j00ru.vexillium.org/syscalls/nt/64/. [cit. 2024-2-14].

[31] Kanaris, O. a Pouwelse, J. Mass Adoption of NATs: Survey and experiments on carrier-grade NATs. *ArXiv preprint arXiv:2311.04658*, 2023.

[32] Kerrisk, M. *System Calls Manual* online. Prosinec 2023. Dostupné z: https://man7.org/linux/man-pages/man2/syscalls.2.html. [cit. 2024-02-14].

[33] Kew, N. *The Apache Modules Book: Application Development with Apache*. Pearson Education, 2007. Pearson Open Source Software Development Series. ISBN 9780132704502.

[34] key-networks; lideming a Martignène, N. ztncui. *GitHub* online. 2023. Dostupné z: https://github.com/key-networks/ztncui/blob/master/README.md. [cit. 2025-02-3].

[35] kitware, inc. *CMake: The Standard Build System* online. 2024. Dostupné z: https://cmake.org/features/. [cit. 2024-2-14].

[36] Kulíšek, V. *Síťová komunikace a kolaborativní sdílení dat pro strategickou hru na webové platformě* online. Brno, 2022. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Jan Pečiva, P. Dostupné z: https://www.vut.cz/www_base/zav_prace_soubor_verejne.php?file_id=245368.

[37] Kulíšek, V. ZeroTier Controller Client. *Excel@FIT*, 2025. Dostupné z: https://excel.fit.vutbr.cz/submissions/2025/058/58.pdf.

[38] Kwame, A. E.; Martey, E. M. a Chris, A. G. Qualitative assessment of compiled, interpreted and hybrid programming languages. *Communications*. Foundation of Computer Science (FCS), NY, USA, 2017, sv. 7, č. 7, s. 8–13.

[39] Lin, D. A. Y.; Malis, A. G.; Heinanen, D. J.; Gleeson, B. a Armitage, D. G. *A Framework for IP Based Virtual Private Networks* RFC 2764. RFC Editor, únor 2000. Dostupné z: https://doi.org/10.17487/RFC2764.

[40] LogMeIn, Inc. *VPN.net – Hamachi by LogMeIn* online. 2018. Dostupné z: https://vpn.net/. [cit. 2024-1-18].

[41] Matthews, P.; Rosenberg, J.; Wing, D. a Mahy, R. *Session Traversal Utilities for NAT (STUN)* RFC 5389. RFC Editor, říjen 2008. Dostupné z: https://doi.org/10.17487/RFC5389.

[42] McKenzie, C. *How to setup PHP on Nginx with fastCGI (PHP-FPM) example* online. 2022. Dostupné z: https://www.theserverside.com/blog/Coffee-Talk-Java-News-Stories-and-Opinions/Nginx-PHP-FPM-config-example. [cit. 2025-05-09].

[43] MDN contributors. *JavaScript* online. Září 2023. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript. [cit. 2024-02-15].

[44] MDN contributors. *HTML: HyperText Markup Language* online. Duben 2025. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTML. [cit. 2025-05-10].

[45] MikroTik. *Mikrotik: hAP ax lite* online. 2023. Dostupné z: https://mikrotik.com/product/hap_ax_lite. [cit. 2025-5-4].

[46] NetBird. *Introduction to NetBird: Self-hosting quickstart guide (5 min)* online. 2025. Dostupné z: https://docs.netbird.io/selfhosted/selfhosted-quickstart. [cit. 2025-2-3].

[47] netmaker.io. *NetMaker Docs: Quick Install* online. 2025. Dostupné z: https://docs.netmaker.io/docs/server-installation/quick-install. [cit. 2025-2-3].

[48] ngrok, Inc. *Docs: HTTP Endpoints* online. 2025. Dostupné z: https://ngrok.com/docs/http/. [cit. 2025-1-31].

[49] ngrok, Inc. *Docs: TCP Endpoints* online. 2025. Dostupné z: https://ngrok.com/docs/tcp/. [cit. 2025-1-31].

[50] ngrok, Inc. *Ngrok: How does ngrok work* online. 2025. Dostupné z: https://ngrok.com/docs/how-ngrok-works/. [cit. 2025-4-16].

[51] ngrok, Inc. *Ngrok: Pricing* online. 2025. Dostupné z: https://ngrok.com/pricing. [cit. 2025-1-31].

[52] Normunds; Artūrs; Antons; Serhii; Oskars et al. *RouterOS: WireGuard* online. Riga: MikroTik, leden 2025. Dostupné z: https://help.mikrotik.com/docs/spaces/ROS/pages/69664792/WireGuard. [cit. 2025-05-15].

[53] Normunds; Artūrs; Antons; Serhii; Oļegs et al. *RouterOS: Container* online. Riga: MikroTik, prosinec 2024. Dostupné z: https://help.mikrotik.com/docs/spaces/ROS/pages/84901929/Container. [cit. 2025-04-30].

[54] OpenJS Foundation a Electron contributors. *Application Packaging* online. Leden 2023. Dostupné z: https://www.electronjs.org/docs/latest/tutorial/application-distribution. [cit. 2024-02-15].

[55] OpenJS Foundation a Electron contributors. *Electron* online. 2023. Dostupné z: https://www.electronjs.org/. [cit. 2023-12-21].

[56] Oskars; Māris; Matīss; Serhii; Edgars et al. *MikroTik documentation wiki: L2TP* online. Riga: MikroTik, prosinec 2024. Dostupné z: https://help.mikrotik.com/docs/spaces/ROS/pages/2031631/L2TP. [cit. 2025-04-23].

[57] Parekh, S. *Apache Module Registry* online. 1999. Dostupné z: https://web.archive.org/web/19990222075112/http://modules.apache.org/doc/Intro_API_Prog.html. [cit. 2023-12-16].

[58] Pennarun, A. *How Tailscale works* online. Toronto: Tailscale Inc., March 2020. Dostupné z: https://tailscale.com/blog/how-tailscale-works/. [cit. 2023-10-01].

[59] PHP Documentation Group. *FastCGI Process Manager (FPM)* online. únor 2022. Dostupné z: https://www.php.net/manual/en/install.fpm.php. [cit. 2025-05-09].

[60] Pinto, C. M. a Coutinho, C. From Native to Cross-platform Hybrid Development. In: *2018 International Conference on Intelligent Systems (IS)* online. IEEE, Září 2018. ISBN 978-1-5386-7097-2. Dostupné z: https://doi.org/10.1109/is.2018.8710545.

[61] Potter, J. *Npm trends: electron* online. 2023. Dostupné z: https://npmtrends.com/electron. [cit. 2023-12-20].

[62] Rauschmayer, A. *Speaking JavaScript: an in-depth guide for programmers* online. 1. vyd. "O'Reilly Media, Inc.", 2014. ISBN 978-1-4493-6499-1. Dostupné z: https://exploringjs.com/es5/.

[63] Rosenberg, J.; Huitema, C.; Mahy, R. a Weinberger, J. *STUN – Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs)* RFC 3489. RFC Editor, březen 2003. Dostupné z: https://doi.org/10.17487/RFC3489.

[64] Sanner, M. F. et al. Python: a programming language for software integration and development. *J Mol Graph Model*, 1999, sv. 17, č. 1, s. 57–61.

[65] Satrapa, P. *IPv6* online. CZ.NIC. Praha: CZ.NIC, z. s. p. o., 2019. ISBN 978-80-88168-46-1. Dostupné z: https://knihy.nic.cz/cs/detail/22/. [cit. 2024-10-17].

[66] SoftEther Project. Virtual Hub Security Features. *SoftEther VPN Project* online. 2025. Dostupné z: https://www.softether.org/4-docs/1-manual/3/3.5. [cit. 2025-01-30].

[67] SoftEther Project. Dynamic DNS and NAT Traversal. *SoftEther VPN Project* online. 2025. Dostupné z: https://www.softether.org/4-docs/2-howto/6.VPN_Server_Behind_NAT_or_Firewall/1.Dynamic_DNS_and_NAT_Traversal. [cit. 2025-01-29].

[68] SoftEther Project. Ultimate Powerful VPN Connectivit. *SoftEther VPN Project* online. 2025. Dostupné z: https://www.softether.org/1-features/1._Ultimate_Powerful_VPN_Connectivity. [cit. 2025-01-30].

[69] SoftEther Project. *VPN Azure Cloud Service* online. 2025. Dostupné z: https://www.vpnazure.net/en/. [cit. 2025-01-31].

[70] Strods; Krisjanisj; Arturszon; Janisk; Pauls et al. *MikroTik documentation wiki: Manual:IP/Cloud* online. Riga: MikroTik, červen 2022. Dostupné z: https://wiki.mikrotik.com/wiki/Manual:IP/Cloud#DDNS. [cit. 2023-09-24].

[71] Sun Microsystems. *Java Look and Feel Design Guidelines*. Addison-Wesley, 1999. Java Series. ISBN 9780201615852.

[72] Sun Microsystems, Inc. Java™: An Overview. *Sun Microsystems Laboratories The First Ten Years*, 2001.

[73] Synology Inc. *Synology: Container Manager* online. 2025. Dostupné z: https://www.synology.com/cs-cz/dsm/feature/docker. [cit. 2025-5-4].

[74] Tailscale Inc. *Pricing* online. 2025. Dostupné z: https://tailscale.com/pricing. [cit. 2025-2-3].

[75] Tailscale Inc. *Tailscale: Download* online. 2025. Dostupné z: https://tailscale.com/download. [cit. 2025-2-2].

[76] Tailscale Inc. *Tailscale: Connect to network attached storage (NAS)* online. 2025. Dostupné z: https://tailscale.com/kb/1307/nas. [cit. 2025-2-2].

[77] Tailscale Inc. *Tailscale: Open source at Tailscale* online. 2025. Dostupné z: https://tailscale.com/opensource. [cit. 2025-2-3].

[78] team lighty. *The CGI-Module* online. 2023. Dostupné z: https://redmine.lighttpd.net/projects/lighttpd/wiki/Mod_cgi. [cit. 2023-12-16].

[79] The Apache Software Foundation. *Electron Platform Guide* online. 2023. Dostupné z: https://cordova.apache.org/docs/en/12.x/guide/platforms/electron/.

[80] W3Techs. *W3Techs – World Wide Web Technology Surveys: Usage statistics and market shares of JavaScript libraries* online. 2025. Dostupné z: https://w3techs.com/technologies/overview/javascript_library. [cit. 2025-05-09].

[81] Wadhwa, P.; dlorenc; Johnson, J.; Rickard, M.; Kubala, N. et al. *Kaniko: Build Images In Kubernetes* online. 2025. Dostupné z: https://github.com/GoogleContainerTools/kaniko/blob/main/README.md. [cit. 2025-5-6].

[82] Wei, Y.; Yamada, D.; Yoshida, S. a Goto, S. A new method for symmetric NAT traversal in UDP and TCP. *Network*, 2008, sv. 4, č. 8.

[83] Weil, J.; Kuarsingh, V.; Donley, C.; Liljenstolpe, C. a Azinger, M. *IANA-Reserved IPv4 Prefix for Shared Address Space* RFC 6598. RFC Editor, duben 2012. Dostupné z: https://doi.org/10.17487/RFC6598.

[84] Wojtczyk, M. a Knoll, A. A Cross Platform Development Workflow for C/C++ Applications. In: *2008 The Third International Conference on Software Engineering Advances*. 2008, s. 224–229. ISBN 978-1-4244-3218-9.

[85] ZeroTier, Inc. *Protocol Design Whitepaper* online. Irvine: ZeroTier, Inc., 2023. Dostupné z: https://docs.zerotier.com/zerotier/manual. [cit. 2023-09-29].

[86] ZeroTier, Inc. *ZeroTier Documentation: ZeroTierOne Service API* online. 2024. Dostupné z: https://docs.zerotier.com/service/v1/. [cit. 2024-1-25].

[87] ZeroTier, Inc. *Pricing* online. 2025. Dostupné z: https://www.zerotier.com/pricing/. [cit. 2025-2-3].

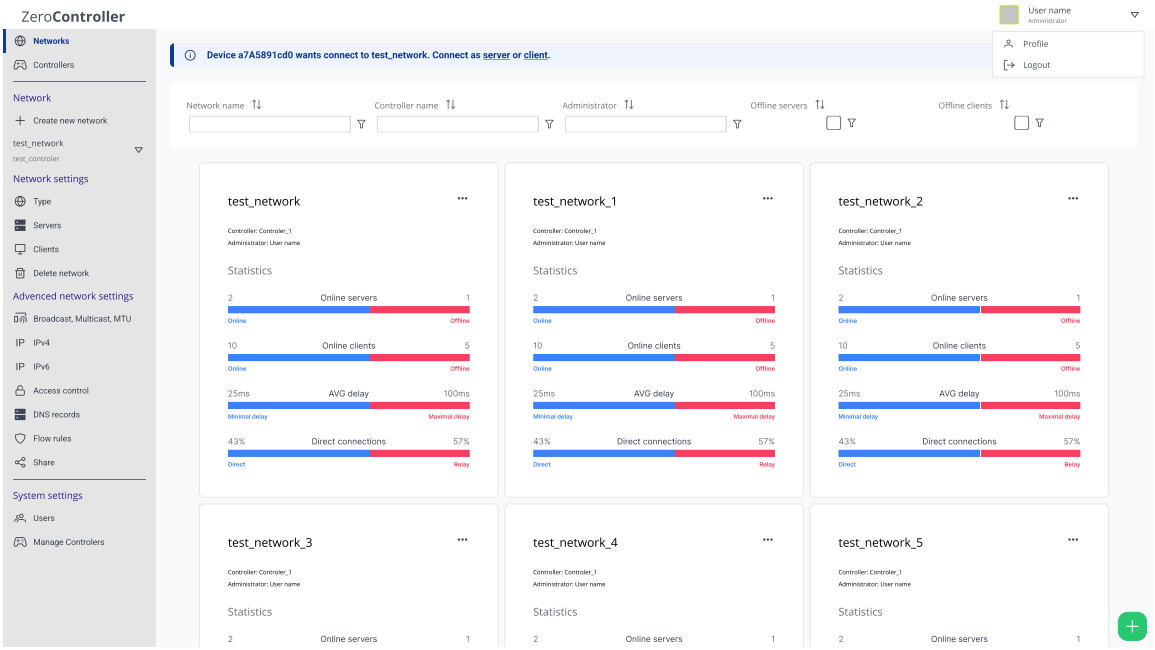
[88] ZeroTier, Inc. *ZeroTier: Download* online. 2025. Dostupné z: https://www.zerotier.com/download/. [cit. 2025-2-3].

[89] ZeroTier, Inc. *ZeroTier: Welcome to the ZeroTier Community* online. 2025. Dostupné z: https://www.zerotier.com/community/. [cit. 2025-2-3].

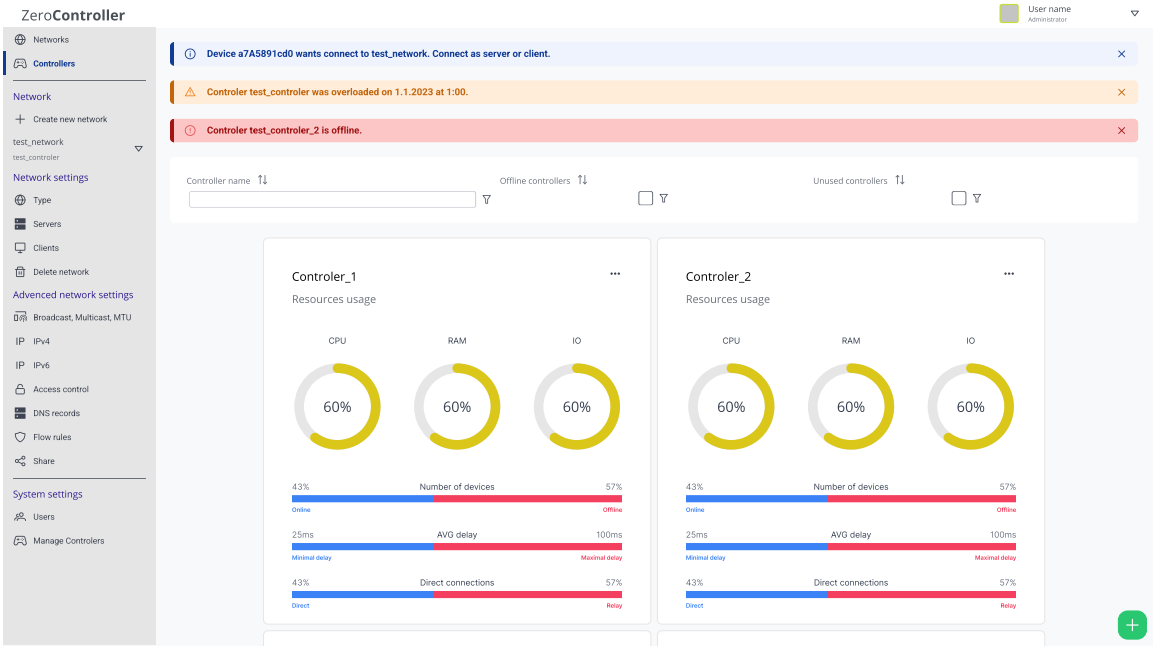
[90] Zhao, C. *V0.1.0: Update node: use node's implementation of setImmediate.* online. 2013. Dostupné z: https://github.com/electron/electron/releases/tag/v0.1.0. [cit. 2023-12-20].

Příloha A

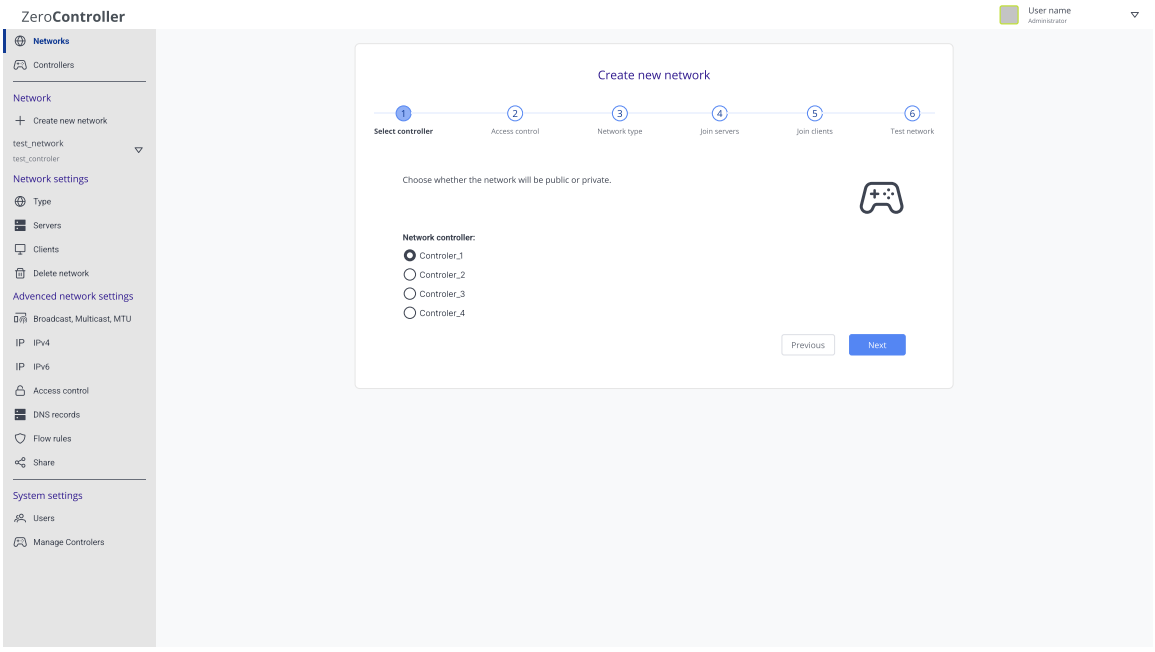
Návrh uživatelského rozhraní



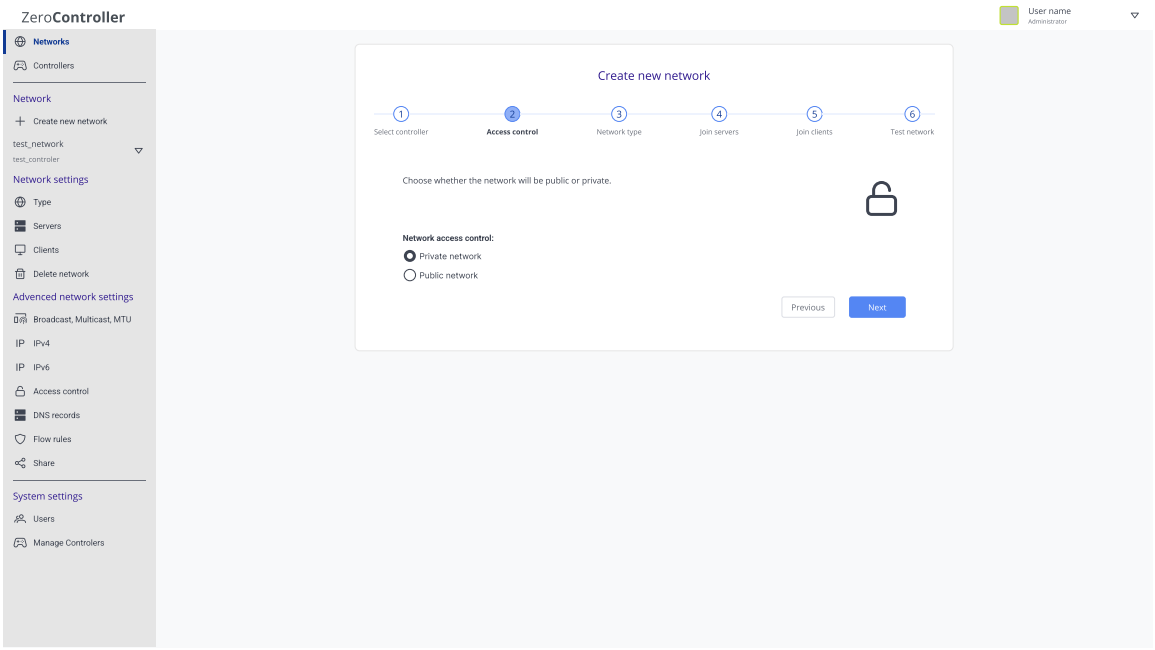
Obrázek A.1: Statistiky sítí



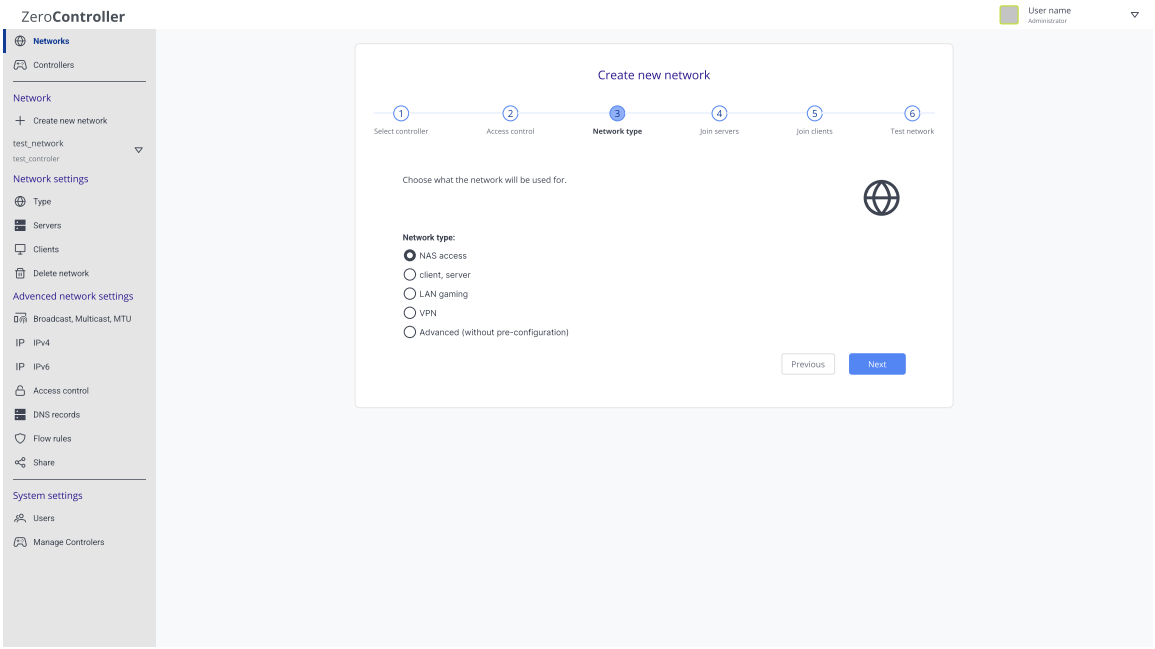
Obrázek A.2: Statistiky kontrolérů



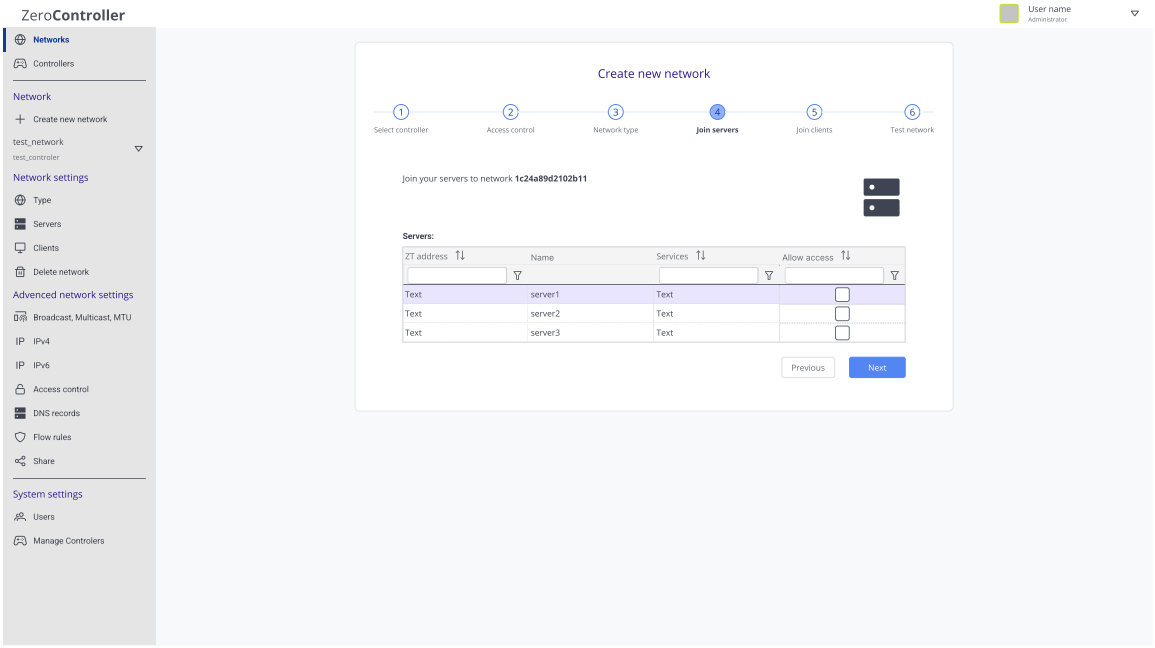
Obrázek A.3: Vytvoření nové sítě krok první



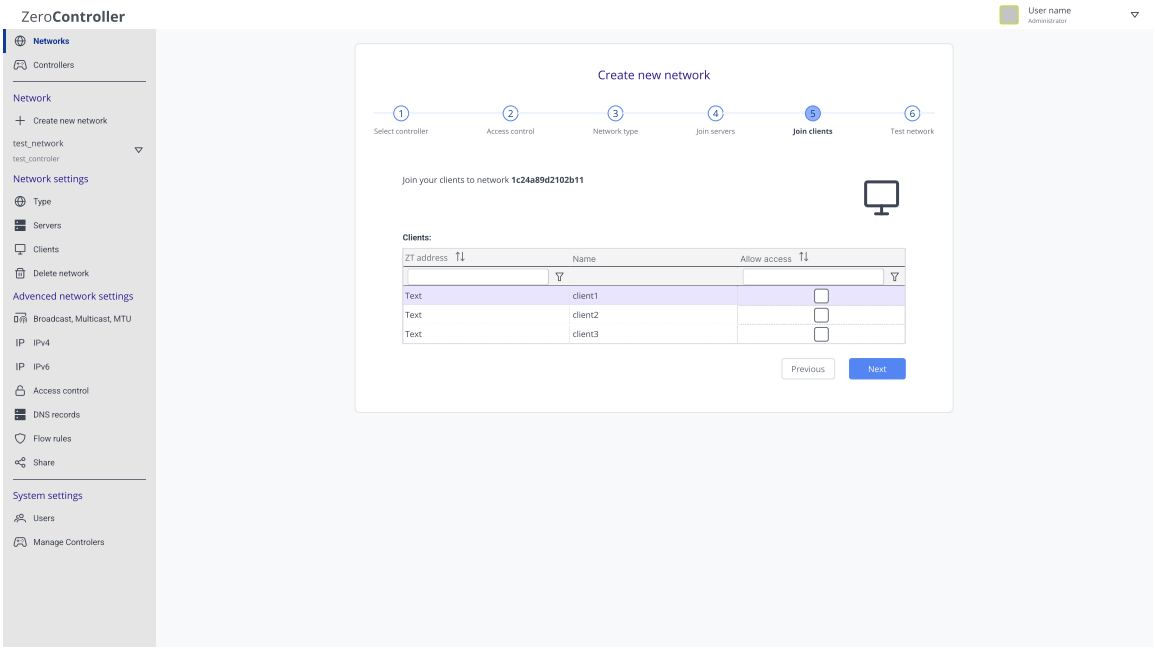
Obrázek A.4: Vytvoření nové sítě krok druhý



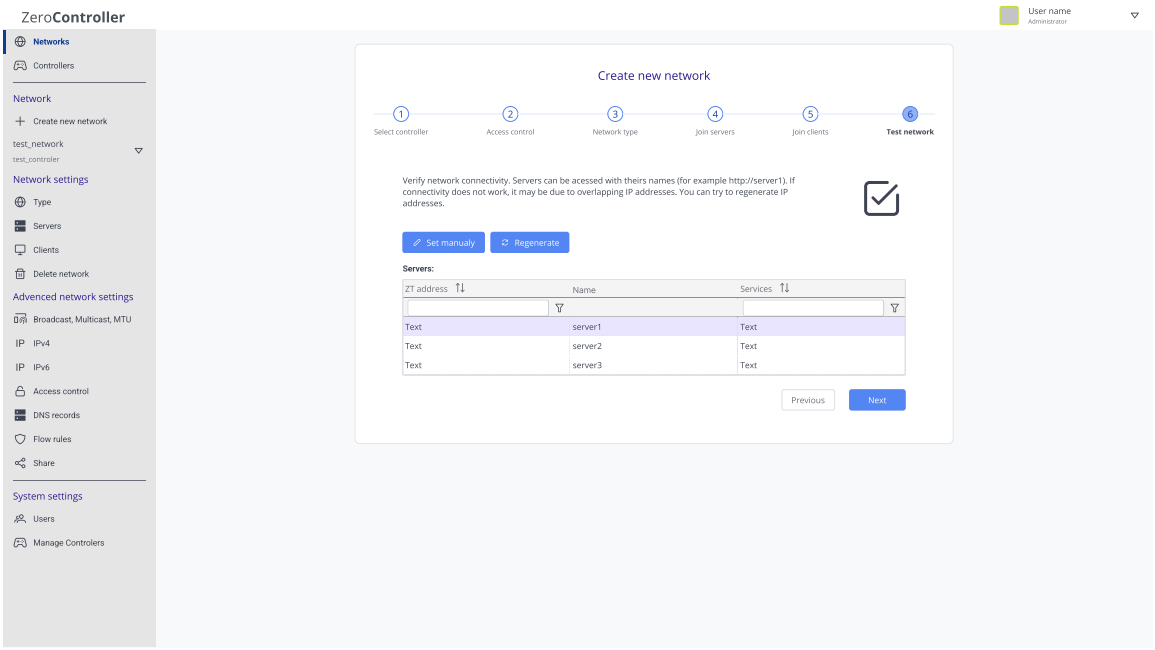
Obrázek A.5: Vytvoření nové sítě krok třetí



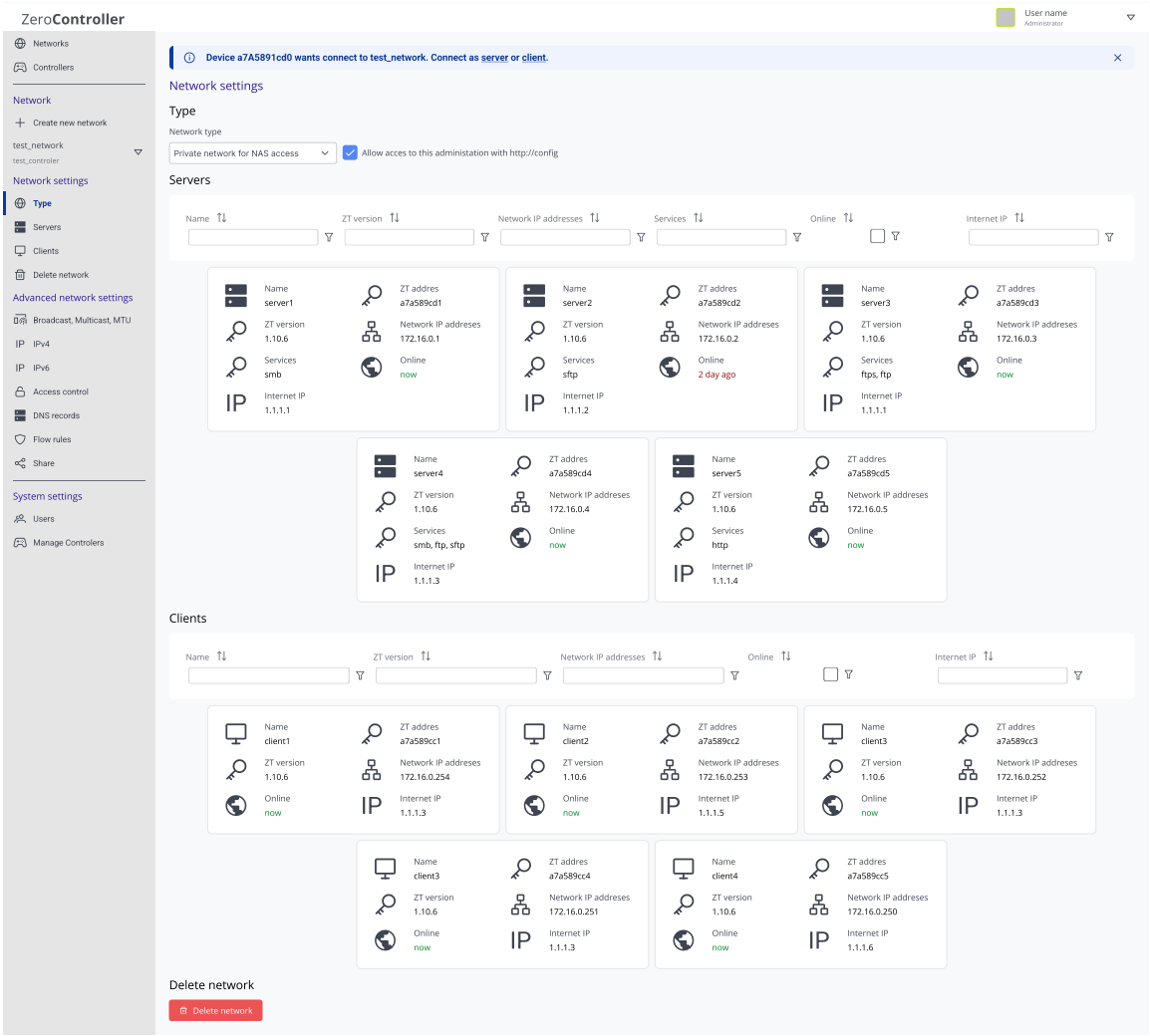
Obrázek A.6: Vytvoření nové sítě krok čtvrtý



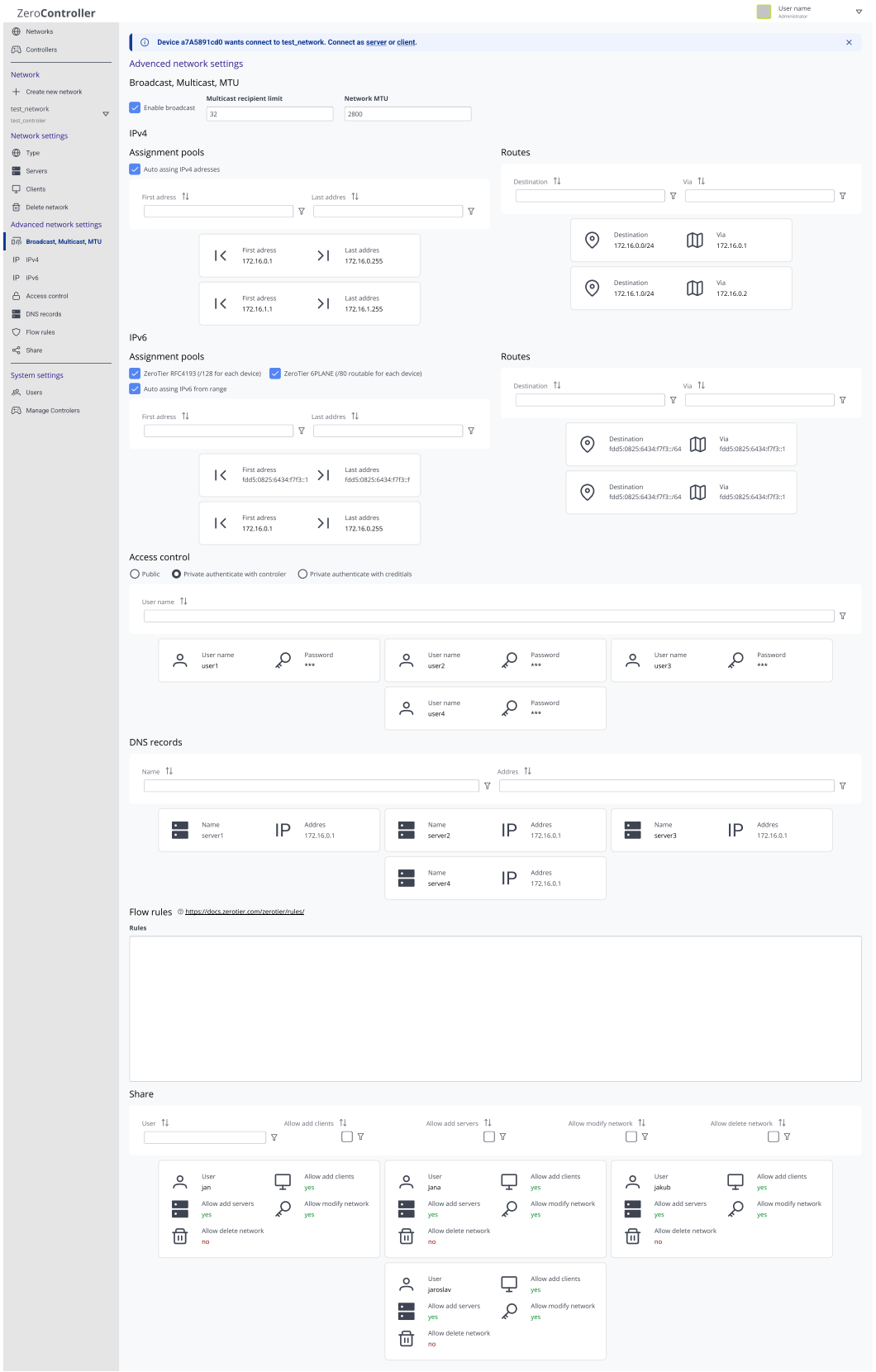
Obrázek A.7: Vytvoření nové sítě krok pátý



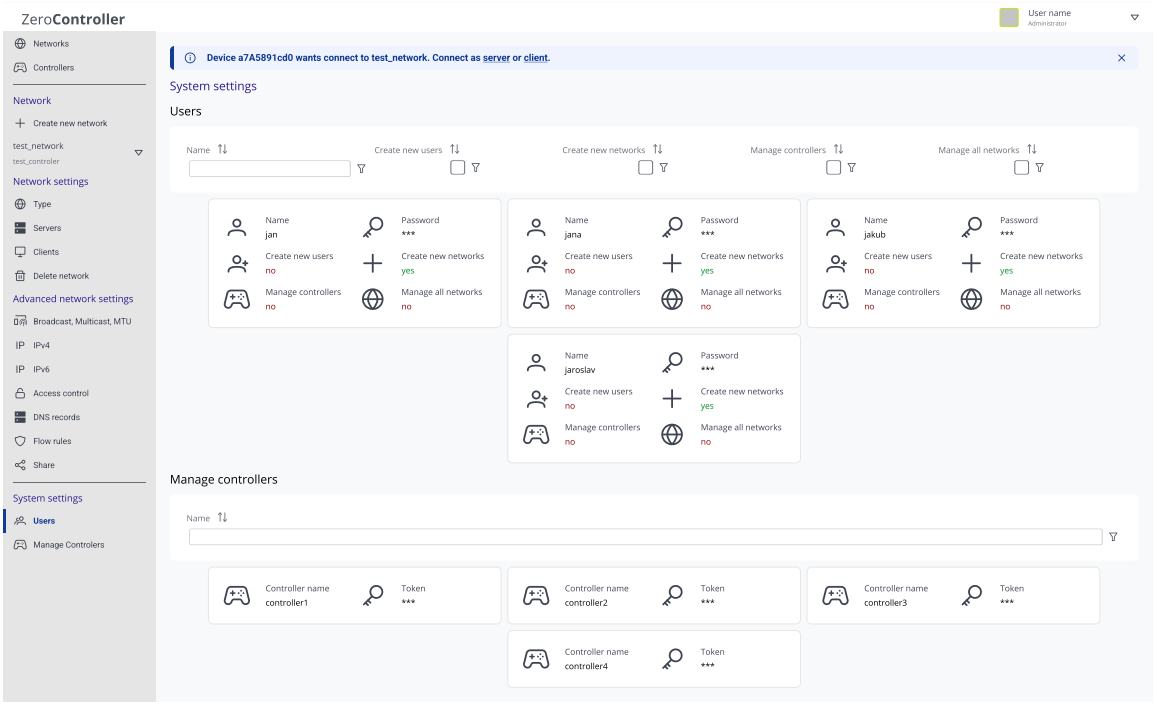
Obrázek A.8: Vytvoření nové sítě krok šestý



Obrázek A.9: Základní nastavení



Obrázek A.10: Pokročilá nastavení

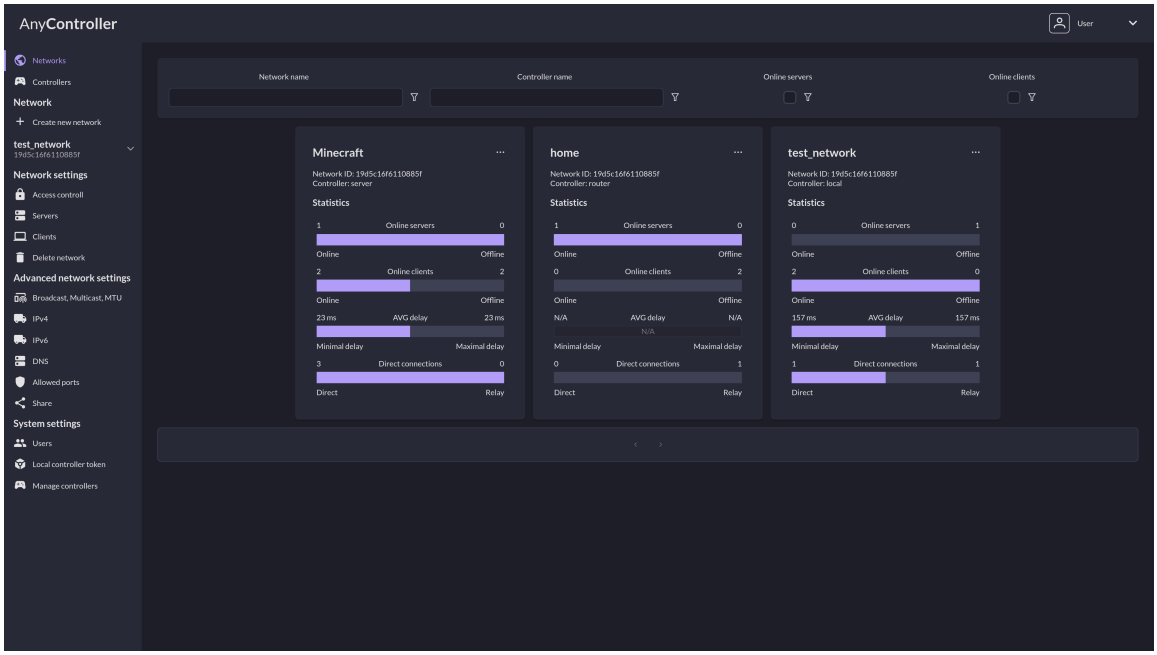


Obrázek A.11: Nastavení uživatelů a přidání kontroléru

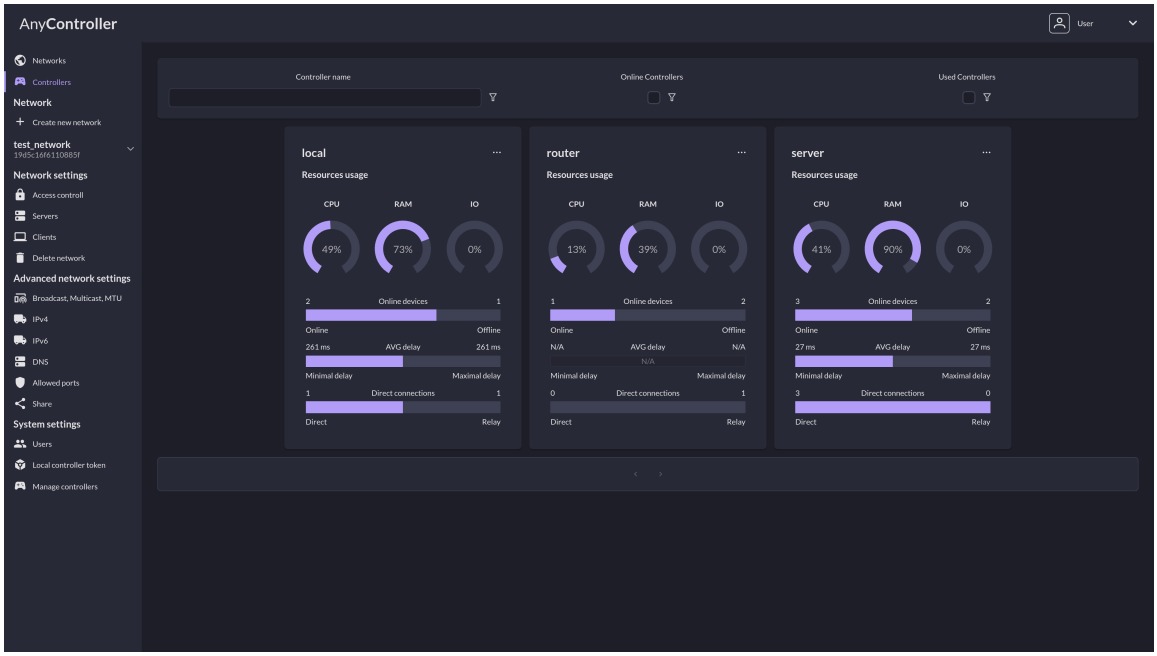
Obrázek A.12: Ikony aplikace

Příloha B

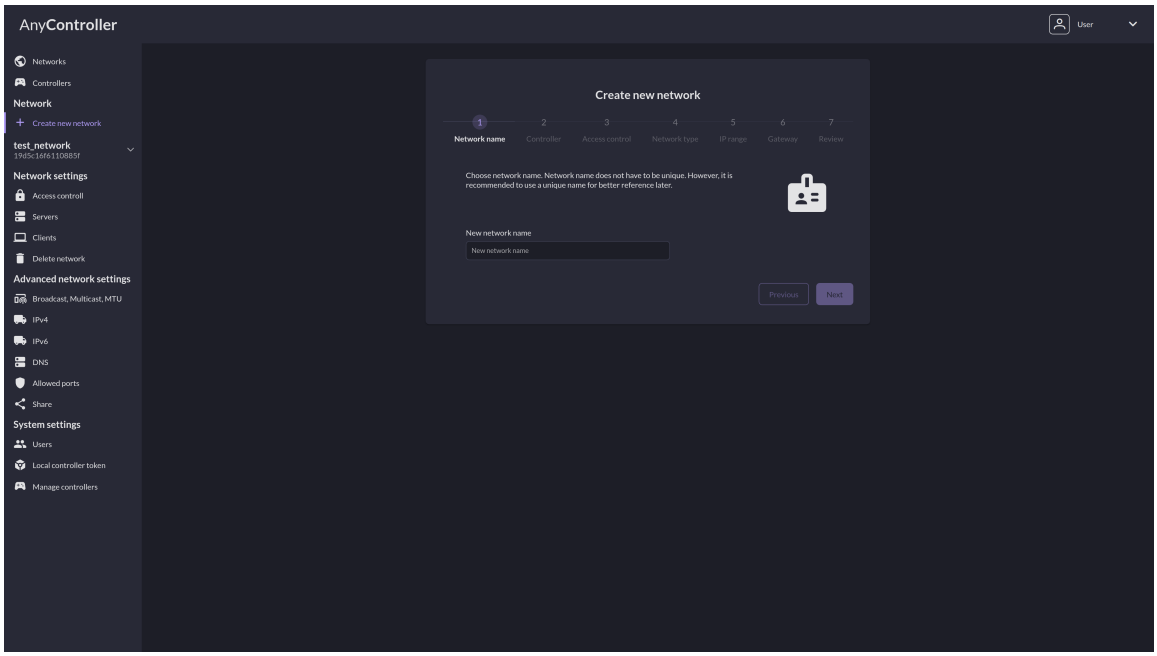
Implementovaná aplikace



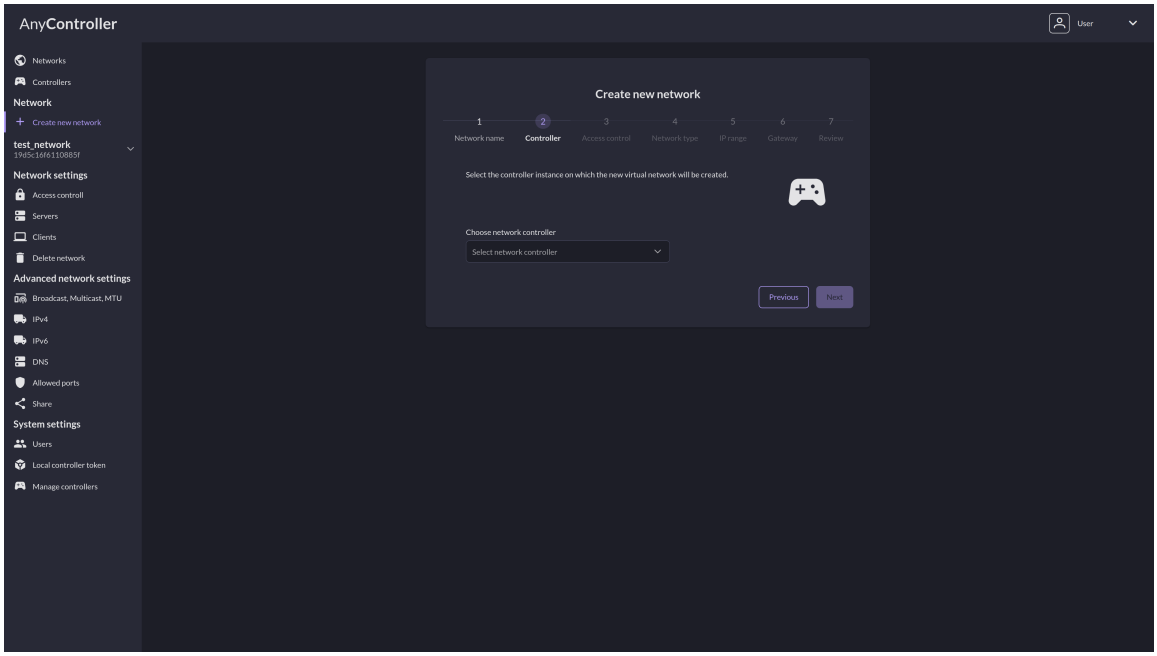
Obrázek B.1: Statistiky sítí



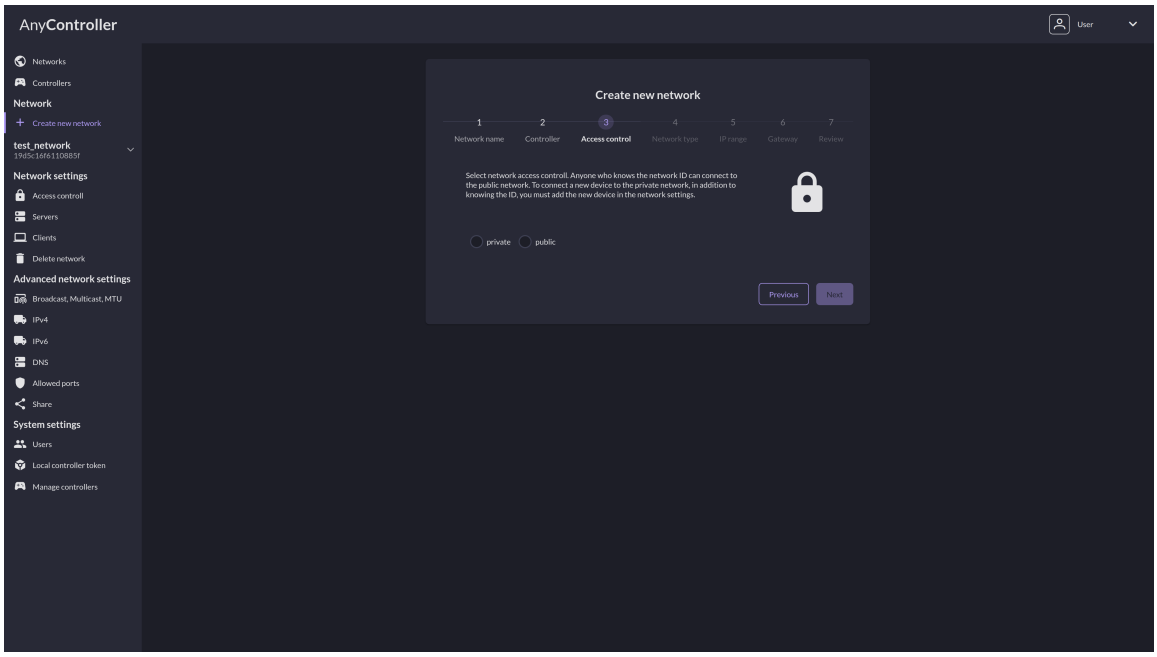
Obrázek B.2: Statistiky kontrolérů



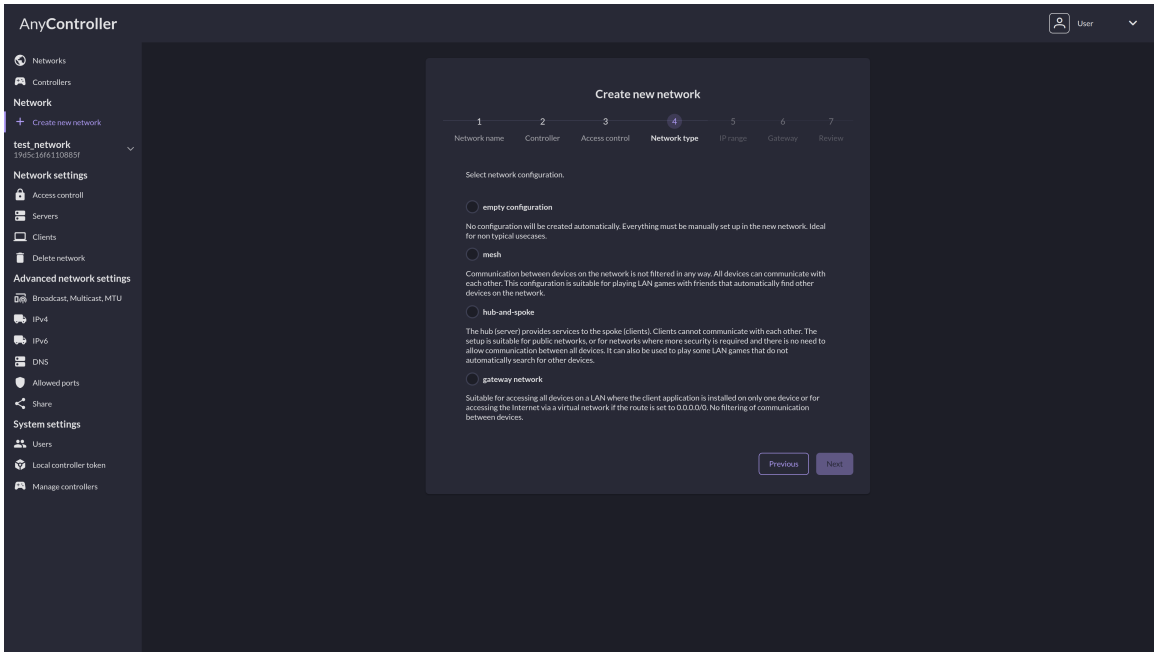
Obrázek B.3: Vytvoření nové sítě krok první



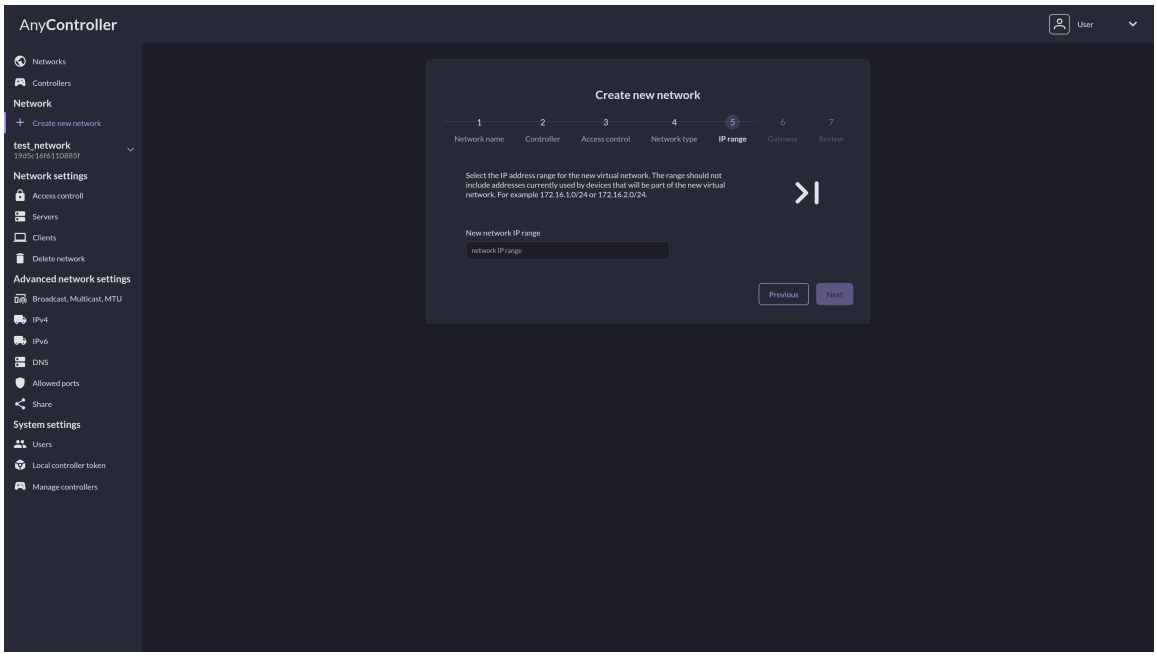
Obrázek B.4: Vytvoření nové sítě krok druhý



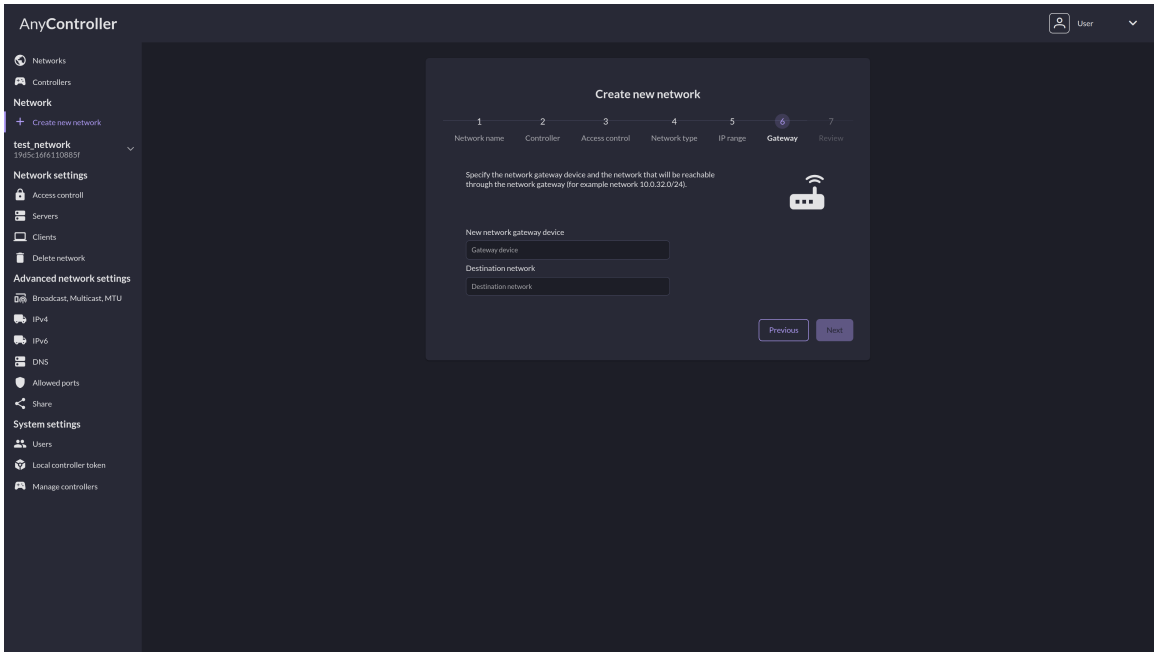
Obrázek B.5: Vytvoření nové sítě krok třetí



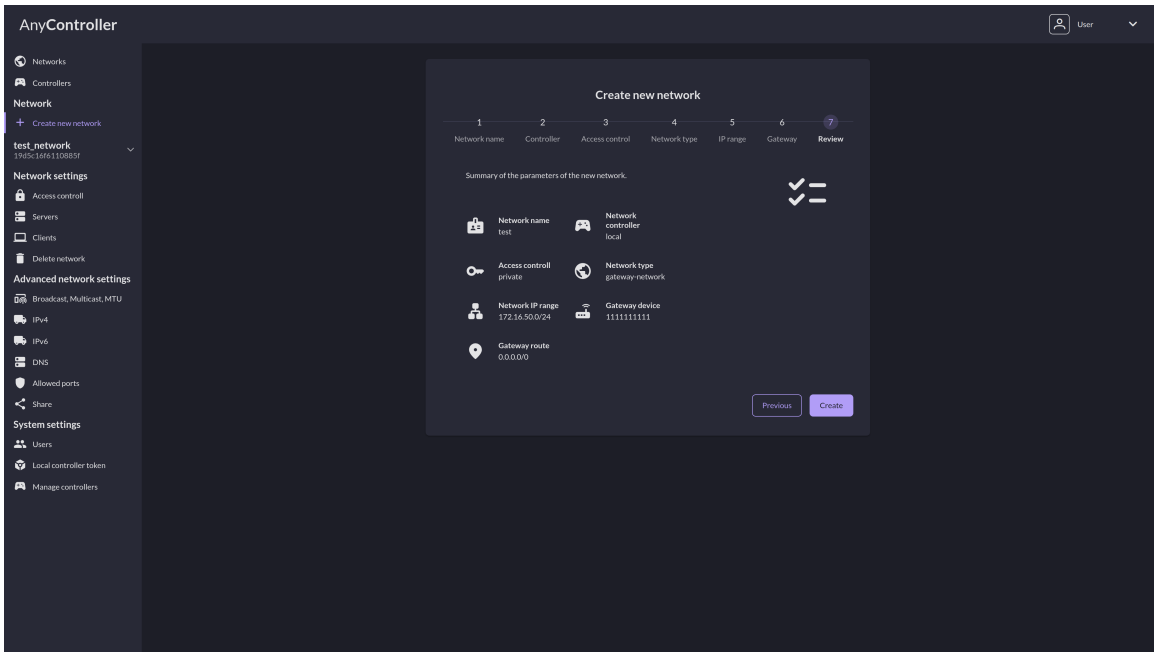
Obrázek B.6: Vytvoření nové sítě krok čtvrtý



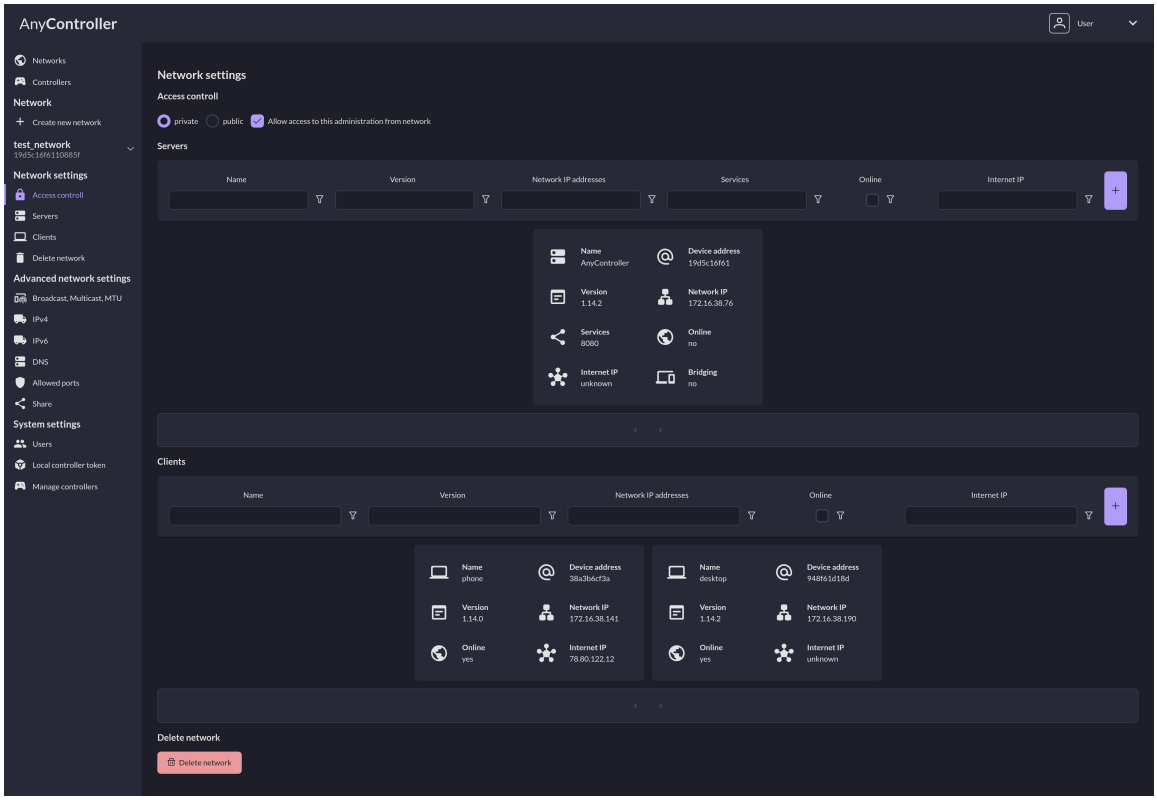
Obrázek B.7: Vytvoření nové sítě krok pátý



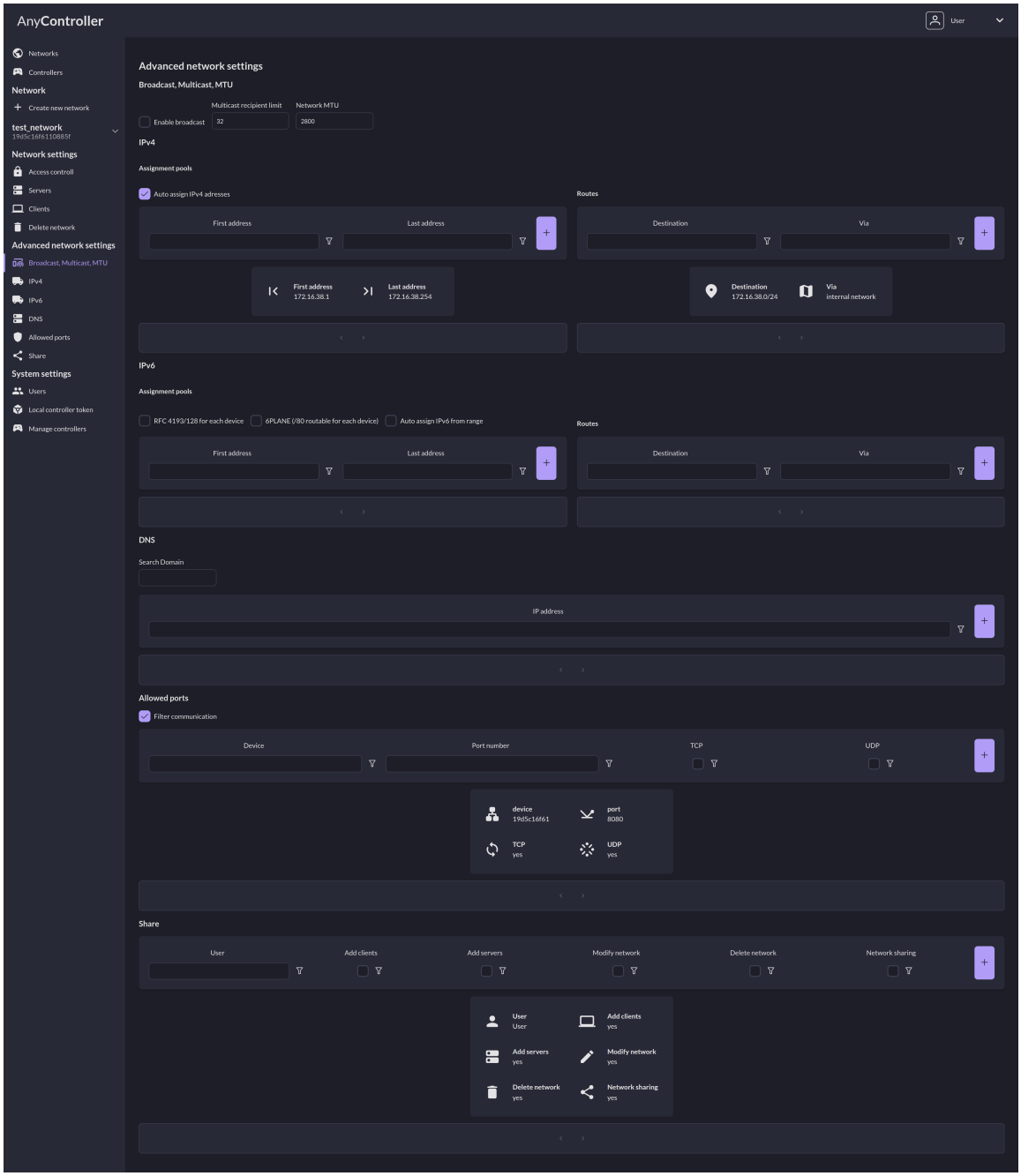
Obrázek B.8: Vytvoření nové sítě krok šestý



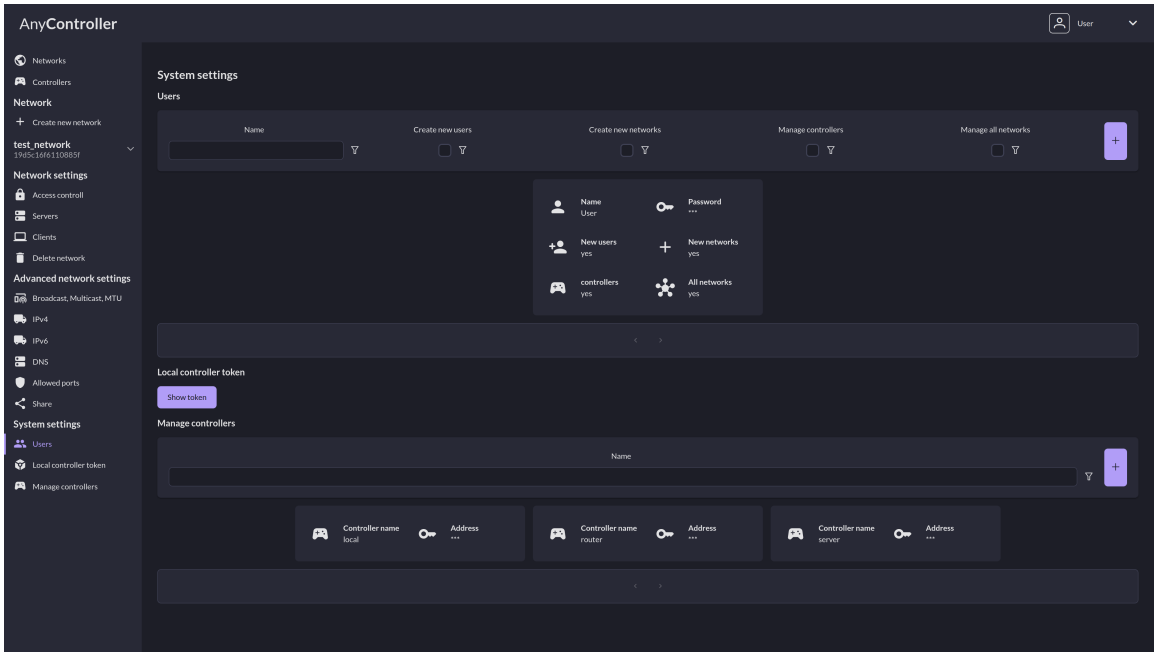
Obrázek B.9: Vytvoření nové sítě krok sedmý



Obrázek B.10: Základní nastavení



Obrázek B.11: Pokročilá nastavení



Obrázek B.12: Nastavení uživatelů a přidání kontroléru