

**SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY**

Evidenčné číslo: FEI-16605-111186

**VPN MANAŽÉR
BAKALÁRSKA PRÁCA**

2023

Viktor Bojda

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Evidenčné číslo: FEI-16605-111186

VPN MANAŽÉR
BAKALÁRSKA PRÁCA

Študijný program: Aplikovaná informatika
Názov študijného odboru: Informatika
Školiace pracovisko: Ústav informatiky a matematiky
Vedúci záverečnej práce: Ing. Peter Jakubis

Bratislava 2023

Viktor Bojda



ZADANIE BAKALÁRSKEJ PRÁCE

Študent: **Viktor Bojda**
ID študenta: 111186
Študijný program: aplikovaná informatika
Študijný odbor: informatika
Vedúci práce: Ing. Peter Jakubis
Vedúci pracoviska: doc. Ing. Milan Vojvoda, PhD.
Miesto vypracovania: Ústav informatiky a matematiky FEI STU

Názov práce: **VPN manažér**

Jazyk, v ktorom sa práca vypracuje: slovenský jazyk

Špecifikácia zadania:

Cieľom bakalárskej práce je oboznámiť sa s VPN nástrojom Pritunl.

Úlohy:

1. Naštudujte oblasť VPN a jej manažovania.
2. Navrhňte a vytvorte systém, ktorý umožní v nástroji Pritunl riadiť smerovanie (Routes) na viacerých serveroch súčasne. Systém by mal umožniť automatické spustenie/zastavenie servera počas nasadzovania zmien, prekladať DNS domény na IP adresy a poskytnúť prehľad o prístupoch.
3. Overte vytvorené riešenie a zdokumentujte ho.

Táto bakalárska práca bude vypracovaná v spolupráci so spoločnosťou Piano Software, ktorá je lídrom v poskytovaní SaaS produktov v oblasti médií.

Zoznam odbornej literatúry:

1. SCOTT, Ch., WOLFE, P. a ERWIN, M. Virtual Private Networks: Turning the Internet Into Your Private Network. 2. vyd. O'Reilly Media, 1998. isbn 978-1-56592- 529-8.
2. SNADER, Jon C. VPNs Illustrated: Tunnels, VPNs, and IPsec. 1. vyd. AddisonWesley Professional, 2005. isbn 978-0-321-24544-1.

Termín odovzdania bakalárskej práce:	02. 06. 2023
Dátum schválenia zadania bakalárskej práce:	01. 06. 2023
Zadanie bakalárskej práce schválil:	prof. Dr. rer. nat. Martin Drozda – garant študijného programu

SÚHRN

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Študijný program:	Aplikovaná informatika
Autor:	Viktor Bojda
Bakalárska práca:	VPN manažér
Vedúci záverečnej práce:	Ing. Peter Jakubis
Miesto a rok predloženia práce:	Bratislava 2023

Cieľom tejto práce bola detailná analýza VPN služby Pritunl, identifikácia potenciálnych nedostatkov a následná implementácia vlastného riešenia, ktoré by tieto nedostatky odstránilo a pridalo požadovanú funkcionality. Výsledkom tejto snahy bola webová aplikácia - VPN Manager. Aplikácia poskytuje nástroje na spravovanie klientov, organizácií, serverov a routes. Implementuje vlastné REST API, ktoré komunikuje s Pritunl API a dopĺňa chýbajúcu funkcionality. Aplikácia disponuje webovým rozhraním, ktoré ponúka nástroje na interakciu s vytvoreným REST API. Implementuje tiež autentifikačný a autorizačný systém, ktorý chráni vytvorené API a webové rozhranie. Výsledná aplikácia spĺňa bezpečnostné požiadavky a umožňuje efektívnejšiu prácu so službou Pritunl.

Kľúčové slová: VPN, Pritunl, webová aplikácia

ABSTRACT

SLOVAK UNIVERSITY OF TECHNOLOGY IN BRATISLAVA

FACULTY OF ELECTRICAL ENGINEERING AND INFORMATION TECHNOLOGY

Study Programme:	Applied Informatics
Author:	Viktor Bojda
Bachelor's thesis:	VPN Manager
Supervisor:	Ing. Peter Jakubis
Place and year of submission:	Bratislava 2023

The aim of this thesis was to analyse Pritunl VPN service in detail, identify potential shortcomings and then implement a solution that would eliminate these shortcomings and add the required functionality. The result of this effort was a web application - VPN Manager. The application provides tools to manage clients, organizations, servers and routes. It implements a custom REST API that communicates with Pritunl API and adds missing functionality. The application has a web interface that provides tools to interact with the built REST API. It also implements an authentication and authorization system that protects the created API and web interface. The resulting application meets security requirements and allows users to work more efficiently with the Pritunl service.

Keywords: VPN, Pritunl, web application

Podakovanie

Rád by som vyjadril vďaku môjmu vedúcemu práce, Ing. Petrovi Jakubisovi, za jeho trpezlivé vedenie, podporu a odborné rady počas tvorby tejto bakalárskej práce. Chcem sa tiež poďakovať spoločnosti Pritunl za poskytnutie študentskej licencie, ktorá bola nevyhnutná na realizáciu tejto práce.

Obsah

Úvod	1
1 VPNs a riešenie Pritunl	2
1.1 Úvod do Virtual Private Networks	2
1.2 OpenVPN	3
1.3 Pritunl	5
1.4 Prehľad existujúcich riešení	5
1.5 Zhrnutie	6
2 Návrh riešenia	7
2.1 Konceptuálny návrh	7
2.1.1 Funkcionálne požiadavky	7
2.1.2 Nefunkcionálne požiadavky	8
2.1.3 Zhrnutie	8
2.2 Použité technológie	9
2.2.1 Python	9
2.2.2 Django	10
2.2.3 Django Rest Framework	10
2.2.4 REST API	11
2.2.5 Django Template Language	11
2.2.6 JavaScript a jQuery	12
2.3 Architektonický návrh	12
2.3.1 Prezentačná vrstva	12
2.3.2 Aplikačná vrstva	13
2.3.3 Dátová vrstva	14
3 Implementácia riešenia	16
3.1 Komunikácia s Pritunl	16
3.2 API špecifikácia a realizácia	17
3.2.1 Vytvorenie organizácie	17
3.2.2 Získanie zoznamu organizácií	17
3.2.3 Vytvorenie klienta	18
3.2.4 Získanie zoznamu klientov	18
3.2.5 Vytvorenie servera	19
3.2.6 Získanie zoznamu serverov	21

3.2.7	Ovládanie servera	22
3.2.8	Vytvorenie route pre server	23
3.2.9	Pripojenie organizácie ku serveru	24
3.3	Autentifikácia a autorizácia	25
3.4	Pritunl eventy	27
3.5	Webové rozhranie	29
3.5.1	Klienti a organizácie	30
3.5.2	Servery	34
4	Vyhodnotenie riešenia	38
4.1	Bezpečnosť	38
4.2	Funkcionalita	39
4.3	Webové rozhranie	40
Záver		42
Zoznam použitej literatúry		43
Prílohy		I
A Zdrojový kód		II
B Inštalačná a používateľská príručka		III

Zoznam obrázkov a tabuliek

Obrázok 1	Konceptuálny návrh pre VPN Manager	9
Obrázok 2	Sekvenčný diagram návrhu pre detekciu zmien pomocou long pollingu	14
Obrázok 3	VPN Manager: Prihlasovacie rozhranie	27
Obrázok 4	VPN Manager: Klienti a organizácie - žiadne vytvorené organizácie	31
Obrázok 5	VPN Manager: Klienti a organizácie - vytvorenie organizácie (vľavo), vytvorenie klienta (stred), hromadné vytvorenie klientov (vpravo)	31
Obrázok 6	VPN Manager: Klienti a organizácie - zoznam s organizáciami a ich klientami	32
Obrázok 7	VPN Manager: Klienti a organizácie - prehľad odkazov na stia- hnutie klientských profilov (vľavo), odstránenie vybraných entít (vpravo)	33
Obrázok 8	VPN Manager: Klienti a organizácie - filtrovanie organizácií (vľavo), filtrovanie klientov (vpravo)	34
Obrázok 9	VPN Manager: Servery - žiadne vytvorené servery	35
Obrázok 10	VPN Manager: Servery - vytvorenie servera (vľavo), vytvorenie route (stred), hromadné vytvorenie routes (vpravo)	36
Obrázok 11	VPN Manager: Servery - pripojenie organizácie (vľavo), odstrá- nenie vybraných entít (vpravo)	37
Obrázok 12	VPN Manager: Servery - spustený server	38
Tabuľka 1	API špecifikácia: Vytvorenie organizácie	18
Tabuľka 2	API špecifikácia: Získanie zoznamu organizácií	18
Tabuľka 3	API špecifikácia: Vytvorenie klienta	19
Tabuľka 4	API špecifikácia: Získanie zoznamu klientov	19
Tabuľka 5	API špecifikácia: Vytvorenie servera	21
Tabuľka 6	API špecifikácia: Získanie zoznamu serverov	22
Tabuľka 7	API špecifikácia: Ovládanie servera	22
Tabuľka 8	API špecifikácia: Vytvorenie route pre server	25
Tabuľka 9	API špecifikácia: Pripojenie organizácie ku serveru	25
Tabuľka 10	Prehľad Pritunl eventov	28
Tabuľka 11	Testovanie: Hromadné vytváranie routes	40

Tabuľka 12 Testovanie: Hromadné odstraňovanie routes a odpájanie organizácií 40

Zoznam skratiek

3DES	Triple Data Encryption Standard
AES	Advanced Encryption Standard
IKE	Internet Key Exchange
MD5	Message Digest Algorithm 5
ORM	Object-Relational Mapping
OSI	Open Systems Interconnection
PFS	Perfect Forward Secrecy
REST	Representational State Transfer
SHA	Secure Hash Algorithm
SSE	Server-Sent Events
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol
VPNs	Virtual Private Networks

Úvod

Žijeme vo svete, v ktorom sa dáta stali jednou z najcennejších komodít na trhu. Zachovanie bezpečnosti informácií sa preto stalo prioritným záujmom organizácií po celom svete. Obzvlášť v dnešnej dobe, kde podstatná časť zamestnancov pracuje z pohodlia domova, potrebovali organizácie spôsob ako bezpečne a efektívne svoje dáta zdieľať a pristupovať ku nim na diaľku. Táto skutočnosť vyniesla do popredia používanie Virtual Private Networks (VPNs).

VPNs sa stali preferovaným nástrojom na sprostredkovanie bezpečného pripojenie naprieč internetom. Jedným z obľúbených VPN riešení je aj Pritunl, vďaka svojmu užívateľsky prívetivému webovému rozhraniu a komplexným metódam na vytvorenie bezpečnej komunikácie. Napriek robustným VPN funkcionalitám, ktoré Pritunl ponúka, naskytuje sa príležitosť využitia poskytnutého API na implementáciu vlastného riešenia, ktoré by rozšírilo jeho služby o ďalšie funkcie.

Takéto rozšírenie má potenciál zefektívniť riadenie sieťových pripojení, poskytnúť lepšiu kontrolu nad prístupmi používateľov a umožniť zautomatizovať alebo dokonca úplne odstrániť niektoré procesy. Práve implementácia takéhoto vlastného riešenia, konkrétne webovej aplikácie, bude hlavnou témou tejto práce.

V úvodnej časti tejto práce sme sa venovali definícii pojmov a konceptov spojených s fungovaním VPN, ktoré sú nevyhnutné pre pochopenie danej problematiky. Zanalyzovali sme nástroje a funkcie, ktoré Pritunl poskytuje a spravili sme si prehľad existujúcich riešení. Na základe získaných poznatkov sme následne definovali požiadavky, ktoré naša aplikácia musí spĺňať a pripravili sme podrobný návrh riešenia.

V ďalšej časti sme prešli na softvérovú implementáciu a poskytli sme podrobný popis funkcií, ktorými naša aplikácia disponuje. Na záver sme poskytli pohľad na konečnú bezpečnosť a funkcionálnosť a vykonali sme sériu testov na našej aplikácii. Výsledky testovania sme potom porovnali so vstavaným rozhraním Pritunl a zhodnotili sme, do akej miery sa nám podarilo splniť stanovené požiadavky a aké problémy naša aplikácia rieši.

1 VPNs a riešenie Pritunl

V tejto kapitole sa zoznámime s Virtual Private Networks, predstavíme si ich použitie a identifikujeme zložky, ktoré by mala bezpečná VPN obsahovať. Potom prejdeme na analýzu protokolu OpenVPN. Následne si podrobne rozoberieme riešenie Pritunl, kde si objasníme, ako funguje a detailne popíšeme niektoré základné koncepty pre lepšie pochopenie problematiky, ktorou sa v tejto práci zaoberáme.

1.1 Úvod do Virtual Private Networks

Virtual Private Networks (VPNs) sú dôležitým aspektom modernej internetovej bezpečnosti. Umožňujú totiž dosiahnutie bezpečnej komunikácie medzi *vzdialenými sieťami*¹ a používateľmi naprieč verejnou sieťou ako je internet. VPNs boli vytvorené aby riešili rastúcu potrebu dosiahnuť zabezpečenie a taktiež cenovo efektívnu komunikáciu a to primárne medzi zamestnancami a vnútornou sieťou organizácií [1]. Táto potreba je jasne vidieť, keďže podľa štúdie [2] až 93% organizácií využíva služby minimálne jednej VPN služby.

V minulosti bolo jediným spôsobom ako dosiahnuť takéto bezpečné spojenie vytvorenie „pravých“ súkromných sietí, ktoré poskytujú ochranu prostredníctvom fyzickej separácie. Takáto separácia je dosiahnutá prenájmom telekomunikačných liniek alebo použitím vyhradeného hardvéru na šifrovanie dát. Naproti tomu, VPNs fungujú na princípe šifrovania a autentifikácie pomocou softvéru a teda bez nutnosti zaobstarania cenovo náročných hardvérových riešení. Okrem znížených nákladov, je vďaka širokej dostupnosti internetu vo väčšine rozvinutých oblastí možnosť pripojenia ku vnútornej sieti organizácie značne uľahčená a urýchlená. Vzhľadom na využitie robustných *cryptographic primitives*² a protokolov, sa dajú VPNs považovať za bezpečnejšie ako vyhradené hardvérové riešenia. [3]

Sprostredkovanie bezpečnej komunikácie je dosiahnuté vytvorením chráneného spojenia medzi klientmi³ a VPN serverom alebo vzdialenou sieťou. Takéto spojenie zaručuje, že dáta prenesené medzi dvomi bodmi zostanú súkromné, nezmenené a prístupné len pre oprávnených používateľov. Pre správne fungovanie takéhoto spojenia je potrebná spolupráca niekoľkých kľúčových komponentov:

Tunelovanie: Pred prenosom sú pôvodné dátové pakety zapuzdrené v inom pakete cez proces, ktorý sa nazýva tunelovanie. Toto zapuzdrenie vytvára tzv. „tunel“, ktorý umožňuje bezpečný prenos dát naprieč verejnou sieťou. Na nadviazanie a spravovanie

¹siete, ku ktorým sa používateľ pripája cez služby VPN

²osvedčených nízkoúrovňových šifrovacích algoritmov

³používateľskými zariadeniami

takéhoto tunelu, klient a server používajú tzv. *tunelové protokoly*. Keď sa dáta dostanú na druhý koniec tunela, zapuzdrenie je odstránené. Následne sú pôvodné dáta preposlané do ich určenej destinácie. [3]

Šifrovanie: Proces prevodu pôvodných dát do nečitateľného formátu, ktorý zabezpečuje utajenosť dát sa nazýva šifrovanie. Dátové pakety prenášané naprieč VPN tunelom sú šifrované pomocou algoritmov ako sú napríklad AES, Blowfish, 3DES a iné. Pred odoslaním dát cez tunel, VPN klient zašifruje dáta pomocou symetrického šifrovacieho kľúča. Následne iba VPN server alebo vzdialená sieť, ktorá má prístup ku symetrickému kľúču dokáže takto zašifrované dáta dešifrovať. [4]

Výmena kľúčov: So šifrovaním je nepochybne spätá aj výmena šifrovacích kľúčov medzi klientom a serverom. Pre zachovanie utajenosti a integrity šifrovaných dát bolo nutné prísť s riešením ako sprostredkovať takúto bezpečnú výmenu. Z tohoto dôvodu boli vyvinuté viaceré protokoly ako sú Internet Key Exchange (IKE) pre IPsec VPNs alebo Diffie-Hellmanova výmena kľúčov, využívaná napríklad v OpenVPN. [4]

Autentifikácia: Na oddelenie autorizovaných používateľov od potenciálnych narušiteľov sa využíva autentifikácia. Jedná sa o proces overenia identity VPN klienta, servera alebo vzdialenej siete. Bez autentifikácie by nebolo možné zabrániť neautorizovaným používateľom pristupovať ku VPN alebo zachytávať a následne manipulovať so zašifrovanými dátami. Existuje množstvo overovacích techník s rozličnými stupňami ochrany. Ako príklad je možné uviesť zadanie používateľského mena a hesla, digitálne certifikáty alebo využitie kryptografických kľúčov. [4]

Integrita dát: Nutnosťou je takisto overiť či dáta, ktoré boli prenesené cez tunel zostali neporušené a nezmenené. Na odhalenie manipulácie alebo úpravy dát počas prenosu sa využívajú techniky na kontrolu integrity dát ako je napríklad kryptografické hašovanie. Ak sa v prijatých dátach zistí rozdiel, môže to indikovať, že nastala manipulácia s dátami. V takom prípade sa môžu dotknuté dáta vyradiť alebo sa označia za podozrivé a bude ich nutné ďalej prešetriť. Medzi takéto hašovacie algoritmy patrí napríklad MD5, SHA-1, SHA-2, SHA-3 a iné. [4, 5]

1.2 OpenVPN

OpenVPN je populárny, open-source VPN protokol, ktorý ponúka spoľahlivé a vysoko konfigurovateľné riešenie na vytváranie zabezpečených point-to-point alebo site-to-site spojení. Na šifrovanie, autentifikáciu a výmenu kľúčov využíva OpenSSL knižnicu. Protokol funguje na aplikačnej vrstve v OSI modeli. Je veľmi flexibilný a je ho možné nastaviť a spustiť na širokej škále súčasne používaných operačných systémov. Medzi podporované

patria napríklad Windows, macOS a Linux, no aj mobilné operačné systémy ako sú Android a iOS. [6]

Používatelia si pre prenos dát môžu vybrať medzi User Datagram Protocol (UDP) a Transmission Control Protocol (TCP). Pre aplikácie bežiacie v reálnom čase je UDP zvyčajne tá lepšia voľba. Je totiž rýchlejší a efektívnejší. Naproti tomu, TCP ponúka pripojenie, ktoré je spoľahlivejšie. Poskytuje detekciu a opravu chýb, garanciu doručenia dát a to aj v poradí v akom boli vyslané. [6]

Jednou z hlavných predností OpenVPN je použitie Secure Sockets Layer/Transport Layer Security (SSL/TLS) pre výmenu kľúčov. Takéto riešenie dovoľuje nadviazanie bezpečného spojenia aj bez potreby vopred zdieľaných kľúčov. Namiesto toho je zdieľaný tajný kľúč vytvorený dynamicky počas vytvárania spojenia, bez nutnosti prenášať ho cez sieť. Táto funkcionálna je dosiahnutá využitím digitálnych certifikátov. Slúžia na autentifikáciu a napomáhajú procesu výmeny kľúčov. Okrem toho OpenVPN poskytuje bezpečnostnú funkciu Perfect Forward Secrecy (PFS). Tá vytvára nový šifrovací kľúč pre každú session, čím chráni v prípade kompromitácie kľúča ovplyvnenie ostatných sessions. [6]

Protokol podporuje rôzne typy šifrovacích algoritmov, musia však byť podporované knižnicou OpenSSL. Medzi ne patrí napríklad AES, podporujúci kľúče s dĺžkou 128, 192 a 256 bitov, a Blowfish, s kľúčami o dĺžke až 448 bitov. Ako predvolený je zvolený Blowfish, ktorý sa síce považuje za menej bezpečný ako napríklad AES, dosahuje však vyššej rýchlosti. [6, 7]

Vďaka vysokej úrovni konfigurácie môžu používatelia upravovať mnohé parametre vrátane *network routing*⁴, kompresie a šifrovacích nastavení. Týmto je možné dosiahnuť optimálnu rovnováhu medzi výkonom a bezpečnosťou. Podpora vytvárania virtuálnych sieťových rozhraní známych ako TUN (pre IP tunelovanie) a TAP (pre Ethernetové tunelovanie) je ďalším spôsobom, ktorým zvyšuje flexibilitu a kompatibilitu OpenVPN s rôznymi sieťovými konfiguráciami. [6]

Aj napriek tomu, že je OpenVPN široko považovaný za jeden z najbezpečnejších VPN protokolov má aj niekoľko nedostatkov. Vysoká konfigurovateľnosť spôsobuje taktiež značnú komplexnosť, ktorá môže odradiť veľké množstvo používateľov [8]. V porovnaní s inými protokolmi [9] môže zvýšená výpočtová náročnosť, spôsobená náročnými šifrovacími algoritmami a závislosťou na SSL/TLS pre výmenu kľúčov, ovplyvniť nižšie prenosové rýchlosti.

⁴v preklade: smerovanie sietí

1.3 Pritunl

Pritunl je open-source VPN riešenie postavené na OpenVPN protokole, ktorého funkcionality bola neskôr rozšírená o podporu pre WireGuard protokol [10]. Ponúka webové rozhranie pre riadenie a nasadzovanie VPN serverov a klientov. Vďaka tomu, že je postavený na OpenVPN dedí jeho širokú kompatibilitu naprieč platformami. Samotný Pritunl server je však možné prevádzkovať len na vybraných distribúciách Linuxu [10]. Navrhnutý bol tak aby bol škálovateľný, takže je vhodný ako pre menšie tak aj pre väčšie typy organizácií. Na ukladanie nastavení a používateľských dát využíva NoSQL databázu MongoDB. Pritunl funguje na báze licencií s niekoľkými cenovými kategóriami [10]. Ponúka aj bezplatnú verziu, tá je však značne okresaná o funkcionality.

Pre nadviazanie spojenia s VPN serverom vyvinul Pritunl aplikáciu „Pritunl Client“, ktorá zjednodušuje proces pripojenia, riadi klientské nastavenia a umožňuje monitorovanie stavu pripojenia. Táto aplikácia je však dostupná iba pre desktopové zariadenia, pre mobilné zariadenia odporúčajú používať aplikáciu „OpenVPN Connect“.

Centrálnym nástrojom pre prácu s Pritunl je už spomenuté webové rozhranie. Do neho sa môžu prihlásiť iba vybraní administrátori⁵, ktorí majú prístup ku všetkým funkciám a nastaveniam. Rozhranie je rozdelené na viacero častí, pričom medzi najdôležitejšie patria podstránky „Users“ a „Servers“. V časti Users je možné vytvárať tzv. organizácie, do ktorých sa dajú potom pridávať používatelia. Bez organizácie nemôžu používatelia existovať. Tieto organizácie sú vlastne iba skupiny, ktoré slúžia na zaradenie používateľov do jednotlivých serverov. Tým sa dostávame ku časti Servers. Tu je možné vytvárať, ovládať a konfigurovať vlastnosti VPN serverov. Akékoľvek vykonávanie zmien je možné iba na vypnutom serveri. Ku každému serveru vieme pridávať *routes*⁶ a pripájať organizácie. Routes rozhodujú, ktoré dáta sa posielajú cez VPN pripojenie a ktoré idú priamo cez štandardné internetové pripojenie používateľa. Pokiaľ napríklad chceme aby používatelia mali prístup do konkrétnej private network, pridáme jej adresu medzi routes daného servera.

1.4 Prehľad existujúcich riešení

Pred začatím práce na návrhu a implementácii nášho riešenia bolo dôležité vykonať prieskum pre už existujúce riešenia. Pritunl je pomerne populárna VPN služba a preto je možné nájsť viacero open-source projektov, ktoré s ním pracujú. Žiaden z týchto projektov

⁵v prípade inej ako najvyššej licencie existuje iba jeden administrátor

⁶v preklade: smerovania

však nevyhovuje našim požiadavkám, keďže sa jedná len o wrappers⁷ na základné ovládanie služieb Pritunl.

1.5 Zhrnutie

Virtual Private Networks sú už v dnešnej dobe neoddeliteľnou súčasťou internetovej bezpečnosti. Zaručujú súkromie, integritu dát a prístup iba pre oprávnených používateľov. Túto skutočnosť potvrdzuje aj vysoké percento organizácií, ktoré sa už spoliehajú na ochranu, ktorú im VPNs ponúkajú. Je preto dôležité poznať spôsob akým je táto bezpečnosť zaručená, aby mohli používatelia zvoliť pre nich správne VPN riešenie. Medzi kľúčové komponenty na dosiahnutie bezpečného spojenia sa radí: tunelovanie, šifrovanie, výmena kľúčov, autentifikácia a integrita dát.

Populárnym VPN protokolom je OpenVPN, ktorý je známy svojou bezpečnosťou, spoľahlivosťou a mnohými možnosťami pre nastavenie zabezpečeného spojenia. Disponuje širokou kompatibilitou naprieč operačnými systémami. Pre prenos dát umožňuje vybrať medzi UDP a TCP. Podporuje viacero overených šifrovacích algoritmov ako sú AES alebo Blowfish. Jeho vysoká konfigurovateľnosť síce poskytuje flexibilitu, ale môže v porovnaní s inými protokolmi spôsobiť aj zložitosť a potenciálne nižšie prenosové rýchlosti.

Jedným z obľúbených VPN riešení je Pritunl, ktorý je postavený na OpenVPN. Vďaka zabudovanej škálovateľnosti je vhodný pre organizácie rôznych veľkostí. Na zakrytie vysokej komplexnosti OpenVPN protokolu implementuje webové rozhranie, ktoré slúži na ovládanie VPN serverov, vytváranie používateľov a riadenie ich prístupov. Takéto rozhranie však nespĺňa naše požiadavky.

Ako vyplýva z podkapitoly 1.4, momentálne nevidujeme žiadne riešenie, ktoré by sa zaoberalo problémom popisovaným v tejto práci.

Poznámka: Od tohto momentu budeme jasne rozlišovať medzi termínmi „používateľ“ a „klient“. „Používateľ“ bude odkazovať na osobu, ktorá využíva webové rozhranie. „Klient“ bude označovať osobu, ktorá sa pripája cez VPN službu - toto zodpovedá použitiu termínu „user“ v kontexte Pritunl [11]. Takéto rozlíšenie je dôležité pre zachovanie konzistencie medzi Pritunl a naším riešením.

⁷enkapsulácia funkcionality pre uľahčenie používania

2 Návrh riešenia

Vzhľadom na uvedené poznatky a zistenia sme sa rozhodli navrhnúť softvérové riešenie, ktoré sme pomenovali *VPN Manager*. Naším cieľom je poskytnúť používateľom intuitívne webové rozhranie, ktoré je jednoduché na ovládanie. Rozhranie musí umožňovať efektívnu prácu so službou Pritunl a byť schopné rozlišovať medzi rôznymi typmi používateľov. Snažíme sa zjednodušiť interakciu s rozhraním, eliminovať redundanciu a presunúť niektoré procesy do pozadia.

2.1 Konceptuálny návrh

Niektoré systémové požiadavky sme už načrtli v úvode tejto kapitoly. Pred tým než však môžeme ďalej pokračovať, je nutné si ich riadne zadať.

2.1.1 Funkcionálne požiadavky

1. **Webové rozhranie:** Systém musí poskytovať webové rozhranie
2. **Spravovanie organizácií:** Systém musí umožňovať správu organizácií. Toto zahŕňa CRUD⁸ operácie a pripojenie ku serverom.
3. **Spravovanie klientov:** Systém musí umožňovať správu klientov. Toto zahŕňa CRUD operácie, hromadné vytváranie, zobrazenie stavu pripojenia a sťahovanie VPN profilov
4. **Spravovanie serverov:** Systém musí umožňovať správu serverov. Toto zahŕňa CRUD operácie, zobrazenie detailov, ovládanie a automatické spustenie/zastavenie servera počas nasadzovania zmien.
5. **Spravovanie routes:** Systém musí umožňovať správu routes. Toto zahŕňa CRUD operácie, hromadné vytváranie a preklad DNS domén na IP adresy.
6. **Filtrovanie:** Systém musí poskytovať možnosti filtrovania pre organizácie a klientov.
7. **Autentifikácia:** Systém musí zabezpečiť autentifikáciu na zabránenie neoprávnenému prístupu.
8. **Autorizácia:** Systém musí vedieť rozlišovať medzi prihlásenými používateľmi. Požadované sú dve role: admin a read-only. Admin musí mať absolútny prístup ku

⁸v tejto sekcii budeme pod týmto výrazom rozumieť: prehľad entít, ich vytváranie, modifikovanie a mazanie

všetkému čo systém ponúka. Read-only musí mať zobrazený prehľad o všetkých entitách a mať prístup ku filtrovaniu. Ostatné akcie dovolené nemá.

2.1.2 Nefunkcionálne požiadavky

1. **Bezpečnosť:** Systém by mal dodržiavať najnovšie štandardy a protokoly v oblasti kybernetickej bezpečnosti
2. **Udržateľnosť:** Systém by mal byť navrhnutý tak, aby sa dal ľahko aktualizovať alebo upravovať.
3. **Použiteľnosť:** Webové rozhranie poskytované systémom by malo byť intuitívne a užívateľsky prívetivé.
4. **Odolnosť:** Systém by mal zostať funkčný aj v prípade určitých typov porúch.

2.1.3 Zhrnutie

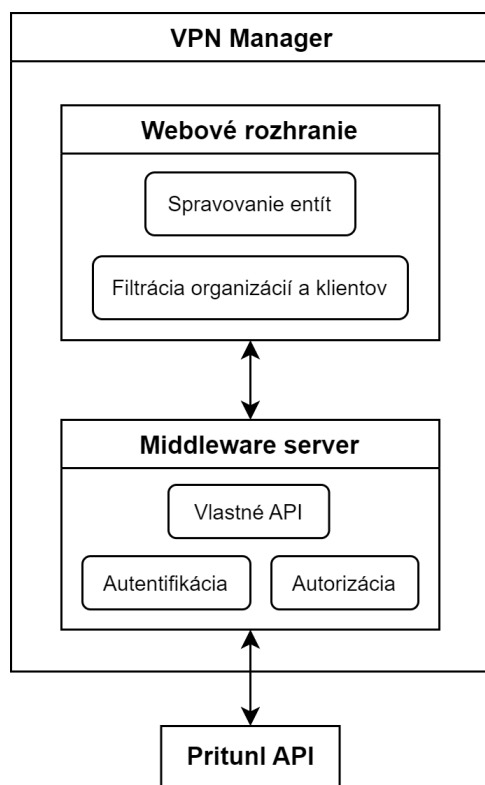
Na základe uvedených požiadaviek sme určili, že naše riešenie sa má skladať z dvoch častí: webové rozhranie a middleware server (obr. 1). Používatelia majú môcť pracovať s webovým rozhraním, ktoré bude slúžiť ako hlavný ovládací panel pre VPN Manager. Toto webové rozhranie má poskytovať nástroje na spravovanie organizácií, klientov, serverov a routes.

Komunikácia webového rozhrania má byť obmedzená výlučne na middleware server, ktorý následne komunikuje s Pritunl API. To znamená, že webové rozhranie by nikdy nemalo priamo komunikovať s Pritunl API. Existencia middleware servera má niekoľko dôvodov:

Prvým je implementácia rozširujúcich funkcií pre prácu s Pritunl, ako je napríklad automatické spustenie/zastavenie servera počas nasadzovania zmien a podobne. Tieto funkcie by boli veľmi náročné na implementáciu iba pomocou frontendu a priniesli by viaceré bezpečnostné riziká.

Druhým dôvodom je vytvorenie systému pre prihlásenie a rôzne úrovne autorizácie. Autentifikačný systém, ktorý poskytuje Pritunl je veľmi jednoduchý a pozná iba jeden typ používateľa. Preto bude potrebné implementovať na našom middleware serveri nový systém, ktorý je možné v prípade potreby ďalej rozširovať.

Tretím dôvodom je separácia logiky. Naš middleware server má implementovať vlastné API, ktoré zahŕňa celú logiku pre prácu so službou Pritunl. Vďaka tomu bude možné nahradiť alebo úplne odstrániť webové rozhranie, ak by to bolo požadované.



Obr. 1: Konceptuálny návrh pre VPN Manager

2.2 Použité technológie

Pred vývojom našej aplikácie si je nutné zvoliť s akými technológiami budeme jednotlivé časti vyvíjať. Pri výbere sme brali do úvahy najmä splnenie stanovených požiadaviek.

Middleware server sme sa rozhodli naprogramovať v jazyku Python v kombinácii s frameworkom Django a Django REST framework (DRF). Naše vlastné API budeme budovať v súlade so štýlom REST API. Pre implementáciu webového rozhrania sme si vybrali Django Template Language (DTL), HTML, CSS, JavaScript a doplnili sme ich o knižnice jQuery a Bootstrap.

V tejto časti si rozoberieme niektoré z využitých technológií a vysvetlíme, prečo sme sa rozhodli práve pre ne.

2.2.1 Python

Jadrom logiky na strane servera je Python. Jedná sa o vysoko-úrovňový, univerzálny programovací jazyk, ktorý je obľúbený najmä vďaka svojmu jednoduchému syntaxu a dobrej čitateľnosti [12]. Toto značne uľahčuje údržbu a pridávanie novej logiky, čím zabezpečuje dlhú životnosť a škálovateľnosť aplikácie. Jednou z jeho najväčších predností je aj široký výber dostupných knižníc a frameworkov, čo môže výrazne skrátiť čas vývoja.

2.2.2 Django

Django je populárny webový framework, vytvorený v Pythone, ktorý sa riadi filozofiou „batteries included“ [13]. To znamená, že ponúka množstvo predpripravených riešení pre bežné úlohy, s ktorými sa je možné pri vývoji webových stránok stretnúť.

Django je postavený na architektonickom návrhovom vzore Model-View-Controller (MVC). Avšak v Django kontexte sa často hovorí o Model-View-Template (MVT), čo presnejšie vystihuje jeho implementačnú štruktúru [14]. Prehľad MVT je nasledovný:

- **Model:** Dátová vrstva, ktorá spracováva všetky aspekty súvisiace s dátami. Definuje sa tu prístup k dátam, ich validácia, správanie a vzťahy medzi jednotlivými dátovými entitami.
- **View:** Miesto kde je implementovaná „biznis logika“. View spracuje webovú požiadavku a vráti odpoveď. Odpoveď môže byť vo forme HTML, JSON, presmerovania, chybovej správy a podobne.
- **Template:** Šablóny sú spôsob akým Django generuje dynamický HTML obsah. Používa na to Django Template Language.

Na komunikáciu s databázou implementoval Django vlastný ORM systém [15]. Tento systém nahrádza písanie často komplikovaných SQL dopytov a poskytuje Pythonské rozhranie, ktoré pracuje iba s definovanými modelmi. To redukuje potencionálne chyby a bezpečnostné hrozby, a vedie k čitateľnejšiemu a lepšie udržiateľnému kódu.

Django tiež poskytuje middleware systém, na báze zásuvných modulov, ktorý spracováva webové požiadavky a odpovede predtým, ako sa dostanú do ich cieľovej destinácie. Pomocou tohto systému Django implementuje autentifikačný systém a komplexnú vstavanú ochranu proti mnohým typom bezpečnostných hrozieb, ako sú napríklad Cross Site Scripting (XSS), Cross Site Request Forgery (CSRF), SQL Injection, a Clickjacking [16].

Ďalšou veľmi praktickou funkciou pre správu obsahu je Django Admin [17]. Ide o vstavané administrátorské webové rozhranie, ktoré je automaticky generované na základe definovaných modelov. Umožňuje plnohodnotne pracovať s dátovými modelmi v databáze aj bez nutnosti písania kódu. Je veľmi užitočné počas vývoja, no je ho možné využívať naďalej aj v prevádzke, keďže je chránené autentifikačným systémom.

2.2.3 Django Rest Framework

DRF je sada nástrojov na tvorbu webových API v prostredí Django [18]. Ponúka množstvo funkcií a modulov, ktoré umožňujú efektívne budovanie API rozhraní. Medzi ne patrí

napríklad serializácia a deserializácia, a autentifikácia spolu s riadením používateľských oprávnení.

Serializačný systém poskytuje jednoduchý mechanizmus pre konverziu komplexných dátových typov na dátové typy jazyka Python, ktoré je potom možné ľahko previesť do formátu JSON alebo XML. Deserializácia funguje opačne, konverziou dát z formátu JSON alebo XML do formátu komplexných dátových typov. Predtým, však nad týmito dátami vykoná validáciu [19]. Takýto systém umožňuje plynulú výmenu dát medzi rôznymi vrstvami aplikácie a prípadne aj s úplne inými aplikáciami.

Silnou stránkou DRF je tiež jeho autentifikačný systém a obsluha oprávnení, ktorá je plynulo integrovaná do existujúceho systému vstavaného do Django. Poskytuje niekoľko autentifikačných metód, vrátane overovania na základe session alebo tokenu [20]. Rovnako tak zaručuje prístup len pre používateľov s určenými oprávneniami [21]. Má vstavaných niekoľko tried oprávnení, ako sú napríklad „IsAuthenticated“, „IsAdminUser“ a podobne. Okrem toho však podporuje aj vytváranie vlastných tried. Kombináciou týchto nástrojov vieme jednoducho chrániť prístup ku nášmu middleware API či webovému rozhraniu.

2.2.4 REST API

Representational State Transfer (REST) je architektonický štýl, ktorý definuje sadu požiadaviek, ktoré by pri vytváraní webových služieb mali byť dodržané. REST API sa riadi týmito požiadavkami a využíva na to existujúce protokoly, typicky HTTP [22]. Metódy HTTP sú bezstavové a používajú sa na vykonávanie CRUD (Create, Retrieve, Update, Delete) operácií, na ktoré slúžia metódy POST, GET, PUT a DELETE, v tomto poradí.

Používanie REST API významne prispieva ku škálovateľnosti a výkonu aplikácie. Tým, že je bezstavové, musí každá webová požiadavka na server obsahovať všetky potrebné dáta pre správne spracovanie takejto požiadavky. Takéto fungovanie zabezpečuje rýchle spracovanie, keďže server si medzi požiadavkami nemusí o používateľovi nič pamätať.

2.2.5 Django Template Language

Django Template Language (DTL) je nástroj určený na tvorbu šablón, čo sú v podstate dynamicky generované HTML súbory [23]. Keďže na strane servera využívame Django a DTL je jeho vstavanou súčasťou, DTL bol prirodzenou voľbou pre tvorbu HTML. Umožňuje nám prácu s premennými, používať podmienky, cykly a ďalšie logické funkcie. Používa syntax, ktorý pripomína Python a je plne integrovaný s ostatnými Django funkciami.

Medzi kľúčové aspekty, ktoré DTL ponúka, je aj dedičnosť. Dovoľuje nám vytvárať tzv. „base“ šablóny. Tie okrem vlastného obsahu disponujú placeholdermi pre obsah,

ktorý môžu ostatné šablóny doplniť. Táto funkcia nám značne urýchli vývoj a zamedzí zbytočnému opakovaniu kódu, čím nám pomáha dodržiavať princíp DRY (Don't Repeat Yourself). DTL tiež obsahuje zabudovanú ochranu proti XSS útokom prostredníctvom HTML Escaping, ktorý konvertuje potenciálne nebezpečné znaky na bezpečné.

Ďalším užitočným bezpečnostným prvkom je použitie tzv. „CSRF cookie“. Tento cookie poskytuje ochranu proti CSRF útokom tým, že vyžaduje jeho odoslanie na server pri každej chránenej požiadavke, ako je napríklad POST, PUT, DELETE a podobne.

2.2.6 JavaScript a jQuery

Pre vývoj moderných webových stránok je už JavaScript neodmysliteľnou súčasťou. Vďaka JavaScriptu sú webové stránky dynamické a umožňujú používateľovi interakciu, ktorá siaha za rámec jednoduchej navigácie na stránke.

Napriek svojmu využitiu je JavaScript v určitých oblastiach pomerne náročný na prácu, najmä pokiaľ ide o manipuláciu s DOM a obsluhu asynchrónnych operácií. Z tohto dôvodu bola vyvinutá knižnica jQuery, ktorá rieši mnohé zo spomenutých problémov. Poskytuje jednoduché metódy na prehľadávanie a manipuláciu DOM, obsluhu udalostí a AJAX metódy [24].

Kombinácia JavaScriptu a jQuery nám umožňuje efektívnejší a jednoduchší vývoj nášho dynamického webového rozhrania.

2.3 Architektonický návrh

Ako vyplýva z konceptuálneho návrhu naše riešenie sa bude riadiť 3-vrstvovým architektonickým modelom [25]. VPN Manager ich bude mať definované ako:

- **Prezentačná vrstva:** Webové rozhranie
- **Aplikačná vrstva:** Middleware server
- **Dátová vrstva:** Pritunl API a databáza

Každá z vrstiev bude zabezpečovať odlišnú časť výslednej aplikácie a navzájom budú komunikovať len prostredníctvom REST API. Takéto rozdelenie vytvára logickú separáciu, ktorá zvyšuje bezpečnosť a umožňuje škálovanie jednotlivých vrstiev nezávisle od ostatných.

2.3.1 Prezentačná vrstva

Prezentačná vrstva zodpovedá za UI a UX, a teda je to vrstva, s ktorou koncový používateľ priamo interaguje. Primárnym cieľom tejto vrstvy je zobrazovanie informácií a zbieranie dát od používateľa.

Vrstva by mala obsahovať dve hlavné stránky. Prvá stránka by mala slúžiť na obsluhu organizácií a ich klientov, pričom druhá stránka je určená pre servery spolu s priradenými organizáciami a routes. Obe stránky musia byť prístupné len pre autentifikovaných používateľov.

Každá zo stránok by mala poskytnúť funkcie na správu jej príslušných entít, ktoré sme si zadefinovali v podkapitole 2.1. Sem patrí napríklad formulár pre tvorbu a úpravu klientov či organizácií, filtrácia, stav jednotlivých VPN serverov, tlačidlá na ich ovládanie a podobne.

Na realizáciu takýchto akcií musí prezentačná vrstva vedieť komunikovať s API endpointami aplikačnej vrstvy, prostredníctvom štruktúrovaných HTTP požiadaviek. Využijeme na to AJAX metódy, ktoré zabezpečia asynchrónne posielanie a prijímanie dát z middleware servera.

Nutné je tiež zabezpečiť aktualizáciu informácií o jednotlivých entitách v prípade ich zmeny. Táto zmena môže pochádzať priamo od používateľa našej aplikácie, ale musí byť schopná reagovať na všetky zmeny, ktoré sa udejú na Pritunl servery. Pokiaľ niekto napríklad pracuje so vstavaným Pritunl rozhraním alebo iba s REST API nášho middleware servera, musia sa aktualizovať entity aj na našom webovom rozhraní v čo najkratšom čase.

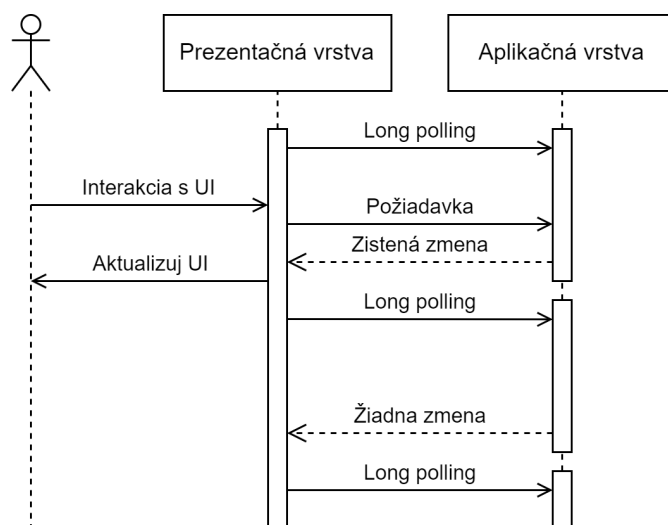
Pre dosiahnutie takejto funkcionality využijeme long polling [26] na middleware server (obr. 2). Ten nás bude informovať ak nastane na Pritunl servery nejaká zmena a ktorej entity sa to týka. Posledná časť je dôležitá, pretože ak by sme nevedeli, ktorá entita bola zmenená, museli by sme pri každej zmene obnoviť všetky dáta zo servera. Takáto operácia by bola časovo náročná, čo by výrazne zhoršilo svižnosť aplikácie a tým aj celkový používateľský dojem.

2.3.2 Aplikačná vrstva

Aplikačná vrstva definuje operácie, ktoré sa budú s dátami vykonávať. Ide o centrálnu vrstvu, ktorá pôsobí ako most medzi prezentačnou a dátovou vrstvou.

Táto vrstva sa má skladať z dvoch systémov. Prvý má zabezpečiť vytvorenie vlastného REST API, ktoré bude na pozadí komunikovať s Pritunl API. Druhý má poskytovať autentifikáciu a autorizáciu pre vytvorené API endpointy a stránky prezentačnej vrstvy.

Naše REST API má implementovať všetky endpointy pre zabezpečenie plnohodnotnej práce so službou Pritunl. Musia dodržiavať odporúčané REST konvencie pre názvy a vykonávať validáciu dát. Pre každú entitu má byť definovaný vlastný logický modul. Ten bude obsahovať triedy, ktoré budú vykonávať potrebnú validáciu a prípadnú serializáciu prijatých JSON dát pre jednotlivé endpointy. V prípade zachytenia chyby, napríklad kvôli



Obr. 2: Sekvenčný diagram návrhu pre detekciu zmien pomocou long polling

chýbajúcim dátam alebo nesprávnemu dátovému typu, je žiadané používateľa informovať HTTP stavovým kódom spolu s vhodným chybovým hlásením.

Je potrebné vytvoriť generickú metódu, ktorú budú endpointy používať na komunikáciu s Pritunl API. Táto metóda musí byť tiež schopná odchytať prípadné chyby, ktoré Pritunl vygeneruje a informovať o nich používateľa.

Na autentifikáciu využijeme metódu na základe sessions, ktorá je vstavanou súčasťou Django a doplníme ju o nástroje z DRF. Náš systém musí poskytovať prihlasovaciu službu, ktorá bude schopná overiť identity používateľov a umožniť im bezpečný prístup ku službám našej aplikácie.

Pokiaľ používateľ nemá rolu admin, musí mu byť zabránený prístupu ku nebezpečným metódam⁹. DRF štandardne takúto triedu oprávnení neponúka, je teda nutné si vytvoriť vlastnú, ktorá obsiahne takúto funkcionality.

2.3.3 Dátová vrstva

Dátová vrstva je zodpovedná za správu a uchovávanie dát. Niekedy sa jej hovorí aj databázová vrstva, keďže zvyčajne pozostáva z databázového systému ako je napríklad MySQL, PostgreSQL a iné.

V našom prípade, keďže nepracujeme s dátami Pritunl priamo ale iba prostredníctvom poskytnutých API endpointov, budeme považovať Pritunl API za súčasť našej dátovej vrstvy. Naša aplikácia bude však využívať vlastný model používateľa a ten je takisto nutné uložiť do databázy. Na toto nám už len Pritunl API stačiť nebude.

⁹ako nebezpečné sa označujú metódy, ktoré menia stav servera (POST, PUT, DELETE)

Pritunl ukladá všetky dáta do databázy MongoDB, ktorá beží popri Pritunl serveri. Toto môžeme využiť tým, že pripojíme našu aplikačnú vrstvu priamo k tej istej databáze a budeme ukladať našich používateľov tam. Takto sa nám podarí ušetriť zdroje, ktoré by sme inak museli použiť na prevádzku ďalšej databázy.

3 Implementácia riešenia

Na základe pripraveného architektonického návrhu a s pomocou predstavených technológií sme implementovali softvérové riešenie, ktoré ponúka nástroje na prácu so službou Pritunl a rozširuje ju o funkcionality podľa požiadaviek definovaných v podkapitole 2.1.

V tejto kapitole si postupne rozoberieme jednotlivé komponenty, ktoré tvoria našu aplikáciu VPN Manager.

3.1 Komunikácia s Pritunl

Pre prácu s Pritunl API sú potrebné autorizačné API Token a API Secret. Podrobný postup, ktorý popisuje, kde ich je možné nájsť a ako ich použiť, je detailne rozobraný v prílohe B. V čase písania tejto práce je pre prístup ku Pritunl API potrebná najvyššia licencia Enterprise [27].

Pred implementáciou našich API sme museli vytvoriť metódu, ktorú budú využívať na komunikáciu s Pritunl. Pomohli sme si poskytnutou funkciou z Pritunl dokumentácie [27], ktorú sme doplnili o odchytyvanie výnimiek a ich spracovanie:

```
def auth_request(
    method, path, headers=None, data=None, raise_err=False
) -> requests.Response:
    # ... poslanie HTTP požiadavky na Pritunl API endpoint ...
    if raise_err:
        try:
            response.raise_for_status()
        except requests.HTTPError:
            try:
                err_dict = json.loads(response.text)
            except json.JSONDecodeError:
                raise PritunlAPIException(
                    detail=response.text, status_code=response.status_code
                )
            else:
                raise PritunlAPIException(
                    err_dict.get('error_msg', None),
                    err_dict.get('error', None),
                    response.status_code)
    return response
```

Bežné chyby, ktorých sa môže používateľ pri používaní nášho webového rozhrania dopustiť sú zachytené a vyriešené ešte počas validácie na middleware servery. Nie je však možné odhadnúť alebo predísť nečakaným zlyhaniam na strane Pritunl servera. Navyše výnimky, ktoré Pritunl generuje nemajú jednotnú štruktúru. Z tohto dôvodu sme vytvorili vlastnú triedu výnimky `PritunlAPIException`, ktorá nečakaným chybovým hláseniam dodá potrebnú štruktúru pre správny výpis na webovom rozhraní.

3.2 API špecifikácia a realizácia

V tejto podkapitole si rozoberieme jednotlivé API nášho middleware servera. Pre prehľad ich budeme zapisovať do tabuliek, ktoré poskytnú náhľad na štruktúru HTTP požiadaviek a odpovedí. Pritunl dokumentácia neobsahuje žiadny zoznam ani popis ponúkaných API endpointov, namiesto toho na ich dohľadanie odporúčajú použiť Chrome Developer Tools [27]. Budeme preto uvádzať aj Pritunl API endpointy, ktoré sú na pozadí volané.

Kvôli veľkému množstvu API v našej aplikácii sa zameriame iba na vybrané, ktoré sa využívajú najčastejšie. Pre ušetrenie miesta v tabuľkách sme neuvádzali celú URL ale iba konkrétne endpointy. Základná URL je vždy adresa servera, na ktorom je konkrétna služba spustená.

3.2.1 Vytvorenie organizácie

Pre vytvorenie organizácie je potrebný len jeden parameter, názov novej organizácie. Tento parameter odosielame v tele požiadavky, formátovanej ako JSON. Vzhľadom na to, že ide o chránenú metódu POST, je v hlavičke HTTP požiadavky okrem parametra `sessionid` potrebné zahrnúť aj `csrftoken`. V prípade úspešného vytvorenia, je odpoveďou objekt novej organizácie. HTTP požiadavka a príklad tela požiadavky je detailne definovaný v tabuľke 1.

3.2.2 Získanie zoznamu organizácií

Potrebovali sme spôsob ako získať prehľad o všetkých existujúcich organizáciách. Dané API poskytuje zoznam organizácií vo formáte JSON. Parametre, ktoré každá organizácia obsahuje sú: ID, meno, počet klientov, atď. Keďže sa jedná o bezpečnú metódu GET, v hlavičke požiadavky je možné vypustiť parameter `csrftoken`. HTTP požiadavka je podrobnejšie definovaná v tabuľke 2.

HTTP metóda	POST
Endpoint	/api/organizations/create
Pritunl endpoint	/organization
HTTP hlavičky	Content-Type: application/json Cookie: csrftoken=<csrftoken> ¹⁰ sessionid=<sessionid>
Telo požiadavky	{ "name": "org1" }
Odpoveď	Stavový kód 201 Created Telo Vytvorená organizácia

Tabuľka 1: API špecifikácia: Vytvorenie organizácie

HTTP metóda	GET
Endpoint	/api/organizations
Pritunl endpoint	/organization
HTTP hlavička	Cookie: sessionid=<sessionid>
Odpoveď	Stavový kód 200 OK Telo Zoznam organizácií

Tabuľka 2: API špecifikácia: Získanie zoznamu organizácií

3.2.3 Vytvorenie klienta

Existencia klienta je neoddeliteľne spätá s prítomnosťou organizácie. Preto je pri procese vytvárania klienta nevyhnutné špecifikovať jeho príslušnosť k určitej organizácii. To dosahujeme prostredníctvom uvedenia ID danej organizácie ako path parametra. Zvyšné parametre sú uvádzané klasicky v tele požiadavky. Povinným parametrom pre vytvorenie klienta je len jeho meno. Možno je ale zadať aj ďalšie údaje, ako napríklad email a názvy tzv. skupín, do ktorých klient patrí. HTTP hlavičky sú rovnaké ako tie, ktoré sú použité pri vytváraní organizácie, keďže sa v oboch prípadoch jedná o POST metódu. HTTP požiadavka a príklad jej tela sú podrobnejšie špecifikované v tabuľke 3.

3.2.4 Získanie zoznamu klientov

Podobne ako pre organizácie tak aj pre klientov sme potrebovali spôsob ako získať ich zoznam. Rozdiel je v tom, že nevieme získať úplný zoznam všetkých klientov, ale

¹⁰v tomto texte, bude symbol <> použitý ako zástupný znak pre konkrétnu premennú

HTTP metóda	POST
Endpoint	/api/organizations/<org-id>/users/create
Pritunl endpoint	/user/<org-id>
HTTP hlavičky	Content-Type: application/json Cookie: csrftoken=<csrftoken> sessionid=<sessionid>
Telo požiadavky	{ "name": "user1" "email": "user1@example.com" "groups": ["group1", "group2"] }
Odpoveď	Stavový kód 201 Created Telo Vytvorený klient

Tabuľka 3: API špecifikácia: Vytvorenie klienta

len prehľad klientov spadajúcich pod konkrétnu organizáciu. Tá je dodaná vo forme ID organizácie ako path parameter. Odpoveďou je zoznam klientov, ktorý zahŕňa už spomenuté parametre: ID, meno, email a skupiny, ale aj ďalšie informácie, ako je status pripojenia, čas posledného pripojenia, klientove zariadenia a podobne. Detailnejšie informácie o HTTP požiadavke sú definované v tabuľke 4.

HTTP metóda	GET
Endpoint	/api/organizations/<org-id>/users
Pritunl endpoint	/user/<org-id>
HTTP hlavičky	Cookie: sessionid=<sessionid>
Odpoveď	Stavový kód 200 OK Telo Zoznam klientov

Tabuľka 4: API špecifikácia: Získanie zoznamu klientov

3.2.5 Vytvorenie servera

Vytvorenie servera je kľúčový proces, keďže každý z nich predstavuje nezávislý OpenVPN server. Výber správnych nastavení počas jeho vytvárania je obzvlášť dôležitý, pretože hoci dodatočné úpravy sú možné, môžu si vyžadovať opätovné stiahnutie klientskych profilov kvôli potrebe spárovania klienta so serverom.

V tele požiadavky sa posiela viacero parametrov ako: meno, virtuálna sieť, skupiny,

číslo portu, protokol, sieťový režim, šifrovací algoritmus a hašovací algoritmus. Až na skupiny sú všetky parametre povinné.

Virtuálna sieť smie obsahovať iba validnú IPv4 súkromnú sieťovú adresu spolu s maskou podsiete v CIDR formáte. Súkromné siete sa delia na tri kategórie a to trieda A, trieda B a trieda C. Výber adresy zo správnej triedy je dôležitý, keďže sa od nej odvíja maximálny počet klientov, ktorí sa môžu ku serveru pripojiť. Rozsah adries je nasledovný [28]:

- trieda A: 10.0.0.0 - 10.255.255.255, maximálny počet adries: 16 777 216
- trieda B: 172.16.0.0 - 172.31.255.255, maximálny počet adries: 1 048 576
- trieda C: 192.168.0.0 - 192.168.255.255, maximálny počet adries: 65 536

Skupiny sa využívajú na obmedzenie prístupu k serveru len pre určitých klientov z organizácie. V prípade zadania skupín sa porovnávajú s tými, ktoré sme definovali pri vytváraní klienta. Prístup k serveru je následne povolený iba pre tých klientov, ktorí sú členmi tých istých skupín. Ak server nemá definované žiadne skupiny, potom je automaticky povolený prístup pre všetkých klientov z danej organizácie.

Číslom portu zadefinujeme, na ktorom porte má vytvorený VPN server počúvať pre prichádzajúce VPN pripojenia. Povolený rozsah je od 1 do 65535. V praxi sa však často využívajú porty od 1024 do 49151. Dôvodom je, že porty od 1 do 1023 sú väčšinou rezervované pre štandardné a dobre známe služby. Napríklad, HTTP na porte 80, HTTPS na porte 443, FTP na porte 21 a podobne. Na druhej strane, porty od 49152 do 65535 sú definované ako dynamické alebo súkromné porty, ktoré sú často používané na dočasné, automaticky pridelené služby.

Protokolom určíme metódu pre posielanie a prijímanie dát cez sieť. Keďže na pozadí beží OpenVPN na výber máme medzi protokolmi TCP a UDP.

Sieťový režim určuje akou metódou má server vytvoriť VPN pripojenie. Na výber máme medzi metódami tunel a most. Spôsob akým funguje tunel sme si už dôkladne rozobrali v podkapitole 1.1. Naproti tomu metóda most spôsobí, že klienti sa javia, ako keby boli priamo pripojení k lokálnej sieti. Takáto metóda môže byť užitočná ak potrebujeme aby mali klienti prístup ku sieťovým službám, ktoré vyžadujú priamy prístup ku sieti. To môže byť napríklad objavovanie sietí alebo vysielanie správ. Takéto pripojenie je síce rýchlejšie ale zároveň je aj menej bezpečné, keďže prenášané dáta nie sú šifrované [29]. Navyše, Pritunl dokumentácia [30] metódu most neodporúča používať keďže je náročná na výkon a jeho podpora je obmedzená.

Šifrovacie a hašovacie algoritmy a ich využitie sme si takisto už rozobrali v podkapitole 1.1. Spomenieme však dostupné možnosti výberu. Na šifrovanie si môžeme vybrať medzi algoritmami Blowfish o dĺžke 128 a 256 bitov a medzi AES v 128, 192 a 256 bitovom prevedení. Možnosť je takisto nepoužiť žiadne šifrovanie, aj keď to nie je odporúčané. Čo sa týka hašovacích algoritmov, môžeme si vybrať medzi MD5, SHA-1, SHA-256 a SHA-512.

Odpoveďou pri úspešnej požiadavke je objekt vytvoreného servera v JSON formáte. Novo vytvorený server má zároveň automaticky pridanú route 0.0.0.0/0, ktorá spôsobuje, že všetky dáta sú najprv smerované cez VPN server a až potom do ich cieľovej destinácie. Takéto nastavenie smerovania sa nazýva aj „full tunnel“. Celá HTTP požiadavka a príklad jej tela je bližšie popísaný v tabuľke 5.

HTTP metóda	POST
Endpoint	/api/servers/create
Pritunl endpoint	/server
HTTP hlavičky	Content-Type: application/json Cookie: csrftoken=<csrftoken> sessionid=<sessionid>
Telo požiadavky	{ "name": "server1" "network": "192.168.216.0/24" "groups": ["group1", "group2"] "port": 19542 "protocol": tcp "network_mode": "tunnel" "cipher": "aes128" "hash": "sha1" }
Odpoveď	Stavový kód 201 Created Telo Vytvorený server

Tabuľka 5: API špecifikácia: Vytvorenie servera

3.2.6 Získanie zoznamu serverov

Podobne ako pri ostatných entitách, aj v kontexte serverov bolo nevyhnutné zabezpečiť možnosť získania ich zoznamu. Odpoveďou na dané API je teda zoznam serverov, obsahujúci identické parametre, aké sme uviedli pri procese jeho vytvárania. Okrem týchto parametrov

sa v zozname nachádzajú napríklad aj informácie o aktuálnom stave servera, čas vyjadrený v sekundách od posledného spustenia, celkový počet klientov pripojených cez organizácie, počet aktuálne pripojených klientov atď. Bližšie informácie o HTTP požiadavke sú v tabuľke 6.

HTTP metóda	GET
Endpoint	/api/servers
Pritunl endpoint	/server
HTTP hlavičky	Cookie: sessionid=<sessionid>
Odpoveď	Stavový kód 200 OK
	Telo Zoznam serverov

Tabuľka 6: API špecifikácia: Získanie zoznamu serverov

3.2.7 Ovládanie servera

Ovládanie servera realizujeme prostredníctvom HTTP požiadaviek, ktoré sú detailne popísané v tabuľke 7. Tieto požiadavky nám umožňujú spustiť server v prípade, že je momentálne vypnutý, a zastaviť alebo reštartovať, pokiaľ je spustený. Identifikáciu konkrétneho servera, nad ktorým sa má vykonať ovládanie, zabezpečuje path parameter ID servera. Ako pri metóde POST, tak aj pri metóde PUT je nevyhnutné v hlavičke odoslať `csrftoken`. Po úspešnom vykonaní požiadavky je odpoveďou server s aktualizovaným stavom. Ak bol server spustený, jeho stav sa zmení na online a aktivuje sa interné počítadlo času. Pri zastavení servera sa stav zmení na offline, počítadlo času sa zastaví a jeho hodnota sa nastaví na null. V prípade reštartu servera ostáva stav online, avšak počítadlo času sa resetuje.

HTTP metóda	PUT
Endpoint	/api/servers/<server-id>/[start, stop, restart] ¹¹
Pritunl endpoint	/server/<server-id>/operation/[start, stop, restart]
HTTP hlavičky	Cookie: csrftoken=<csrftoken> sessionid=<sessionid>
Odpoveď	Stavový kód 200 OK
	Telo Ovládaný server

Tabuľka 7: API špecifikácia: Ovládanie servera

¹¹ide o tri samostatné endpointy, kvôli podobnosti štruktúr sme ich však zlúčili

3.2.8 Vytvorenie route pre server

V prípade, že nechceme smerovanie všetkých dát cez server, ale len dát určených pre konkrétnu adresu, je nevyhnutné vytvoriť route pre danú adresu. Takto vytvárame koncept známy ako „split tunneling“. Tento prístup umožňuje, aby dáta smerujúce na inú adresu, než je definovaná v routes, neboli smerované cez VPN server, ale prostredníctvom verejnej siete.

Proces vytvorenia route zahŕňa špecifikáciu ID servera ako path parameter. V tele požiadavky sa potom posiela jediný povinný parameter - adresa siete a nepovinný parameter - komentár. Adresa siete môže byť uvedená buď vo formáte CIDR alebo DNS. Preklad adresy z formátu DNS na IP adresu realizujeme pomocou Python knižnice `socket`. V prípade, že je rozpoznaná adresa vo formáte DNS a súčasne nebol uvedený žiaden komentár, tak sa ako komentár použije pôvodný názov domény. Takéto riešenie bolo zvolené s cieľom zlepšenia UX, keďže výpis routes sa realizuje výlučne vo formáte CIDR. Pre používateľa by potom mohlo byť ťažké identifikovať, na ktorú doménu sa konkrétna adresa vzťahuje, pokiaľ by zabudol pridať komentár. Ukážka funkcií:

```
def is_dns_name(addr):
    try:
        socket.gethostbyname(addr)
        return True
    except socket.gaierror:
        return False

def create_route(*, server_id: str, **kwargs) -> Dict:
    # ...
    network = kwargs.get("network", None)
    if network and is_dns_name(network):
        kwargs["network"] = socket.gethostbyname(network)
        comment = kwargs.get("comment", None)
        if comment is None:
            kwargs["comment"] = network
    # ...
```

Pred pokračovaním v procese vytvárania route je nevyhnutné najprv skontrolovať stav servera, keďže so serverom nie je možné pracovať pokiaľ je spustený. V prípade, že je spustený, je potrebné ho najprv zastaviť. Po zastavení servera je možné pokračovať vo vytváraní route. V rámci zlepšenia UX, ak bol server zastavený, musí byť po dokončení procesu znova spustený. Ukážka funkcií:

```
def stop_server_if_online_and_verify(server_id) -> bool:
```

```
    was_online = False
```

```
    server = get_server_by_id(server_id)
```

```
    if server["status"] != "offline":
```

```
        server = stop_server(server_id)
```

```
        if server["status"] != "offline":
```

```
            raise PritunlAPIException(
```

```
                detail="Failed to stop server.",
```

```
                code="server_stop_failed",
```

```
                status_code=500,
```

```
            )
```

```
        was_online = True
```

```
    return was_online
```

```
def start_server_if_offline_and_verify(server_id):
```

```
    server = get_server_by_id(server_id)
```

```
    if server["status"] != "online":
```

```
        server = start_server(server_id)
```

```
        if server["status"] != "online":
```

```
            raise PritunlAPIException(
```

```
                detail="Failed to start server.",
```

```
                code="server_start_failed",
```

```
                status_code=500,
```

```
            )
```

```
def create_route(*, server_id: str, **kwargs) -> Dict:
```

```
    was_online = stop_server_if_online_and_verify(server_id)
```

```
    # ...
```

```
    if was_online:
```

```
        start_server_if_offline_and_verify(server_id)
```

```
    return response.json()
```

Odpoveďou v prípade úspešného vytvorenia je nový objekt route v JSON formáte. Podrobnejšie je HTTP požiadavka zhrnutá v tabuľke 8.

3.2.9 Pripojenie organizácie ku serveru

Pre umožnenie pripojenia klientov na VPN server je potrebné aby bola ich príslušná organizácia najprv asociovaná s daným serverom. Server a organizácia, ktoré sa majú

HTTP metóda	POST
Endpoint	/api/servers/<server-id>/routes/create
Pritunl endpointy	/server/<server-id>/route /server/<server-id>/operation/[stop, start]
Telo požiadavky	{ "network": "example.com" "comment": "example comment" }
HTTP hlavičky	Content-Type: application/json Cookie: csrftoken=<csrftoken> sessionid=<sessionid>
Odpoveď	Stavový kód 201 Created Telo Vytvorený route

Tabuľka 8: API špecifikácia: Vytvorenie route pre server

navzájom prepojiť sú určené podľa path parametrov ID servera a ID organizácie.

Podobne ako v prípade vytvárania route, aj tento proces vyžaduje, aby bol server zastavený. Využijeme na to funkcie `stop_server_if_online_and_verify` a `start_server_if_offline_and_verify`, ktoré sme predstavili v predchádzajúcej sekcii 3.2.8. Detailnejšia špecifikácia HTTP požiadavky je uvedená v tabuľke 9.

HTTP metóda	PUT
Endpoint	/api/servers/<server-id>/organizations/<org-id>/attach
Pritunl endpointy	/server/<server-id>/organization/<org-id> /server/<server-id>/operation/[stop, start]
HTTP hlavičky	Cookie: csrftoken=<csrftoken> sessionid=<sessionid>
Odpoveď	Stavový kód 200 OK Telo Meno a ID pripojenej organizácie a ID servera

Tabuľka 9: API špecifikácia: Pripojenie organizácie ku serveru

3.3 Autentifikácia a autorizácia

Ako sme už načrtli v sekcii 2.3.2, autentifikáciu implementujeme s využitím sessions. Skombinovaním Django a DRF bolo vytvorenie takéhoto typu autentifikácie veľmi

jednoduché. Do konfiguračných nastavení projektu stačilo pridať:

```
"DEFAULT_AUTHENTICATION_CLASSES": [  
    "rest_framework.authentication.SessionAuthentication",  
]
```

Táto trieda umožňuje automatické spravovanie sessions na každom API a View. Bez takéhoto nastavenia by bolo nutné v každom endpointe, kde sa má použiť autentifikácia, explicitne nastaviť `authentication_classes` na `SessionAuthentication`.

Session autentifikácia funguje tak, že po úspešnom prihlásení sa pre používateľa vygeneruje session, ktorá sa uloží do databázy. Používateľovi sa priradí jedinečné session ID, ktoré sa uchováva vo forme cookie na strane používateľa. Pri každej požiadavke, ktorá sa odošle na server, sa poslané session ID porovná s tým, ktoré je uložené v databáze, čím sa overí totožnosť používateľa.

Treba zdôrazniť, že proces overenia ako taký neudeluje a ani neodmieta prístup prichádzajúcim požiadavkám. Potvrďuje iba platnosť údajov, s ktorými bola požiadavka odoslaná [20]. Na kontrolu prístupov je potrebné nastaviť príslušné `permission_classes`.

Pred implementáciou kontroly prístupov sme si ale museli, podľa odporúčaní z Django dokumentácie [31], vytvoriť vlastný model používateľa. Výhodou takéhoto riešenia je možnosť v budúcnosti model rozširovať o ďalšie polia či metódy. Zatiaľ nám ale postačí jednoduchý model, ktorý obsahuje polia ako: prihlasovacie meno, heslo a boolean - `is_staff`, ktorý určí rolu používateľa.

V súlade s našimi požiadavkami sme stanovili dve úrovne oprávnení - admin a read-only. Admin musí mať neobmedzený prístup ku všetkým API endpointom, zatiaľ čo read-only má mať prístup iba ku endpointom, ktoré používajú bezpečné metódy. Na dosiahnutie takejto funkcionality sme vytvorili vlastnú triedu oprávnení:

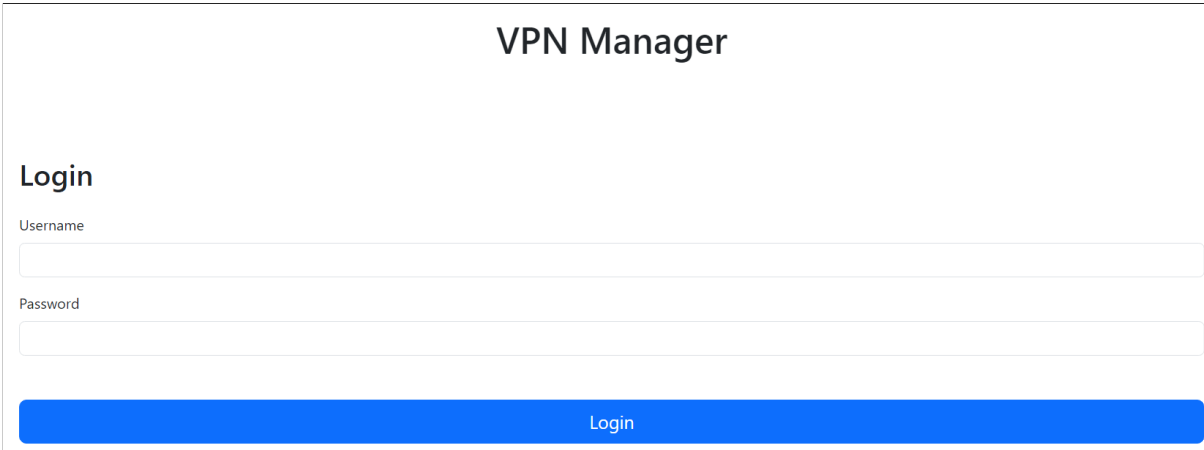
```
class IsAdminUserOrReadOnly(BasePermission):  
    def has_permission(self, request, view):  
        return bool(  
            request.user and request.user.is_authenticated and  
            (request.method in SAFE_METHODS or request.user.is_staff)  
        )
```

Túto triedu, podobne ako pri autentifikácii, pridáme do konfiguračných nastavení projektu:

```
"DEFAULT_PERMISSION_CLASSES": [  
    "accounts.permissions.IsAdminUserOrReadOnly"  
]
```

Týmto krokom dosiahneme automatickú kontrolu oprávnení aj bez nutnosti explicitne definovať `permission_classes` pre každý endpoint samostatne.

Na pridávanie nových používateľov sme sa rozhodli využiť vstavané rozhranie Django Admin. Prístup k tomuto rozhraniu má iba `superuser`, ktorý je vytvorený prostredníctvom príkazového riadka na servery aplikácie. Môže tu vykonávať všetky CRUD operácie na modeli používateľa, vrátane modifikácie atribútu `is_staff`. Na autentifikáciu sme vytvorili jednoduché prihlasovacie rozhranie (obr. 3), ktoré je dostupné z endpointu `/login`. Slúži ako vstupný bod do ostatných častí nášho webového rozhrania.



The image shows a web form titled "VPN Manager". Under the title is the heading "Login". Below this heading are two text input fields. The first field is labeled "Username" and the second is labeled "Password". At the bottom of the form is a prominent blue button with the text "Login" in white.

Obr. 3: VPN Manager: Prihlasovacie rozhranie

3.4 Pritunl eventy

Predtým, ako prejdeme k rozboru implementácie jednotlivých stránok, je potrebné si objasniť mechanizmus na automatickú aktualizáciu entít. Pritunl implementuje systém, ktorý upozorňuje o všetkých zmenách na serveri prostredníctvom generovania tzv. „eventov“.

Každá akcia vyvolá jej zodpovedajúci event, napríklad vytvorenie novej organizácie spustí event typu `organizations_updated` a podobne. Rovnako ako v prípade Pritunl API endpointov, zoznam eventov a ani akcie, ktoré ich generujú, nie sú zverejnené. Museli sme preto použiť Chrome Developer Tools počas interakcie so vstavaným Pritunl webovým rozhraním aby sme odhalili štruktúru a logiku vytvárania eventov. Podrobný prehľad jednotlivých eventov, príslušných entít a akcií, ktoré ich spustia, je detailne zobrazený v tabuľke 10.

Eventy typu `users_updated`, `server_routes_updated` a `server_organizations_updated` obsahujú aj atribút `resource_id`. Toto ID patrí ich rodičovskej entite. V prípade klientov to znamená ID organizácie, v prípade routes a pripojených organizácií

Typ eventu	Entity a akcie
organizations_updated	organization => [create, update, delete] user => [create, delete]
users_updated	user => [create, update, delete] organization => [attach, detach] server => [update, start, restart, stop]
servers_updated	server => [create, update, delete, start, stop, restart] organization => [delete, attach, detach] user => [create]
server_routes_updated	route => [create, update, delete] server => [update] organization => [attach, detach]
server_organizations_updated	organization => [delete, attach, detach]

Tabuľka 10: Prehľad Pritunl eventov

zas ID servera. Vďaka tejto informácii nám postačí získavať z middleware servera iba potrebné dáta entít patriacich pod rodiča s daným ID, čím výrazne optimalizujeme proces ich aktualizácie.

Tak ako sme už uviedli v sekcii 2.3.1, k získavaniu eventov sa využije metóda long polling na príslušnom endpointe middleware servera, ktorý nám sprostredkuje vygenerované Pritunl eventy. Ako vyplýva z tabuľky 10, niektoré akcie môžu spustiť viaceré eventy súčasne. Z toho dôvodu bolo nevyhnutné vytvoriť funkciu, ktorá ich klasifikuje a vykoná požadované aktualizácie. Každý event má navyše pridelené unikátne ID, ktoré je nutné sledovať a posilať spolu s požiadavkou na server. Pre správne fungovanie totiž Pritunl očakáva ID posledného vygenerovaného eventu. Ukážka funkcie:

```
function parseEvents(events) {
  let lastEventID = null;
  events.forEach(event => {
    lastEventID = event.id;
    switch (event.type) {
      case "servers_updated":
        ifExistsCall ("rebuildServers");
        break;
      case "organizations_updated":
        ifExistsCall ("rebuildOrgs");
```



```

        ifExistsCall ("refreshAttachOrgModal");
        break;
    case "server_routes_updated":
        ifExistsCall ("rebuildRoutesByServerID", event.resource_id);
        break;
    case "server_organizations_updated":
        ifExistsCall ("rebuildAttachedOrgsByServerID", event.resource_id);
        break;
    case "users_updated":
        ifExistsCall ("rebuildUsersByOrgID", event.resource_id);
        break;
    default:
        console.log("Unknown event type: " + event.type);
        break;
    }

});
return lastEventID;
}

```

Funkcia `ifExistsCall()` overí, či argument mena funkcie v kontexte programu existuje. Ak áno, zavolá ju aj s poskytnutým argumentom. Takéto riešenie sme implementovali pretože viaceré eventy a funkcie s nimi spojené sú zdieľané naprieč stránkami. Takto sa nám podarí odstrániť zbytočnú redundanciu, ktorá by inak nastala.

3.5 Webové rozhranie

Pri implementácii stránok nášho webového rozhrania sme sa usilovali o zachovanie dizajnu podobnému tomu, ktorý používa aj Pritunl vo svojom vstavanom webovom rozhraní. Snažili sme sa udržať konzistentné názvy prvkov a ich rozmiestenie, pričom sme brali do úvahy aj nami doplnené prvky. Cieľom tohto rozhodnutia bolo uľahčiť a urýchliť adaptáciu na naše rozhranie pre používateľov, ktorí sú už zvyknutí na používanie webového rozhrania Pritunl. Na každej stránke je prítomný navigačný panel umožňujúci prechod medzi jednotlivými stránkami a tlačidlo na odhlásenie.

Často používanou funkciou je `rebuildElements()`. Jedná sa o generickú funkciu, ktorá udržiava entity synchronizované s databázou. Je zodpovedná za ich vytváranie, upravovanie, mazanie a dopĺňanie informácií, ako napríklad ID. Ako parametre prijíma dáta o entitách, ktoré sú získané z API volaní, a množstvo doplňujúcich nastavení. Medzi

tieto patria aj tzv. „callback“ funkcie, ktoré sa vykonávajú po dokončení synchronizácie a ktoré nám umožňujú rozšíriť funkcionality špecifickú pre danú entitu.

3.5.1 Klienti a organizácie

Po úspešnom prihlásení je používateľ presmerovaný na stránku *Users and Organizations*, ktorá je určená na správu klientov a organizácií. Táto stránka obsahuje menu so štyrmi tlačidlami na realizáciu rôznych akcií, disponuje zoznamami existujúcich organizácií a ich klientov a zároveň poskytuje možnosť filtrovania týchto zoznamov.

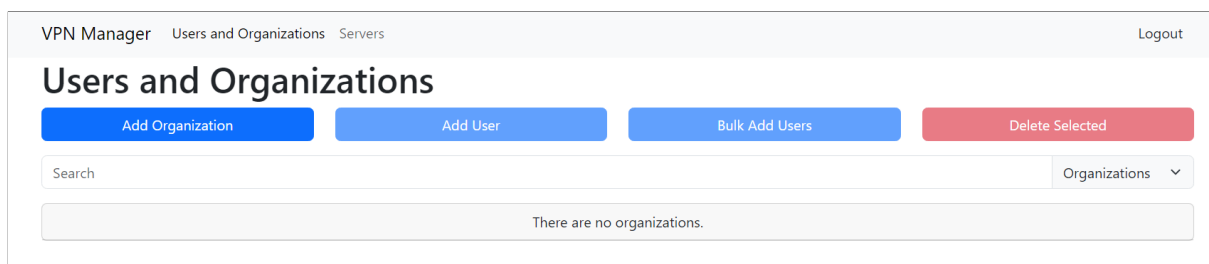
Po prvotnom načítaní stránky je vykonaná kontrola, ktorá vyhodnotí či má prihlásený používateľ rolu admin alebo read-only. Informáciu o role používateľa vkladá middleware server v podobe premennej `is_readonly` do kontextu Django šablóny, ktorá ju potom umiestni na stránku. Táto informácia je následne prečítaná a uložená. V prípade, že má používateľ rolu read-only, je mu umožnené len prezerať existujúce entity a filtrovať ich. Zvyšné funkcie stránky sú pre neho neaktívne. Takéto riešenie je bezpečné, keďže na stránku má rola čisto vizuálny vplyv a skutočná kontrola oprávnení sa vykonáva až na úrovni servera.

Po získaní role používateľa je spustená funkcia `rebuildAllData()`, ktorá je zodpovedná za prvotné naplnenie zoznamu organizácií a ich klientov. Tá ako prvé volá funkciu `rebuildOrgs()`, ktorá odošle požiadavku na API pre získanie zoznamu organizácií. Prijaté dáta potom vloží do funkcie `rebuildElements()`, ktorá zoznam organizácií aktualizuje. Volaný je aj callback `configureMenuBtns()`, ktorý sa postará o obnovu tlačidiel v menu.

Po úspešnej aktualizácii organizácií je pre každú z nich volaná funkcia `rebuildUsersByOrgID()`, ktorá ako parameter prijíma ID danej organizácie. Táto funkcia zašle požiadavku na API pre získanie zoznamu klientov príslušnej organizácie. Tieto dáta následne vkladá do funkcie `rebuildElements()`, ktorá aktualizuje zoznam klientov.

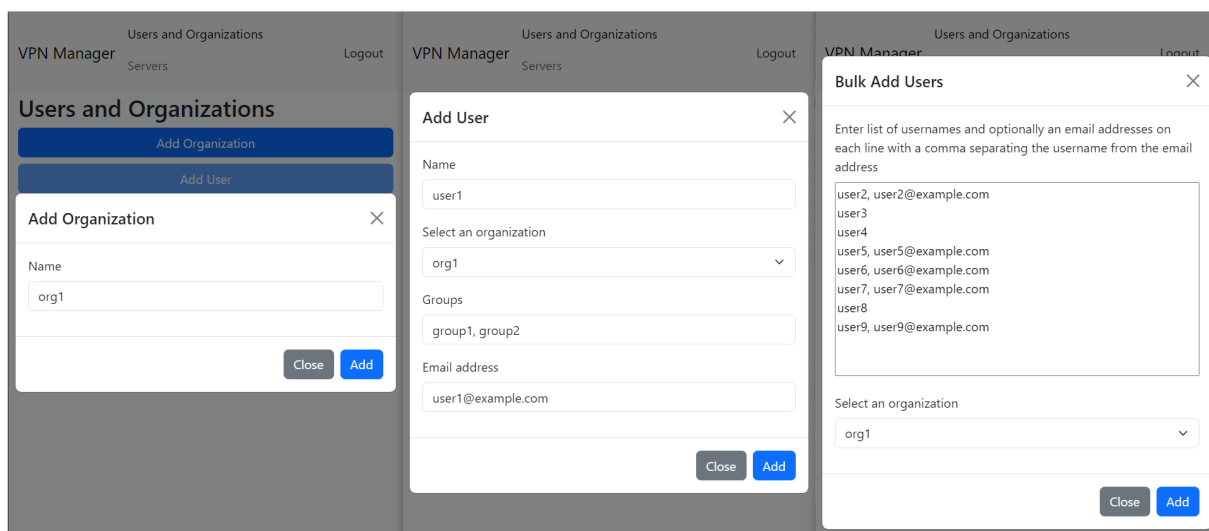
Nakoniec je spustená funkcia `listenForEvents()`, ktorá zahájí long polling na middleware server a vkladá eventy do funkcie `parseEvents()`. Tento proces sme si bližšie popísali v podkapitole 3.4.

V prípade, že nie sú vytvorené žiadne organizácie, je zoznam organizácií prázdny a jediné aktívne tlačidlo je *Add Organization* (obr. 4). Po jeho stlačení sa zobrazí formulár na vytvorenie organizácie (obr. 5 vľavo). Po zadaní názvu a potvrdení formulára je zavolaná funkcia `cleanFormData()`. Tá z dát odstráni nevyplnené polia a zbytočné medzery a transformuje ich do objektu s požadovanou štruktúrou. Následne sú takto upravené dáta vložené do funkcie `createOrgApi()`, ktorá odošle požiadavku na API pre vytvorenie organizácie. Ak prebehla táto akcia úspešne, vygenerovaný event spôsobí zavolanie už spomínaného `rebuildOrgs()`.



Obr. 4: VPN Manager: Klienti a organizácie - žiadne vytvorené organizácie

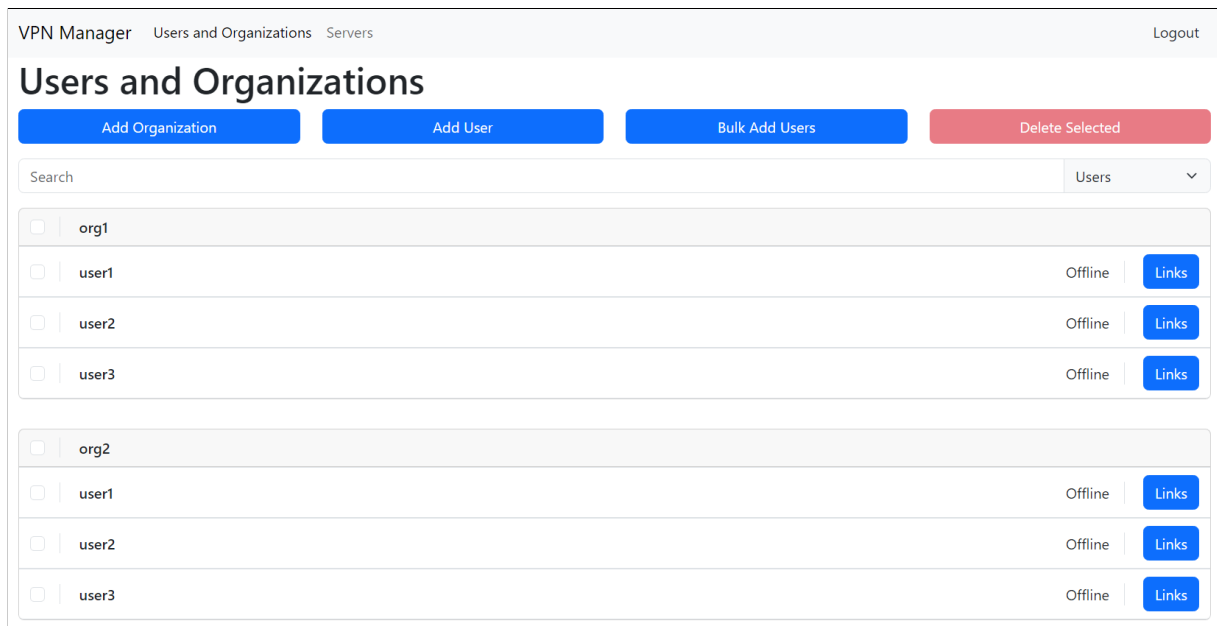
Ak existuje aspoň jedna organizácia, tlačidlá *Add User* a *Bulk Add Users* sú aktívne. Obidve tieto tlačidlá zobrazia formulár určený na pridávanie klientov, avšak s rozdielnou funkcionalitou. Zatiaľ čo *Add User* umožňuje vytvorenie len jedného klienta, *Bulk Add Users* ponúka hromadné vytvorenie viacerých klientov naraz. Formulár na vytvorenie jedného klienta je užívateľsky prívetivejší a ponúka vyššiu mieru kontroly, keďže umožňuje definovanie skupín, do ktorých má klient patriť (obr. 5 stred). Naproti tomu, formulár na vytvorenie viacerých klientov neumožňuje pridať klientom skupiny. Navrhnutý je ale tak, aby zefektívnil hromadné operácie a keďže dáta sú vkladané vo forme voľného textu, je možné požadované dáta vopred pripraviť a vložiť ich do formulára ako celok (obr. 5 vpravo). Ak je to potrebné, je možné takto vytvoreným klientom dodatočne pridať skupiny cez formulár na ich modifikáciu.



Obr. 5: VPN Manager: Klienti a organizácie - vytvorenie organizácie (vľavo), vytvorenie klienta (stred), hromadné vytvorenie klientov (vpravo)

V oboch prípadoch je proces, nasledujúci po potvrdení formulára, podobný tomu, ktorý sme opísali pri vytváraní organizácie. Vyplnené dáta sú najprv funkciou `cleanFormData()`

pripravené do požadovaného stavu, v prípade hromadného vytvorenia sú ešte navyše rozparsované do pola objektov. Takéto dáta sú potom poslané v požiadavke na príslušné API, ktoré je zodpovedné za ich vytvorenie. Ak boli klienti úspešne vytvorený, tak vyvolaný event a jeho `resource_id` spôsobí zavolanie už spomenutej funkcie `rebuildUsersByOrgID()` (obr. 6).



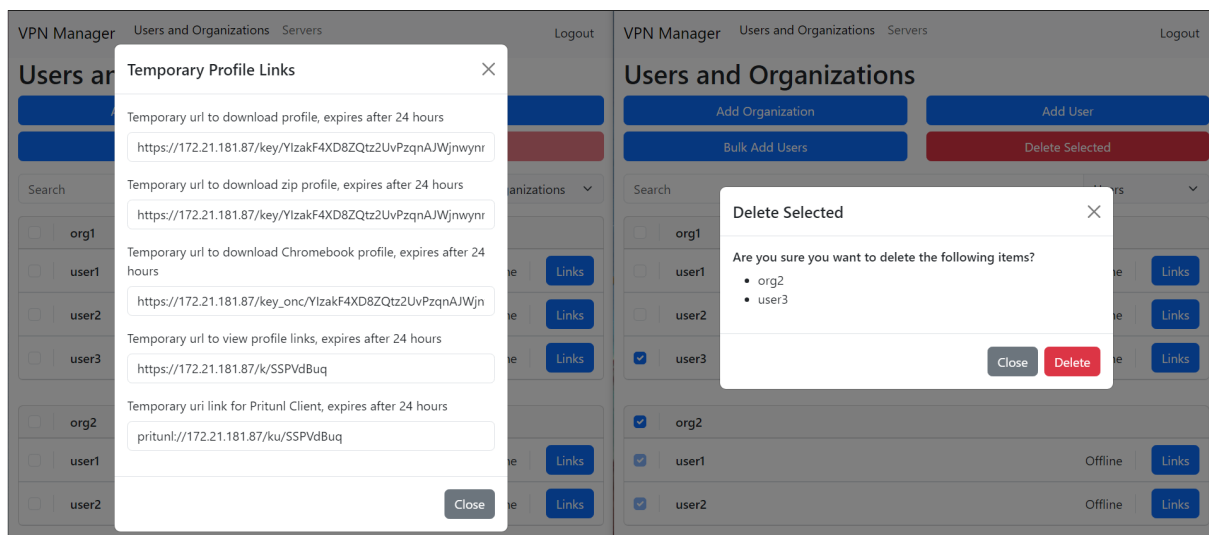
Obr. 6: VPN Manager: Klienti a organizácie - zoznam s organizáciami a ich klientami

Po kliknutí na meno organizácie alebo klienta je otvorený formulár, ktorý je určený na ich modifikáciu. Uvedený formulár je v zásade identický s formulárom, ktorý bol použitý pri ich pôvodnom vytváraní a aj celý proces prebieha podobne. Jediný rozdiel spočíva v charaktere volaných API, ktoré sú špecificky určené na modifikáciu existujúcich entít.

Klienti v zozname tiež obsahujú informáciu o stave ich pripojenia (online/offline) a tlačidlo *Links*. Jeho stlačením sa zavolá funkcia `fetchUserLinksApi()`, ktorá komunikuje s API na generovanie dočasných odkazov na stiahnutie klientských profilov (obr. 7 vľavo). Stiahnuté profily potom môžu klienti použiť na nadviazanie spojenia s VPN serverom pomocou aplikácií ako napríklad Pritunl Client alebo v prípade mobilných zariadení OpenVPN Connect.

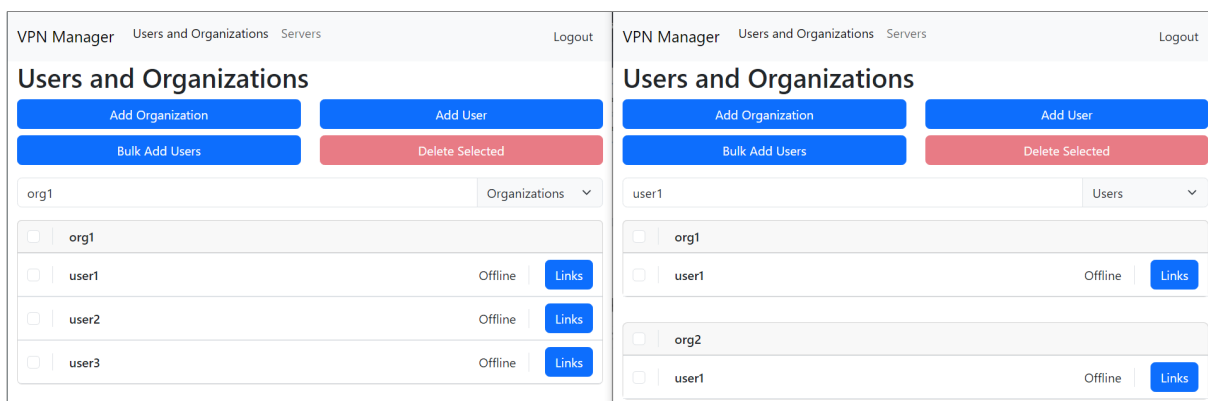
Organizácie ako aj klienti obsahujú zaškrŕavacie políčka slúžiace na označenie entity, ktorá má byť zmazaná. Pokiaľ je vybraná aspoň jedna entita, tlačidlo *Delete Selected* sa aktivuje. Jeho stlačením sa zobrazí potvrdzovacie okno, ktoré zároveň obsahuje výpis entít, určených na odstránenie (obr. 7 vpravo). Po potvrdení výberu, sú zo zoznamu vybraných entít získané ich ID, ktoré tam boli vložené pri ich aktualizácii. Následne sú

získané ID postupne vkladané do funkcií `deleteOrgApi()` alebo `deleteUserApi()`. Tie pošlú požiadavku na API, ktoré zabezpečí odstránenie požadovaných entít. Po úspešnom dokončení nasleduje proces, analogický predchádzajúcim prípadom, ktorý zahŕňa vyvolanie eventu a následnú aktualizáciu dotknutých entít.



Obr. 7: VPN Manager: Klienti a organizácie - prehľad odkazov na stiahnutie klientských profilov (vľavo), odstránenie vybraných entít (vpravo)

Filtrácia zoznamu je ovládaná prostredníctvom vstupného pola, ktoré je umiestnené pod menu a ponúka dva režimy filtrovania - organizácie a klienti. Filtrovanie je realizované funkciou `searchOrgsOrUsers`, ktorá na základe vloženého textu a zvoleného režimu prehľadáva príslušné zoznamy pre výskyt zhody textu s menom entity. Pri prehľadávaní organizácií sú zobrazené len tie organizácie, pri ktorých nastala zhoda, a tiež sú zobrazený všetci jej klienti (obr. 8 vľavo). Analogicky, v prípade prehľadávania klientov sú zobrazený iba klienti, ktorých meno sa zhoduje s poskytnutým textom a zobrazené sú tiež organizácie, ku ktorým patria (obr. 8 vpravo).



Obr. 8: VPN Manager: Klienti a organizácie - filtrovanie organizácií (vľavo), filtrovanie klientov (vpravo)

3.5.2 Servery

Stránka *Servers* slúži na správu serverov, routes a na pripájanie organizácií. Obsahuje menu s piatimi tlačidlami a zoznam serverov, ich routes a pripojených organizácií.

Postup pri načítaní tejto stránky je identický s postupom na stránke *Users and Organizations*. Začína sa získaním a nastavením používateľovej role, po ktorej nasleduje volanie funkcií `rebuildAllData()` a `listenForEvents()`.

Funkcia `rebuildAllData()` ako prvé spustí funkciu `rebuildServers()`, ktorá pošle požiadavku na API s cieľom získať zoznam serverov. Získané dáta potom vloží do `rebuildElements()`, ktorá aktualizuje zoznam serverov a spolu s ním aj informácie ako je stav servera (online/offline), dobu behu, počet aktívnych a celkových klientov, počet pripojených zariadení, adresu siete a port s protokolom prenosu dát. Každý server tiež disponuje tlačidlami pre jeho spustenie, zastavenie alebo reštartovanie. O to aby boli zobrazené len počas zodpovedajúceho stavu servera zodpovedá callback `configureServerControlBtns()`.

Po úspešnej aktualizácii serverov sa vyvolá funkcia `refreshAttachOrgModal()`, ktorá vo formulári na pripájanie organizácií obnoví ponuku serverov a dostupných organizácií na pripojenie. Tento proces vyžaduje získanie zoznamu organizácií pomocou príslušného API.

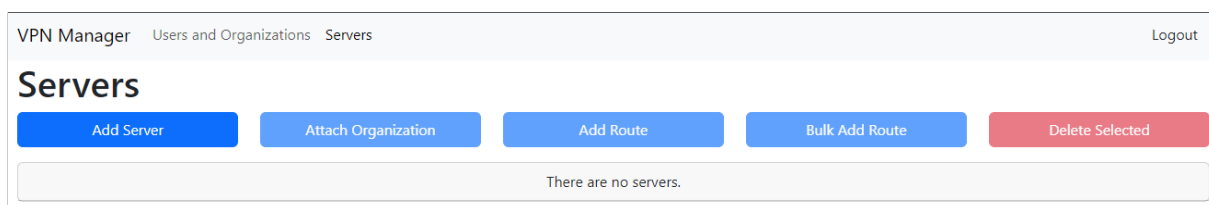
Následne sú pre každý server spustené dve funkcie: `rebuildRoutesByServerID()` a `rebuildAttachedOrgsByServerID()`. Prvá funkcia obnovuje zoznam routes a druhá funkcia zas zoznam pripojených organizácií. Postup je rovnaký ako v predošlých prípadoch. Najprv sú volané API na získanie potrebných dát a tie sú následne vložené do funkcie `rebuildAllData()`, ktorá jednotlivé zoznamy aktualizuje.

Za zmienku stojí spomenúť, že medzi zoznamom routes sa nachádza aj virtuálna

sieť servera, ktorú vieme identifikovať pomocou booleanu `virtual_network`. Keď pri aktualizovaní routes narazíme na takúto sieť, zaškrŕavacie políčko deaktivujeme, aby sme predišli možnému pokusu používateľa o jej odstránenie.

Ak server nemá pripojené žiadne organizácie, nemalo by mu byť umožnené byť spustený. Teda, ak API pre získanie pripojených organizácií ku serveru vráti prázdny zoznam, musí byť tlačidlo na spustenie servera deaktivované.

Pokiaľ neexistuje žiadny server, je zoznam serverov prázdny a jediné aktívne tlačidlo je *Add Server* (obr. 9). Slúži na zobrazenie formulára pre vytvorenie servera (obr. 10 vľavo). Jednotlivé polia, ich formát a význam sme si podrobne popísali v sekcii 3.2.5. Podobne ako v predchádzajúcich prípadoch tak aj tu sa po vyplnení polí a potvrdení formulára, dáta najprv upraví funkciou `cleanFormData()`. Vložené sú potom do funkcie `createServerApi()`, ktorá pošle požiadavku na API pre vytvorenie servera. Ak prebehla úspešne, vygenerovaný event vyvolá spustenie už popísanej funkcie `rebuildServers()`.

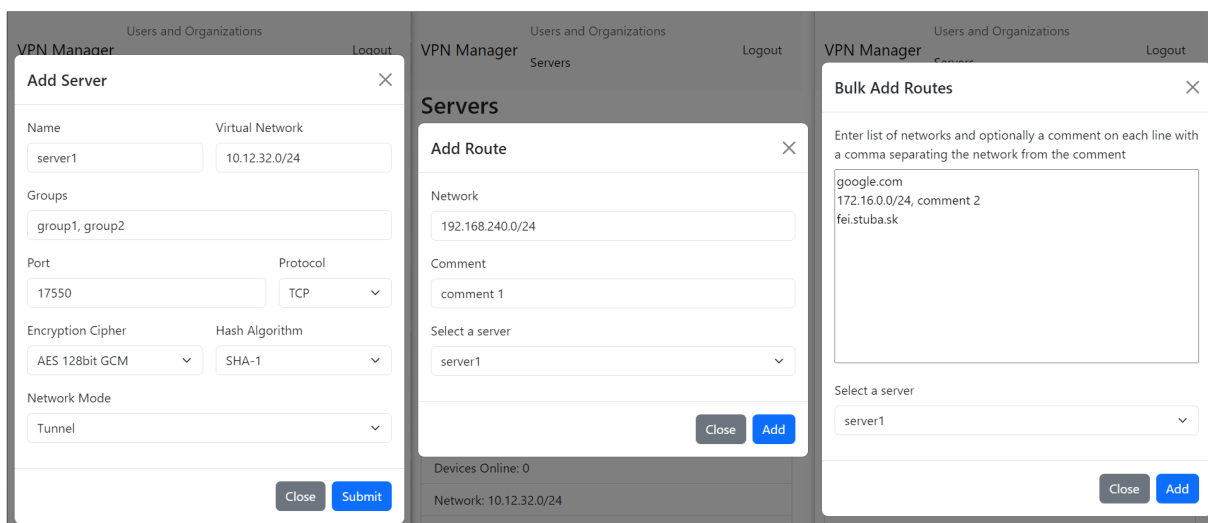


Obr. 9: VPN Manager: Servery - žiadne vytvorené servery

Ak je vytvorený aspoň jeden server, tlačidlá *Add Route* a *Bulk Add Routes* sú aktívne. Obidve tlačidlá sú zodpovedné za zobrazenie formulárov určených na vytváranie routes. Dôležité je rozlíšiť, že *Add Route* slúži na vytvorenie jednej route (obr. 10 stred), zatiaľ čo *Bulk Add Routes* je určené na hromadné vytváranie (obr. 10 vpravo). Využitie a dôvod za zavedením týchto dvoch akcií je totožný ako na predchádzajúcej stránke, pri vytváraní jedného alebo viacerých klientov.

Po očistení zadaných dát je spustená funkcia `createRouteApi()` alebo `bulkCreateRoutesApi()`, ktorá následne pošle požiadavku na korešpondujúci API pre vytvorenie jednej alebo viacerých routes. Podrobne sme si tento proces ako aj s tým spojené zastavenie/spustenie servera už opísali v sekcii 3.2.8. Ak je vytvorenie úspešné, vygenerovaný event spôsobí volanie funkcie `rebuildRoutesByServerID()`, ktorú sme si už tiež predstavili.

Ak okrem servera existuje zároveň aspoň jedna organizácia, aktívne je aj tlačidlo *Attach Organization*. Zodpovedné je za zobrazenie formulára na pripojenie organizácie ku serveru (obr. 11 vľavo). Po výbere z dostupných možností a po potvrdení je volaná funkcia `attachOrgApi()`, ktorá následne posieľa požiadavku na relevantný API. Podobne



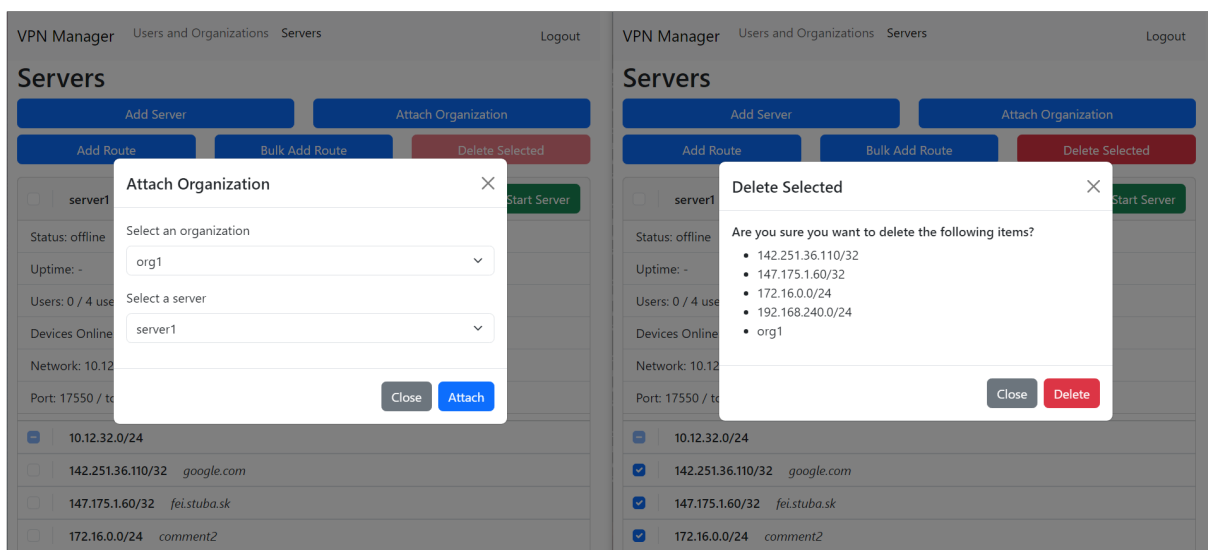
Obr. 10: VPN Manager: Servery - vytvorenie servera (vľavo), vytvorenie route (stred), hromadné vytvorenie routes (vpravo)

ako pri vytváraní routes, aj v tomto prípade je nevyhnutné zabezpečiť, aby bol daný server vypnutý. Detailnejšie sme tento proces rozobrali v sekcii 3.2.9. Úspešné pripojenie organizácie spôsobí, z dôvodov nešpecifikovaných zo strany Pritunl, generovanie všetkých typov eventov, ktoré sa týkajú serverov a ich entít. Vzhľadom na túto skutočnosť a nemožnosť ovplyvniť ju bez významného obmedzenia funkcionality, dochádza ku volaniu funkcií zodpovedných za aktualizáciu všetkých serverov, routes a pripojených organizácií. V tomto kontexte je pre nás zaujímavá iba posledná časť, konkrétne volanie funkcie `rebuildAttachedOrgsByServerID()`, ktorej fungovanie sme si už rozobrali.

Vytvorené servery a routes je takisto možné dodatočne upravovať. Kliknutím na meno servera alebo na adresu route sa otvorí formulár na ich úpravu. V prípade servera je možné modifikovať všetky atribúty, avšak, ak klienti na pripojenie ku serveru nepoužívajú Pritunl Client, tak modifikácia portu, protokolu, šifrovacieho algoritmu či sieťového režimu si vyžaduje opätovné stiahnutie klientských profilov. Pokiaľ ide o routes, jediný atribút, ktorý je možné upraviť je priložený komentár. Zmenu adresy nie je možné realizovať, je potrebné vymazať existujúcu route a vytvoriť novú s požadovanou adresou.

Toto nás vedie k procesu mazania, ktorý je rovnaký ako na predchádzajúcej stránke. Vyberanie entít sa uskutočňuje pomocou zaškrŕavacích políčk. Pokiaľ je zaškrŕnuté aspoň jedno, sprístupní sa tlačidlo *Delete Selected*, ktoré zobrazí potvrdzovací formulár vrátane výpisu entít určených na odstránenie (obr. 11 vpravo). V prípade pripojených organizácií sa nejedná o zmazanie organizácie, ale o jej odpojenie od servera.

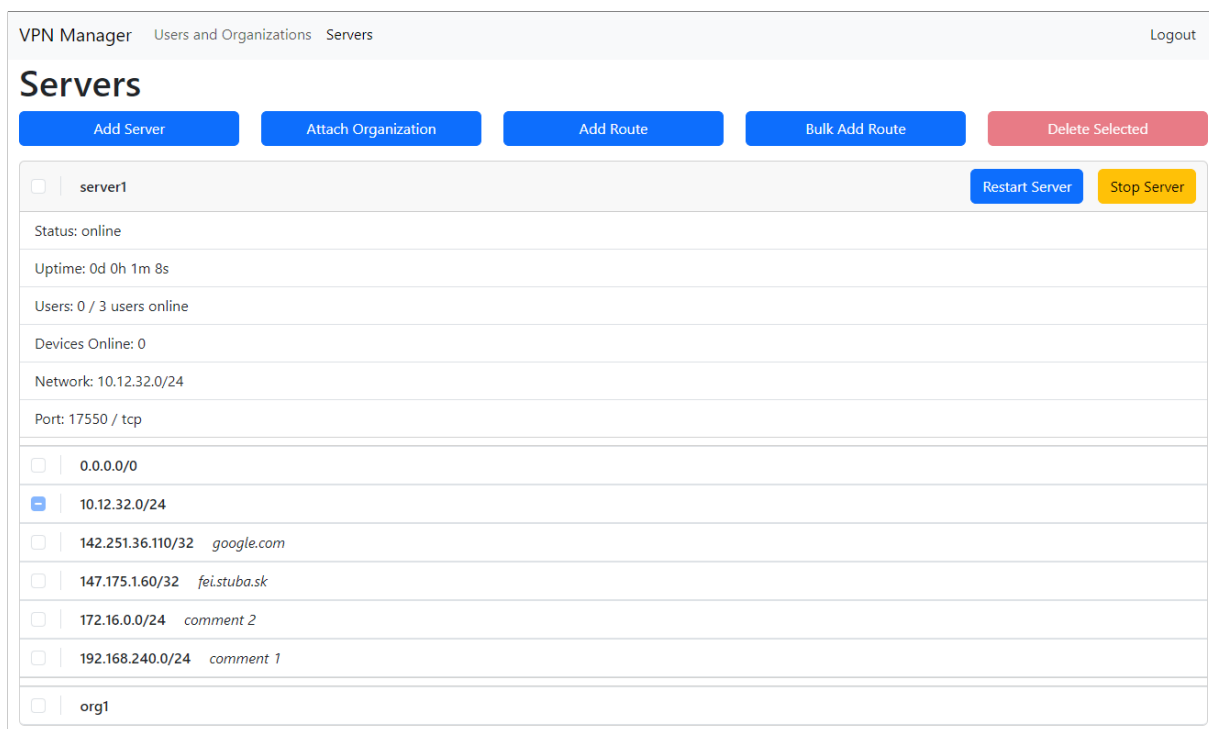
Po potvrdení sa pre servery volá funkcia `deleteServerApi()`, pre routes a pripojené



Obr. 11: VPN Manager: Servery - pripojenie organizácie (vľavo), odstránenie vybraných entít (vpravo)

organizácie spoločná funkcia `deleteRoutesAndOrgsApi()`. Pre routes a organizácií sme vytvorili API, ktoré slúži na mazanie oboch entít naraz a v tele požiadavky očakáva objekt, ktorý obsahuje typy entít a ich ID. Implementácia takéhoto, pre REST neštandardného, riešenia bola potrebná v dôsledku toho, že server musí byť pred vykonaním akcie zastavený a po jej dokončení opätovne spustený. Ak by sme poslali dve samostatné požiadavky, jednu pre routes a druhú pre organizácie, server by bol zbytočne spúšťaný a potom znovu zastavovaný. Naše riešenie úspešne predchádza tomuto problému a zvyšuje celkovú efektivitu aplikácie. Po dokončení mazania, prebehne aktualizácia dotknutých entít na základe vyvolaných eventov.

Ako bolo už predtým spomenuté, každý server disponuje tlačidlami na jeho ovládanie. Po ich stlačení je zavolaná funkcia `controlServerApi()`, ktorá ako parameter očakáva ID servera a typ akcie (start, restart, stop). Táto funkcia na základe zvolenej akcie pošle požiadavku na príslušné API. Spôsob fungovania tohto API a dôsledky jednotlivých akcií na server sme definovali v sekcii 3.2.7. Po úspešnom vykonaní akcie je vyvolaný event na aktualizáciu servera, ktorý následne spustí už viackrát spomínanú `rebuildServers()` (obr. 12).



Obr. 12: VPN Manager: Servery - spustený server

4 Vyhodnotenie riešenia

V tejto kapitole si vyhodnotíme aké problémy rieši nami implementovaná aplikácia a či sa nám podarilo splniť zadané požiadavky. Okrem toho uvedieme jej nedostatky a potenciálne oblasti na zlepšenie.

4.1 Bezpečnosť

Middleware server implementuje autentifikačný a autorizačný systém na základe sessions, ktorý poskytuje solídnu ochranu proti bežným hrozbám. Disponuje vlastným modelom používateľa, ktorý je navrhnutý tak, aby sa v prípade potreby dal jednoducho rozšíriť o požadované atribúty alebo metódy. V súlade s požiadavkami ponúka dve autorizačné úrovne - admin a read-only. Tie sa používajú na rozlíšenie a obmedzenie prístupu používateľov ku vytvoreným API. Autentifikácia je realizovaná prostredníctvom prihlasovacieho rozhrania, ktoré overuje prihlasovacie meno a heslo. Jedná sa o jednoduchú formu overenia, ktorá síce spĺňa naše požiadavky, avšak má priestor na zlepšenie.

Potenciálnym vylepšením by mohlo byť zavedenie dvojfaktorovej autentifikácie, ktorá by okrem prihlasovacieho mena a hesla vyžadovala zadanie jednorázového kódu. Ten by mohol byť generovaný aplikáciou na smartfóne alebo odoslaný na email používateľa.

Ďalším spôsobom ako zvýšiť bezpečnosť by mohla byť napríklad implementácia throttlingu, ktorý obmedzí počet pokusov o prihlásenie v danom časovom intervale. Ak by bol limit prekročený, systém by na určitý čas blokoval ďalšie pokusy o prihlásenie. Takéto opatrenie by prispelo k ochrane systému pred brute-force útokmi.

4.2 Funkcionalita

Middleware server tiež disponuje vlastným REST API, ktoré nášmu webovému rozhraniu sprostredkuje komunikáciu so službou Pritunl. Implementované boli API endpointy, ktoré umožňujú spravovať organizácie, klientov, servery a routes. Doplnili sme ich o funkcionality, ktorú Pritunl natívne neposkytuje. Sem patrí napríklad kontrola stavu servera a jeho automatické zastavenie/spustenie v API endpointoch, ktoré by inak pri pokuse o ich vykonanie, v prípade spusteného servera, vygenerovali chybu na Pritunl servery. Takéto riešenie významne zlepšuje UX, keďže eliminuje zbytočné chybové hlásenia o nemožnosti vykonania požadovanej akcie kvôli spustenému serveru.

Inšpirovaní funkciou hromadného vytvárania klientov, ktorú ponúka aj Pritunl, sme implementovali API na hromadné vytváranie routes. V prípade, že chce používateľ vytvoriť väčšie množstvo routes naraz, táto API mu na to poskytuje rýchly a efektívny spôsob. Ku endpointom na vytváranie routes sme tiež pridali možnosť zadať adresu route aj vo formáte DNS. Tá je automaticky rozpoznaná a pokiaľ je platná, je preložená na zodpovedajúcu IP adresu.

Pri hromadnom mazaní routes a odpájaní organizácií zo serverov sme narazili na problém s výkonom v dôsledku redundantného zastavovania/spúšťania servera. Implementovali sme preto API, ktoré vykoná kontrolu stavu servera iba raz a umožní hromadne odstraňovať routes a odpájať organizácie. Takéto riešenie sa nám osvedčilo, keďže okrem odstránenia spomenutého problému, stačí poslať len jednu API požiadavku, namiesto zvlášť požiadavky pre každú jednu entitu, čím je zvýšená efektívnosť a rýchlosť spracovania požadovanej akcie.

Naše REST API poskytuje nástroje na efektívnejšiu prácu s Pritunl, a keďže funguje ako samostatný logický celok, je ho možné využívať aj bez vytvoreného webového rozhrania. Vhodné je však najmä pre základné riadenie služby Pritunl, keďže komplexnejšie funkcie nie sú implementované alebo sú implementované len čiastočne. Medzi takéto chýbajúce funkcie patria napríklad niektoré pokročilé nastavenia pri vytváraní klientov, serverov a routes, automatické zasielanie odkazov na klientské profily cez email a podobne.

4.3 Webové rozhranie

Webové rozhranie komunikuje s vytvoreným REST API a sprostredkuje nástroje na jeho ovládanie. Navrhnuté je tak, aby bolo intuitívne, užívateľsky prívetivé, responzívne a štruktúrou podobné vstavanému Pritunl rozhraniu. Prístup ku nemu je podmienený už spomenutou autentifikáciou a na základe autorizačnej úrovne používateľa, upravuje dostupný obsah na stránkach podľa definovaných požiadaviek.

Implementované bolo filtrovanie vytvorených klientov a organizácií podľa ich mena. Narozdiel od jednoduchého filtrovania klientov pre konkrétnu organizáciu, ktoré poskytuje vstavané Pritunl rozhranie, to naše je globálne a teda je možné vyhľadávať klientov naprieč všetkými organizáciami.

Rozhodli sme sa porovnať čas, ktorý je potrebný na vykonávanie určitých akcií v klasickom Pritunl rozhraní a v našom riešení. Zdefinovali sme si scenár zo spusteným serverom, ktorému chce používateľ pridať desať rôznych routes. Po vykonaní všetkých akcií musí byť server opäť spustený. Používateľ nepozná IP adresy, má však zoznam ich DNS ekvivalentov. Výsledky merania sú v tabuľke 11.

Prostredie	Čas
Pritunl rozhranie	4 minúty a 17 sekúnd
VPN Manager	9 sekúnd

Tabuľka 11: Testovanie: Hromadné vytváranie routes

Ďalší scenár, ktorý sme si zdefinovali, bol pre testovanie hromadného mazania routes a odpájania organizácií. Začíname so spusteným serverom, ktorý má desať routes a päť pripojených organizácií. Chceme ich všetky odstrániť a ponechať iba prázdny server. Namerané hodnoty sú v tabuľke 12. Ako vyplýva z testovania, naše riešenie má potenciál používateľovi značne urýchliť a zefektívniť prácu so službou Pritunl.

Prostredie	Čas
Pritunl rozhranie	45 sekúnd
VPN Manager	16 sekúnd

Tabuľka 12: Testovanie: Hromadné odstraňovanie routes a odpájanie organizácií

Jedným z problémov, ktoré sme identifikovali, je pomalší čas na detekciu zmien v databáze. Webové rozhranie na to využíva techniku long polling na middleware server, ktorý následne vykonáva long polling na Pritunl server. Tento medzikrok spôsobuje oneskorenie, nemôžeme ho však vynechať, keďže ho potrebujeme na vykonávanie autentifikácie

požiadaviek. Potenciálnym riešením by mohlo byť napríklad nahradenie long pollingu zo strany webového rozhrania technikou SSE. Tá je efektívnejšia [32] a jej využitie by mohlo prispieť ku rýchlejšej detekcii zmien, čím zlepšíme odozvu systému.

Ďalším spôsobom na optimalizáciu webového rozhrania by mohla byť implementácia paginácie, pre zoznamy organizácií a serverov. Zobrazovali a aktualizovali by sa len tie dáta, ktoré používateľ aktuálne potrebuje. V prípade veľkého počtu entít, by paginácia mohla značne zlepšiť rýchlosť a efektívnosť systému, keďže by sa prenášali len potrebné dáta. Má tiež potenciál prispieť k lepšej prehľadnosti zobrazených dát.

Záver

Cieľom tejto práce bola podrobná analýza VPN služby Pritunl, jej vstavaného webového rozhrania a identifikácia možných nedostatkov. Na základe týchto poznatkov sme mali implementovať vlastné riešenie, ktoré by zistené nedostatky odstránilo a doplnilo požadovanú funkcionality.

Vytvorili sme webovú aplikáciu VPN Manager, ktorá komunikuje so službou Pritunl a ponúka nástroje na spravovanie klientov, organizácií, serverov a routes. Aplikácia je štrukturovaná do dvoch hlavných častí - middleware server a webové rozhranie. Middleware server implementuje vlastné REST API, ktoré je zodpovedné za spracovanie požiadaviek a interakciu s Pritunl API. Navrhnuté je tak, aby bolo ľahko rozšíriteľné a v prípade potreby, použiteľné aj bez webového rozhrania. Štandardnú Pritunl funkcionality rozširuje o automatické zastavenie/spustenie servera počas nasadzovania zmien, hromadné vytváranie routes, preklad DNS domén na IP adresy a hromadné odstraňovanie routes a pripojených organizácií zo serverov.

Disponuje tiež autentifikačným a autorizačným systémom, ktorý spĺňa bezpečnostné požiadavky a chráni prístup ku REST API a webovému rozhraniu. Ponúka dve autorizačné úrovne - admin a read-only, ktoré umožňujú lepšie kontrolovať prístup ku funkciám systému a minimalizujú riziko neoprávnených zmien.

Webové rozhranie je intuitívne a užívateľsky prívetivé, pričom poskytuje prehľad o vytvorených entitách a prístupoch. Ponúka nástroje na prácu s vytvoreným REST API a na základe autorizačnej úrovne používateľa upravuje dostupný obsah. Súčasťou je aj globálny filter klientov a organizácií. Vykonané testovanie ukázalo, že funkcie a nástroje, ktoré implementuje aplikácia VPN Manager majú potenciál výrazne urýchliť a zefektívniť prácu so službou Pritunl.

V mnohých oblastiach, má však aplikácia priestor na zlepšenie. Chýba implementácia niektorých komplexnejších funkcií či nastavení, ktoré Pritunl ponúka. Autentifikačný systém by mohol byť rozšírený o prvky ako je dvojfaktorová autentifikácia alebo throttling pokusov o prihlásenie. Implementácia paginácie a vhodnejšej techniky na rozpoznávanie zmien v databáze by prispeli ku zvýšenej efektívnosti a minimalizácii odozvy pre webové rozhranie.

Zoznam použitej literatúry

1. SCOTT, Ch., WOLFE, P. a ERWIN, M. *Virtual Private Networks: Turning the Internet Into Your Private Network*. 2. vyd. O'Reilly Media, 1998. ISBN 978-1-56592-529-8.
2. *2021 VPN RISK REPORT* [online]. Cybersecurity Insiders, 2021 [cit. 2023-04-08]. Dostupné z : <https://recursos.bps.com.es/files/991/04.pdf>.
3. SNADER, Jon C. *VPNs Illustrated: Tunnels, VPNs, and IPsec*. 1. vyd. Addison-Wesley Professional, 2005. ISBN 978-0-321-24544-1.
4. KUROSE, James F. a ROSS, Keith W. *Computer Networking: A Top-down Approach*. 8. vyd. Pearson, 2020. ISBN 978-0-13-668155-7.
5. DEBNATH, S., CHATTOPADHYAY, A. a DUTTA, S. Brief review on journey of secured hash algorithms. In: *2017 4th International Conference on Opto-Electronics and Applied Optics (Optronix)*. 2017, s. 1–5. Dostupné z DOI: 10.1109/OPTRONIX.2017.8349971.
6. CRIST, Erif F. a KEIJSER, Jan Just. *Mastering OpenVPN*. Packt Publishing, 2015. ISBN 978-1-78355-313-6.
7. *2x HOW TO* [online]. OpenVPN, 2023 [cit. 2023-04-05]. Dostupné z : <https://openvpn.net/community-resources/how-to/#larger-symmetric-keys>.
8. *The best VPN protocols* [online]. NordVPN, 2022 [cit. 2023-04-05]. Dostupné z : <https://nordvpn.com/blog/protocols/#openvpn>.
9. OSSWALD, L., HAEBERLE, M. a MENTH, M. Performance Comparison of VPN Solutions. 2020.
10. *Enterprise Distributed OpenVPN, IPsec and WireGuard Server* [online]. Pritunl, 2023 [cit. 2023-04-07]. Dostupné z : <https://pritunl.com/>.
11. *User configuration* [online]. Pritunl, 2020 [cit. 2023-04-15]. Dostupné z : <https://docs.pritunl.com/docs/configuration>.
12. KUHLMAN, Dave. *A Python Book: Beginning Python, Advanced Python, and Python Exercises*. Platypus Global Media, 2011. ISBN 978-0-9842212-3-3.
13. *contrib packages* [online]. Django Software Foundation, 2023 [cit. 2023-04-16]. Dostupné z : <https://docs.djangoproject.com/en/4.2/ref/contrib/>.

14. *Understanding the MVC pattern in Django* [online]. Medium, 2023 [cit. 2023-04-16]. Dostupné z : <https://medium.com/shecodeafrica/understanding-the-mvc-pattern-in-django-edda05b9f43f>.
15. *What is django ORM?* [online]. Tutorials Point, 2023 [cit. 2023-04-18]. Dostupné z : <https://www.tutorialspoint.com/what-is-django-orm#>.
16. *Security in Django* [online]. Django Software Foundation, 2023 [cit. 2023-04-15]. Dostupné z : <https://docs.djangoproject.com/en/4.2/topics/security/>.
17. *The Django admin site* [online]. Django Software Foundation, 2023 [cit. 2023-04-15]. Dostupné z : <https://docs.djangoproject.com/en/4.2/ref/contrib/admin/>.
18. *Django REST framework* [online]. Encode OSS Ltd., 2023 [cit. 2023-04-17]. Dostupné z : <https://www.django-rest-framework.org/>.
19. *Serializers* [online]. Encode OSS Ltd., 2023 [cit. 2023-04-18]. Dostupné z : <https://www.django-rest-framework.org/api-guide/serializers/>.
20. *Authentication* [online]. Encode OSS Ltd., 2023 [cit. 2023-04-17]. Dostupné z : <https://www.django-rest-framework.org/api-guide/authentication/>.
21. *Permissions* [online]. Encode OSS Ltd., 2023 [cit. 2023-04-17]. Dostupné z : <https://www.django-rest-framework.org/api-guide/permissions/>.
22. *What is a REST API?* [online]. Red Hat, 2020 [cit. 2023-04-09]. Dostupné z : <https://www.redhat.com/en/topics/api/what-is-a-rest-api>.
23. *Templates* [online]. Django Software Foundation, 2023 [cit. 2023-04-16]. Dostupné z : <https://docs.djangoproject.com/en/4.2/ref/templates/language/>.
24. *What is jQuery?* [online]. OpenJS Foundation, 2023 [cit. 2023-04-20]. Dostupné z : <https://jquery.com/>.
25. *What is three-tier architecture?* [online]. IBM, 2023 [cit. 2023-04-15]. Dostupné z : <https://www.ibm.com/topics/three-tier-architecture>.
26. *What is HTTP Long Polling?* [online]. PubNub Inc., 2023 [cit. 2023-04-24]. Dostupné z : <https://www.pubnub.com/blog/http-long-polling/>.
27. *API Access* [online]. Pritunl, 2022 [cit. 2023-05-02]. Dostupné z : <https://docs.pritunl.com/docs/api>.
28. *Private Address Ranges* [online]. IBM, 2023 [cit. 2023-05-04]. Dostupné z : <https://www.ibm.com/docs/en/networkmanager/4.2.0?topic=translation-private-address-ranges>.

29. *VPN Bridge vs Tunnel: Understanding the Differences* [online]. VPNCentral, 2023 [cit. 2023-05-05]. Dostupné z : <https://vpncentral.com/vpn-bridge-vs-tunnel/>.
30. *Server configuration* [online]. Pritunl, 2021 [cit. 2023-05-10]. Dostupné z : <https://docs.pritunl.com/docs/configuration-3>.
31. *Using a custom user model when starting a project* [online]. Django Software Foundation, 2023 [cit. 2023-04-29]. Dostupné z : <https://docs.djangoproject.com/en/4.2/topics/auth/customizing/#using-a-custom-user-model-when-starting-a-project>.
32. APPELQVIST, R. a ÖRNMYR, O. *Performance comparison of XHR polling, Long polling, Server sent events and Websockets*. 2017. Blekinge Institute of Technology, Blekinge.

Prílohy

A	Zdrojový kód	II
B	Inšalačná a používateľská príručka	III

A Zdrojový kód

/vpn-manager

- adresár, ktorý obsahuje zdrojový kód aplikácie VPN Manager

B Inštalačná a používateľská príručka

Zdrojový kód ku aplikácii VPN Manager sa nachádza v prílohe A alebo je dostupný na GitHub repozitári cez tento odkaz: <https://github.com/ViktorBojda/vpn-manager>. Pred začatím inštalácie je potrebné mať:

- nainštalovaný operačný systém, ktorý Pritunl podporuje
- nainštalovaný Pritunl¹² a zakúpenú licenciu Enterprise
- nainštalovaný Python vo verzií 3.10
- vytvorený účet na MongoDB Atlas

Poznámka: Naše riešenie bolo testované na WSL2 s nainštalovaným operačným systémom Ubuntu 20.04

Podrobný postup inštalácie a nastavenia aplikácie VPN Manager:

1. V termináli prejdeme do koreňového priečinku zdrojového kódu a vykonáme príkaz:
`python3.10 -m venv .venv`
2. Po úspešnom vytvorení virtuálneho prostredia ho aktivujeme príkazom:
`source .venv/bin/activate` a príkazom: `python --version` overíme, že Python má verziu 3.10
3. Spustíme príkaz: `pip install -r requirements.txt`, čím sa nainštalujú potrebné knižnice.
4. V koreňovom priečinku vytvoríme súbor `.env`, do ktorého skopírujeme obsah zo súboru `.env.example`. Všetky nasledujúce zmeny sa budú týkať len súboru `.env`
5. Zadáme hodnotu pre premennú `SECRET_KEY`, ktorú môžeme získať napríklad z generátora na odkaze: <https://djecrety.ir/>
6. Do premennej `BASE_URL` vložíme URL adresu servera, na ktorom je spustený Pritunl (v tomto prípade je to adresa používaného počítača). Príklad formátu takejto adresy: `https://172.21.79.130` (dôležité je na konci nevkladať lomku)

¹²zoznam operačných systémov a návod na inštaláciu je dostupný na odkaze: <https://docs.pritunl.com/docs/installation>

7. Prihlásime sa do svojho účtu MongoDB Atlas a prejdeme na podstránku *Database*. Tam klikneme na tlačidlo *Build a Database*.
8. Zvolíme požadované nastavenia a zadáme meno vytváraného clustru. Klikneme na *Create*, ktorý nás presmeruje na podstránku *Security Quickstart*.
9. Ako spôsob autentifikácie zvolíme *Username and Password* a vytvoríme používateľa vložení prihlasovacieho mena a hesla a klikneme na *Create User*.
10. Ako spôsob pripojenia zvolíme *My Local Environment* a klikneme na *Add My Current IP Address*. Po pridaní IP adresy by sa malo aktivovať tlačidlo *Finish and Close*, na ktoré klikneme.
11. Vo vytvorenom clustri klikneme na *Connect* a zvolíme možnosť *Drivers*. Uistíme sa, že driver je nastavený na **Python** a verzia je **3.6 or later**. Skopírujeme connection string (mal by sa začínať na `mongodb+srv://`) a vložíme ho do súboru `.env` ako hodnotu pre premennú `MONGODB_URI`.
12. Musíme ho ešte dodatočne upraviť. Časť `<password>` nahradíme heslom používateľa z bodu 9. Následne za časť `mongodb.net/`, vložíme meno vytvoreného clustru z bodu 8.
13. Adresu z `BASE_URL`, vložíme do prehliadača. Ak máme Pritunl nainštalovaný správne, malo by sa tam nachádzať okno *pritunl database setup*. Do terminálu zadáme príkaz `sudo pritunl setup-key` a vrátenú hodnotu vložíme do kolónky *Enter Setup Key*. Hodnotu z `MONGODB_URI` vložíme do kolónky *Enter MongoDB URI* a klikneme na *Save*.
14. Ak prebehlo nastavovanie úspešne, mali by sme byť presmerovaný na prihlasovacie rozhranie. Je však pravdepodobné, že prehliadač zobrazí upozornenie o nezabezpečenom prístupe kvôli chýbajúcim SSL certifikátom. Odsúhlasíme riziko a pokračujeme ďalej.
15. Do terminálu zadáme príkaz: `sudo pritunl default-password` a použijeme vygenerované údaje na prihlásenie. Následne budeme presmerovaný na vstavané Pritunl rozhranie, ktoré nás vyzve na zmenu údajov a iné nastavenie. Môžeme to preskočiť kliknutím na *Setup Later*.

16. Klikneme na tlačidlo *Upgrade to Enterprise!*. Zobrazí sa nám okno kde si aktivujeme licenciu Enterprise zadáním licenčného kľúča, ktorý nám bol poslaný pri zakúpení licencie.
17. Budeme presmerovaný na webové rozhranie určené pre používateľov s licenciou Enterprise. Prejdeme na podstránku *Administrators*, kde klikneme na meno administrátora (ak sme nemenili nastavenia bude to `pritun1`).
18. Zobrazí sa nám okno na jeho modifikáciu, kde potvrdíme možnosť *Enable Token Authentication*. Zobrazia sa nám hodnoty s označením *API Token* a *API Secret*. Skopírujeme ich a okno potvrdíme. Skopírované hodnoty vložíme do súboru `.env` do ich zodpovedajúcich premenných `API_TOKEN` a `API_SECRET`.
19. Uistíme sa, že v terminály sa nachádzame v koreňovom priečinku a spustíme príkaz: `python manage.py createsuperuser` pre vytvorenie superusera. Postupujeme podľa výziev terminálu a zadáme prihlasovacieho meno a heslo.
20. Ak prebehlo vytvorenie úspešne, môžeme spustiť aplikáciu VPN Manager príkazom: `python manage.py runserver`
21. Dostupná je na predvolenej adrese `127.0.0.1:8000` a Django Admin rozhranie sa nachádza na `127.0.0.1:8000/admin`
22. Aplikácia je teraz nastavená a pripravená na používanie.