Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií

FIIT-100241-102986

Richard Kello

Operačný systém ako web CMS (WCM) - Manažment používateľov a používateľských skupín v operačnom systéme cez webové rozhranie

Bakalárska práca

Pedagogický vedúci: Ing. Katarína Jelemenská, PhD.
Vedúci práce: Ing. Gabriel Szabó

Máj 2023

Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií

FIIT-100241-102986

Richard Kello

Operačný systém ako web CMS (WCM) - Manažment používateľov a používateľských skupín v operačnom systéme cez webové rozhranie

Bakalárska práca

Študijný program: B-INFO4 informatika

Študijný odbor: Informatika

Miesto vypracovania: Ústav počítačového inžinierstva a aplikovanej informatiky (FIIT)

Pedagogický vedúci: Ing. Katarína Jelemenská, PhD.

Vedúci práce: Ing. Gabriel Szabó

Máj 2023

Slovenská technická univerzita v Bratislave
Ústav počítačového inžinierstva a aplikovanej informatiky

Fakulta informatiky a informačných technológií
Akademický rok: 2022/2023
Evidenčné číslo: FIIT-100241-102986

ZADANIE BAKALÁRSKEJ PRÁCE

Študent:	**Richard Kello**
ID študenta:	102986
Študijný program:	informatika
Študijný odbor:	informatika
Vedúci práce:	Ing. Gabriel Szabó
Vedúci pracoviska:	Ing. Katarína Jelemenská, PhD.
Pedagogická vedúca práce:	Ing. Katarína Jelemenská, PhD.

Názov práce: **Operačný systém ako web CMS (WCM) – Manažment používateľov a používateľských skupín v operačnom systéme cez webové rozhranie**

Jazyk, v ktorom sa práca vypracuje: slovenský jazyk

Špecifikácia zadania:

Operačné systémy sú komplexné systémy, ktoré obsahujú hotové a funkčné riešenia pre rôzne typy úloh, ktoré sú bežne riešené aj v rámci WCM systémov. Typickými príkladmi je manažment používateľov, alebo zdielaný prístup k dátam. Cieľom tohto zadania je vystaviť určité funkcie operačného systému na báze Linuxu cez webové rozhranie. Zanalyzujte požiadavky a identifikujte základné funkcie webových portálov a WCM systémov na manažment používatelov, používateľských skupín a zdielanie dát. Porovnajte požadované funkcionality s funkciami dostupnými v operačnom systéme Linux. Navrhnite rozšírenie aktuálne dostupných Linuxových funkcii o potenciálne chýbajúce pomocou jednoduchých skriptov (Bash alebo podobne). Na základe výsledkov z časti 1 zadania vytvorte webové rozhranie na manažment používateľov v operačnom systéme Linux.

Rozsah práce: 40

Termín odovzdania bakalárskej práce:	22. 05. 2023
Dátum schválenia zadania bakalárskej práce:	18. 04. 2023
Zadanie bakalárskej práce schválil:	doc. Ing. Valentino Vranić, PhD. – garant študijného programu

Čestné prehlásenie

Čestne vyhlasujem, že som túto prácu vypracoval samostatne, na základe konzultácií a s použitím uvedenej literatúry.

V Bratislave dňa 22. mája 2023 Richard Kello

Poďakovanie

Chcel by som vyjadriť poďakovanie vedúcemu práce Ing. Gabrielovi Szabóvi za jeho odborné rady a vedenie práce. Okrem toho chcem poďakovať pedagogickej vedúcej Ing. Kataríne Jelemenskej, PhD. Nakoniec by som rád poďakoval všetkým, ktorí ma v práci podporovali.

Annotation

Slovak University of Technology Bratislava
Faculty of Informatics and Information Technologies
Degree Course: B-INFO4 Computer Science

Author: Richard Kello
Bachelor thesis: Operating system as web CMS (WCM) – Management of users and user groups in the operating system via a web interface
Pedagogical supervisor: Dr. Katarína Jelemenská
Supervisor: Ing. Gabriel Szabó
May 2023

The user and user group management application is designed to streamline user management processes in a Linux environment. The application is built with a backend programmed in FastAPI and offers a seamless and efficient solution for managing users, groups, passwords and access control. The application leverages the security and flexibility of JWT tokens to authorize endpoints, ensuring that only authenticated users have access to protected resources. The authentication process is performed using the „oauth2_scheme“. Through FastAPI endpoints, administrators can create, modify, and delete groups, enabling fine-grained access control and resource allocation in a Linux environment. In addition, the application simplifies Linux user and password management by providing user endpoints. By combining the power of FastAPI, JWT tokens, and Linux integration, the application offers a secure, efficient, and user-friendly solution for managing users, groups, passwords, and access control in a Linux environment. It enables administrators to streamline their administrative tasks, enhance security and optimize resource allocation.

Anotácia

Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií
Študijný program: B-INFO4 informatika

Autor: Richard Kello
Bakalárska práca: Operačný systém ako web CMS (WCM) - Manažment používateľov a používateľských skupín v operačnom systéme cez webové rozhranie
Pedagogický vedúci projektu: Ing. Katarína Jelemenská, PhD.
Vedúci bakalárskeho projektu: Ing. Gabriel Szabó
Máj 2023

Aplikácia na správu používateľov a používateľských skupín je určená na zefektívnenie procesov správy používateľov v prostredí Linux. Aplikácia je vytvorená s backendom naprogramovaným v rozhraní FastAPI a ponúka bezproblémové a efektívne riešenie na správu používateľov, skupín, hesiel a riadenie prístupu. Aplikácia využíva bezpečnosť a flexibilitu tokenov JWT na autorizáciu koncových bodov, čím zabezpečuje, že k chráneným zdrojom majú prístup len overení používatelia. Proces overovania sa vykonáva pomocou systému „oauth2_scheme“. Prostredníctvom koncových bodov FastAPI môžu správcovia vytvárať, upravovať a odstraňovať skupiny, čo umožňuje jemné riadenie prístupu a prideľovanie zdrojov v prostredí Linux. Okrem toho aplikácia zjednodušuje správu používateľov a hesiel systému Linux tým, že poskytuje používateľské koncové body. Spojením výkonu FastAPI, tokenov JWT a integrácie systému Linux ponúka aplikácia bezpečné, efektívne a používateľsky prívetivé riešenie na správu používateľov, skupín, hesiel a riadenia prístupu v prostredí Linux. Umožňuje správcom zefektívniť ich administratívne úlohy, zvýšiť bezpečnosť a optimalizovať prideľovanie zdrojov.

Obsah

Zoznam skratiek

WCMS Web Content Management System

JSON JavaScript Object Notation

JWT JSON Web Token

UAT User Acceptance Testing

OS Operating System

CPU Central Processing Unit

RAM Random Access Memory

CMS Content Management System

XML Extensible Markup Language

.NET Domain NETwork

CMA Content Management Application

CDA Content Delivery Application

SQL Structured Query Language

MDPI Multidisciplinary Digital Publishing Institute

XSS Cross-Site Scripting

OWASP Open Worldwide Application Security Project

DOM Document Object Model

CSV Comma-separated Values

CDN) Content delivery network

WAF Web Application Firewall

DDOS Distributed Denial-of-Service

NFS Network File System

GUI Graphical User Interface

SSH Secure Shell Protocol

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol Secure

FTP File Transfer Protocol

ORM Object-Relational Mapping

API Application Programming Interface

JLAT Java Linux Administration Tool)

VPS Virtual Private Server

ACL Access Control Lists

URL Uniform Resource Locator

Kapitola 1

Úvod

Rýchly technologický pokrok viedol k rozšíreniu komplexných operačných systémov, ktoré slúžia ako základ pre rôzne softvérové aplikácie. Jedným z takýchto operačných systémov je Linux, ktorý je známy svojou robustnosťou, škálovateľnosťou a otvoreným zdrojovým kódom. Keďže dopyt po webových systémoch na správu obsahu (WCMS) neustále rastie, je nevyhnutné preskúmať potenciálnu integráciu operačných systémov na báze Linuxu s WCMS, aby sa využili ich hotové a funkčné riešenia pre úlohy, ktoré sa bežne riešia vo WCMS.

Hlavným cieľom tejto práce je vystaviť a analyzovať funkcie operačného systému na báze Linuxu prostredníctvom webového rozhrania. Konkrétne sa zameria na identifikáciu základných funkcií webových portálov a systémov WCMS pre správu používateľov a skupín používateľov. Cieľom tohto výskumu je porovnanie požadovaných funkcií s funkciami dostupnými v systéme Linux a navrhnútie rozšírenia alebo vylepšenia existujúcich funkcií systému Linux pomocou jednoduchých skriptov, ako je Bash alebo podobné skriptovacie jazyky.

Práca je rozdelená do niekoľkých kapitol, pričom každá z nich sa zaoberá

základnými aspektmi danej problematiky. Úvodné kapitoly poskytujú hĺbkovú analýzu operačných systémov, WCMS a ich funkcií. Skúmajú fungovanie WCMS, ich výhody a nevýhody a bezpečnostné aspekty spojené s WCMS. Okrem toho sa popri operačnom systéme Linux preskúmajú existujúce riešenia WCMS, ako sú WordPress a HubSpot.

V ďalších kapitolách sa pozornosť presunie na správu používateľov v operačnom systéme Linux vrátane jej bezpečnostných aspektov. Podrobne sa rozoberú backendové rámce ako Flask, Django a FastAPI spolu s metódami autentifikácie a autorizácie, ako sú JSON Web Token (JWT) a OAuth2. Komplexné porovnanie dostupných funkcií WCMS na správu používateľov s funkciami, ktoré ponúka systém Linux, poskytne cenné poznatky o možných zlepšeniach.

Druhá časť práce sa točí okolo opisu navrhovaného riešenia. Zahŕňa opis funkčných a nefunkčných požiadaviek, pričom sa načrtnú komponenty potrebné na úspešnú implementáciu. Navrhované riešenie bude vyvinuté a implementované prostredníctvom softvérovej implementácie, pričom sa bude zaoberať kľúčovými aspektmi, ako je autentifikácia a autorizácia, správa skupín a správa používateľov.

Na zabezpečenie účinnosti a životaschopnosti riešenia sa navrhnú a vykonajú rôzne testovacie scenáre a scenáre používateľského testovania (UAT). Výsledky testovania potvrdia správnosť riešenia a preukážu jeho praktickosť v reálnych scenároch.

Na konci tejto práce čitatelia získajú komplexné znalosti o vlastnostiach a funkciách operačných systémov na báze Linuxu a ich potenciálnej integrácii s WCMS. Navrhované riešenie poslúži ako cenný príspevok do tejto oblasti, preklenie medzeru medzi WCMS a Linuxom a ponúkne rozšírené možnosti správy používateľov a zdieľania údajov.

Kapitola 2

Analýza

2.1 Operačný systém

Operačný systém (OS) definujeme ako softvér, ktorý slúži na premostenie medzi používateľom počítača a jeho hardvérom. Je to softvér, ktorý riadi zdieľanie úloh medzi používateľmi a koordináciu hardvérových zdrojov. Operačný systém je súbor nástrojov, pomocných programov a systémových aplikácií, ktoré riadia počítačový hardvér a poskytujú všestranné služby pre klientsky aplikačný softvér. Akonáhle je operačný systém funkčný, spracovanie špecifík a možností zápisu sa stáva jeho hlavnou úlohou. Aby každý softvér fungoval správne, operačný systém bude pracovať v súlade s centrálnou procesorovou jednotkou (CPU), pamäťou (RAM) a úložiskom (pevný disk hdd alebo mechanika s nepohyblivým médiom teda ssd disk) každého počítača. Operačný systém spúšťa používateľské aplikačné programy a ponúka vhodné rozhranie na interakciu s hardvérom strojov. Primárnymi funkciami OS je správa počítačových zdrojov a regulácia toku údajov. Pamäť, procesory, vstupné/výstupné zariadenia a trvalé úložné zariadenia sú len niektoré z týchto zdrojov.

V dnešnej dobe existuje veľa operačných systémov, ktoré môžu byť zamerané pre server, smartfón, mikropočítač, osobné počítače a podobne. Medzi niektoré príklady patria:

1. Microsoft Windows
2. Apple macOS
3. Android OS
4. Linux
5. TempleOS

Naša bakalárska práca sa bude zameriavať na operačný systém Linux, konkrétne na linuxovú distribúciu Ubuntu.

2.2 WCMS

Používateľ môže spravovať digitálne informácie na webovej lokalite pomocou systému správy obsahu webu (WCMS), čo je typ systému správy obsahu (CMS), vývojom a správou materiálu bez predchádzajúcej znalosti webového programovania alebo markup jazykov.[12] V našom prípade to bude premostenie medzi operačným systémom Linux a používateľom cez webové rozhranie.

2.2.1 Fungovanie WCMS

Používatelia môžu spravovať, kontrolovať, meniť a rekonštruovať obsah na webovej lokalite pomocou WCMS. Používatelia môžu zostaviť materiál pomocou flexibilného jazyka ako XML alebo .NET a uložiť ho do databázy. Používatelia môžu použiť webový prehliadač na prístup k WCMS a potom použiť rozhranie založené na prehliadači na úpravu obsahu a prispôsobenie rozloženia.[12]

Dve základné časti WCMS:

1. **Aplikácia pre správu obsahu (CMA)** - je používateľské rozhranie, ktoré umožňuje používateľom navrhovať, upravovať, meniť a odstraňovať materiál z webovej lokality bez zásahu oddelenia IT. Medzi používateľov, ktorí môžu používať toto rozhranie, patria napríklad marketéri a tvorcovia obsahu.

2. **Aplikácia pre doručovanie obsahu (CDA)** - ponúka služby typu back-end, ktoré transformujú materiál, ktorý používatelia vytvárajú v CMA, na webovú stránku, ktorú si návštevníci môžu prezerať.

2.2.2 Základné nevýhody WCMS

1. **Požiadavky na úložný priestor** - Bežné webové stránky často obsahujú kombináciu textu, grafiky a fotografií. S rastúcim množstvom grafiky alebo obrázkov však rastie aj množstvo pamäte potrebnej na uloženie každej stránky. Výsledkom je, že ak sa nepoužije kompresia, na uchovanie celej stránky je potrebné väčšie množstvo úložného priestoru. V skutočnosti to má za následok obmedzenie počtu webových stránok, ktoré môžu byť prepojené so stránkou, ale tiež výrazne znižuje efektivitu triedenia a získavania údajov spojených s touto stránkou[25].

2. **Nízka flexibilita týkajúca sa inovácií webových stránok** - Webová stránka musí byť vždy dostupná pre firemných používateľov, ktorí chcú mať stálu online prítomnosť. Keď je však potrebné pridať nové stránky alebo zmeniť alebo odstrániť zastarané stránky, webové stránky vytvorené a udržiavané pomocou konvenčných metód zvyčajne vyžadujú uvedenie celej stránky do režimu offline.[25]

3. **Bezpečnostné riziká** - Pokiaľ nie sú zraniteľnosti WCMS neustále adreso-

vané administrátormi hackeri môžu diery v bezpečnosti využiť kedykoľvek. Správcovia musia sledovať a spravovať rôzne pohyblivé časti WCMS, vrátane MySQL, softvéru webového servera a akýchkoľvek doplnkov alebo doplnkov, aby sa znížili bezpečnostné hrozby.[12]

4. **Potreba špecializovaného tímu údržby webových stránok** - Technický tím správy databáz je často povinný spracovať údaje tak, aby dodržiavali potrebný formát, a pridať tieto údaje do databázy webovej stránky, aby mohol spravovať dátový obsah webovej stránky. Podľa zdroja[25] to výrazne zvyšuje náklady na údržbu webovej stránky a znižuje flexibilitu procesu aktualizácie, ako aj predlžuje čas potrebný na aktualizáciu webovej stránky, čím sa zvyšuje riziko, že údaje sú pri zverejnení na webových stránkach neaktuálne.

2.2.3 Základné Výhody WCMS

1. **Jednoduché na používanie** - WCMS sú zvyčajne jednoduché na používanie. Z toho dôvodu predstavujú veľkú výhodu pre ľudí, ktorí nie sú zručný v programovaní alebo s ním nemajú žiadne skúsenosti.

2. **Nízka cena** - Prevádzkové náklady na WCMS sú zvyčajne nízke v porovnaní s tým, čo ponúka používateľom alebo firmám. V niektorých prípadoch sa môže jednať aj o bezplatné predplatné.

3. **Nenáročné na spravovanie** - Väčšina WCMS je nenáročná na prevádzku z pohľadu administrátorov. Poskytujú celú radu nástrojov a možností, ako si napríklad upraviť pracovné toky či spravovať používateľov.

2.2.4 Bezpečnosť WCMS

Každý informačný systém, ktorý je pripojený na internet, musí byť bezpečný, inak môžu používatelia a operátori utrpieť vážne následky, ako napríklad

odcudzenie informácií o ich kreditnej karte alebo zákazníkovi. Open source WCMS sú pre útočníkov príťažlivým cieľom pre ich široké využitie. Používatelia so zlými úmyslami môžu spustiť útoky proti mnohým, ak nie všetkým, aplikáciám vytvoreným pomocou určitého WCMS, ak sa dozvedia o jeho zraniteľnosti.[19] WCMS sú zvyčajne podľa MDPI[18] cieľmi útokov ako: Manipulácia dát napríklad za pomoci SQL injekcie. Phishing dát ako bankové účty alebo iné používateľské dáta za pomoci aj XSS útoku. Spúšťanie kódu pomocou aj jednoduchých grafických súborov. Spam, kedy bežný webový "crawler"prechádza lokalitu a hľadá validné emailové adresy pre použitie u tohoto typu útoku. Napodobovanie WCM portálu, kedy útočníci použijú upravené formuláre na stránkach poskytovaných daným WCMS a čakajú, kým sa obete autorizujú, aby získanie ich prihlasovacích údajov. Poskytnuté informácie od MDPI[18] sa zhodovali aj s OWASP top 10, čo je zoznam naj využívanejších kyber útokov.[21]

Podľa analýzy údajov na obrázku 1.1[2], 49.8% týchto útokov na webové prehliadače sú útoky XSS, ktoré teda tvoria skoro polovicu napadnutí. Pri XSS útoku používateľ objaví spôsob, ako zadať časť škodlivého kódu na webovú stránku [19]. Inak povedané, útok XSS vloží do renomovanej webovej stránky zákernú sériu pokynov, ktoré sa vykonávajú vo webovom prehliadači návštevníka (bez vedomia návštevníka), čím útočníkovi poskytne prístup k citlivým údajom používateľa vrátane tokenov relácie a uložených súborov cookie, v prehliadači [27].

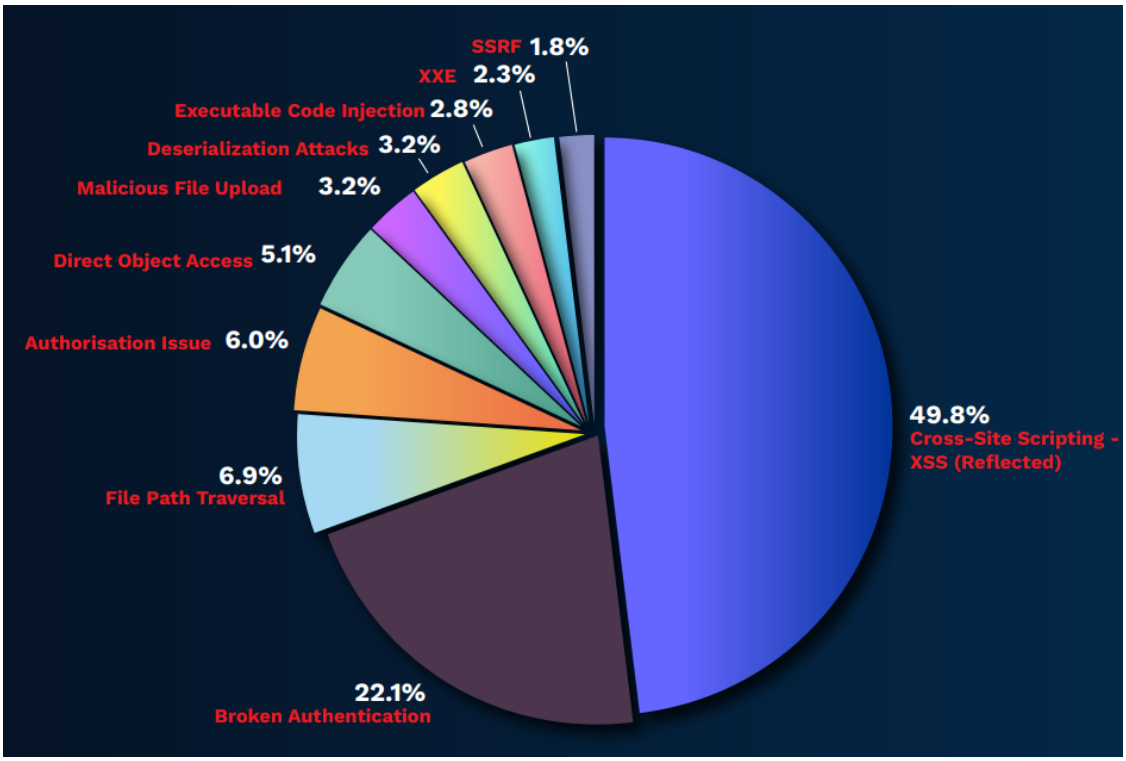
Niektoré varianty útoku XSS:

1. **Reflected XSS útok** - Tento útok používa iné komunikačné prostriedky, aby sa dostal k svojim cieľom, ako sú falošné odkazy v e-mailoch alebo iných webových stránkach, ktoré hlásia útok do webového prehliadača používateľa. Keďže skript pochádza z „dôveryhodného servera“, webový prehliadač ho môže spustiť. Netrvalé alebo XSS útoky typu I sú iným názvom pre tento

druh útoku.[10]

2. **Stored XSS útok** - Keď obeť odošle dotaz, škodlivý skript sa uloží niekde na webový server (napríklad do databázy, správy vo fóre, denníkov, komentárov atď.). Trvalé alebo XSS typu II sú ďalšie názvy pre tento typ útoku.[10]

3. **DOM-Based (Document Object Model) útok** - Na rozdiel od predchádzajúcich typov, kde skript servera spracováva údaje používateľa a vkladá ich späť na webovú stránku, tento druh injekcie vykonáva používateľ.[9]



Obr. 2.1: Distribúcia rôznych techník útoku na prehliadač. Obrázok prevzatý z [2].

Ďalším útokom s vysokým zastúpením bol podľa [2] práve útok porušenia autentifikácie. Jedná sa o útok súvisiacimi s autentifikáciou a potvrdením identity používateľa. Z analýzy údajov na obrázku 1.1[2], celkový počet týchto útokov tvoril až 22.1% celkových útokov.

Z informácií z dostupných článkov[12, 2, 27, 18] môžeme dedukovať že pred typmi útokov na Obr. 1.1, sa vieme chrániť napríklad nasledovne:

1. Vytváranie rutinnej zálohy WCMS (súbory a databázy).
2. Vedenie služieb v skúsenej hostingovej spoločnosti, aby sme sa vyhli útokom ako SQL injekcia.
3. Používanie najnovšej verzie WCMS a doplnkov.
4. Používanie špecializovaných bezpečnostných doplnkov ako JHackGuard.
5. Obmedziť prístup k súborom a priečinkom k administrácii.
6. Odstránenie inštalačných skriptov ako napríklad install.php
7. Vytvorenie bezpečných používateľských rolí a povinná zmena predvoleného hesla.
8. Povolenie captcha pre anonymných používateľov, aby sa zabránilo spamu.
9. E-mailové adresy by mali byť skryté, aby sa zabránilo nežiaducemu spamu.
10. Úprava nastavení globálnych parametrov webových stránok.
11. Pokiaľ je to možné, počas procedúry inštalácie je vhodná zmena predvolenej predpony databázy.
12. Nezobrazovať súkromné údaje WCMS v klientskom rozhraní.

2.2.5 Existujúce riešenia

Medzi existujúce riešenia, ktoré sú dostupné na internete patria napríklad:

1. HubSpot

2. WooCommerce
3. WordPress
4. Joomla
5. Wix
6. Drupal
7. BigCommerce
8. Ghost
9. Magento
10. Textpattern
11. TYPO3

2.2.6 WordPress

Najznámejším WCM systémom v dobe písania tejto práce je WordPress. Zdroj WordPress[7] uvádza, že až 42% web content management systémov používa práve túto platformu. Výhody pri používaní wordpress zahŕňajú: Blockový editor, ktorý zabezpečuje jednoduchosť pri implementácii web portálov. Používatelia, teda nemusia mať žiadne znalosti v oblasti programovania. Medzi výhody rovnako patria aj stovky pluginov a tém[7], ktoré sú platené alebo aj zdarma. Obrovskou výhodou WordPress je fakt, že to je opensource platforma. Používatelia, teda môžu vyhľadať rôzne komunitné skupiny v prípade, že narazia na nejaký problém a nemusia sa spoliehať na support team produktu. Aj napriek množstvu výhod, ktoré WordPress poskytuje, nie je to bez nevýhod. Bezpečnosť na webovom portáli si používatelia musia zabezpečovať sami. Je to spôsobené tým, že WordPress je práve opensource platforma. Z rovnakého dôvodu si vlastník WCMS bude

musieť hradiť aj vedenie doménového mena alebo, aj robenia záloh.

User Management - Je jedným z pluginov pre správu používateľov voľne dostupných v portáli WrodPress. Dáva možnosť spravovať používateľov a ich údaje z jedného dashboardu. Import, export a aktualizácia používateľských údajov pomocou rolí a filtrov. Okrem toho ponúka správu používateľov pre WordPress, ktorá umožňuje správcom webových stránok importovať alebo exportovať informácie o používateľoch cez CSV súbor.[24]

2.2.7 HubSpot

HubSpot je ďalším známym web content management systémom. Rovnako, ako WordPress ani pri používaní tohoto web content management systému používatelia nepotrebujú žiadne programátorské znalosti vďaka ich drag-and-drop editoru.[15]. Pre developerov ponúka HubSpot príkazový riadok, ktorý z vlastných skúseností s podobnou funckionalitou značne uľahčuje a urýchľuje prácu. Nakoľko sa jedná o platený produkt, obsahuje aj vbudované bezpečnostné funkcionality ako: Content delivery network (CDN), teda doručovanie obsahu, ktorý obsahuje citlivé údaje, chráni heslom. Zašifruje určité súbory na doručovanie obsahu a podobne. No rovnako produkt poskytuje aj web application firewall (WAF), a tak isto, aj dedikovaný bezpečnostný tím, ktorý zabezpečuje stránky pred DDoS útokmi, hakermi a inými bezpečnostnými porušeniami.[15]

Manažment používateľov pre HubSpot je zabudovaný a nie je potreba inštalácie žiadnych ďalších pluginov.[23] Pridávanie je riešené priamo cez nastavenia používateľov a teamov. HubSpot poskytuje rôzne templaty[23] pre nastavenie používateľských práv ako: Super admin, bežný používateľ, vedúci služby a podobne. Tieto práva sú aplikovateľné aj pre používateľské teamy, čo značne uľahčuje prácu nakoľko stačí raz nastaviť práva pre team a ďalej už len pridávať ľudí do daného

tímu.

2.3 Linux

Naša implementácia WCMS bude zahŕnať Linux. Jedná sa o open-source operačný systém navrhnutý pánom Linus Torvalds. Pre C, C++, Pascal, Modula-2 a 3, Oberon, Smalltalk a Fortran poskytuje špičkové kompilátory[6]. Existujú rôzne verzie editorov ako vi a Emacs. Virtuálna pamäť, multitasking, viacnásobné prihlásenia, zabezpečenie heslom a ochrana súborov sú plne podporované. Veľké siete teraz umožňujú vzdialené prihlásenie, vzdialené shelly a e-maily vďaka pokrokom v sieťovaní Linuxu[6]. Pre Linux bol vytvorený variant Network File System (NFS). To umožňuje zdieľanie súborového systému medzi niekoľkými počítačmi, takže spotrebúva menej miesta na pevnom disku a vyžaduje menej práce so správou systému[22]. Používatelia Linuxu majú prístup k systému na spracovanie textu ako TeX/LaTeX, ale aj kresliacim programom (ghostview a xdvi) a nástrojom na náhľad (xfig a idraw)[6]. Neplatia sa žiadne licenčné poplatky a všetky tieto funkcie sú bezplatné čo predstavuje obrovskú výhodu pre developmente napríklad WCM systému.

Medzi distribúcie Linuxu patria:

1. Kali Linux - Zameraný na digitálnu forenziu a penetračné testovanie.
2. Ubuntu - Pôvodne vydaný napríklad pre servery a osobné počítače.
3. Fedora Linux - Prispôsobený pre osobné počítače, cloud computing, servery a iné.
4. Arch Linux - Pre používateľov osobných počítačov ktorý chcú voľnosť vo svojom operačnom systéme.

5. Gentoo - Distribúcia zameraná pre power user-ov.

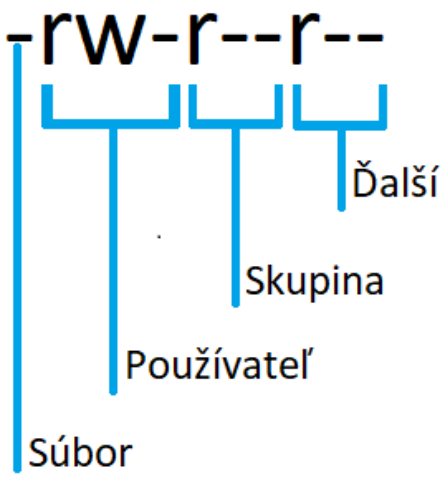
2.3.1 Manažment používateľov v operačnom systéme Linux

Pre začiatok v tejto časti budeme rozoberať povolenia súborov. Existujú 3 základné zaradenia používateľov na základe ktorých sa v Linuxe rozdeľuje vlastníctvo súboru. Patria sem používateľ, skupina a ďalší. Používateľ bude tá osoba, ktorá vytvorila daný súbor, teda vlastník súboru. Pod skupinou rozumieme začlenenie jedného alebo viacerých používateľov, ktorí budú mať rovnaké povolenia k súboru pokiaľ to nenastavíme inak pre jednotlivca. Skupiny zjednodušujú manažment používateľov nakoľko nám stačí nastaviť povolenia pre spúšťanie, čítanie alebo zapisovanie raz pre určitú skupinu a následne už len pridávať používateľov do danej skupiny. Do tretej skupiny zaradenia, ďalší, patria používateia, ktorí buď nevytvorili daný súbor alebo nie sú v žiadnej skupine, ktorá má jedno z troch spomínaných povolení k súboru. Linuxové povolenia možno rozdeliť na r(read teda čítanie), w(write, teda zapisovanie), x(execute, teda spúšťanie, napríklad skriptov a podobne) a nakoniec pomlčku, ktorá hovorí o tom, že dané zaradenie(používateľ, skupina, ďalší) nemá žiadne povolenie k súboru. Na obrázku 1.2 sme vy zobrazili, ako môžu vyzerať povolenia k súboru. Čiara súbor nám hovorí o druhu súboru. V prípade, že by sa jednalo o adresár, miesto pomlčky by sme tam videli písmeno d(directory). Ďalej na obrázku môžeme vidieť, že používateľ, teda tvorca súboru má povolenia pre čítanie(r) a zápis(w). Skupina a ďalší majú povolenia len na čítanie súboru. Z obrázku môžeme vidieť, že povolenia na spúšťanie nemá žiadne z troch zaradení. Povolenia môžeme v konzole zobraziť napríklad príkazom "ls -l". Povolenia k súboru by sme vedeli editovať príkazom chmod, ktorý by mohol vyzerať nasledovne: chmod u+x <názov súboru>. Príkaz by nám nahradil pomlčku u používateľa za x(execute). Povolenia by, teda vyzerali nasledovne: -rwxr–r–. Pre

odstránenie povolenia by sme v pôvodnom príkaze nahradili + za -: chmod u-x <názov súboru>. Výsledné povolenie by, teda opäť vyzeralo ako na obrázku 1.2. V príkaze chmod vieme ďalej nahrádzať zaradenia a písmená(rwx) aj číslami, ktoré predstavujú nasledujúce povolenia:

Číslo	**Povolenie**
0	Žiadne povolenie
1	Spúšťanie
2	Zapisovanie
3	Spúšťanie a zapisovanie
4	Čítanie
5	Čítanie a spúšťanie
6	Čítanie a zapisovanie
7	Čítanie, zapisovanie a spúšťanie

Príkladom upraveného príkazu chmod by bolo: chmod 777 <názov súboru>. Čísla v poradí predstavujú rovnakú hierarchiu ako na obrázku 1.2. Teda prvé číslo je pre používateľa, druhé pre skupinu a posledné pre ďalší. Predchádzajúci príkaz by nám nastavil povolenia nasledovne pri napísaní príkazu "ls -l": -rwxrwxrwx. Teda všetci zo zaradení by mali povolenie vykonávať každú operáciu. Takýto prístup sa všeobecne v praxi nevyužíva nakoľko to predstavuje bezpečnostnú hrozbu, ako to je opísané v pod sekcii 1.3.2, šiestej odrážke "Povolenia na prístup k súborom".



Obr. 2.2: Povolenia súboru v OS Linux.

Pre manažment používateľov v operačnom systéme Linux najskôr potrebujeme používateľov a skupiny. Ďalej sa, teda budeme zameriavať na ich vytváranie.

Používateľský účet v systéme vieme vytvoriť príkazom: sudo useradd <meno používateľa. Tento príkaz nám však vytvorí len používateľa bez domovského adresára bez možnosti používateľa prihlásiť sa. Aby sa však novo vytvorený používateľ vedel aj prihlásiť do svojho účtu musíme použiť nasledujúci príkaz: sudo useradd -m -s /bin/bash <meno používateľa>. Predchádzajúci príkaz nám okrem používateľa vytvorí aj jeho domovský adresár. Pre zaistenie bezpečnosti musíme nastaviť heslo používateľovi, čo dosiahneme príkazom: sudo passwd <meno používateľa>. Do konzole následne napíšeme heslo pre používateľa. Odstránenie používateľov v operačnom systéme vieme dosiahnuť dvomi spôsobmi. Prvý zahŕňa samostatné vymazanie používateľa a ďalej samostatné vymazania jeho domovského adresára. Druhý a lepší spôsob je vymazanie týchto dvoch častí naraz. Príkaz pre druhý spôsob by vyzeral nasledovne: sudo deluser --remove-home <meno používateľa>.

Ako sme už spomínali, používateľské skupiny v systéme Linux hrajú ob-

rovskú rolu z hľadiska jednoduchosti manažmentu používateľov. Znamená to, že rovnako, ako používateľom cez príkaz chmod aj skupinám nastavujeme povolenia r,w alebo x a následne prideľujeme používateľov do vytvorených skupín odkiaľ tieto povolenia zdedia. Vytváranie používateľských skupín realizujeme cez príkaz: sudo groupadd <meno skupiny>. Používateľov do skupiny následne pridávame za pomoci príkazu: sudo usermod -aG <meno skupiny> <meno používateľa>. Správnou praktikou je vytvorenie skupiny a nastavenie povolení predtým, ako pridáme používateľa do skupiny, aby sme predišli možným nepovoleným operáciám, ktoré používateľ vie uskutočniť a nemal by podľa hierarchie spoločnosti mať oprávnenia na vykonanie týchto operácií. Mazanie používateľov zo skupiny vieme vykonať cez príkaz: sudo gpasswd -d <meno používateľa> <meno skupiny>. V prípade, že niektorá z existujúcich skupín v systéme nie je potrebná, vieme ju vymazať príkazom: sudo groupdel <meno skupiny>.

2.3.2 Bezpečnosť operačného systému Linux

Táto časť vysvetľuje, ako chrániť systém Linux pred vnútornými aj vonkajšími útokmi. Tieto techniky môžu zahŕňať používanie úložísk na zabezpečenie systému, používanie anti-vírusu na kontrolu, či stiahnutý softvér nie je napadnutý vírusmi, opatrnosť pri spúšťaní softvéru Windows na systémoch Linux, aktualizáciu softvéru, konfiguráciu pravidiel brány firewall, správu hesiel a používanie rôznych prístupových povolení pre rôznych používateľov.

1. **Zabezpečenie prostredníctvom úložísk** - Softvér v linuxových distribúciách sa často sťahuje a inštaluje cez úložiská, ktoré obsahujú veľké množstvo balíkov, ktoré môžu používatelia používať a sťahovať. [13] Kali Linux napríklad obsahuje nástroje navrhnuté špeciálne na penetračné testovanie. [3] Pretože vývojári týchto linuxových distribúcií schválili repozitáre, ktoré sú

povolené s predvolenou dodávkou operačného systému, tento spôsob inštalácie softvéru je všeobecne považovaný za vysoko bezpečný. [26] Repozitáre však nedávajú všetko. Vo všeobecnosti by sa používatelia mali snažiť vyhnúť inštalácii softvéru z internetu alebo z iných zdrojov a namiesto toho sa zamerať na používanie repozitárov, ktoré ponúka linuxová distribúcia, ktorú si vybrali.

2. **Použitie antivírusu: ClamAV** - Pokiaľ používateľ nemá inú možnosť, ako stiahnuť softvér, ktorý sa nachádza mimo repozitára jeho distribúcie, mal by použiť antivírusový program. Dostupný priamo v repozitároch distribúcií alebo na oficiálnej stránke, ClamAV je jednou z možností.[26] Vo svojej podstate je podobný známemu antivírusovému programu Windows Defender. Oba obsahujú možnosť plánovania kontroly systému, v prípade, že používateľ obľubuje písanie skriptov vie ClamAV ignorovať dané repozitáre.[8] Obsahuje konzolovú verziu, ale aj verziu s GUI pre používateľov menej zdatných s konzolou. Databáza hrozieb pre ClamAV je neustále aktualizovaná a práve aj z tohoto dôvodu považujeme program za validnu možnosť pre Linux distribúcie.

3. **Aktualizácia softvéru** - Stiahnuté aplikácie aj systémové balíky je potrebné aktualizovať, aby bol systém Linux bezpečný. Pre aktualizáciu celého softvéru, ktorý bol získaný z úložísk, ako je prehliadač alebo jadro, je potrebné navštíviť buď webovú lokalitu, z ktorej bol softvér stiahnutý, a nainštalovať najnovšie verzie, alebo použiť terminál na vykonanie pre distribúciu špecifického sledu príkazov. Záverom možno povedať, že tragédiám, ako je strata údajov, sa dá predísť udržiavaním aktuálnych systémov Linuxu.

4. **Firewall** - Firewally môžu byť nainštalované na ochranu pred útokmi zo zariadení pripojených k rovnakej sieti. Firewall je súbor pravidiel, ktoré ur-

čujú, ktoré porty sú prístupné počítačom zvonku a ktoré správy môže lokálny počítač prenášať na iné počítače. [4] Napríklad SSH(Secure shell protocol) je sieťový protokol, ktorý umožňuje bezpečné spojenie medzi počítačmi cez sieť. Uskutočňovanie určitých spojení by však nemalo byť povolené, ak sú ostatní používatelia siete neznámi alebo ak ide o verejnú sieť umiestnenú vo verejnej oblasti. Pracovná stanica s príliš veľkým počtom otvorených portov je tiež zraniteľná voči útokom DDoS(Distributed denial of service), čo môže spôsobiť nefunkčnosť systému, kým nebude reštartovaný.[26] Pravidlá môžu byť nastavené na zastavenie takýchto útokov pomocou softvéru iptables, ktorý je prítomný vo väčšine linuxových repozitárov. Napríklad brána firewall určená výhradne pre domácu sieť môže byť menej prísna, aby umožňovala pripojenia ako SSH(Secure shell protocol). Pre verejné siete je možné vytvoriť nový súbor smerníc, ktoré povolia iba HTTP(Hypertext Transfer Protocol) a HTTPS(Hypertext Transfer Protocol Secure), aby boli aktívne, ale zakázali ostatným používateľom v tej istej sieti používať a útočiť na porty SSH(Secure shell protocol) a FTP(File transfer protocol). Keď sú tieto dve sady pravidiel vytvorené, možno ich zameniť pomocou príkazu „iptables-restore < rules“, kde pravidlá sú názov súboru pravidiel brány firewall, ktorý sa nachádza v aktuálnom pracovnom adresári.[26]

5. **Správa hesiel** - Používatelia sú často požiadaní, aby si zaregistrovali používateľské konto pri inštalácii distribúcie Linuxu na počítač, ktorý generuje adresár pre všetky údaje tohto používateľa v rámci súborového systému. Aby sa predišlo neúmyselnému alebo úmyselnému zničeniu dôležitých systémových súborov používateľa, tieto súbory musia byť uložené oddelene od súborov používateľa root. Okrem toho musia byť pre každého používateľa v systéme vytvorené samostatné účty, ak existuje niekoľko používateľov. Jednotlivé používateľské súbory môžu byť oddelené od seba vďaka tejto izolácii,

ktorá tiež zabraňuje neoprávneným používateľom v prístupe k údajom. Používateľské heslá a heslá root sa musia navzájom líšiť. Dobrou praktikou je používať silné heslá, ktoré ideálne používateľ nepoužíva nikde inde.

6. **Povolenia na prístup k súborom** - V systéme musia byť nakonfigurované rôzne povolenia, aby sa ostatným používateľom zabránilo v prístupe k súborom, ku ktorým by nemali mať prístup. Na dokončenie je možné použiť príkazy ako chmod. Príkaz chgrp je možné použiť aj na vytváranie skupín používateľov, aby bolo možné nastaviť povolenia pre viacerých používateľov naraz. Používatelia nebudú môcť manipulovať alebo spúšťať súbory alebo potenciálne meniť systém spôsobom, akým by nemali, nastavením povolení pre osoby, ktoré môžu čítať, zapisovať a spúšťať súbory.[26]

2.4 Backend rámce

Nasledujúca sekcia obsahuje informácie o populárnych rámcoch Pythonu - Flask, Django a FastAPI. V texte tiež uvádzame, podrobnejšie porovnanie a zdôvodnenie, prečo sme si FastAPI vybrali namiesto ostatných rámcov. Flask, Django a FastAPI sú populárne rámce Pythonu, ktoré sa často používajú pri vývoji webových stránok. Každý z týchto rámcov má svoje vlastné silné stránky a unikátne vlastnosti, ktoré ich robia vhodnými pre rôzne typy projektov.

2.4.1 Flask

Flask je ľahký a flexibilný rámec, ktorý sa zameriava na jednoduchosť a minimalizmus. Jeho hlavnou výhodou je, že poskytuje základné nástroje a funkcionality pre vývoj webových aplikácií, ale zároveň je veľmi prispôsobiteľný. Je ideálny pre malé až stredne veľké projekty, kde je dôležité mať rýchly a jednoduchý vývojový cyklus.

Flask je ľahký a flexibilný mikrorámec, čo znamená, že poskytuje len základné komponenty na vývoj webových stránok. Má minimalistický dizajn a umožňuje vývojárom väčšiu kontrolu nad štruktúrou kódu. Flask je vhodný pre malé až stredne veľké aplikácie, prototypy a projekty, ktoré si vyžadujú prispôsobenie. Má relatívne plochú krivku učenia a poskytuje veľkú flexibilitu, čo umožňuje vývojárom robiť vlastné rozhodnutia týkajúce sa knižníc a komponentov. Flask nemá niektoré funkcie öut-of-the-box", napríklad integráciu s databázou a validáciu formulárov, ktoré môžu vyžadovať ďalšie knižnice[20]. Je vysoko rozšíriteľný a podporuje používanie rozšírení tretích strán na pridanie funkcií. Flask často uprednostňujú vývojári, ktorí uprednostňujú jednoduchosť, kontrolu a odľahčený rámec.

2.4.2 Django

Django je plnohodnotný rámec, ktorý sa riadi zásadou "batérie sú súčasťouä poskytuje komplexnú sadu funkcií hneď po vybalení. Obsahuje ORM (Object-Relational Mapping) na interakciu s databázou, šablónovací engine a zabudované administrátorské rozhranie na správu obsahu[11]. Django sa riadi prístupom convention-over-configuration, čo znamená, že má preddefinovanú štruktúru a konvencie, ktoré uľahčujú rýchly vývoj. Má vynikajúcu dokumentáciu a veľkú komunitu, vďaka čomu je ľahké nájsť riešenia a zdroje. Django je vhodný pre komplexné aplikácie, webové stránky s veľkým množstvom obsahu a projekty, pri ktorých je rozhodujúca bezpečnosť a škálovateľnosť. Môže mať strmšiu krivku učenia sa kvôli rozsiahlemu súboru funkcií a konvencií. Django uprednostňujú vývojári, ktorí oceňujú produktivitu, škálovateľnosť a zabudované funkcie.

2.4.3 FastAPI

FastAPI je relatívne nový rámec, ktorý si získal značnú popularitu vďaka svojmu vysokému výkonu a moderným funkciám. Je postavený nad rámcom Starlette a využíva asynchrónne programovanie na dosiahnutie výnimočného výkonu. FastAPI poskytuje automatické generovanie dokumentácie API pomocou OpenAPI (predtým Swagger) a zabudovanú podporu validácie schémy JSON. Má vynikajúcu podporu pre typové anotácie a využíva typové nápovedy jazyka Python, ktoré umožňujú automatickú validáciu a serializáciu údajov. FastAPI je navrhnutý tak, aby zvládal vysoké zaťaženie, vďaka čomu je vhodný na vytváranie robustných a škálovateľných rozhraní API. Dobre sa integruje s ďalšími rámcami jazyka Python, ako je napríklad SQLAlchemy na interakciu s databázou. FastAPI uprednostňujú vývojári, ktorí uprednostňujú výkon, typovú bezpečnosť a efektívne spracovanie koncových bodov API.

FastAPI sme si vybrali miesto ostatných uvedených rámcov z niekoľkých dôvodov, ktoré súvisia s našimi špecifickými požiadavkami. Jedným z hlavných dôvodov je voľnosť, ktorú ponúka, pretože nevnucuje vopred definovanú štruktúru ako Django. Táto flexibilita nám umožňuje prispôsobiť architektúru aplikácie našim špecifickým potrebám a preferenciám. Okrem flexibility poskytuje FastAPI aj určité funkcie, ktoré nie sú dostupné vo Flasku, a preto je pre náš projekt vhodnejšou voľbou. Tieto dodatočné funkcie zlepšujú proces vývoja a prispievajú k celkovej efektívnosti webovej aplikácie. Celkovo sa kombinácia flexibility, súboru funkcií a výkonu FastAPI stala optimálnou voľbou pre potreby nášho špecifického projektu, čo nám umožnilo vytvoriť vysoko prispôsobiteľnú a efektívnu webovú aplikáciu.

2.5 Autentifikácia a autorizácia

V nasledujúcej sekcii sa pozrieme na overovanie a prideľovanie oprávnení používateľom a taktiež spôsoby akými sa to dá dosiahnuť. Autentifikácia a autorizácia spolupracujú na zaistení bezpečného prístupu k systémom a zdrojom. Autentifikácia overuje identitu, zatiaľ čo autorizácia určuje úroveň prístupu udeleného autentifikovaným používateľom. Kombináciou týchto dvoch procesov môžeme presadzovať bezpečnostné politiky, chrániť citlivé informácie a kontrolovať činnosti používateľov v rámci našej aplikácie a systému.

2.5.1 Autentifikácia

Autentifikácia je proces overovania identity používateľa alebo systému. Zabezpečuje, že subjekt, ktorý žiada o prístup k prostriedku, je tým, za koho sa vydáva. Autentifikácia zahŕňa poskytovanie poverení, ako sú používateľské mená a heslá, biometrické údaje alebo digitálne certifikáty, na preukázanie totožnosti. Proces overovania sa zvyčajne uskutočňuje na začiatku relácie alebo pri prístupe k zabezpečeným prostriedkom.

2.5.2 Autorizácia

Autorizácia je proces udelenia alebo zamietnutia prístupu ku konkrétnym prostriedkom alebo akciám na základe oprávnení autentifikovaného používateľa. Po určení identity používateľa prostredníctvom overenia sa autorizáciou určuje, čo môže používateľ robiť alebo k čomu má prístup. Zahŕňa nastavenie oprávnení a definovanie kontrol prístupu, aby sa zabezpečilo, že používatelia môžu vykonávať len tie činnosti alebo pristupovať len k tým zdrojom, ktoré sú oprávnení používať.

2.5.3 Použité metódy JWT (JSON Web Token) a OAuth2

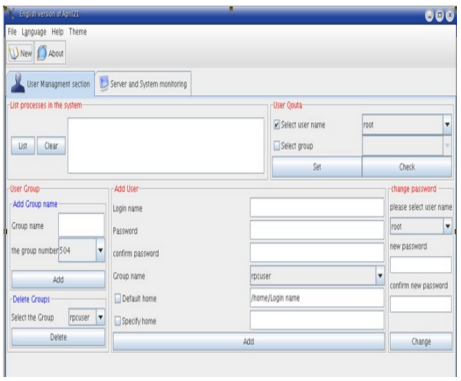
1. **Účel a rozsah pôsobnosti:** JWT je formát tokenu, ktorý reprezentuje tvrdenia kompaktným a samostatným spôsobom. Zameriava sa predovšetkým na poskytovanie bezpečného spôsobu prenosu a ukladania informácií medzi stranami[16]. OAuth 2.0 je rámec, ktorý definuje protokoly a pracovné postupy na autorizáciu a delegovanie prístupu. Umožňuje používateľom udeliť obmedzený prístup k svojim zdrojom aplikáciám tretích strán bez toho, aby museli zdieľať svoje prihlasovacie údaje[14].

2. **Funkčnosť:** JWT sa používa predovšetkým na overovanie a výmenu informácií. Poskytuje mechanizmus na bezpečný prenos tvrdení medzi stranami a overovanie ich pravosti[16]. OAuth 2.0 sa používa predovšetkým na autorizáciu a delegovanie prístupu. Definuje protokoly na získavanie a používanie prístupových tokenov na autorizáciu aplikácií tretích strán na prístup k chráneným zdrojom[14].

2.6 Porovnanie dostupných WCMS funkcií na manažment používateľov s Linuxom

Podobným riešením našej bakalárskej práce bol projekt z roku 2006 s názvom JLAT(Java Linux administration tool).[5] Jednalo sa o projekt, ktorý mal za úlohu administráciu operačného systému Linux cez rozhranie napísané v jazyku Java. Na obrázku 1.3[5] je ukázané, ako nástroj vyzeral. JLAT bol projekt, ktorý nefungoval na WCMS, ako to bude v prípade našej bakalárskej práce, ale miesto toho to bola aplikácia, ktorú si používateľ mohol nainštalovať do operačného systému Linux. Jednou nevýhodou JLAT bolo používanie. Nástroj mohol používať len super user[5](používateľ počítačového systému so špeciálnymi oprávneniami po-

trebnými na správu a údržbu systému; správca systému) nakoľko mnoho príkazov, ktoré boli implementované do GUI potrebovali sudo oprávnenia. V našom riešení sa chceme vyhnúť podobným problémom nakoľko naša implementácia WMCS by mala fungovať aj pre koncových používateľov.



Obr. 2.3: Java Linux administration tool GUI. Obrázok prevzatý z[5].

2.7 Stručný opis riešenia

Na implementáciu našej bakalárskej práce sme sa rozhodli pre linuxovú distribúciu Ubuntu vo verzii 22.04, ktorá vychádza z distribúcie Debian. Distribúciu Linuxu sme volili aj z dôvodu jednoduchosti inštalácie a používania oproti iným distribúciám, ako napríklad Arch Linux. Ďalším dôvodom prečo sme volili spomínanú distribúciu bola stabilita tohoto operačného systému. Dôležitým faktorom pri výbere bola aj aktuálnosť a popularita. Nakoľko je Ubuntu neustále aktualizované a zároveň najpoužívanejšie medzi linuxovými distribúciami, zaručí nám to aktuálne pomôcky pri riešení našej bakalárskej práce. Pre vyriešenie posielania požiadaviek na server sme volili program Postman.

Kapitola 3

Opis riešenia

3.1 Opis požiadaviek

V nasledujúce sekcii sa budeme zaoberať opisom funkčných a nie-funkčných požiadaviek našej práce. Rozoberieme ich oddelene so zameraním na požiadavky našej práce. Do funkčných požiadaviek sme zaradili špecifikácie a potreby na funkcionalitu a správanie implementovaného systému. Nie-funkčné požiadavky sme zamerali na vlastnosti a kvalitu systému, ktoré priamo nesúvisia s funkcionalitou.

3.1.1 Funkčné požiadavky

1. Prvú funkčnú požiadavku sme si určili ako prevenciu neautorizovaného prístupu do infraštruktúry, aplikácie alebo k dátam.
2. Ako ďalšiu požiadavku v tejto kategórii sme zaradili ukladanie údajov používateľov a poverení.
3. V našom projekte budeme ďalej zabezpečovať poskytovanie prihlasovacieho

mechanizmu pre koncových používateľov.

4. Systém je bez používateľov len prázdnou entitou. A preto sme si ako funkčnú požiadavku určili aj registráciu, nastavovanie a obnovenie hesiel.

5. Keďže budeme administrátormi nami vytvoreného systému, ako požiadavku sme si určili aj prideľovanie používateľských prav k systémom a službám

6. V neposlednej rade požiadavka ktorá vychádza z predchádzajúcej bude správa užívateľských oprávnení v rámci služieb a systémov.

3.1.2 Nie-funkčné požiadavky

Do nie-funkčných požiadaviek radíme rýchlosť, správnosť a efektívnosť. Môžeme ich aplikovať napríklad na autorizácie keďže chceme, aby používateľ nečakal dlho pri prihlasovaní a rovnako, aby sa správne kontrolovali údaje poskytnuté používateľom s databázou používateľov. Jednoduchosť používania sme rovnako zaradili do tohoto druhu požiadaviek. Chceme, aby administrácia používateľov v našom WCMS bola priamočiara a nespôsobovala ťažkosti správcom. Keďže aplikácie sú často neprehľadné naším cieľom bude vytvoriť administráciu, v ktorej sa používatelia/administrátori nebudú strácať a zároveň bude aj ľahko naučiteľná a jednoducho zapamätateľná pre opakované používanie.

3.2 Návrh

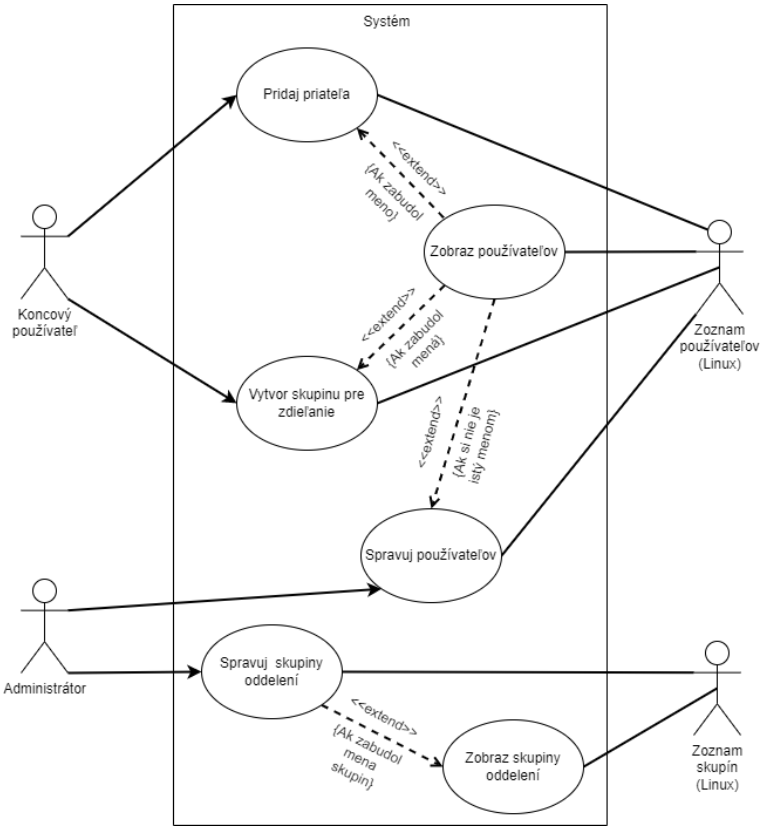
Naše riešenie budeme realizovať cez spomínanú distribúciu Linuxu, Ubuntu. Náš backend bude tvoriť webový rámec FastAPI a bude spolupracovať s programom Postman. Spomínaný Postman nám bude poskytovať požiadavky, ktoré bude následne posielať na náš backend, ktorý bude požiadavky spracovávať a následne na ne odpovedať. Výslednú odpoveď pošle, FastAPI opäť na Postman. Registráciu

plánujeme riešiť ako žiadosť na backend, čo bude tvoriť serverovú stranu, ktorá overí či sa daný používateľ už nenachádza v naše databáze a v prípade, že nie, vytvorí nového s poskytnutými informáciami. V prípade, že používateľ existuje (pod poskytnutím menom) backend pošle odpoveď, ktorá bude obsahovať správu o chybe. Rovnaká odpoveď bude zaslaná na Postman aj v prípade, že používateľ zadal heslo v zlom tvare, teda použil nepovolené znaky, bolo príliš krátke/dlhé a podobne. Po vytvorení používateľa mu backend zároveň aj vytvorí a pridelí adresár, ktorý bude obsahovať používateľovu prácu. Zároveň sa vytvorí skupina používateľovi priatelia. Do priečinka nebude mať prístup nikto iný okrem konkrétneho vlastníka priečinka a administrátorov. V prípade, že už používateľ účet má pokračuje priamo k prihláseniu. Autentifikáciu budeme riešiť voči backendu(serveru), kde sa skontroluje JWT token. Pridanie do skupiny, ako priatelia bude musieť používateľ poznať meno druhej osoby, ktorú si chce pridať. V opačnom prípade má možnosť zavolať koncové body pre vypísanie všetkých nesystémových účtov. Po odoslaní požiadavky sa používateľovi môže vrátiť hláška s úspešnou akciou alebo nejaký druh HTTP chyby podľa druhu požiadavky. Používateľské a skupinové oprávnenia sa budú držať v rámci Linuxu a my k ním budeme len pristupovať cez príkazy pre zjednodušenie riešenia.

3.3 Implementácia

Pri implementácii sme zvolili Linux, v súlade s naším pôvodným návrhom. Pre dosiahnutie potrebnej funkcionality sme použili rámec FastAPI, ktorý bol detailne opísaný v sekcii 1.5 Analýzy, konkrétne v podsekcii FastAPI. V programe Postman sme pridali jednotlivé koncové body, ktoré následne voláme prostredníctvom grafického rozhrania. Po vykonaní požiadavky zobrazujeme výsledok požiadavky v Postmanovi. V závislosti na správnosti požiadavky sa môžeme stretnúť s rôz-

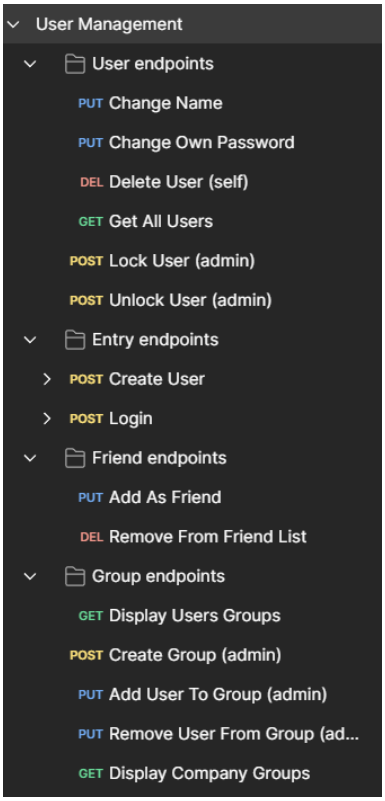
nymi typmi HTTP chýb, ako napríklad neautorizovanou požiadavkou, nesprávnou požiadavkou alebo chybou z Linux konzoly a úspešným výsledkom požiadavky. Napríklad pri vytvorení skupiny zobrazíme správu: "Skupina <názov skupiny> bola úspešne vytvorená". Podrobnejšie popisy jednotlivých typov požiadaviek sú uvedené v nasledujúcich pod-sekciách. Ďalšou odlišnosťou od pôvodného návrhu bola implementácia databázy. Vzhľadom na rozsah tejto bakalárskej práce sme sa rozhodli nezahrnúť do našej práce integrovaný databázový server. Rozhodli sme sa tak preto, že systém Linux nám poskytoval väčšinu potrebných informácií pre implementáciu, s výnimkou JWT tokenov, ktoré sme však jednoducho ukladali do súboru pre jednoduchosť implementácie. Na obrázku 3.1 Opisujeme manažment skupín v našom riešení manažmentu skupín.



Obr. 3.1: Manažment skupín a používateľov.

3.3.1 Programová implementácia riešenia

Pre účely tejto bakalárskej práce sme zvolili Linux s konkrétnou verziou 22.0.4, ktorá nám poskytla všetky potrebné nástroje a prostredie na implementáciu nášho riešenia. Tento výber nám umožnil vyhnúť sa ďalším úpravám a modifikáciám, pretože Linux distribúcia sama o sebe poskytovala všetky potrebné komponenty pre náš FastAPI projekt. Celý systém je nasadený na virtuálnom súkromnom serveri (VPS), na ktorý priamo z programu Postman posielame požiadavky, ktoré sme definovali podľa potrieb našej práce. Na implementáciu logiky nášho riešenia sme využili vývojové prostredie PyCharm vo verzii 2022.3.2. Vzhľadom na to, že FastAPI nemá pevne stanovenú štruktúru súborov, mali sme voľnosť v tom, ako sme organizovali náš projekt.



Obr. 3.2: Rozdelenie koncových bodov v programe Postman.

Rozhodli sme sa ho rozdeliť do logických celkov, ktoré najviac zodpovedali našim potrebám a ukazovali sa ako najviac zmysluplné. Presný popis štruktúry súborov je uvedený v technickej dokumentácii, ktorá sprevádza našu prácu. Ďalej sme na obrázku 3.2 ukázali rozmiestnenie ednpointov v rámci programu Postman.

3.3.2 Autentifikácia a autorizácia

Táto časť sa zameriava na implementáciu spoľahlivých autentifikačných a autorizačných mechanizmov na zabezpečenie bezpečnej kontroly prístupu a ochrany citlivých údajov v systéme. Pri implementácii autentifikácie sme zvolili JWT tokeny, ktoré vytvárame pomocou Python knižnice "jwt", ktorú využívame na generovanie, ale aj dekódovanie týchto tokenov v prípade, že si potrebujeme z nich vybrať nejaké informácie, ako meno používateľa, ktorý požiadavku na server vytvoril. Tokeny držíme v súbore, ktorý sa nachádza na servery. Pri prihlásení používateľa sa naša API pozrie do súboru a hľadá token. V prípade, že tam rovnaký token nájde, len zmení trvanlivosť tokenu podľa hodnoty, ktorú definujeme v .py súbore. Pokiaľ sa tam token nenachádza, zapíše ho do tohoto súboru. V rámci správy tokenov ďalej na pozadí API beží proces z knižnice „asyncio“, ktorý každých tridsať sekúnd kontroluje trvanlivosť tokenov a pokiaľ nájde token ktorému skončila platnosť vymaže ho zo súboru. Zmena v súbore ďalej nastáva aj pri zmene mena používateľa. Token jednoducho prepíšeme. Autorizáciu vyžaduje každý nami definovaný koncový bod, aby sme predišli neautorizovaným požiadavkám na server. V požiadavke sa, teda musí nachádzať token, ktorý sa musí nachádzať v súbore s aktívnymi tokenmi. To či sa samotný token nachádza v súbore sa kontroluje za pomoci funkcie, ktorá vráti odpoveď neplatný token v prípade, že sa používateľov token v tomto súbore nenachádza. Všetky požiadavky musia byť od prihláseného používateľa. Formát tokenu definuje schéma „oauth2_scheme“, ktorá sa používa pri každom koncovom bode s výnimkou prihlásenia. V prípade prihlásenia pou-

žívame „OAuth2PasswordRequestForm“, čo definuje štruktúru informácií v požiadavke.

Proces autorizácie sme implementovali ako koncový bod, kde sa volá samotná funkcia pre autorizáciu používateľa. Návratová hodnota koncového bodu sa posiela podľa výsledku z funkcie, ktorá je volaná. Obrázok 3.3 zobrazuje implementáciu.

```
@router.post("/user/auth")
async def auth_user(token: str = Depends(token_verification_middleware)):
    response = authorize_user(token)
    return response
```

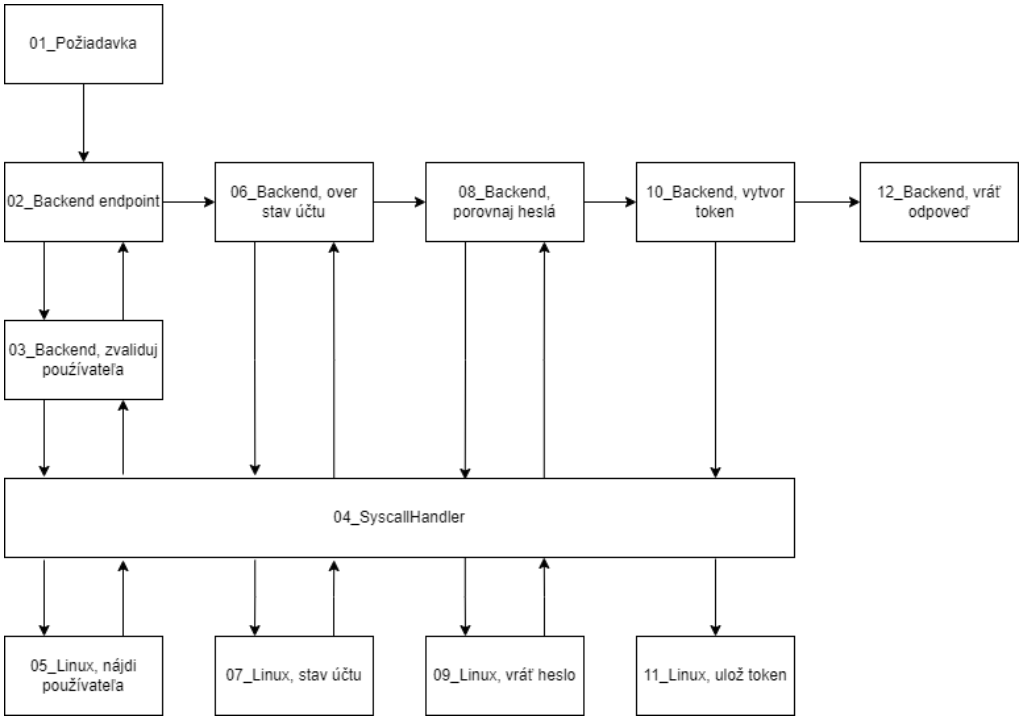
Obr. 3.3: Autorizačný koncový bod.

```
def authorize_user(token: str):
    # Verify the JWT token
    try:
        payload = jwt.decode(token, JWT_SECRET_KEY, algorithms=[JWT_ALGORITHM])
    except jwt.DecodeError:
        raise HTTPException(status_code=401, detail="Invalid token")

    # Check if user is admin
    command = f"cat /etc/group | grep administrator | grep {payload}"
    user_manager.run(command)

    if valid_user(payload) and user_manager.get_stdout():
        return {"user_type": "admin", "payload": payload}
    elif valid_user(payload):
        return {"user_type": "user", "payload": payload}
    else:
        return{"Message": "Invalid user", "payload": -1}
```

Obr. 3.4: Funkcia pre autorizáciu.

Na obrázku 3.4 je funkcia ktorú volá koncový bod autorizácie. Ako parameter dostane token a najprv ho dekóduje. Nasleduje príkaz pre spustenie do Linux

Bashu ktorého výstup overuje či poslaný token patrí administrátorskému účtu. Podmienky nachádzajúce sa na konci funkcie overujú či sa skutočne jednalo o administrátorský účet. Pokiaľ nie, nasleduje podmienka ktorá kontroluje používateľa. V prípade že sa meno získané z tokenu nachádza v zozname Linuxu vraciame odpoveď http 200 OK. Prípad kedy neprešla ani jedna z podmienok vyššie a vykonáva sa else označuje neznámeho používateľa v systéme Linux čo sa zobrazí v odpovedi.



Obr. 3.5: Autentifikácia používateľa cez koncový bod Login.

Na obrázku číslo 3.5 sme vyzobrazili proces autentifikácie používateľa v našom programe. Komunikáciu medzi našou API a Linux serverom zabezpečuje (04_SyscallHandler) modul[17]. Proces začína požiadavkou (01_Požiadavka) na backend (02_Backend endpoint) odkiaľ sa najprv validuje (03_Backend, zvaliduj používateľa) či sa daný používateľ nachádza v systéme Linux (05_Linux, nájdi po-

užívateľa). Pokiaľ táto kontrola prejde proces ide na overenie stavu účtu (06_Backend, over stav účtu). To znamená kontrolu uzamknutia účtu. Môže sa stať, že používateľ porušil pravidlá a administrátor mu uzamkol účet, čo mu znemožní prihlásenie. Tento stav účtu sa do Linuxu zapisujem pridaním znaku „!“ k používateľovmu účtu (07_Linux, stav účtu). Pokiaľ kontrola prejde backend ďalej porovnáva heslo poskytnuté používateľom s heslom vyžiadaním z Linuxu (09_Linux, vráť heslo) pre dané používateľské meno (08_Backend, porovnaj heslá). Proces pri úspešnom porovnaní pokračuje na vytvorenie JWT tokenu (10_Backend vytvor token), ktorý zapisuje do súboru umiestnenom v Linuxe (11_Linux, ulož token). Posledný stav procesu posiela odpoveď na vytvorenú požiadavku (12_Backend, vráť odpoveď. Pokiaľ ktorýkoľvek z procesov počas autentifikácie zlyhá, backend posiela odpoveď okamžite na API odkiaľ bola požiadavka vytvorená.

Zároveň sa všetky koncové body autorizujú voči načítanému súboru obsahujúceho aktívne tokeny používateľov. Token sa do tohoto súboru zapisuje po prihlásení používateľa a má platnosť jeden týždeň. Na obrázku 3.6 sme vyzobrazili funkciu, ktorú volá každý koncový bod pri požiadavke na daný koncobý bod. Najskôr si tokeny uložíme z funkcie, ktorá otvára súbor nahraný na VPS a druhým krokom je for cyklus, ktorý kontroluje či sa poskytnutý token z parametrov zhoduje so záznamami z načítaného súboru. Pokiaľ sa nenájde vraciame používateľovi odpoveď z http chybou 401 a správou neplatný token.

```python
def token_verification_middleware(token: str = Depends(oauth2_scheme)):
    # Load the token data from the active_tokens.json file
    tokens_data = read_tokens_file()

    # Check if the token exists in the loaded token data
    if token not in [token_data["token"] for token_data in tokens_data]:
        raise HTTPException(status_code=401, detail="Invalid token")

    return token
```

Obr. 3.6: Funckia pre overenie JWT tokenu.

Autentifikácia sa volá pri procese prihlasovania, ktorý opisujeme nižšie. Jedná sa o samotnú funkciu pre tento proces a triedu so statickou metódou, ktorú funkcia volá. Prvé čo sa vykonáva je zavolanie metódy čo pošle príkaz na získanie hesla používateľa zo systému Linux. Odpoveď sa vracia do funkcie ako premenná, ktorú pomocou knižnice crypt[1] a hmac porovnávame s poskytnutím heslom v požiadavke na server. Knižnica najskôr zahashuje heslo z požiadavky v tvare reťazca a potom ho porovná s heslom, ktoré sa vrátilo z triedy. Hash hesla sa vykonáva podľa saltu, ktorý si heslo z triedy drží v sebe. Obrázky 3.7 pre triedu a 3.8 pre funkciu ukazujú implementáciu spomínaných komponentov.

```
class HashedUserPassword:
    2 usages  xkello
    @staticmethod
    def execute_command(username: str) -> str:
        user_manager.run(f"sudo getent shadow | grep {username}")
        return user_manager.get_stdout().strip().split(':')[1]
```

Obr. 3.7: Trieda so statickou metódou.

```
def authenticate_user(username: str, password: str):
    # Command to execute, execution and stdout
    passwd = HashedUserPassword.execute_command(username)

    # Compare user's hashed password from linux with plain text password from request
    if compare_hash(crypt(password, passwd), passwd):
        return True
    else:
        raise HTTPException(status_code=401, detail="Invalid password")
```

Obr. 3.8: Funkcia pre autentifikáciu.

3.3.3 Správa skupín

V nasledujúcej sekcii sa pozrieme na opis riešenia v rámci správy skupín. Tu sme využili zabudované funkcionality linuxovej distribúcie Ubuntu. To znamená, že v našom prevedení sme riešili pridávanie do skupín, odoberanie zo skupín, mazanie

skupín no nie držanie informácií o skupinách. Pre ukázanie povolení v rámci správy skupín sme zvolili vennov diagram, ktorý sa nachádza na obrázku 3.9.

Obr. 3.9: Štruktúra správy skupín.

Niektoré koncové body, ako vytváranie skupín v rámci štruktúry sme povolili len administrátorom. Urobili sme tak nakoľko sme riešenie prispôsobovali nejakej fiktívnej spoločnosti. To znamená, aby nám v riešení nepribúdali nadbytočné skupiny v rámci oddelení spoločnosti, ktoré môžu mať podobný názov ako dané oddelenie. Ide o ochranu pred vytváraním skupín s veľmi podobným názvom, ktoré by mohli pomýliť niektorých používateľov. Jedine tieto koncové body majú voľne definovateľný názov skupiny. Pri volaní chránených enpointov tohoto typu sa do funkcie .run()[17] posiela s Bash príkazom aj meno používateľa, ktorý chce príkaz spustiť. V prípade, že používateľ má skupinu definovanú v priečinku /etc/sudoers.d/ dostáva oprávnenie na spustenie príkazu. Taktiež však do týchto príkazov musíme definovať, že sú spúšťané pod sudo používateľom. Spomínaný priečinok bolo nutné editovať cez Linux Bash priamo. Na obrázku 3.10 ukazujeme, ako sme upravili /etc/sudoers.d/admins súbor pre splnenie opísanej funkcionality.

```
%administrator ALL=NOPASSWD: /usr/bin/dmesg, /usr/sbin/usermod, /usr/sbin/
groupadd, /usr/bin/gpasswd, /usr/bin/grep, /usr/sbin/cat, /usr/sbin/grep
```

Obr. 3.10: Súbor admins pre konfiguráciu oprávnení administrátorov.

Okrem tejto pevnej štruktúry skupín sme ďalej za definovali niečo podobné zoznamu priateľov. Ide o skupiny, ktoré si používatelia spravujú samy. Každému používateľovi sa pri vytvorení účtu zároveň vytvorí aj skupina <používateľ>_friends. Pridávanie do skupiny priatelia funguje cez koncový bod /addfriend. Ide o predprípravu pre časť III v rámci našej témy, zdieľaný prístup k dátam. Výhodou týchto zoznamov priateľov je, že aj v prípade ne-zdieľania skupiny v rámci oddelenia môžu byť súbory posielané medzi používateľmi bez potreby riešenia linuxových ACL (Access Control Lists) relatívne jednoduchým spôsobom. Ďalej sme v implementácií poskytli možnosť vytvárania skupín one-to-many. Pre tvorenie skupín tohoto typu sme implementovali koncový bod, no rovnako aj Bash skript pre rozšírenie rozmanitosti práce. Samotné funkcie skriptu sa nachádzajú na obrázku 3.11. Funguje na princípe delenia reťazca zo vstupu na časti. Obsahuje niekoľko funkcii pre oddelenie jednotlivých akcii, ktoré vykonáva a zároveň pre zlepšenie estetiky kódu. Prvou časťou skriptu je funkcia, ktorá slúži na rozdelenie reťazca znakov pre použitie v pridávaní do skupiny. Nasleduje funkcia na generovanie náhodného reťazca a inicializovanie mena skupiny zatiaľ do premennej. Tretia funkcia v skripte slúži na vytvorenie skupiny a priradenie používateľa, ktorý zavolal koncový bod alebo samotný skript. V poslednej funkcii sa pridávajú používatelia do už vytvorenej skupiny. Main slúži na nastavenie poradia, v ktorom sa funkcie vykonajú. Aby skript volaný koncovým bodom alebo priamo z Linux Bashu prešiel úspešne musia byť všetky časti reťazca poskytnutého v požiadavke korektne napísané. Meno skupiny sa rovnako ako pri priateľoch definuje pevne, teda obsahuje meno používateľa, ktorý o založenie skupiny požiadal a náhodný reťazec znakov o dĺžke 10. Reťazec sme volili pre jednoduchosť implementácie. Formát ta-

kýchto skupín je: <meno odosielateľa požiadavky>_to_<reťazec znakov o dĺžke 10>.

```
# Split string into needed variables for successful group creation
function split_string(){

  owner=$(echo "$input_str" | cut -d'_' -f1)
  users_str=$(echo "$input_str" | cut -d"_" -f3)
  IFS='-' read -r -a users <<< "$users_str"
}

# Generate random string and assign group name to variable
function assign_group_name(){

  rand_string=$(echo $RANDOM | md5sum | head -c 10; echo;)
  group_name="${owner}_to_${rand_string}"
}

# Create group and add her creator
function bash_commands(){

  groupadd "$group_name"
  usermod -aG "$group_name" "$owner"
}

# Add all the other users to the created group in a loop
function add_users_to_group(){

  for user in "${users[@]}"; do
    usermod -aG "$group_name" "$user"
  done
}
```

Obr. 3.11: One-to-many skript.

Koncový bod, ktorý spomíname vyššie sa nachádza na obrázku 3.13. Ako vstup očakáva požiadavku vo formáte json. Pre úspešné vykonanie skriptu, ktorý sa tu volá je potrebné upraviť prijaté dáta do požadovanej formy ako opisujeme vyššie.

Predchádza tomu však overovanie každého používateľského mena z požiadavky pre verifikáciu existujúcich účtov. Po úspešnom zbehnutí sa volá samotný Bash skript a vracia odpoveď na stranu z ktorej prišla požiadavka.

```
@router.post("/otmgroup")
async def otm_group(users: Request, token: str = Depends(token_verification_middleware)):
    users_data = await users.json()
    username = decode_token(token)

    # Check if all provided users are exist and add them to the users string
    if valid_user(username):
        string = username+"_to_"
        for index, user in enumerate(users_data["users"]):
            if valid_user(user):
                if index > 0:
                    string += "-"+user
                else:
                    string += user
            else:
                raise HTTPException(status_code=401, detail="User {} is not a valid user".format(user))
    else:
        raise HTTPException(status_code=401, detail="Unauthorized")

    command = f'echo "{string}" | {otm_script_path} '
    user_manager.run(command)

    if user_manager.get_return_code() == 0:
        raise HTTPException(status_code=200, detail="Group named {} has been created".format(string))
    else:
        raise HTTPException(status_code=400, detail="Bad request")
```

Obr. 3.12: Štruktúra správy skupín.

3.3.4 Správa používateľov

V neposlednom rade bolo pre našu implementáciu kľúčové na-implementovať správu používateľov. Koncové body v tejto časti rozdeľujeme do administrátorskej časti a časti koncového používateľa. To znamená, že administrátor má k dispozícii chránené koncové body, ako napríklad zamknutie účtu používateľa. Koncový bod môže využiť v prípade, že používateľ porušil niektoré pravidlá alebo mu administrátor chce zamedziť prístup na server. Koncový používateľ (samozrejme aj admin) má k dispozícii zmenu hesla, zmenu mena, ktorá v sebe obsahuje, aj zmenu

mena domovského adresára a všetky skupiny, kde sa jeho meno môže nachádzať. Mazanie používateľov sme nechali prístupné len v rámci jednotlivých účtov. To znamená, že účet si môže odstrániť len používateľ, ktorý spravil požiadavku na server. Mazanie účtov sme pridali z legálnych dôvodov. Pokiaľ si používateľ už neželá byť zapísaný v našom systéme, má plné právo svoj účet aj s dátami vymazať. Pre vyriešenie tejto funkcionality sme použili enpoint, ale aj Bash skript. V koncovom bode riešime samotné zmazanie používateľa no mazanie skupín funguje cez skript. Urobili sme tak z dôvodu jednoduchosti implementácie. Prvotne sa, teda zavolá koncový bod, kde sa do dočasného súboru zapíšu všetky skupiny vytvorené používateľom, teda öne-to-many"skupiny kde zdieľal dáta, ďalej jeho zoznam priateľov. Ďalej sa púšťa skript pre zmazanie týchto skupín. Následne sa maže samotný používateľ a jeho domovský adresár.

3.3.5 Opis API koncových bodov

Vrámci riešenia sme mali implementovať spôsob pre manažment používateľov. Opisom v tejto časti chceme poukázať aj na použitie vyššie opísanej autorizácie. Pridelenie oprávnení zaraďujeme k nevyhnutnej časti správy používateľov. Nasleduje opis „/lockuser“. Jedná sa o chránenú časť riešenia, čo znamená, že prístup je povolený výhradne administrátorom. Pri zavolaní je očakávaný vstup meno používateľa v tele a autorizačný token v hlavičke požiadavky. Nasleduje funkcia na obrázku 3.13, ktorý sa spúšťa cez spomínaný koncový bod. Prvým krokom je dekódovanie JWT tokenu pre získanie používateľského mena, ktoré spravilo požiadavku na server. Ďalej sa kontroluje autorizácia používateľa to znamená či má právo na spustenie chráneného koncového bodu. Pre kontrolu mena poskytnutého v tele požiadavky sa overí či žiadané meno patrí medzi existujúcich používateľov v systéme Linux. Po týchto overeniach sa pred spustením samotného príkazu na uzamknutie účtu ešte preverí status používateľa. Jedná sa o overenie či už nie je

zamknutý. Nasleduje samotný príkaz na zamknutie účtu a ďalšie podmienky pre uistenie správnosti výpisu. V prípade, že sa používateľ pokúsi prihlásiť do účtu po uzamknutí dostane odpoveď z podmienky if. Prepínač -L v príkaze usermod hovorí o tom že chceme uzamknúť meno používateľa, ktoré ho nasleduje.

```python
@router.post("/lockuser")
async def lock_user(user: str = Form(), token: str = Depends(token_verification_middleware)):
    # Decode JWT token to get user's username
    username = decode_token(token)

    # Check if requesting user is admin
    if authorize_user(token)["user_type"] == "admin":

        # Check if requested user exists
        if valid_user(user):

            # Check if account isn't locked already
            command = f"grep {user} /etc/shadow"
            user_manager.run(command)
            if user_manager.stdout.split(":")[1].split("$")[0] == "!":
                return "User already locked"

            # If user is added to the .run(), linux checks if user has perms to run the command
            command = f"sudo usermod -L {user}"
            user_manager.run(cmd=command, user=username)

            if user_manager.get_return_code() == 0:
                return "User {} successfully locked".format(user)
            elif user_manager.get_return_code() == 6:
                return "User {} doesn't exist".format(user)
            else:
                return "Something went wrong"
        else:
            return valid_user(user)
    else:
        raise HTTPException(status_code=401, detail="Unauthorized")
```

Obr. 3.13: Koncový bod pre uzamknutie účtu.

Prihlasovanie a registrácia sú kľúčovou súčasťou každého systému. V našom prípade sme pri registrácii museli okrem samotného pridania používateľa vyriešiť aj ďalšie veci. Máme na mysli nastavenie hesla, vytvorenie a pridanie do skupiny zoznamu priateľov používateľa a ďalej vloženie nového účtu do základnej skupiny koncových používateľov. V koncovom bode sme taktiež ošetrovali dĺžku zadaného hesla ako aj znaky použité v hesle. Znaky bolo nutné kontrolovať pre zamedzenie spúšťania príkazov z registrácie, ku ktorej by sa potencionálne mohol dostať niekto z vonku. Samotný opisovaný koncový bod sa nachádza na obrázku 3.14.

```python
@router.post("/registration")
async def create_user(user: User, credentials: HTTPAuthorizationCredentials = Depends(bearer_scheme)):
    # Verify the JWT token
    try:
        payload = jwt.decode(credentials.credentials, JWT_SECRET_KEY, algorithms=[JWT_ALGORITHM])
    except jwt.DecodeError:
        raise HTTPException(status_code=401, detail="Invalid token")

    # Check password length and if contains only letters and numbers
    if 3 < len(user.password) < 32 and user.password.isalnum():



        # Register user, create home dir for the new user and create group user friends for file sharing
        command = f"useradd -mU {user.username} " \
                  f"&& echo \"{user.username}:{user.password}\" | chpasswd " \
                  f"&& groupadd {user.username}_friends" \
                  f"&& usermod -aG end_users {user.username} " \
                  f"&& usermod -a -G {user.username}_friends {user.username}"
        user_manager.run(command)
        if user_manager.get_return_code() == 0:
            return "User {} has been successfully created".format(user.username)
        elif user_manager.get_return_code() == 9:
            return "User named {} already exists".format(user.username)
    else:
        raise HTTPException(status_code=403, detail="Bad password")
```

Obr. 3.14: Koncový bod pre uzamknutie účtu.

V koncovom bode prihlasovania na obrázku 3.15 bolo potrebné overenie existencie účtu. V kóde ďalej nasleduje príkaz a jeho spustenie, ktorého výstup sa kontroluje pre získanie stavu účtu. Tu mohol nastať prípad kedy sa používateľ s uzamknutím účtom pokúša prihlásiť. Nasleduje samotná autentifikácia, ktorú opisujeme vyššie v sekcii 3.3.2. Po jej úspešnom vykonaní sa vytvára JWT token a posiela sa odpoveď používateľovi. Funckiu pre vytváranie JWT tokenov opisujeme vyššie v sekcii 3.3.2.

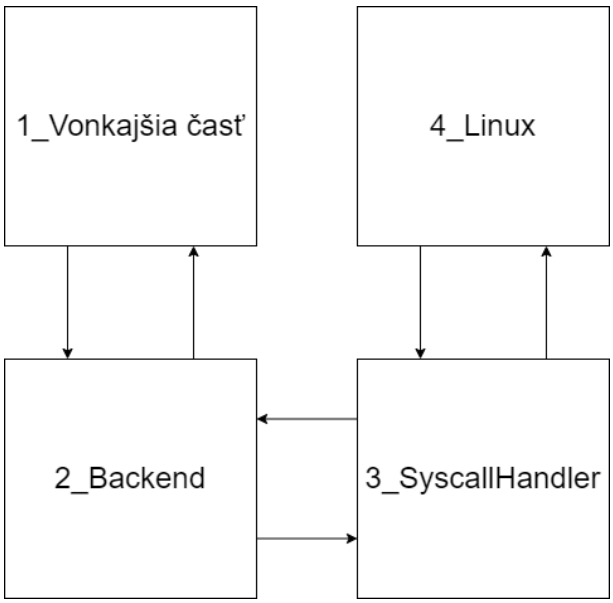
```python
@router.post("/login")
async def login(form_data: OAuth2PasswordRequestForm = Depends()):
    username = form_data.username
    password = form_data.password

    # Check if user exists
    user = valid_user(username)
    if user:

        # Check if account wasn't locked by administrator
        command = f"grep {username} /etc/shadow"
        user_manager.run(command)
        if user_manager.stdout.split(":")[1].split("$")[0] == "!":
            raise HTTPException(status_code=403, detail="Forbidden")

        # Authenticate the user
        if not authenticate_user(username, password):
            raise HTTPException(status_code=401, detail="Invalid credentials")

        # Create a JWT token
        access_token = create_access_token({"sub": username})

        # Return the JWT token in header
        response = {"Successfully logged in"}
        headers = {"Authorization": f"Bearer {access_token}"}
        return response, headers
    else:
        return user
```

Obr. 3.15: Koncový bod pre prihlásenie.

3.3.6 Štruktúra API na backende

Pre štruktúrovanie nášho backendu sme používali koncové body, ktoré sú volané cez http požiadavky. Enpointy v každom prípade volajú hlbšie do programu nakoľko sa vždy overí aspoň či je používateľ autentifikovaní. Ďalej koncové body v každom prípade volajú aj linuxové príkazy cez modul 3_SyscallHandler[17], ako sme spomínali vyššie podľa druhu koncového bodu, ktorý bol zavolaný. Požiadavky

sa však môžu spracovať aj viac vrstvovo. To znamená, že koncový bod zavolá konkrétnu funkciu, ktorú potrebuje na spracovanie údajov. Daná funkcia potom volá ďalšie funkcie alebo sa odkazuje na potrebnú triedu definovaný v rámci štruktúry programu. Rovnako, ako koncové body aj funkcie alebo triedy môžu volať linuxové príkazy pokiaľ to je potrebné. Naša implementácia má podľa opisu 3 základné úrovne, a to 4_Linux, 2_Backend a 1_Vonkajšiu časť odkiaľ sa dajú posielať požiadavky. Vonkajšiu časť môže tvoriť čokoľvek, čo je schopné posielať požiadavky na náš backend. Komunikáciu medzi štruktúrami sme vyzobrazili na obrázku 3.16. Dáta v požiadavkách v našom prípade vždy berieme z tela a nepoužívame query parametre. Http odpovede najčastejšie môžu byť: 200 (dobrá požiadavka), 400 (zlá požiadavka), 401 (neautorizovaná požiadavka), a 403 (zakázaný prístup).



Obr. 3.16: Štruktúra správy skupín.

3.4 Overenie riešenia

Na účely overenia riešenia prostredníctvom používateľského testovania sme definovali scenáre používateľského akceptačného testovania (UAT). Tieto písomné scenáre boli poskytnuté osobám, ktoré súhlasili so spracovaním svojich odpovedí. Testovanie zahŕňalo prechod cez konkrétne koncové body programu Postman. Overenie riešenia bolo založené na testovaní niekoľkých implementovaných funkcionalít nášho programu. Išlo o overenie autentifikácie používateľa prostredníctvom prihlásenia, autorizácie každej požiadavky a výpisu všetkých používateľov, ktorí nepatria do skupiny používateľov systému. Okrem toho sme otestovali pridávanie priateľov alebo osôb do skupín firemnej štruktúry, čo je povolené len administrátorským účtom. V rámci pridávania do štruktúr si scenáre vyžadovali použitie koncového bodu na načítanie preddefinovaných skupín v systéme, keďže testeri nepoznali názvy týchto skupín. Keďže naše riešenie neobsahuje frontend, museli sme najprv vysvetliť, ako pracovať s programom Postman, v ktorom prebiehalo testovanie. Okrem výsledkov scenárov UAT sme po ukončení testovania požiadali osoby, s ktorými sme spolupracovali, aby vyplnili dotazník. Cieľom dotazníka bolo posúdiť primeranosť funkčnosti riešenia, logické usporiadanie koncových bodov do balíkov, informatívnosť odpovedí servera a zahrnuli sme aj jednu otvorenú otázku na návrhy na zlepšenie našej implementácie. To znamená, že napriek absencii implementovaného frontendu sme prezentovali koncové body usporiadané do súborov tak, ako by ich používatelia našli roztriedené do rôznych zoznamov na webovej stránke. Dotazník bol zameraný na celkovú funkčnosť, neobmedzoval sa na scenáre UAT. Po dokončení scenárov UAT sme testerom umožnili prezrieť všetky koncové body a balíky. Venovali sme sa všetkým otázkam, ktoré mali počas procesu testovania a po ňom. Na prezentáciu výsledkov testovania sme využili grafy a ich výsledky sú opísané v časti 3.4.1.1. Scenáre UAT a dotazník boli kľúčové pri získavaní spätnej väzby a poznatkov od účastníkov testovania.

3.4.1 UAT scenáre

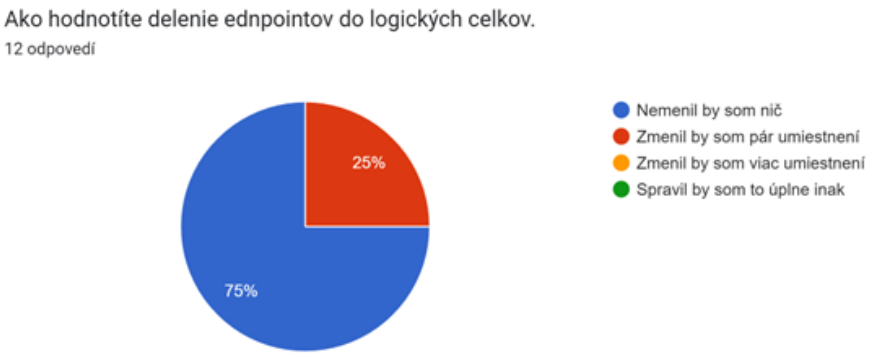
UAT1: Pridanie priateľa s prihlásením	
Vstupné podmienky	Používateľ má otvorený program Postman Používateľ má nahraný JSON file v Postmanovy Používateľ je registrovaný Používateľ je odhlásený Používateľ je oboznámený zo základnými oknami Postmana
Výstupné podmienky	Používateľ si úspešne pridal priateľa
Postup	1. Používateľ sa naviguje do balíčka Entry endpoints v časti „Collections“ kde volí Login. 2. Zadá svoje údaje z registrácie do „Body“ v časti požiadavky a kliká na tlačidlo Send. 3. Dostane odpoveď v „Body“ v časti odpoveď kde skopíruje hodnotu „Authorization“ záznamu. 4. Používateľ nepozná meno nikoho v aplikácii tak sa presunie do balíčka User enpoints v časti „Collections“. 5. Tu vyberá potrebný endpoint Get All Users 6. Presunie sa do záložky „Headers“ v časti požiadavky kde do pola „Authorization“ vloží skopírovanú hodnotu z kroku číslo 3. 7. Kliká na tlačidlo „Send“. 8. Používateľ si vyberá jedného z používateľov ktorý sa mu vrátili v zozname v okne odpovede. 9. Ďalej sa používateľ naviguje do balíčka Friend endpoints v okne „Collections“ kde vyberá endpoint Add As Friend. 10. Tu opäť kopíruje svoj JWT token do „Headers“ v okne požiadavky do pola „Authorization“. 11. V rovnakom okne kliká na „Body“ kde do hodnoty „friend_to_add“ pridá vybraného používateľa. 12. Používateľ kliká na tlačidlo „Send“. 13. Používateľ sa pozrie na výstupnú hlášku v okne odpovedí v „Body“.
Výsledok:	Pass/Fail

UAT2: Pridanie nového používateľa do štruktúry firmy	
Vstupné podmienky	Používateľ má otvorený program Postman Používateľ má nahraný JSON file v Postmanovy Používateľ má prístup k administrátorskému účtu Používateľ je odhlásený Používateľ je oboznámený zo základnými oknami Postmana
Výstupné podmienky	Nový používateľ je zapísaný v jednej skupine firmy
Postup	1. Používateľ sa prihlási do administrátorského účtu. 2. Používateľ si skopíruje autorizačný token v okne odpovede. 3. Ďalej sa naviguje do balíčka Group endpoints v okne „Collections“ kde volí endpoint Display Company Groups. 4. Používateľ sa naviguje v okne požiadavky do časti „Headers“ kde do hodnoty „Authorization“ vloží skopírovany token. 5. Ďalej používateľ kliká na tlačidlo „Send“. 6. Používateľ vyberá a zapamätá si ľubovolnú skupinu zo zoznamu ktorý sa mu vrátil v okne odpovede. 7. Používateľ sa naviguje ďalej do endpointu Add User To Group kde v okne požiadavka vloží do „Headers“ pola „Authorization“ skopírovaný token z bodu 2. 8. Následne používateľ prejde vrámci okna do záložky „Body“ kde do kolónky user vloží svoje meno ktoré zadal počas registrácie ktorá predchádzala testovanie. 9. V zápätí vloží do kolónky group skupinu ktorú si mal zapamätať a stláča tlačidlo „Send“ 10. Používateľ sa pozrie na výstupnú hlášku v okne odpovedí v „Body“.
Výsledok:	Pass/Fail

3.4.1.1 Výsledky testovania

Výsledok dotazníka ktorý sme po otestovaní dávali vyplniť používateľom sme zobrazili do grafu pod tabuľkou spolu z opýtanou otázkou a možnosťami. Nakoľko všetci používatelia docielili výsledok „PASS“ nevyhodnocovali sme UAT scenáre ako celok. Celkovo sme dosiahli priaznivé odpovede čo nám potvrdilo správnosť riešenia vzhľadom na opýtané otázky. Musíme však podotknúť že všetci testeri mali aspoň základy v informatike no nie nutne z manažmentom používateľov. Myslíme si že toto malo za následok 100% úspešnosť UAT scenárov.

Obr. 3.17: Dotazník, otázka číslo 1



Obr. 3.18: Dotazník, otázka číslo 2

Obr. 3.19: Dotazník, otázka číslo 3

V nasledujúcej časti sme spravili vyhodnotenie dotazníka, ktorí sme dávali vyplniť naším testerom po vykonaní oboch UAT scenárov. Odpovede respondentov boli celkovo priaznivé no našli sa prípady kedy ich miatli odpovede servera. Po konverzácii s osobami sme usúdili, že odpovede na požiadavky mohli byť menej jasné používateľom nakoľko nevedeli, čo sa deje na pozadí serveru, teda backendu. Dvaja testeri mali problém s vyhľadaním samotného prihlásenia, ktoré hľadali v balíčku „User endpoints“ miesto jeho skutočného umiestnenia v balíčku „Entry endpoints“. Aj napriek tomu, že takéto množstvo používateľov tvorí jednu šestinu celkového počtu to neberieme ako chybu v rozdelení na našej strane. Je to odôvodnené tým, že v reálnom scenári kedy sa zobrazí stránka s prihláseným, ako prvá takáto chyba nestala. Prvý používateľ, ktorý skúšal UAT1 taktiež narazil na chybu výpisu priateľov. Bolo to spôsobené chybou na našej strane nakoľko sa v zozname účtov v Linuxe nachádzali ešte používatelia bez zoznamu priateľov. Tieto záznamy slúžili predošle na testovanie funkcionality koncových bodov. Nakoľko to bol tento prípad daných používateľov sme zmazali a požiadali osobu, aby tieto záznamy ignorovala. Nižšie sme uviedli všetky odpovede na voľnú otázku, ktorú sme sa pýtali v dotazníku.

1. za mňa ako informatika v poriadku
2. trošku som nerozumel odpovediam, riadil som sa iba podla vykonaného statusu a ze to bolo oznacene OK
3. Po vysvetlení ako obsluhovať postman to bolo fajn a nemal som problem, no sam by som na to neprisiel
4. Fungovalo vsetko tak ako ma, nebol najmensi problem s navigáciou medzi endpointami a vytváranim jednotlivých requestov, taktiež to bolo logicky štruktúrované
5. Pomenovanie logických celkov
6. Výpis priateľov
7. chcelo by to nejake moznosti pre endpointy nieco ako swagger alebo openapi

Vrámci overenia riešenia sme zakomponovali metódu testovania funkčnosti. V našom prípade sme sa zameriavali výhradne na skúšanie takých koncových bodov ktorých implementácia ešte nebola odskúšaná UAT scenármi. Metódu testovania sme aplikovali my. Pre vykonanie akcii sme používali už existujúceho používateľa, ktorý mal administrátorské práva pre potreby niektorých testovaný koncových bodov. Výsledky sme zhrnuli do tabuľky ktorú sme uviedli nižšie. Používali sme výrazy „Pass“ čo predstavuje úspešné prejdenie. Stav mohol nastať len vtedy keď sme od servera dostali odpoveď ktorá bola zrozumiteľná a zároveň obsahovala http kód 200 čo predstavuje úspešnú požiadavku. A zároveň aj prípady kedy bola poslaná požiadavka so zlým vstupom pokiaľ to bolo možné. Opačný prípad bola možnosť „Fail“, ktorá nastala v každom ďalšom prípade. Nasleduje tabuľka testvania 3.1.

Výsledky testovania funkcionality	
Koncový bod	Výsledok
1. POST - Lock User	Pass
2. POST - Unlock User	Pass
3. PUT - Change Name	Pass
4. PUT - Change Own Password	Pass
5. PUT - Remove From Friend List	Pass
6. GET - Display Users Groups	Pass
7. POST - Create Group	Pass
8. PUT - Remove User From Group	Pass
9. DELETE - Delete User (self)	Pass

Tabuľka 3.1: Testovanie funkcionality

Kapitola 4

Záver

4.1 Zhrnutie

Hlavným cieľom tejto práce bolo vystaviť a analyzovať určité funkcionality operačných systémov na báze Linuxu so zameraním na správu používateľov a skupín vo webových portáloch a systémoch WCMS. Porovnaním požadovaných funkcií s tými, ktoré sú k dispozícii v systéme Linux, sa výskum zameral na navrhnutie rozšírení alebo vylepšení existujúcich funkcií systému Linux pomocou jednoduchých skriptov, ako je Bash alebo podobné skriptovacie jazyky.

Práca zahŕňala rôzne kapitoly, pričom každá sa týkala základných aspektov predmetu. Úvodné kapitoly Analýzy poskytli hlbší náhľad do operačných systémov, WCMS a ich funkcionalít, skúmali fungovanie, výhody, nevýhody a bezpečnostné aspekty spojené s WCMS. Okrem toho boli spolu s operačným systémom Linux preskúmané aj existujúce riešenia WCMS, ako sú WordPress a HubSpot. Nasledujúce kapitoly vrámci Analýzy presunuli pozornosť na správu používateľov v operačnom systéme Linux, vrátane jeho bezpečnostných aspektov. Boli prediskutované backendové rámce ako Flask, Django a FastAPI spolu s metódami

autentifikácie a autorizácie, ako sú JSON Web Token (JWT) a OAuth2. Komplexné porovnanie medzi dostupnými funkciami správy používateľov WCMS a funkciami, ktoré ponúka systém Linux, poskytlo cenné informácie o potenciálnych zlepšeniach.

Druhá časť bakalárskej, Implementácia, práce bola zameraná na popis navrhovaného riešenia, načrtnutie funkčných a nefunkčných požiadaviek a identifikáciu potrebných komponentov pre úspešnú implementáciu. Navrhované riešenie bolo vyvinuté a riešené prostredníctvom softvérovej implementácie, ktorá rieši kľúčové aspekty, ako je autentifikácia, autorizácia, správa skupín a správa používateľov. Na zabezpečenie efektívnosti a životaschopnosti riešenia boli navrhnuté a spustené rôzne testovacie scenáre a scenáre užívateľského akceptačného testovania (UAT). Výsledky testovania potvrdili správnosť riešenia a preukázali jeho praktickosť v reálnych scenároch. Práca priniesla poznatky o vlastnostiach a funkcionalitách operačných systémov na báze Linuxu a ich potenciálnej integrácii s WCMS. Navrhované riešenie slúži ako príspevok v tejto oblasti, premosťuje medzeru medzi WCMS a Linuxom a ponúka rozšírenú správu používateľov a možnosti zdieľania údajov.

4.2 Plány do budúcnosti

Na základe výskumu a implementácie vykonanej v tejto práci možno identifikovať niekoľko možností pre budúcu prácu a vylepšenia. Tieto potenciálne oblasti vývoja sú zamerané na ďalšie zlepšenie funkčnosti a použiteľnosti operačného systému na báze Linuxu integrovaného s webovými portálmi a systémami WCMS. Plány do budúcnosti sme definovali v nasledujúcich paragrafoch.

Implementácia zoznamov na riadenie prístupu (ACL): S cieľom zvýšiť bezpečnosť a kontrolu nad oprávneniami používateľov by sa budúca práca mohla

zamerať na implementáciu zoznamov riadenia prístupu (ACL) v rámci operačného systému založeného na Linuxe. Zoznamy ACL by umožnili jemnejšiu kontrolu prístupu a umožnili by správcom definovať špecifické oprávnenia pre jednotlivých používateľov alebo skupiny používateľov na súbory a adresáre.

Vývoj frontendu: Na zabezpečenie komplexnejšieho a používateľsky príveti-vejšieho prostredia by sa implementácia mohla rozšíriť o vývoj frontendového rozhrania. Tento frontend by fungoval ako brána pre komunikáciu medzi API Linux syscall, API správy súborov a API správy používateľov prostredníctvom komunikácie HTTP. Frontend by umožnil používateľom komunikovať so systémom intuitívnejším a prístupnejším spôsobom. V nadväznosti na vývoj frontendu by sa budúca práca mohla zamerať na zlepšenie používateľského rozhrania webového portálu. Mohlo by to zahŕňať návrh intuitívneho používateľského rozhrania, začlenenie zásad responzívneho dizajnu a optimalizáciu používateľského prostredia s cieľom zabezpečiť jednoduché používanie a efektívnosť.

Integrácia so systémom WCMS: Na ďalšie zlepšenie integrácie medzi systémami Linux a WCMS by sa budúca práca mohla zamerať na bezproblémovú integráciu navrhovaného riešenia s existujúcimi platformami WCMS. Táto integrácia by umožnila používateľom využívať možnosti správy používateľov a zdieľania údajov operačného systému založeného na Linuxe priamo v prostredí WCMS.

Vylepšenia zabezpečenia: Pokračovanie výskumu v oblasti bezpečnosti by bolo cenné pre zabezpečenie robustnosti a odolnosti operačného systému založeného na Linuxe. Budúca práca by sa mohla zamerať na implementáciu ďalších bezpečnostných opatrení, ako je šifrovanie citlivých údajov, implementácia bezpečných komunikačných protokolov (napr. HTTPS) a vykonávanie dôkladných bezpečnostných auditov a hodnotení zraniteľností.

Literatúra

[1] 2022. URL: https://docs.python.org/3/library/crypt.html (cit. 20. 05. 2023).

[2] *2022 vulnerability statistics report*. 2022. URL: https://www.edgescan.com/2022-vulnerability-statistics-report-lp/#form (cit. 20. 10. 2022).

[3] Lee Allen, Tedi Heriyanto a Shakeel Ali. *Kali Linux-Assuring security by penetration testing*. Packt Publishing Ltd, 2014. (Cit. 18. 11. 2022).

[4] David Barrera, Ian Molloy a Heqing Huang. „IDIoT: Securing the Internet of Things like it's 1994". In: (dec. 2017). arXiv: 1712.03623 [cs.CR]. (Cit. 04. 12. 2022).

[5] Ahmed Bentiba, Ahmed Mohmed a Jamal Zemerly. „Java Linux Administration Tool". In: *2006 IEEE GCC Conference (GCC)*. IEEE. 2006, s. 1–4. (Cit. 07. 12. 2022).

[6] SN Bokhari. „The Linux operating system". In: *Computer* 28.8 (1995), s. 74–79. (Cit. 28. 10. 2022).

[7] *Build a site, Sell your stuff, start a blog & more.* URL: https://wordpress.com/?aff=190 (cit. 23. 10. 2022).

[8] *ClamAV documentation*. URL: https://docs.clamav.net/ (cit. 04. 12. 2022).

[9] *Common weakness enumeration*. URL: https://cwe.mitre.org/data/definitions/79.html (cit. 20. 10. 2022).

[10] *Cross site scripting (XSS)*. URL: https://owasp.org/www-community/attacks/xss/ (cit. 20. 10. 2022).

[11] Samuel Dauzon, Aidas Bendoraitis a Arun Ravindran. *Django: web development with Python*. Packt Publishing Ltd, 2016, s. 41. (Cit. 27. 12. 2022).

[12] Demetra Edwards et al. *What is a web content management system (WCMS)?* 2021. URL: https://www.techtarget.com/searchcontentmanagement/definition/web-content-management-WCM (cit. 16. 10. 2022).

[13] J A Galindo, D Benavides a S Segura. „Debian Packages Repositories as Software Product Line Models. Towards Automated Analysis“. In: *the 1st International Workshop on Automated Configuration and Tailoring of Applications*. 2010. (Cit. 15. 11. 2022).

[14] Dick Hardt. *The OAuth 2.0 authorization framework*. Tech. spr. 2012, s. 4–5. (Cit. 14. 01. 2023).

[15] HubSpot. *HubSpot website builder and Marketing Free*. URL: https://www.hubspot.com/marketing/am_website-builder-hsmf?irclickid=TWW1z\%3A3vxxyNRuXWgJQMRRfEUkAzBZUpDy-IVo0\&irgwc=1\&mpid=11535\&utm_id=am11535\&utm_me-dium=am\&utm_source=am11535\&utm_campaign=amcid_TWW1z\%3A3vxxy-NRuXWgJQMRRfEUkAzBZUpDy-IVo0_irpid_11535\&utm_content=wordpress (cit. 28. 10. 2022).

[16] Michael Jones, John Bradley a Nat Sakimura. *Json web token (jwt)*. Tech. spr. 2015, s. 2–3. (Cit. 14. 01. 2023).

[17] Jakub Kuska. *Syscallhandlerpublic/modules/base.py*. 2023. URL: https://gitlab.com/three-brave-axolotls/syscallhandlerpublic/-/blob/main/modules/base.py (cit. 19. 05. 2023).

[18] Jose-Manuel Martinez-Caro et al. „A comparative study of web content management systems“. In: *Information* 9.2 (2018), s. 27. (Cit. 19. 10. 2022).

[19] Michael Meike, Johannes Sametinger a Andreas Wiesauer. „Security in Open Source Web Content Management Systems“. In: *IEEE Security & Privacy* 7.4 (2009), s. 44–51. DOI: 10.1109/MSP.2009.104. (Cit. 16. 10. 2022).

[20] Mohammad Robihul Mufid et al. „Design an MVC Model using Python for Flask Framework Development“. In: *2019 International Electronics Symposium (IES)*. 2019, s. 214–219. DOI: 10.1109/ELECSYM.2019.8901656.

[21] *Owasp Top Ten*. URL: https://owasp.org/www-project-top-ten/ (cit. 23. 10. 2022).

[22] Spencer Shepler et al. *Network file system (NFS) version 4 protocol*. Tech. spr. 2003. (Cit. 12. 11. 2022).

[23] Support. *Add HubSpot users*. 2018. URL: https://knowledge.hubspot.com/settings/add-and-remove-users (cit. 28. 10. 2022).

[24] WPExperts a Uzair Ahmed. *User management*. 2022. URL: https://wordpress.org/plugins/user-management/ (cit. 23. 10. 2022).

[25] Ming-Ju Yang et al. „A User-Friendly Web Content Management System“. In: *2008 3rd International Conference on Innovative Computing Information and Control*. IEEE. 2008, s. 367–367. (Cit. 16. 10. 2022).

[26] Matthew R. Yaswinski, Md Minhaz Chowdhury a Mike Jochen. „Linux Security: A Survey“. In: *2019 IEEE International Conference on Electro Information Technology (EIT)*. 2019, s. 357–362. DOI: 10.1109/EIT.2019.8834112. (Cit. 20. 11. 2022).

[27] Imran Yusof a Al-Sakib Khan Pathan. „Mitigating Cross-Site Scripting Attacks with a Content Security Policy“. In: *Computer* 49.3 (2016), s. 56–63. DOI: 10.1109/MC.2016.76. (Cit. 20. 10. 2022).

Dodatok A

Plán práce na riešení projektu

Týždeň	Popis práce
1.	Študovanie problematiky témy
2.	Študovanie problematiky témy
3.	Študovanie problematiky témy
4.	Analýza operačných systémov
5.	Analýza WCM systémov
6.	Analýza použitého operačného systému Linux
7.	Analýza dostupných backend rámcov
8.	Analýza riešení autentifikácie a autorizácie
9.	Dokončenie analýzy
10.	Vytvorenie kapitoly opisu riešenia
11.	Vytvorenie kapitoly opisu požiadaviek
12.	Dokončenie dokumentu pre Bakalársky projekt I

Tabuľka A.1: Plán práce pre Bakalársky projekt I

Týždeň	Popis práce
1.	Vytvorenie kapitoly návrh
2.	Implementácia programovej časti
3.	Implementácia autorizácie a autentifikácie
4.	Implementácia spravy skupín
5.	Implementácia správy používateľov
6.	Vytvorenie kapitoly implementácia
7.	Oprava chýb v celkovej implementácii
8.	Testovanie UAT scenárov
9.	Vyhodnotenie testovania UAT scenárov
10.	Dokončenie implementácie
11.	Vytvorenie kapitoly overenie riešenia
12.	Dokončenie dokumentu Bakalársky projekt II

Tabuľka A.2: Plán práce pre Bakalársky projekt II

Dodatok B

Technická dokumentácia

Táto časť práce bude venovaná technickému opisu. Vymenujeme tu použité knižnice vrámci backend implementácie a opíšeme ich. Pozrieme sa bližšie aj na príkazy volané koncovými bodmi, ktoré v práci používame. V krátkosti opíšme aj druhy požiadaviek a na čo slúžia. Na konci taktiež vysvetlíme smerovače, ktorý sme v práci použili vrámci implementácie FastAPI.

B.1 Použité knižnice

V implementácii aplikácie sme použili knižnice, ktoré nám uľahčili vytvorenie riešenia. Išlo o knižnice:

1. JWT: Táto knižnica umožňuje kódovať a dekódovať webové tokeny JSON, ktoré sa používajú na bezpečné overovanie a výmenu údajov medzi stranami. Tokeny JWT pozostávajú z kompaktného objektu JSON bezpečného pre URL, ktorý možno podpísať a zašifrovať na zabezpečenie integrity a dôvernosti. V našom prípade sa používali, ako je spomenuté v sekcii Implementácia.

2. fastapi.security: Táto knižnica je súčasťou rámca FastAPI, čo je vysoko výkonný webový rámec na vytváranie rozhraní API pomocou jazyka Python. Modul fastapi.security poskytuje rôzne komponenty a nástroje súvisiace s bezpečnosťou, napríklad mechanizmy autentifikácie a autorizácie vrátane OAuth2, JWT a ďalších. Pri implementácii sme knižnicu použili pre definovanie schém tvaru JWT tokenu a tvaru hesla.

3. crypt: Tento modul poskytuje funkcie na jednosmerné hashovanie hesiel pomocou rôznych hashovacích algoritmov, napríklad DES, MD5 a SHA-256. Používa sa predovšetkým na zabezpečenie hesiel ich konverziou na nezvratné hash hodnoty. My sme použili knižnicu na zahashovanie hesla, ktoré poskytol používateľ pri prihlásení.

4. hmac: Modul hmac je tiež súčasťou štandardnej knižnice jazyka Python. Poskytuje podporu pre algoritmus HMAC (Hash-based Message Authentication Code). HMAC sa používa na bezpečné overovanie správ kombináciou tajného kľúča a správy na generovanie hodnoty hash. Pomáha zabezpečiť integritu a pravosť údajov. Knižnicu sme použili na porovnanie zahashovaného hesla knižnicou crypt a hesla, ktoré sme dostali z OS Linux.

5. json: Knižnica json v jazyku Python poskytuje funkcie na prácu s údajmi JSON (JavaScript Object Notation). Umožňuje serializovať objekty jazyka Python do reťazcov JSON a deserializovať reťazce JSON do objektov jazyka Python. JSON je široko používaný formát na výmenu údajov medzi systémami. Použili sme ju pre načítavanie tokenov zo súboru v tvare json súboru a zapisovanie do súboru v rovnakom tvare.

6. asyncio: Knižnica asyncio je súčasťou štandardnej knižnice jazyka Python a poskytuje rámec na písanie asynchrónneho kódu pomocou ko-rutín, slučiek udalostí a úloh. Umožňuje písať súbežný a neblokujúci kód, čo je užitočné

najmä pre aplikácie viazané na vstupy a výstupy a aplikácie založené na sieti. V implementácii používame funkciu, ktorá beží každých tridsať sekúnd práve za pomoci tejto knižnice.

7. datetime: Modul datetime je súčasťou štandardnej knižnice jazyka Python a poskytuje triedy a funkcie na prácu s dátumami, časmi a časovými intervalmi. Umožňuje vytvárať, manipulovať, formátovať a vykonávať výpočty s dátumami a časmi. Pre kontrolu expirácie tokenov sme potrebovali prekonvertovať formát, v ktorom sa čas ukladal. Práve tu bola knižnica kľúčová pre naše riešenie.

8. time: Modul time je tiež súčasťou štandardnej knižnice jazyka Python a poskytuje funkcie na prácu s časom. Umožňuje prístup k systémovému času, meranie časových intervalov a vykonávanie jednoduchých operácií súvisiacich s časom. Na porovnanie času bola dôležitá aj knižnica time, ktorá slúžila na získavanie aktuálneho času.

9. fastapi: FastAPI je vysoko výkonný webový rámec na vytváranie rozhraní API v jazyku Python. Kombinuje najlepšie vlastnosti moderných webových rámcov, ako je rýchle vykonávanie, typové anotácie, automatická validácia a generovanie dokumentácie. FastAPI je známy svojou rýchlosťou a jednoduchosťou pri vytváraní robustných webových rozhraní API. Jadrom riešenia bola práve knižnica fastapi. Využili sme mnoho funkcionalít, ktoré ponúkla. Išlo napríklad o HTTP odpovede, formát informácií v požiadavke, smerovače a požiadavky.

B.2 Použité príkazy a operátory

Vrámci komunikácie s Linuxom sme volali príkazy pre vykonanie akcie podľa druhu koncového bodu. Išlo o príkazy pre úpravu používateľov, skupín, výpisy, filtrovanie a podobne. Vrámci posielania príkazov sme používali aj operátory pre pipovanie alebo nadväzovanie príkazov. Pre vykonanie niektorých príkazov však boli kľúčové práve prepínače. Slúžia pre upresnenie príkazu. Máme na mysli spresnenie akcie, ktorá sa má vykonať. Ďalej opisujeme jednotlivé príkazy spolu aj s prepínačmi a operátormi použitými ako celok pre dosiahnutie požadovaného výsledku.

Ako prvý si opíšeme usermod. V systéme Linux sa príkaz používa na úpravu vlastností a nastavení používateľského účtu. Umožňuje správcom systému vykonávať zmeny v existujúcich používateľských účtoch, napríklad upravovať používateľské meno, ID používateľa (UID), ID skupiny (GID), domovský adresár, predvolený shell a ďalšie. V našom prípade ho využívali pre zaradenie používateľa do skupiny s prepínačmi „a“ pre pripojenie do skupiny bez prepísania aktuálnych skupín používateľa a „G“ pre možnosť pridávania do skupín. Ďalej sme v kombinácii s usermod požívali prepínač „d“, ktorý slúži na zmenu mena domovského adresára. Prepínač „l“ sme použili pri zmene mena účtu. Administrátorské účty používali aj prepínač „L“. Jednalo sa o prípad kedy bolo potrebné uzamknutie účtu. Opačný prípad bol prepínač „U“, ktorý slúži na odomknutie účtu.

Nasleduje popis grep. Používa na vyhľadávanie a filtrovanie textu na základe špecifických vzorov. Jedná sa skutočne o nesmierne užitočný a silný príkaz. Žiadna aplikácia na báze Linuxu alebo administrátor by bez príkazu podľa našej úvahu nemohli existovať. Konkrétne použitie v implementácii bolo filtrovanie zoznamu účtov za pomoci operátora „|“ pre získanie len potrebných záznamov. To isté platí pre skupiny a heslá. Grep sme ďalej používali aj v kombinácii s operátorom

„>“ alebo „|“ pre zápis do súboru odkiaľ skripty čítali vstupy.

Príkaz echo sme v riešení používali na pipovanie vstupu cez „|“, ktorý nasledoval v úvodzovkách do príkazov podľa potreby. Nebolo, teda nutné používať viac príkazov pre vykonanie akcie.

Useradd bol ďalším užitočným príkazom, ktorý sme používali. Využitie je síce obmedzené no kľúčové pri implementácii akéhokoľvek WCMS s prepojením na Linux. Príkaz slúži na pridávanie používateľov a spolu s prepínačmi „m“ pre vytvorenie domovského adresára a „U“ pre vytvorenie základnej skupiny používateľa bol nevyhnutný pre riešenie.

Spravovanie skupín sme riešili cez príkazy ako groups pre výpis všetkých skupín používateľa. Na vytváranie skupín sme použili groupadd. V prípade, že administrátor potreboval mal možnosť odobrať používateľa zo skupiny cez príkaz gpasswd s použitím prepínača „d“.

Príkaz cut sa v Linuxe používa na extrahovanie určitých častí (stĺpcov) alebo znakov z riadkov vstupu. Je to univerzálny nástroj príkazového riadka, ktorý umožňuje vybrať a oddeliť časti textu na základe pozície oddeľovača alebo znaku. Spolu s prepínačom „f“ a pozíciou sa príkaz vie správať podobne ako polia v programovacom jazyku python.

Groupmod v Linuxe sa používa na úpravu existujúcich vlastností skupiny. Umožňuje správcom systému vykonávať zmeny atribútov skupiny, ako je názov skupiny, identifikátor skupiny (GID), heslo skupiny a členstvo. Konkrétne sme ho používali pre zmenu mien skupín v zozname.

Spájanie príkazov je v Linuxe bežnou praktikou. Využíva sa tu operátor „&&“. Má však pár nevýhod, ako napríklad, keď sa nevykoná jeden z príkazov v sekvencii ďalšie už sa nespúšťajú. Je preto dôležité písať jednotlivé príkazy ko-

rektne. Využíva ho mnoho koncových bodov pre zredukovanie množstva riadkov v kóde.

B.3 Druhy požiadaviek

V implementácii využívame všetky typy požiadaviek. Máme na mysli PUT, DELETE, POST a CREATE. Tieto metódy požiadaviek HTTP poskytujú klientskym aplikáciám (napríklad webovým prehliadačom alebo mobilným aplikáciám) štandardizovaný spôsob interakcie so zdrojmi servera a vykonávania rôznych operácií s nimi vrátane načítania, vytvárania, aktualizácie a odstraňovania.

B.3.1 GET

Požiadavka GET sa používa na získanie informácií alebo zdrojov z určeného servera. Je to najbežnejší typ požiadavky a zvyčajne sa používa, keď chceme získať údaje zo servera. Keď otvoríme webovú lokalitu alebo klikneme na odkaz, prehliadač odošle požiadavku GET na server, aby získal príslušnú webovú stránku. Požiadavky GET by nemali mať žiadne vedľajšie účinky na server a mali by byť konzistentné, čo znamená, že opakované zadanie tej istej požiadavky GET by malo priniesť rovnaký výsledok.

B.3.2 POST

Požiadavka POST sa používa na odoslanie údajov na server s cieľom vytvoriť alebo aktualizovať zdroj. Bežne sa používa pri odosielaní formulárov alebo nahrávaní súborov. Na rozdiel od požiadaviek GET môžu mať požiadavky POST vedľajšie účinky na server, napríklad pridanie nového záznamu do databázy alebo aktualizácia existujúceho prostriedku. Keď na webovej lokalite vyplníme formulár a klikneme na tlačidlo Ödoslať", zadané údaje sa zvyčajne odošlú na server

pomocou požiadavky POST.

B.3.3 PUT

Požiadavka PUT sa používa na odoslanie údajov na server s cieľom aktualizovať alebo nahradiť zdroj na konkrétnej adrese URL. Je podobná požiadavke POST v tom, že môže upravovať zdroje na strane servera. Hlavný rozdiel však spočíva v tom, že požiadavka PUT je určená na úplné nahradenie zdroja na určenej adrese URL, zatiaľ čo požiadavka POST sa často používa na vytvorenie nových zdrojov alebo aktualizáciu špecifických polí existujúceho zdroja.

B.3.4 DELETE

Požiadavka DELETE sa používa na odstránenie určeného prostriedku zo servera. Používa sa, keď chceme odstrániť zdroj, napríklad záznam v databáze alebo súbor na serveri. Odoslaním požiadavky DELETE na server sa spustí proces odstránenia určeného prostriedku. Je dôležité povedať, že požiadavka DELETE, podobne ako požiadavky POST a PUT, môže mať vedľajšie účinky na server.

B.4 FastAPI smerovače

V rozhraní FastAPI sa smerovače vytvárajú definovaním inštancií triedy APIRouter. Každý smerovač predstavuje kolekciu súvisiacich koncových bodov API, ktoré slúžia na konkrétny účel alebo patria do konkrétnej domény. Zoskupením príbuzných koncových bodov umožňujú smerovače lepšiu organizáciu a udržiavateľnosť kódu. Použili sme ich v súbore main, ktorý importoval smerovače definované v ostatných súboroch, kde sa nachádzali koncové body. Dosiahli sme tak čistejší vzhľad kódu a umožnilo nám to štruktúrovať kód do rôznych súborov miesto jedného dlhého.

Dodatok C

Používateľská príručka

Príručka bude určená na opísanie konfigurácie, inštalácie potrebných programov a spustenia programu implementovaného riešenia.

C.1 Potrebné programy

Pre spustenie implementovaného riešenia sú potrebné nasledujúce programy: Docker, Postman a nejaký druh príkazového riadku, ktorý podporuje Docker. V našom prípade sme používali terminál vývojového prostredia PyCharm s verziou, ako opisujeme v Implementácii. No fungoval aj Windows Command Promt. V prípade Dockeru sme používali Docker desktop. Preto budeme aj spustenie prispôsobovať tomuto programu.

C.2 Konfigurácia spustenia

Program je možné plne konfigurovať podľa potreby. Pre našu implementáciu sme však pred-konfigurovali Docker súbor pre jednoduchosť spustenia. V

prípade potreby je možné súbor editovať podľa požiadaviek. To znamená zmena distribúcie Linux, inštalácia knižníc, zmena používaného shellu, spustenie príkazov po štarte alebo zámena servera, na ktorom sa server spustí. V prípade potreby je možné pridať alebo odobrať knižnice pre FastAPI aplikáciu úpravou textového súboru requirements.txt. Pre spomínané zmeny v docket súbore je možné použiť nasledujúce príkazy: FROM - zmena distribúcie Linux, WORKDIR - zmena pracovného adresára, RUN - spustenie príkazov do Shellu, SHELL - zmena Shellu, COPY - kopírovanie súborov do systému, CMD - spúštanie Docker image. Ďalej sme zhrnuli nami pred definovanú konfiguráciu Docker súboru, requirements.txt a docker-compose.yml.

C.2.1 Docker súbor pre konfiguráciu

Ako spomíname vyššie, nasledujúci obsah Docker súboru definuje, čo všetko sa má vykonať pri spustení. Volí sa Linux distribúcia Ubuntu. Nastavuje sa praconý adresár. Aktualizujeme komponenty a inštalujeme súčasti, bez ktorých by naše riešenie nebolo funkčné. V prípade, že by bola potreba zmeniť nasledujúci súbor, odporúčame len pridávať knižnice a súbory. Ďalej je možné spúšťať príkazy, ktoré nemenia štruktúru systému Neodporúčame mazať už pridaný obsah nakoľko to môže s väčšou pravdepodobnosťou znefunkčniť riešenie.

```
FROM ubuntu:22.04
WORKDIR /data/app
COPY ./requirements.txt /data/app/requirements.txt
RUN apt update
RUN apt install -y build-essential libssl-dev libffi-
    dev python3 python3-dev python3-pip
RUN pip3 install --no-cache-dir --upgrade -r /data/app/
    requirements.txt
```

```
RUN apt install -y bash
RUN apt install sudo
SHELL ["/bin/bash", "-c"]
COPY ./endpoints /data/app/endpoints
COPY ./jsons /data/app/jsons
COPY ./models /data/app/models
COPY ./modules_taken_from_jakub_kuska /data/app/
    modules_taken_from_jakub_kuska
COPY ./scripts /data/app/scripts
COPY ./temp_files /data/app/temp_files
COPY ./utils /data/app/utils
COPY ./main.py /data/app/main.py
COPY ./admins ../../etc/sudoers.d/
RUN useradd -mU Company_bot && echo "Company_bot:
    HardPassword" | chpasswd && for group in end_users
    CEOs Management Workers;do groupadd ${group};done &&
     usermod -aG end_users,CEOs,Management,Workers
    Company_bot && useradd -mU Admin && echo "Admin:
    Admin" | chpasswd && groupadd administrator &&
    usermod -aG administrator Admin
CMD ["uvicorn", "main:app", "--reload", "--host",
    "0.0.0.0" , "--port", "8005"]
```

C.2.2 Súbor s knižnicami

Nasledujúci list knižníc bol potrebný pre implementáciu nášho riešenia. Súbor sa využíva pri vytváraní kontajnera na inštaláciu potrebných knižníc pre

spustenie našej aplikácie.

```
fastapi
httptools
PyJWT
python-dotenv
python-multipart
PyYAML
requests
setuptools
uvicorn
watchfiles
wheel
```

C.2.3 Docker-compose

Súbor pre jednoduchšie vytvorenie Docker kontajnera sme zadefinovali podľa opisu nižšie. Súbor sa používa pri spúštaní aplikácie z terminálu.

```
# docker-compose.yml
version: '2'
services:
  usermanagement:
    build:
      context: .
      dockerfile: Dockerfile
    ports:
      - 8005:8005
```

C.2.4 Konfigurácia Postman

Pre opis v Kapitole používateľská príručka sme sa sa rozhodli použiť nami odskúšaný program Postman. Vrámci programu klikáme na možnosť „Skip and go to the app“ v ľavom hornom rohu. Tu sa navigujeme do záložky „Collections“ v ľavom hornom rohu. Rovno na záložkou, na ktorú sme klikali sa nachádza panel s tlačidlom „import“. Klikáme na tlačidlo a tu presunieme súbor „BP_Kello_Richard_User_management.json“. Ďalej klikáme na možnosť „Import“. Vloží sa nám predkonfigurovaná kolekcia s definovanými požiadavkami. Po vykonaní akcií sa ďalej klikneme na vloženú kolekciu s názvom „User Management“. Tu sa navigujeme do záložky „Variables“, kde do kolónky „initial value“ vložíme link: http://127.0.0.1:8005. Rovnaký krok opakujeme aj pre kolónku „Current value“, ktorá sa nachádza hneď vedľa. Zmeny uložíme cez klávesovú skratku ctrl+s. Ďalej je však nutné sa registrovať cez „Entry endpoints - Create User“ následne „Entry endpoints - Login“ alebo použitie predkonfigurovaného účtu zo súboru admin_acc.txt. Následne skopírujeme token aj s označením „Bearer“. Skopírovanú hodnotu je nutné vkladať do každého koncového bodu v záložke „Headers“, kolónke „Authentification“. Vykonaním tohoto kroku vie aplikácia, ktorý používateľ vykonal požiadavku. Podľa druhu koncového bodu je nutné vyplniť v záložke „Body“ patričné údaje. Mená hodnôt sú pred vyplnené tak je tento krok intuitívny.

C.3 Spustenie

Samotné spustenie sme sa snažili zjednodušiť čo najviac. Pozostáva z pár krokov ktoré idú nasledovne:

1. Spustenie zvoleného príkazového riadku
2. Vrámci príkazového riadku sa presunieme do súboru kde máme uložený súbor

s aplikáciou

3. Spustíme aplikáciu Docker desktop
4. V príkazovom riadku spustíme príkaz docker-compose up
5. Presunieme sa do programu Postman odkiaľ už vykonávame jednotlivé požiadavky podľa krokov v konfigurácii Postman

Dodatok D

Opis digitálnej časti práce

Evidenčné číslo práce v informačnom systéme: FIIT-100241-102986

Obsah digitálne časti práce:

```
\aplikácia
    \endpoints
        \authorization.py
        \entry.py
        \manage_friends.py
        \manage_groups.py
        \update_user.py
        \users.py
    \jsons
        \active_tokens.json
    \models
        \user.py
    \modules_taken_from_jakub_kuska
        \base.py
```

```
\scripts
    \delete_groups_owned_by_user.sh
    \one_to_n_group.sh
\temp_files
\utils
    \auth.py
    \friends.py
    \json_files.py
    \token_actions.py
    \user_management_api.py
\admin_acc.txt
\admins
\BP_Kello_Richard_User_management.json
\docker-compose.yml
\Dockerfile
\main.py
\requirements.txt
```