**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ**
**І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ**

**Факультет ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ**

| **ПОГОДЖЕНО** | **ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ** |
|---|---|
| **Декан факультету** Інформаційних технологій | **Завідувач кафедри** Комп'ютерних систем, мереж та кібербезпеки |
| _____Ігор БОЛБОТ (підпис) | _____Дмитро КАСАТКІН (підпис) |
| «___»_____2026 р. | «___»_____2026 р. |

# БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

На тему: «Автоматизована система керування безпечним VPN-доступом до корпоративних ресурсів»

Спеціальність F7 «Комп'ютерна інженерія»

Освітньо-професійна програма «Комп'ютерна інженерія»

**Гарант освітньої програми**
канд. фіз.-мат. наук, доцент _____ (підпис) **Євгеній НІКІТЕНКО**

**Керівник бакалаврської кваліфікаційної роботи**
канд. техн. наук, доцент _____ (підпис) **Максим МІСЮРА**

**Виконав** _____ (підпис) **Артем МИХАЙЛОВСЬКИЙ**

**КИЇВ-2026**

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ**
**І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ**

**Факультет ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ**

**ЗАТВЕРДЖУЮ**
**Завідувач кафедри**
комп'ютерних систем, мереж та кібербезпеки
канд. пед. наук, доцент_____Дмитро КАСАТКІН
«_____»_____20____ р.

# З А В Д А Н Н Я

# ДО ВИКОНАННЯ БАКАЛАВРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ

**ЗДОБУВАЧУ**

Михайловському Артему Сергійовичу
(прізвище, ім'я, по батькові)

Спеціальність «Комп'ютерна інженерія »

Освітньо-професійна програма «Комп'ютерна інженерія »

Тема кваліфікаційної бакалаврської роботи
«Автоматизована система керування безпечним VPN-доступом до корпоративних ресурсів »

затверджена наказом ректора НУБіП України від «10»___12_____2026 р. №___3072 «С»___
Термін подання завершеної роботи на кафедру «02»___06_____2026 р.
Вихідні дані до бакалаврської кваліфікаційної роботи
Мова програмування серверної частини – Python. Мова програмування клієнтської частини – JavaScript (HTML/CSS). Векторний графічний редактор для створення діаграм -draw.io.

Перелік питань, що підлягають дослідженню:
Аналіз предметної області. Визначення сервісів продукту. Розробка інтерфейсу користувача
Тестування розробленої системи

Перелік графічного матеріалу (за необхідності).

Дата видачі завдання "__10__"_____12_____2025 р.

**Керівник бакалаврської кваліфікаційної роботи** _____ **Максим МІСЮРА**
канд. техн. наук, доцент (підпис)

**Завдання взяв до виконання** _____ **Артем МИХАЙЛОВСЬКИЙ**

(підпис)

# КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів дипломного проекту (роботи) | Строк виконання етапів проекту ( роботи ) | Примітка |
|---|---|---|---|
| 1 | Аналіз предметної області | 02.02.2026 р. | Виконано |
| 2 | Проектування програмного продукту | 16.03.2026 р. | Виконано |
| 3 | Розробка системи | 10.04.2026 р. | Виконано |
| 4 | Тестування системи | 01.05.2026 р. | Виконано |
| 5 | Оформлення пояснювальної записки | 25.05.2026 р. | Виконано |
| 6 | Оформлення графічного матеріалу | 25.05.2026 р. | Виконано |

**Студент** _____ **А.С. Михайловський**

( підпис ) (ініціали та прізвище)

**Керівник проекту (роботи)** _____ **М.Д. Місюра**

( підпис ) (ініціали та прізвище)

## РЕФЕРАТ

Пояснювальна записка: 84 сторінок, 29 рисунків, 1 таблиця, додаток, 16 джерел.

VPN, DOCKER, AUTHELIA, АУТЕНТИФІКАЦІЯ.

Об’єкт дослідження - процеси забезпечення безпечного віддаленого доступу до корпоративних мережевих ресурсів.

Метою роботи є розробка автоматизованої системи керування безпечним VPN-доступом.

Проект складається з трьох розділів.

Перший розділ присвячено аналізу матеріалу. Проведено аналіз, розробка і тестування програмного продукту.

У другому розділі розкриті питання щодо розробки алгоритму програми та визначення її сервісів.

Третій розділ присвячений тестуванню працездатності програми .

В результаті виконання дипломної роботи проведено аналіз, розробка і тестування програмного продукту.

# ЗМІСТ

**РЕФЕРАТ**.....4
ВСТУП.....6
**1 АНАЛІТИЧНИЙ ОГЛЯД**.....7
1.1 Огляд інформаційних джерел.....7
**1.2 Призначення програмного продукту**.....11
**1.3 Аналіз ТЗ**.....22
**1.4 Вибір мови та середовища програмування**.....23
**1.5 Аналіз стандартів безпеки корпоративної мережі**.....27
**2 ПРОЕКТУВАННЯ ТА РЕАЛІЗАЦІЯ АПАРАТНО-ПРОГРАМНОГО КОМПЛЕКСУ**.....32
**2.1 Архітектура мікросервісів та контейнеризація**.....32
2.2 Розробка схеми алгоритму програми.....35
2.3 Розробка програми.....40
**2.4 Мережева архітектура**.....57
3 ТЕСТУВАННЯ ПРАЦЕСДАТНОСТІ ПРОГРАМИ.....60
**3.1 Запуск та тестування функцій апаратно-програмного комплекса**.....60
**3.2 Апаратні вимоги до серверного обладнання та тестування продуктивності**.....75
ВИСНОВКИ.....82
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....83

# ВСТУП

В епоху, коли майже кожен аспект нашого життя пов'язаний з Інтернетом, захист нашої онлайн-активності став важливішим, ніж будь-коли. Від перегляду веб-сайтів та потокового контенту до обробки конфіденційних фінансових транзакцій, наш цифровий слід робить нас наразеними на сторонніх очей, витоків даних та кіберзагроз. Як для окремих осіб, так і для бізнесу підтримка конфіденційності та безпеки в цьому все більш пов'язаному світі більше не є необов'язковою, а є необхідною. Саме тут на допомогу приходять віртуальні приватні мережі (VPN). Створюючи безпечне, зашифроване з'єднання між вашим пристроєм та Інтернетом, VPN служать цифровим щитом, захищаючи ваші дані від хакерів, рекламодавців і навіть вашого власного інтернет-провайдера (ISP). Незалежно від того, чи користуєтеся ви публічним Wi-Fi в кафе, працюєте віддалено чи просто цінуєте свою конфіденційність, VPN забезпечують безцінний рівень захисту.

# 1 АНАЛІТИЧНИЙ ОГЛЯД

## 1.1 Огляд інформаційних джерел

Що таке VPN?

Віртуальна приватна мережа, або VPN, – це інструмент, розроблений для підвищення конфіденційності та безпеки в Інтернеті шляхом створення безпечного, зашифрованого з'єднання між вашим пристроєм та Інтернетом. Уявіть собі це як приватний тунель, який захищає ваші дані від будь-кого, хто може спостерігати за вами – будь то хакери, рекламодавці чи навіть ваш інтернет-провайдер (ISP). За допомогою VPN ваша онлайн-активність прихована від сторонніх очей, а ваша IP-адреса маскується, що ускладнює відстеження вашого місцезнаходження або особи. По суті, VPN створює безпечний тунель для вашого інтернет-трафіку, надсилаючи його через віддалений сервер, перш ніж він досягне місця призначення. Цей процес не тільки маскує вашу онлайн-особу, але й шифрує дані, що робить практично неможливим перехоплення або розшифрування неавторизованими сторонами.

VPN широко використовуються для різних цілей, включаючи підтримку конфіденційності, доступ до контенту з обмеженим доступом за регіоном та захист віддалених робочих з'єднань. Підприємства часто покладаються на VPN, щоб дозволити співробітникам безпечно працювати з дому, тоді як окремі особи використовують їх для обходу цензури або захисту конфіденційної особистої інформації. Цей подвійний характер робить VPN важливим інструментом як для звичайних користувачів Інтернету, так і для бізнесу. Приховуючи ваш цифровий

слід, VPN дозволяє вам переглядати веб-сторінки, працювати та спілкуватися онлайн з упевненістю та спокоєм. Але як саме VPN досягає такого рівня захисту? Розуміння механізмів, що лежать в основі функціональності VPN, є ключовим для оцінки його ролі в забезпеченні безпеки цифрового середовища.

Як працюють VPN?

VPN-клієнт змінює ваш мережевий шлях, спочатку надсилаючи ваші пакети даних на VPN-сервер, замість того, щоб надсилати їх безпосередньо до кожного пункту призначення в Інтернеті. Ключовим моментом є те, що це зміна маршрутизації та транспортування: ваш пристрій встановлює з'єднання з одним шлюзом (VPN-сервером), а потім трафік передається далі звідти. Без VPN ваш трафік зазвичай спрямовується до мережі вашого інтернет-провайдера, а потім до кожного сайту чи служби. З VPN ваш інтернет-провайдер зазвичай здійснює зашифроване з'єднання з VPN-сервером, а цей сервер пересилає трафік до кінцевого пункту призначення. Простий спосіб подумати про це такий: VPN додає додатковий крок — ваш трафік проходить через VPN-сервер — а також додає зашифровану оболонку для сегмента між вашим пристроєм і цим сервером.

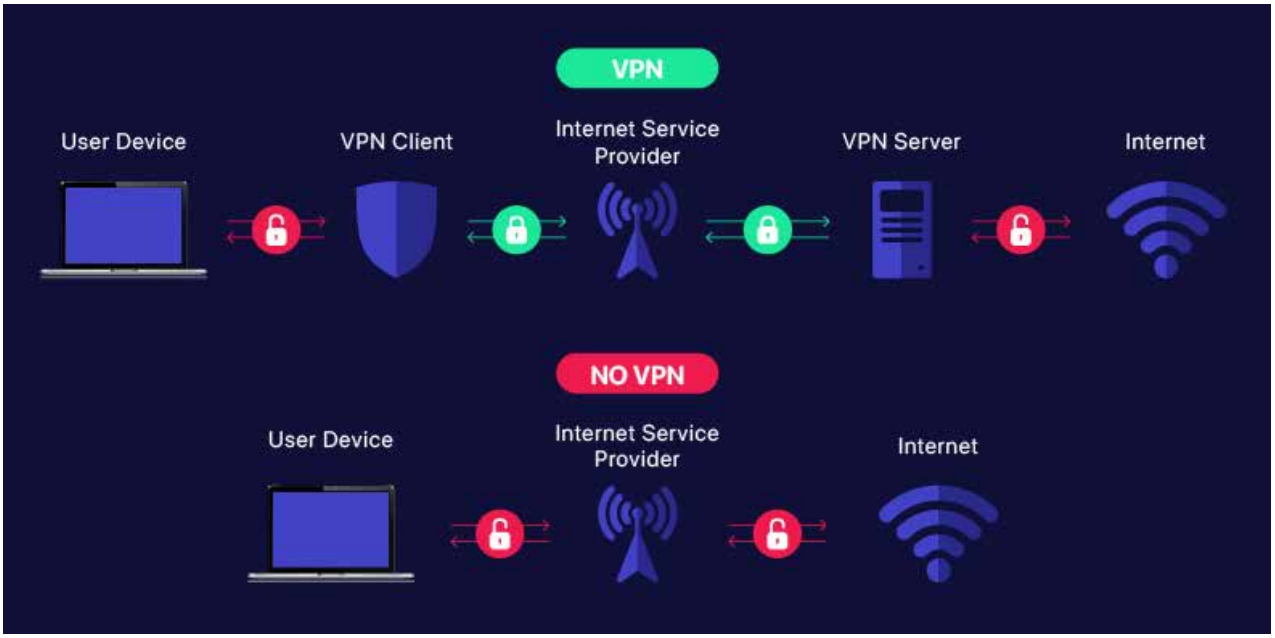


Рисунок 1.1 – Як працює VPN [13]

Зашифрований тунель змінює шлях ваших пакетів, роблячи VPN-сервер першим пунктом призначення, куди ваш пристрій спрямовує трафік. Ключовим моментом є те, що «тунель» створюється за допомогою протоколу тунелювання та шифрування: протокол тунелювання забезпечує структуру для передачі трафіку через з'єднання, а шифрування захищає дані під час передачі в сегменті між VPN-клієнтом та VPN-сервером. Інкапсуляція – це конкретний механізм, що стоїть за тунелем. Замість того, щоб надсилати ваші оригінальні пакети безпосередньо до кожного пункту призначення, VPN-клієнт інкапсулює ці пакети в новий зовнішній пакет. Маршрутизатори в Інтернеті потім маршрутизують на основі адресації цього зовнішнього пакета, який вказує на VPN-сервер. На практиці VPN-сервер діє як ретранслятор. Він отримує тунельований трафік, обробляє його, а потім надсилає трафік далі до пункту призначення. Тим не менш, шифрування VPN пов'язане з сегментом клієнт-сервер; те, що відбувається після VPN-сервера, залежить від того, як VPN пересилає трафік та яке шифрування існує між VPN-сервером та пунктом призначення.

Які бувають типи VPN?

VPN бувають різних форм, кожна з яких розроблена для задоволення конкретних потреб та сценаріїв користувачів. Хоча всі VPN спрямовані на захист онлайн-активності та конфіденційності, їхня функціональність та варіанти використання можуть суттєво відрізнятися. Нижче наведено основні типи VPN:

1. VPN віддаленого доступу VPN віддаленого доступу зазвичай використовуються окремими особами для безпечного підключення до приватної мережі з віддаленого місця. Ці VPN особливо популярні серед співробітників, які працюють вдома або подорожують, оскільки вони дозволяють отримати доступ до внутрішніх систем організації без шкоди для безпеки. Наприклад, віддалений працівник може використовувати VPN для безпечного доступу до ресурсів компанії, таких як бази даних або спільні файли, з будь-якої точки світу.
2. VPN типу «сайт-сайт» VPN типу «сайт-сайт», які часто використовуються великими організаціями, безпечно з'єднують кілька мереж через Інтернет. Наприклад, компанія з офісами в різних містах може використовувати VPN типу «сайт-сайт» для зв'язку своїх мереж, забезпечуючи безперебійний зв'язок та спільний доступ до ресурсів. Цей тип VPN зазвичай реалізується за допомогою спеціалізованого обладнання та ідеально підходить для підприємств, які прагнуть захистити дані, що обмінюються між офісами.
3. Персональний VPN Також відомі як персональні VPN, вони розроблені для окремих користувачів, які прагнуть конфіденційності та безпеки у своїй онлайн-активності. Споживчі VPN широко використовуються для шифрування трафіку в публічних мережах Wi-Fi, обходу географічних

обмежень для доступу до регіонально заблокованого контенту та збереження анонімності. Ці VPN прості у використанні, часто вимагають лише простого встановлення програми.

4. Хмарний VPN. Використовується для безпечного підключення користувачів або мереж до хмарної інфраструктури, що розміщується у провайдерів, таких як AWS або Google Cloud.

Кожен тип VPN служить різним цілям, будь то особиста конфіденційність, безпечне ділове спілкування чи підключення до великомасштабної мережі. Розуміння цих відмінностей може допомогти користувачам вибрати правильне VPN-рішення для своїх конкретних потреб. Маючи чітке розуміння типів VPN, не менш важливо зважити переваги, які вони надають, оскільки ці переваги підкреслюють, чому VPN є незамінними інструментами в сучасному цифровому світі.

## 1.2 Призначення програмного продукту

Вибір VPN-протокола

Коли підключаємося до VPN, то не просто «вмикаємо конфіденційність». За лаштунками, протокол VPN визначає, як шифруються наші дані, наскільки швидко працює ваше з’єднання та наскільки ви насправді захищені. Але з такими варіантами, як OpenVPN, WireGuard та IKEv2, як дізнатися, який з них обрати? Чи має це взагалі значення? Ми розглянемо три найпопулярніші протоколи VPN, порівняємо їхні сильні та слабкі сторони та виберемо, який з них найкраще відповідає вашим конкретним потребам.

Що таке VPN-протокол? VPN-протокол – це набір правил, який визначає, як ваші дані передаються між вашим пристроєм та VPN-сервером. Уявіть собі це як «мову», якою розмовляє ваш VPN. Різні протоколи використовують різні методи шифрування, швидкість з'єднання та підходи до безпеки. Протокол, який ви використовуєте, впливає на:

- Швидкість – швидкість вашого з'єднання
- Безпека – захист ваших даних
- Стабільність – надійність вашого з'єднання
- Сумісність – підтримка пристроїв та мереж

Більшість сучасних VPN-провайдерів пропонують кілька протоколів, що дозволяє вам вибирати між ними залежно від ваших потреб. Давайте розглянемо трьох основних конкурентів.

WireGuard

Сучасний претендент WireGuard – це найновіший великий VPN-протокол, офіційно випущений у 2020 році. Він був розроблений з нуля, щоб бути простішим, швидшим та безпечнішим, ніж старіші протоколи. Як працює WireGuard. Використовує найсучаснішу криптографію: ChaCha20 для шифрування, Curve25519 для обміну ключами та BLAKE2s для хешування. Він працює виключно через UDP та має крихітну кодову базу, що становить лише ~4000 рядків.

Плюсами WireGuard є неймовірно швидкий - значно швидший, ніж OpenVPN (часто в 2-3 рази) Мінімальний код - ~4000 рядків. Сучасна криптографія - найновіші стандарти шифрування. Низьке споживання батареї - ідеально підходить для мобільних пристроїв. Швидке підключення -

підключення за мілісекунди, а не за секунди. Вбудований у ядро Linux - нативна підтримка з Linux 5.6

Мінуси WireGuard Тільки UDP - може блокуватися деякими брандмауерами. Статичні IP-адреси - проблеми з конфіденційністю (VPN-провайдери обходять це). Новіший - менше тестування в реальному світі, ніж OpenVPN. Обмежене обфускація - важче замаскувати під звичайний трафік.

IKEv2/IPsec

IKEv2 (Internet Key Exchange версії 2) у поєднанні з IPsec – це протокол, особливо популярний на мобільних пристроях завдяки своїй здатності швидко відновлювати підключення після змін у мережі.

Як працює IKEv2? Обробляє початкове налаштування з'єднання та обмін ключами, тоді як IPsec забезпечує фактичне шифрування. Він підтримує шифрування AES-256 та використовує MOBIKE (Mobility and Multihoming Protocol) для безперебійної комутації мережі.

IKEv2 чудово підходить для мобільних пристроїв - миттєво перепідключається при перемиканні мереж. Висока швидкість - часто порівнянна з WireGuard. Надійне шифрування - AES-256 з ідеальною прямою секретністю. Вбудована підтримка - вбудовано у Windows, macOS, iOS. Стабільні з'єднання - коректно обробляє зміни в мережі

Недоліками IKEv2 є проблеми з брандмауером - легше заблокувати, ніж OpenVPN. Не з відкритим вихідним кодом - реалізація Microsoft/Cisco є власною Обмежені параметри портів - Використовує UDP 500/4500 (можна заблокувати). Менш гнучкий - менше варіантів налаштування.

OpenVPN

Є золотим стандартом VPN-протоколів з моменту його випуску в 2001 році. Він має відкритий вихідний код, а це означає, що дослідники безпеки з усього світу перевіряли його код на наявність вразливостей.

OpenVPN використовує бібліотеку OpenSSL для шифрування, підтримуючи як AES-256 (шифрування військового рівня), так і різні методи автентифікації. Він може працювати через порти UDP (швидше) або TCP (надійніше).

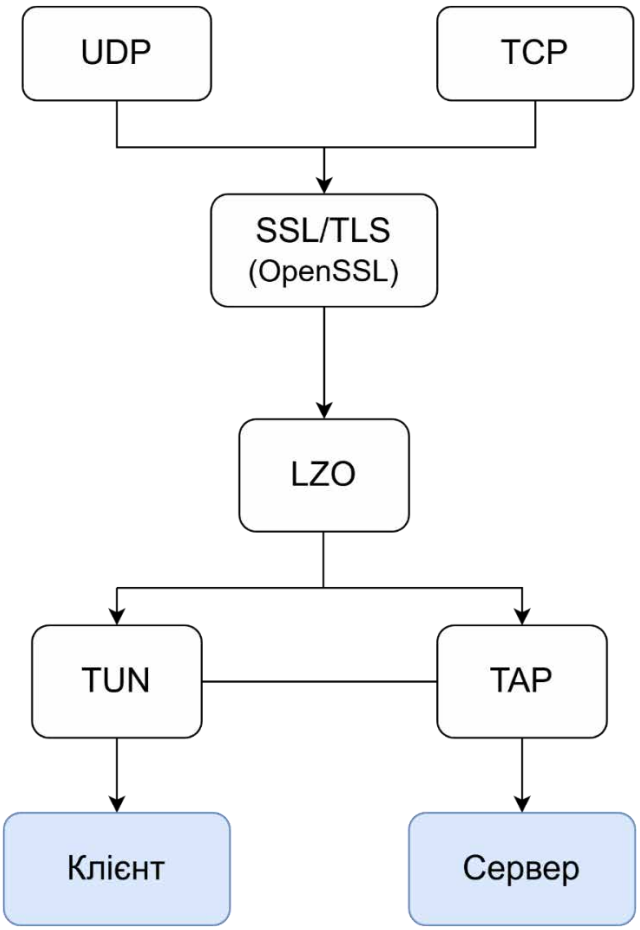


Рисунок 1.2 – Схема OpenVPN

Перевагами OpenVPN є високий рівень безпеки - понад 20 років аудитів безпеки. Високо-конфігурований - можна налаштувати для будь-якого сценарію.

Працює майже на всьому - Windows, macOS, Linux, iOS, Android, маршрутизатори Обходить більшість брандмауерів - може працювати на порту 443 (HTTPS-порт). Відкритий вихідний код - прозорий, аудитований код.

Недоліками OpenVPN є повільніший за новіші протоколи - більше накладних витрат означає зниження швидкості. Вища завантаження процесора - може розряджати батарею на мобільних пристроях. Велика кодова база - ~400 000 рядків коду (більший потенціал для помилок).

Нижче наведено таблицю порівняння протоколів, сформовану на основі джерел інформації:

| **Функція** | **OpenVPN** | **WireGuard** | **IKEv2/IPsec** |
|---|---|---|---|
| **Швидкість** | Добре | Відмінно | Дуже добре |
| **Безпека** | Відмінно | Відмінно | Дуже добре |
| **Стабільність** | Дуже добре | Добре | Відмінно |
| **Продуктивність мобільних пристроїв** | Добре | Відмінно | Відмінно |
| **Обхід брандмауера** | Відмінно | Обмежена | Обмежена |
| **Відкритий код** | Так | Так | Частково |
| **Розмір коду** | ~400,000 рядків | ~4,000 рядків | ~20,000 рядків |
| **Шифрування** | AES-256 | ChaCha20 | AES-256 |

Таблиця 1.1 – Порівняння протоколів [14]

Надано перевагу OpenVPN, так як адаптується як до конфіденційних завдань, так і до потреб звичайного перегляду веб-сторінок - це найкращий вибір для конфіденційності та захисту даних під час обробки критичної інформації. OpenVPN виділяється як один з найкращих VPN-протоколів, що пропонуються більшістю комерційних VPN-провайдерів. Він пройшов кілька сторонніх аудитів

і активно підтримується та вдосконалюється спільнотою розробників для усунення вразливостей.

Має підтримку обох транспортних протоколів з можливістю переходу з UDP на TCP без зміни архітектури. У поточній реалізації використовується UDP як основний транспортний протокол, що забезпечує: мінімальні затримки при передачі даних, вищу пропускну здатність мережі. Водночас, архітектура залишає можливість переходу на TCP у випадках, коли: потрібна стабільна і гарантована доставка даних.

Використовується підтримка вбудованих відкритих ключів PKI, що є основою безпеки даної системи: кожен користувач отримує індивідуальний сертифікат**,** згенерований через Easy-RSA. Також інтегрується із зовнішніми системами автентифікації, що використовуються у проєкті: взаємодіє з Authelia для багатофакторної автентифікації, підтримує plugin-механізм для підключення до LDAP-каталогу.

Чому використано сервіс Authelia?

Authelia — це сервер автентифікації 2FA та SSO, призначений для безпеки програм та користувачів. Його можна вважати розширенням зворотних проксі-серверів, оскільки він надає функції, специфічні для автентифікації. Серед інших функцій ви знайдете:

- кілька методів двофакторної автентифікації;
- перевірка особи під час реєстрації пристроїв з двофакторною автентифікацією;
- самостійне скидання паролів користувачів;

- блокування облікових записів після занадто великої кількості спроб (відоме як регулювання).

Authelia є супутником зворотних проксі-серверів, таких як Traefik, Nginx. Його можна розглядати як розширення цих проксі-серверів, що забезпечує функції автентифікації та портал входу.

Authelia підключена до зворотного проксі-сервера, але ніколи не підключена до серверних частин програм, саме тому корисні навантаження, надіслані клієнтами захищеного API, ніколи не досягнуть Authelia, лише частини автентифікації. Отже, захищені API можуть бути REST, GraphQL або будь-яким іншим типом API поверх HTTP.

Зворотні проксі-сервери налаштовані таким чином, що кожен вхідний запит генерує запит на автентифікацію, що надсилається до Authelia. Authelia відповідає та дає інструкцію зворотному проксі-серверу або дозволити проходження вхідного запиту, або заблокувати його, оскільки користувач не автентифікований або недостатньо авторизований.

Коли перший запит неавтентифікованого користувача потрапляє на зворотний проксі-сервер, Authelia визначає, що користувач не автентифікований, оскільки разом із запитом не було надіслано файл cookie сесії. Відповідно, Authelia перенаправляє користувача на портал автентифікації, наданий самою Authelia. Потім користувач може виконати робочий процес автентифікації, використовуючи цей портал, щоб отримати файл cookie сесії, дійсний для всіх піддоменів домену, захищеного Authelia.

Коли користувач знову відвідує початковий веб-сайт, запит надсилається разом із файлом cookie сесії, який пересилається в запиті на автентифікацію до

Authelia. Цього разу Authelia може перевірити автентифікацію користувача та наказати зворотному проксі пропустити запит.

Authelia працює лише для веб-сайтів, що обслуговуються через HTTPS, оскільки файл cookie сесії може передаватися лише через захищені з'єднання.

Вибір середовища розгортання проєкту OpenVPN

Розгортання є важливою частиною будь-якого проєкту. Ми візьмемо для порівняння 3 середовища для розгортання: Bare Metal(на пряму на сервері), Віртуальна машина і Docker.

Переваги серверів Bare Metal. Повний контроль: з сервером Bare Metal ви маєте повний контроль над апаратним та програмним стеком. Це означає, що можна оптимізувати систему для своїх конкретних потреб та отримати найкращу можливу продуктивність. Фізична ізоляція: оскільки ви єдиний "орендар" на сервері, вам не потрібно турбуватися про проблему «шумного сусіда». Це коли дії іншого "орендаря" впливають на вашу продуктивність. Крім того, фізична ізоляція забезпечує високий рівень безпеки – вам не потрібно турбуватися про атаки побічних каналів або інші вразливості безпеки, які можуть виникнути в спільних середовищах. Відповідність нормативним вимогам: для програм, які повинні відповідати суворим нормативним вимогам або вимогам до дотримання нормативних вимог, розгортання на пряму на сервері може бути єдиним варіантом. Маючи повний контроль над системою, ви можете гарантувати, що вона відповідає всім необхідним стандартам.

Хоча використання Bare Metal має багато переваг, є також деякі недоліки, які слід враховувати. Вартість: сервери можуть бути дорогими для придбання та обслуговування. Вам потрібно буде придбати обладнання заздалегідь, а також

вам знадобиться команда компетентних фахівців для управління системою. Складність управління: оскільки ви маєте повний контроль над системою, ви також несете відповідальність за її управління. Це може бути складним і трудомістким завданням, особливо якщо у вас немає досвіду адміністрування серверів. Масштабованість: сервери може бути важко масштабувати. Якщо вам потрібно більше ресурсів, вам потрібно буде придбати додаткове обладнання та налаштувати його для роботи з вашою існуючою конфігурацією. Це може бути повільним і дорогим процесом.

Віртуальна машина – це програмне забезпечення, яке емулює фізичний комп'ютер. Цей процес відомий як віртуалізація. Віртуальні машини пропонують кілька переваг порівняно з фізичними машинами, зокрема: однією з суттєвих переваг віртуальних машин є економія коштів. Завдяки віртуалізації кілька віртуальних машин можуть використовувати одне й те саме обладнання, що означає більше використання ресурсів. Це зменшує витрати на обладнання та мінімізує споживання енергії, що призводить до зниження витрат для організації.

Масштабування віртуальної машини є більш керованим, ніж купівля та встановлення нового фізичного обладнання. Крім того, деяке програмне забезпечення для віртуалізації може навіть переміщувати запущені віртуальні машини з одного голого обладнання на інше без будь-яких простоїв. Гнучкість: віртуальні машини пропонують бізнесу більшу гнучкість. Завдяки віртуалізації організації можуть вибирати віртуальні машини різного розміру відповідно до своїх робочих навантажень. Вони можуть вибирати віртуальні машини загального призначення з кількома ядрами процесора та кількома ГБ пам'яті або високопродуктивні віртуальні машини з сотнями ядер та терабайтами пам'яті.

Хоча віртуальні машини пропонують кілька переваг, є також деякі недоліки, які слід враховувати. Проблема конкуренції за ресурси: віртуальні машини можуть зазнавати зниження продуктивності через спільне використання апаратних ресурсів. У цій ситуації продуктивність програми може постраждати через обмежену доступність ресурсів. Вразливості безпеки: віртуальні машини, що працюють на одному й тому ж апаратному забезпеченні, використовують одні й ті ж фізичні ядра процесора. Вони вразливі до атак, спрямованих на недоліки конструкції сучасних процесорів. Атаки побічних каналів, такі як Meltdown та Spectre, є одними з відомих прикладів.

Docker — це потужна та широко використовувана платформа контейнеризації, яка дозволяє розробникам створювати, розгортати та запускати програми в гнучкому, легкому середовищі. Розглянемо деякі ключові переваги використання Docker у процесах розробки та розгортання:

- Портативність: Однією з основних переваг Docker є його портативність. Образи Docker розроблені таким чином, щоб бути легкими та автономними, тобто їх можна легко переміщувати з одного середовища в інше. Це спрощує розгортання програм у різних середовищах, включаючи локальні машини розробки, середовища тестування та виробничі сервери.
- Узгодженість: Ще однією перевагою Docker є узгодженість. Образи Docker розроблені таким чином, щоб бути узгодженими в різних середовищах, що гарантує однакову роботу програми незалежно від того, де вона розгорнута. Ця узгодженість дозволяє легко виправляти проблеми та гарантувати, що програма працює належним чином у виробничому середовищі.

- Ефективність: Docker розроблений для ефективності як з точки зору використання ресурсів, так і часу розгортання. Контейнери Docker легкі та потребують мінімальних накладних витрат, що робить їх ефективнішими за традиційні віртуальні машини. Крім того, образи Docker можна створювати швидко та легко, що дозволяє розробникам швидко виконувати ітерації та швидше розгортати зміни.
- Безпека: Docker надає кілька функцій безпеки, які роблять його безпечнішим варіантом розгортання додатків, ніж традиційні технології віртуалізації. Контейнери Docker ізольовані від операційної системи хоста, що забезпечує додатковий рівень безпеки. Крім того, Docker надає інструменти для керування доступом та дозволами користувачів, що дозволяє легко контролювати, хто має доступ до конфіденційних даних та ресурсів.
- Ризики безпеки: контейнери використовують одне й те саме ядро операційної системи, а це означає, що якщо зловмисник отримає доступ до одного контейнера, він зможе отримати доступ до інших контейнерів на тому ж хості. Складність: керування контейнерами та їхніми залежностями може бути складним, особливо у великих масштабах. У розгортанні та управлінні контейнерами бере участь багато рухомих частин, що може призвести до таких проблем, як неправильні конфігурації та вразливості безпеки. Обмежена функціональність: контейнери призначені для запуску однієї програми або служби та можуть не підходити для складних програм, які потребують безперебійної спільної роботи кількох служб або компонентів. Це може ускладнити розгортання та керування певними типами програм у контейнерному середовищі.

На завершення, Docker пропонує багато переваг порівняно з традиційними технологіями віртуалізації. Його портативність, узгодженість, ефективність та безпека роблять його потужним інструментом як для розробників, так і для системних адміністраторів, саме тому я наадаю йому перевагу.

### 1.3 Аналіз ТЗ

Для дипломного проектування  необхідно розробити корпоративний VPN. Він призначений для безпечного доступу до корпоративних ресурсів.

Вхідні дані від користувача:

- логін та пароль для авторизації в Authelia;
- одноразовий код для проходження двофакторної автентифікації (TOTP з додатку Microsoft authenticator/Google authenticator або WebAuthn через апаратний ключ, біометрію пристрою);
- електронна пошта користувача для отримання коду підтвердження при реєстрації двофакторної аутентифікації.

Основні можливості програми:

- авторизація користувачів з обов’язковим проходженням двофакторної аутентифікації;
- автоматичне створення персональних сертифікатів користувачів для підключення Openvpn;
- завантаження особистого конфігураційного файлу для підключення до VPN;
- створення облікових записів користувачів через LLDAP;
- захищена передача даних між клієнтом та сервером з використанням протокола TLS;

- простий та зручний веб-інтерфейс для роботи користувача.

### 1.4 Вибір мови та середовища програмування

Ця робота виконана мовою програмування Python, оскільки для неї існує безліч різноманітних бібліотек, що дозволяють значно спростити виконання будь-яких завдань і реалізацію інтеграцій у порівнянні з іншими мовами.

Python спрощує все завдяки своєму легкозрозумілому синтаксису. Він дозволяє розробникам висловлювати складні ідеї меншою кількістю рядків коду порівняно з деякими іншими мовами, такими як Java. Наприклад, те, що може зайняти 7 рядків у C++, потребує лише 3 рядків у Python. Як результат, Python значно скорочує обсяг шаблонного коду та пришвидшує час розробки. Менша кількість рядків коду також призводить до таких переваг, як менше використання простору та менше налагодження.

Здатність Python гнучко розгортати програми подібна до універсального інструменту для розсилки програм у різні місця. Чи то на серверах, настільних комп'ютерах, чи навіть у хмарі, Python робить цей процес плавним та ефективним. Уявіть собі команду, яка хоче запустити веб-додаток. Адаптивність Python стає очевидною, оскільки Python підтримує різні способи розгортання. Від традиційних серверів до сучасних хмарних платформ, Python дозволяє розробникам вибирати найкращий метод розробки для своїх проектів. Але це ще не все — Python також добре працює з іншими технологіями, такими як Docker, роблячи розгортання ще більш узгодженим. Це як упакувати ваш додаток у спеціальний пакет, щоб він працював однаково, незалежно від того, куди він йде.

Python — це як скарбниця для розробників, наповнена набором бібліотек Python та фреймворків Python, створених спільнотою. Уявіть собі ці бібліотеки

як організовані набори інструментів, кожен з яких розроблений для вирішення конкретних завдань кодування. Понад 30 років розробники Python по всьому світу створили сотні таких бібліотек, щоб спростити життя майбутнім розробникам. Ці бібліотеки охоплюють усе: від веб-розробки до аналізу даних. Ці бібліотеки діють як готові рішення, дозволяючи розробникам зосередитися на своїх проектах, не винаходячи велосипеда. Фреймворки — це ще один потужний аспект Python. Мови програмування такі, як C++ і Java, не можуть зрівнятися з якістю та підтримкою спільноти фреймворків Python. Наприклад, розглянемо Django та Flask.

Для роботи програми використовуються такі бібліотеки:

- Flask - легкий веб-фреймворк для Python. Надає простий спосіб обробки HTTP-запитів, маршрутизації, шаблонів, сесій тощо. Використовується для створення веб-застосунків та API;
- Flask-SQLAlchemy - розширення, яке інтегрує SQLAlchemy з Flask. Спрощує налаштування бази даних та використання ORM у застосунках Flask;
- SQLAlchemy - повнофункціональний інструментарій Python SQL та ORM (Object Relational Mapper). Дозволяє взаємодію з базою даних через об'єкти Python;
- requests - бібліотека Python для HTTP-запитів. Дозволяє робити GET, POST, PUT, DELETE тощо;
- datetime - корисно для позначок часу, математичних обчислень дат, часових поясів тощо;
- pathlib - стандартний модуль Python для шляхів до об'єктно-орієнтованих файлових систем;

- getpass - модуль для безпечного введення паролів. Приховує введені користувачем дані під час набору паролів;
- functools - модуль з функціями вищого порядку та інструментами для функціонального програмування. Використовується для кешування, функцій композиції, декораторів;
- os - стандартний модуль Python, який використовується для взаємодії з операційною системою. Використовується для кросплатформних дій ОС з Python;
- sys - модуль Python для взаємодії інтерпретатора та системи. Корисно в програмах та скриптах CLI;
- re - модуль для регулярних виразів. Добре підходить для розбору та перевірки тексту;
- subprocess - модуль Python для створення процесів, виконання команд оболонки.

Вони спрощують розробку веб-додатків, пропонуючи попередньо створені компоненти та чіткі структури. Ця система спільної підтримки гарантує, що розробники мають багато ресурсів під рукою. Це робить Python мовою програмування для проектів будь-якого розміру та складності.

Для веб-розробки був вибраний Javascript, тому що — це потужна мова програмування, яка дозволяє розробникам фронтенд-інтерфейсів додавати інтерактивність та динамічні функції до веб-сайтів. Окрім покращення взаємодії з користувачем, JavaScript можна використовувати для створення анімації, перевірки форм та обробки введених користувачем даних. JavaScript широко використовується для створення інтерактивних елементів на веб-сторінках, таких як слайдери, спливаючі меню та ефекти прокручування. JavaScript можна

використовувати для отримання даних із зовнішніх джерел, таких як API, та динамічного оновлення вмісту веб-сторінки без необхідності повного оновлення сторінки. Зі зростанням попиту на веб-додатки, які забезпечують насичений взаємодію з користувачем, JavaScript став важливим інструментом для розробників фронтенд-інтерфейсів у створенні динамічних та інтерактивних веб-сайтів. На завершення, HTML, CSS та JavaScript є основою розробки фронтенд-інтерфейсів, працюючи разом для створення візуально привабливих, адаптивних та інтерактивних веб-сайтів та додатків. HTML визначає структуру веб-контенту, CSS стилізує елементи для покращення дизайну, а JavaScript додає інтерактивність та динамічну функціональність. Оволодівши цими трьома технологіями, фронтенд-розробники можуть створювати переконливий користувацький досвід, який стимулює залученість та покращує загальну зручність використання веб-сайту чи застосунку.

Середовищем розробки було вибрано PyCharm. Виділяється як широко використовуване інтегроване середовище розробки, адаптоване для програмування на Python. Розроблене JetBrains, воно охоплює широкий спектр можливостей, що підвищують ефективність та задоволення від розробки на Python. PyCharm — це надійне та насичене функціями IDE, яке багато розробників Python використовують, але воно не позбавлене своїх недоліків. Ці обмеження включають корисне споживання ресурсів, криву освоєння, ціну Professional Edition, повільніший запуск екземплярів та обмежену підтримку мов.

Функції PyCharm:

- Інтелектуальне автодоповнення коду: PyCharm пропонує інтелектуальне автодоповнення коду, яке пропонує фрагменти коду,

назви змінних та назви функцій під час введення тексту, заощаджуючи ваш час та зменшуючи кількість помилок.

- Інтегрований відладчик: інтегрований відладчик дозволяє встановлювати точки зупинки, перевіряти змінні та покроково проходити по коду, щоб виявляти та виправляти проблеми.
- Потужний рефакторинг: можна легко рефакторувати свій код за допомогою PyCharm, роблячи його чистішим та зручнішим у підтримці. Він може допомогти вам перейменовувати змінні, витягувати методи тощо.
- Інтеграція з системами контролю версій: PyCharm бездоганно інтегрується з системами контролю версій, такими як Git, що спрощує відстеження змін та співпрацю з іншими користувачами над вашими проектами.
- Аналіз коду: PyCharm містить інструменти аналізу коду, які можуть виявляти особливості коду та пропонувати покращення для вашого коду, допомагаючи вам писати чистіші та ефективніші програми на Python

### 1.5 Аналіз стандартів безпеки корпоративної мережі

Побудова захищеної корпоративної мережі неможлива без дотримання визнаних міжнародних стандартів та рекомендацій у сфері інформаційної безпеки. Розроблена система VPN-доступу спирається на кілька ключових стандартів, кожен з яких регулює окремий аспект захисту мережевої інфраструктури.

Міжнародний стандарт ISO/IEC 27001 визначає вимоги до системи управління інформаційною безпекою і є основоположним документом для будь-якої організації, що прагне забезпечити комплексний захист інформаційних активів. Стандарт охоплює організаційні, технічні та процедурні аспекти захисту і структурований навколо концепції управління ризиками. У контексті розробленої системи особливу роль відіграють контролі Annex A, зокрема контроль 8.20, що регулює мережеву безпеку, та контроль 8.5, що встановлює вимоги до захищеної автентифікації. Реалізація цих контролів у системі забезпечується через централізоване керування обліковими записами за допомогою LLDAP, що виступає єдиним джерелом істини щодо ідентифікаційних даних користувачів, а також через систему автентифікації Authelia, яка реалізує багатофакторну перевірку особи перед наданням будь-якого доступу до ресурсів. Шифрування каналу передачі даних забезпечується на двох рівнях — на рівні зворотного проксі Nginx з використанням TLS та на рівні VPN-тунелю OpenVPN з алгоритмом AES-256-GCM, що відповідає вимогам контролю 8.24 щодо використання криптографії.

Національний інститут стандартів і технологій США у публікації NIST SP 800-77 формулює рекомендації щодо побудови захищених VPN-з'єднань для корпоративних мереж. Відповідно до цього документа, ефективна VPN-система повинна забезпечувати надійну взаємну автентифікацію сторін з'єднання, стійке шифрування трафіку та механізм перевірки цілісності даних. У розробленій системі автентифікація реалізована на двох незалежних рівнях: на рівні VPN-клієнта через сертифікатну автентифікацію на основі PKI-інфраструктури та на рівні застосунку через Authelia з підтримкою одноразових паролів TOTP. Багаторівнева автентифікація ефективно унеможливлює несанкціонований

доступ, навіть якщо один із факторів буде скомпрометовано. Наприклад, зловмисник, який заволодів паролем користувача, не зможе отримати доступ без фізичного доступу до пристрою з установленим додатком автентифікації. Аналогічно, наявність сертифіката без відповідних облікових даних також не дозволить пройти перевірку в системі Authelia.

Особливе місце в архітектурі системи займає відповідність принципам Zero Trust, сформульованим у публікації NIST SP 800-207. Концепція нульової довіри кардинально відходить від традиційної моделі периметрової безпеки, за якої всі ресурси всередині корпоративної мережі вважаються довіреними за замовчуванням. Натомість Zero Trust вимагає верифікації кожного запиту незалежно від його походження — як із зовнішніх, так і з внутрішніх мереж. Розроблена система реалізує ключові тенети цієї концепції: Authelia перевіряє кожну спробу доступу незалежно від того, чи перебуває користувач всередині VPN-тунелю, доступ до ресурсів надається на основі перевіреної ідентичності, а не мережевого розташування, а використання груп у LLDAP дозволяє реалізувати принцип найменших привілеїв, за яким кожен користувач отримує доступ лише до тих ресурсів, які необхідні для виконання його службових обов'язків.

Стандарт CIS Controls, розроблений Центром інтернет-безпеки, визначає пріоритетний набір заходів захисту корпоративних мереж, впорядкованих за рівнем критичності. З точки зору розробленої системи найбільш релевантними є Control 4, що регулює безпечну конфігурацію корпоративних ресурсів, Control 6, що охоплює керування обліковими записами, та Control 12, присвячений управлінню мережевою інфраструктурою. Відповідність Control 6 забезпечується через централізоване зберігання та управління обліковими записами в LLDAP з

можливістю миттєвого відкликання доступу шляхом одночасної деактивації облікового запису та відкликання сертифіката через список анульованих сертифікатів (CRL). Вимоги Control 12 виконуються завдяки ізоляції всіх сервісів у межах Docker bridge-мережі vpn_net, де єдиною точкою входу ззовні є зворотний проксі Nginx, що суттєво скорочує поверхню атаки.

Загальний регламент захисту персональних даних Європейського Союзу встановлює вимоги до обробки та захисту персональних даних громадян ЄС Розроблена система дотримується принципу мінімізації даних, зберігаючи в LLDAP лише ті атрибути користувача, які потрібні для автентифікації та авторизації. До них належать ідентифікатор, електронна пошта та інформація про належність до груп.Принцип цілісності та конфіденційності забезпечується шифруванням усіх каналів передачі даних, а право на видалення реалізується через механізм деактивації облікового запису, що повністю і незворотно позбавляє користувача будь-якого доступу до корпоративних ресурсів.

Важливим аспектом безпеки є також фізична та логічна ізоляція компонентів системи. Контейнеризація на основі Docker забезпечує ізольоване середовище виконання для кожного сервісу, унеможливлюючи горизонтальне переміщення потенційного зловмисника між компонентами у випадку компрометації одного з контейнерів. Ізольована bridge-мережа vpn_net гарантує, що жоден із внутрішніх сервісів — ні Authelia, ні Redis, ні LLDAP, ні vpn-admin — не є доступним безпосередньо з інтернету. Весь зовнішній трафік обов'язково проходить через Nginx, який виконує функцію єдиної контрольованої точки входу з підтримкою TLS-термінації та інтеграцією із системою автентифікації.

Таким чином, розроблена система VPN-доступу відповідає вимогам ключових міжнародних стандартів інформаційної безпеки. Поєднання централізованої автентифікації через LLDAP та Authelia з багатофакторною перевіркою, PKI-інфраструктури для керування сертифікатами, шифрування на всіх рівнях передачі даних та ізоляції компонентів у контейнерному середовищі формує комплексний захист, що відповідає сучасним вимогам до корпоративних мережевих систем і реалізує принципи як традиційної глибокоешелонованої оборони, так і сучасної концепції нульової довіри.

# 2 ПРОЕКТУВАННЯ ТА РЕАЛІЗАЦІЯ АПАРАТНО-ПРОГРАМНОГО КОМПЛЕКСУ

## 2.1 Архітектура мікросервісів та контейнеризація

Архітектура мікросервісів являє собою підхід до проектування, який структурує додатки як набір незалежно розгортаних служб, кожна з яких відповідає за певну функціональність. На відміну від традиційних монолітних додатків, де вся бізнес-логіка централізована, кожна служба обробляє одну або кілька певних бізнес-функцій і може бути розроблена, розгорнута та масштабована незалежно.

Фундаментальні принципи мікросервісів включають:

Слабке зв'язування гарантує, що мікросервіси можуть бути незалежними з точки зору змін, і жодних хвиль не створюється через їхні зміни по всій системі. Сервіси взаємодіють через чітко визначені API без знання внутрішніх реалізацій.

Під час роботи з мікросервісами ми зосереджуємося на чотирьох вимірах ізоляції: стан, простір, час та збій. Кожна служба керує своїм власним станом і даними, запобігаючи прямому доступу до бази даних між службами. Служба - це працююча сутність, яку можна масштабувати горизонтально без обов'язкового масштабування всієї системи. Це призводить до належного використання ресурсів та зниження витрат.

Різні сервіси можуть використовувати різні технологічні стеки та можуть бути оновлені незалежно, що забезпечує гнучкість у виборі технологій. Контейнеризація Docker відіграє вирішальну роль у розгортанні мікросервісів.

Контейнери Docker потребують менше обчислювальних ресурсів, ніж віртуальні машини. За допомогою Docker можна зменшити накладні витрати на продуктивність та розгорнути тисячі мікросервісів на одному сервері. Docker стандартизує середовища розробки, тестування та виробництва, усуваючи проблему «працює на моїй машині». Кожен контейнер працює незалежно з власним набором залежностей та ресурсів, тобто будь-які зміни чи проблеми в одному контейнері не впливають на інші. Ця ізоляція забезпечує надійність системи та спрощує обслуговування. При розробці системи було застосовано мікросервісну архітектуру на базі Docker контейнерів.

Система складається з наступних незалежних компонентів:

- Контейнер Authelia — сервер автентифікації та авторизації з відкритим кодом, що забезпечує двофакторну автентифікацію та єдиний вхід (SSO) для програм через веб-портал. Він діє як доповнення до зворотних проксі-серверів, дозволяючи, відхиляючи або перенаправляючи запити. У цій системі Authelia забезпечує обов'язкову двофакторну автентифікацію, використовуючи одноразові паролі на основі часу, що генеруються кожні 30 секунд, які користувачі можуть перевірити як другий фактор за допомогою програм Google/Microsoft Authenticator або апаратних ключів WebAuthn. Контейнер інтегрується з серверною частиною LDAP для перевірки облікових даних користувачів та підтримує стан сеансу через сховище Redis.
- Контейнер LLDAP — легкий сервер LDAP, що надає централізовані служби каталогів користувачів. Він зберігає облікові записи користувачів, облікові дані, членство в групах та організаційну

інформацію у структурованій схемі LDAP. Контейнер надає доступ до протоколу LDAP на порту 3890 для інтеграції Authelia та надає веб-інтерфейс адміністрування на порту 17170 для операцій керування користувачами.

- Контейнер OpenVPN — реалізує безпечне тунелювання VPN за допомогою PKI (інфраструктури відкритих ключів), що складається з окремого сертифіката та закритого ключа для сервера та кожного клієнта, а також головного сертифіката та ключа центру сертифікації (CA), який використовується для підписання кожного з сертифікатів сервера та клієнта. OpenVPN підтримує двонаправлену автентифікацію на основі сертифікатів, що означає, що клієнт повинен автентифікувати сертифікат сервера, а сервер повинен автентифікувати сертифікат клієнта, перш ніж буде встановлено взаємну довіру. Контейнер керує генерацією сертифікатів за допомогою інструментарію EasyRSA та обробляє зашифрований UDP-трафік на порту 1194.
- Контейнер VPN-Admin — веб-додаток на основі Python, створений за допомогою framework Flask, який автоматизує надання сертифікатів VPN та розповсюдження файлів конфігурації. Додаток взаємодіє PKI OpenVPN для генерації сертифікатів клієнтів, збирає персоналізовані конфігураційні файли, що містять усі необхідні криптографічні матеріали, та надає кінцеві точки REST API для операцій панелі інструментів. Він підтримує підключення до бази даних PostgreSQL для ведення журналу аудиту та відстеження статистики користувачів.
- Контейнер Nginx — функціонує як зворотний проксі-сервер та точка завершення SSL для всієї інфраструктури. Контейнер спрямовує вхідні HTTPS-запити до відповідних серверних служб, орієнтуючись на URL-

шляхи. Він здійснює перевірку автентифікації шляхом звернення до кінцевої точки Authelia /api/verify перед наданням доступу до захищених ресурсів. Окрім цього, контейнер забезпечує управління SSL/TLS-шифруванням, використовуючи сертифікати Let's Encrypt із підтримкою автоматичного поновлення, що підвищує безпеку та спрощує обслуговування.

- Контейнер PostgreSQL — забезпечує постійне сховище реляційної бази даних для системи адміністрування VPN. Контейнер зберігає таблиці для записів користувачів, журнали підключень з часовими позначками та статистикою трафіку, а також журнали аудиту, що фіксують адміністративні дії. Операції з базою даних використовують стандартні SQL-транзакції, що забезпечують узгодженість даних та цілісність посилань.
- Контейнер Redis — реалізує високопродуктивне сховище ключ-значення в пам'яті для керування сеансами Authelia. Контейнер забезпечує збереження сеансу під час перезапусків контейнера Authelia, підтримує стан автентифікації користувачів за допомогою налаштовуваних політик закінчення терміну дії та забезпечує пошук сеансів з мілісекундною затримкою, підтримуючи тисячі одночасних автентифікованих користувачів.

## 2.2 Розробка схеми алгоритму програми

Блок-схеми — це прості візуальні інструменти, які допомагають нам дуже легко зрозуміти та представити процеси. Вони використовують такі форми, як стрілки, прямокутники та ромби, щоб чітко показати кроки та рішення. Якщо

хтось створює проект або пояснює складне завдання, блок-схеми можуть полегшити розуміння складних ідей.

Блок-схеми вибирають символи/форми, такі як стрілки, прямокутники та ромб, щоб правильно пояснити послідовність кроків, які беруть участь в алгоритмі або процесі. Блок-схеми мають свої варіанти використання в різних галузях, таких як розробка програмного забезпечення, моделювання бізнес-процесів та інженерія.

Блок-схеми працюють за допомогою численних переваг, які вони надають. Нижче наведено деякі важливі причини використання блок-схеми:

- Вони забезпечують ясність та спрощення складних процесів і алгоритмів, які, у свою чергу, разом з іншими людьми легко їх розуміють;
- Блок-схеми забезпечують універсальну візуальну мову, яку можна зрозуміти будь-хто в різних командах, і допомагають зменшити непорозуміння;
- Вони є оптимальним рішенням для документування стандартних операційних процедур, робочих процесів або бізнес-процесів. Це полегшує навчання нових співробітників;
- Блок-схеми допомагають покращити задачі візуалізації, що вирішуються, що дозволяє прийняти більш обґрунтовані рішення на основі даних.

Типи блок-схем

Існує багато типів блок-схем, кожен з яких призначений для представлення різних видів процесів та інформації. Деякі розширені типи блок-схем:

Блок-схема процесу: Вона представляє послідовність кроків у процесі. Вони часто використовуються в моделюванні бізнес-процесів, виробництва та управлінських проектів.

Блок-схема «плавальні смуги»: Вона організовує процес у різних смугах, кожен з яких представляє окрему особу або відділ, і використовується для програмування того, як різні команди або відділи співпрацюють у рамках процесу.

Діаграма робочого процесу: вона відображає систему, як завдання, документи або інформація рухаються через, і зазвичай використовуються в офісних процесах або розробці програмного забезпечення.

Діаграма потоку даних (DFD): Вона зосереджена на деталізації вхідних даних, процесів та виходів. Використовується в проектуванні та аналізі система для моделювання потоку даних у системі.

Блок-схема рішень: Вона зосереджена на відображених точках рішень у процесі та можливих результатів кожного рішення. Використовується в сценаріях прийняття рішень.

Рисунок 2.1 – Блок-схема створення нового користувача

Символи, що використовуються в дизайні блок-схеми

1. Овальний символ позначає Пуск, Зупинку та Зупинку в логічній потоці програми. Пауза/зупинка зазвичай використовується в логіці програми для певних умов помилки. Термінал - це перший та останній символи в блок-схемі.

2. Паралелограм позначає будь-яку функцію типу введення/виведення. Інструкції програми, які приймають вхідні дані від пристроїв введення та відображають вихідні дані на пристроях виведення, позначені паралелограмом у блок-схемі.

3. Прямокутник представляє арифметичні інструкції, певну дію або операцію, яка відбувається як частина процесу. Усі арифметичні процеси, такі як додавання, віднімання, множення та ділення, позначені символом дії/процесу.

4. Символ ромба представляє точку прийняття рішення. Операції, що базуються на рішеннях, таких як питання «так/ні» або «правда/неправда», позначені на блок-схемі ромбом.

5. Щоразу, коли блок-схема стає складною або розширюється на більш ніж одну сторінку, корисно використовувати з’єднувачі, щоб уникнути плутанини. З’єднувачі використовують для позначення переходу від однієї частини блок-схеми до іншої без малювання довгих або складних ліній. З’єднувач на сторінці представлений маленьким колом.

6. Щоразу, коли блок-схема стає складною або розширюється на більш ніж одну сторінку, корисно використовувати з’єднувачі, щоб уникнути плутанини. З’єднувачі використовують для позначення переходу від однієї частини блок-схеми до іншої без малювання довгих або складних ліній. Позасторінковий з’єднувач представлений п’ятикутником.

7. Лінії потоку вказують точну після виконання інструкцій. Стрілки позначають напрямок потоку керування та зв’язок між символами загальної блок-схеми.

### 2.3 Розробка програми

Docker Compose спрощує керування всім стеком застосунків, полегшуючи керування сервісами, в одному файлі конфігурації YAML.

Файл визначає сім взаємопов'язаних сервісів, які разом утворюють інфраструктуру VPN. Docker Compose керує життєвим циклом контейнерів, автоматично запускаючи, зупиняючи та перезапускаючи їх за потреби. Він гарантує, що контейнери створюються та підключаються у правильному порядку, на основі їхніх залежностей Docker.

```
authelia:
  image: authelia/authelia:latest
  container_name: authelia
  volumes:
    - ./authelia:/config
  environment:
    - TZ=UTC
  ports:
    - "9091:9091"
  restart: unless-stopped
  depends_on:
    - redis
    - lldap
  networks:
    - vpn_net
```

Лістинг 2.1 – Контейнер authelia

Використовує офіційний образ authelia/authelia:latest з Docker Hub, забезпечуючи доступ до найновіших оновлень безпеки та функцій.

Прив'язка ./authelia:/config забезпечує постійне сховище для файлів конфігурації (configuration.yml) та бази даних SQLite (db.sqlite3), що містять

секрети TOTP та облікові дані WebAuthn. Ключ шифрування шифрує конфіденційні дані бази даних, включаючи секрети TOTP та облікові дані WebAuthn. Навіть за наявності доступу до файлів бази даних дані залишаються захищеними без ключа шифрування. Локальне сховище використовує базу даних SQLite у файлі db.sqlite3, де зберігаються реєстрації користувачів TOTP, облікові дані WebAuthn, спроби автентифікації для регулювання та журнали аудиту.

Authelia прослуховує всі мережеві інтерфейси на порту 9091 через протокол TCP, що робить його доступним для інших контейнерів у мережі Docker.

Перевірки протоколу мережевого часу вимкнено під час запуску та у разі збоїв. За замовчуванням Authelia перевіряє синхронізацію часу, оскільки коди TOTP критично залежать від точного часу – навіть одна хвилина зсуву годинника призводить до збоїв автентифікації.

JWT secret криптографічно підписує токени для скидання пароля та перевірки електронної пошти. Коли користувачі запитують скидання пароля, Authelia генерує підписаний токен з обмеженим терміном дії та надсилає посилання електронною поштою.

Конфігурація одноразового пароля на основі часу встановлює ім'я видавця, яке відображається в програмах автентифікації під час сканування QR-коду. Період 30 секунд є стандартним інтервалом генерації коду.

Authelia підключається до сервера LLDAP через стандартний протокол LDAP на порту 3890. Параметр реалізації оптимізовано спеціально для схеми LLDAP. Базове DN визначає корінь дерева LDAP, де починається пошук користувачів і груп.

Секрет сеансу шифрує файли cookie сеансу, запобігаючи підробці – навіть якщо їх перехоплено, файли cookie не можна змінити або створити без секрету. Конфігурація файлів cookie визначає дійсність домену, authelia_url для перенаправлення після автентифікації та default_redirection_url для успішного входу в систему.

Інтеграція з Redis забезпечує постійне зберігання сеансів під час перезапусків контейнера Authelia – без Redis усі користувачі виходили б із системи під час кожного перезапуску.

Механізм регулювання захищає від атак методом грубої сили паролів. Максимум повторних спроб встановлює максимальну кількість невдалих спроб автентифікації перед блокуванням. Час пошуку визначає вікно підрахунку спроб – три невдалі спроби протягом двох хвилин. Час заборони визначає тривалість блокування – п'ять хвилин до дозволеної повторної спроби. Блокування відбувається окремо для кожного імені користувача та IP-адреси.

SMTP-сповіщення надсилає сповіщення електронною поштою для перевірки особи під час реєстрації TOTP або WebAuthn, посилання для скидання пароля та сповіщення про критичні події безпеки. Адреса вказує SMTP-сервер і порт - smtp.gmail.com:587 використовує шифрування STARTTLS на стандартному порту надсилання.

```
redis:
  image: redis:alpine
  container_name: redis
  volumes:
    - redis_data:/data
  restart: unless-stopped
  networks:
    - vpn_net
```

Лістинг 2.2 – Контейнер redis

Використовує полегшений варіант redis:alpine, мінімізуючи розмір контейнера, зберігаючи при цьому повну функціональність Redis.

Іменований том(volume) redis_data:/data забезпечує збереження сеансу між відтвореннями контейнерів. Томи представляють спільні каталоги між хостом та контейнерами. Вони дозволяють постійне зберігання та обмін даними. Docker автоматично керує іменованими томами.

Підключено до vpn_net, але не розкриває порти хосту, що гарантує доступ до Redis лише для інших контейнерів у мережі Docker.

```
lldap:
  image: lldap/lldap:stable
  container_name: lldap
  ports:
    - "17170:17170"
  volumes:
    - lldap_data:/data
  environment:
    - LLDAP_JWT_SECRET= YourSecret32
    - LLDAP_LDAP_USER_PASS= YourPassword
    - LLDAP_LDAP_BASE_DN=dc=vpn,dc=local
    - LLDAP_HTTP_URL=http://localhost:17170
  restart: unless-stopped
  networks:
    - vpn_net
```

Лістинг 2.3 – Контейнер lldap

Зіставлення портів:

17170:17170 зіставляє порт контейнера з хостом для локального доступу адміністратора через http://localhost:17170.

Доступ до веб-інтерфейсу:

Щоб увійти як адміністратор, потрібне ім'я користувача admin і пароль з LLDAP_LDAP_USER_PASS.

Іменований том lldap_data:/data persists:

База даних облікових записів користувачів (паролі зберігаються як хеші bcrypt), членство в групах, токени сеансу та ключі підпису JWT.

Authelia підключається до LLDAP за допомогою протоколу LDAP на порту 3890. Authelia надсилає запит на зв'язок LDAP з обліковими даними користувача. LLDAP перевіряє пароль на відповідність хешу bcrypt. У разі успіху Authelia запитує членство в групах. Доступ надається на основі правил access_control, що відповідають групам користувача.

LLDAP_HTTP_URL – канонічна URL-адреса для веб-інтерфейсу. Використовується для URI перенаправлення та створення посилань в електронних листах.

```
nginx:
  image: nginx:alpine
  container_name: nginx
  volumes:
    - ./nginx/nginx.conf:/etc/nginx/nginx.conf:ro
    - ./nginx/ssl:/etc/nginx/ssl:ro
    - ./certbot/conf:/etc/letsencrypt
    - ./certbot/www:/var/www/certbot
  command: "/bin/sh -c 'while :; do sleep 6h & wait $${!}; nginx -s
reload; done & nginx -g \"daemon off;\"'"
  ports:
    - "80:80"
    - "443:443"
  depends_on:
    - authelia
    - vpn-admin
  restart: unless-stopped
  networks:
    - vpn_net
```

Лістинг 2.4 – Контейнер nginx

Автентифікація користувачів у системі винесена за межі Python-додатку і реалізується через зв’язку Nginx та Authelia. Nginx виконує роль зворотного проксі та передає запити до Authelia для перевірки. У разі успішної автентифікації Nginx додає до HTTP-запиту службові заголовки, які передаються до Python-додатку. Таким чином, Flask-додаток не обробляє паролі та не реалізує власну систему автентифікації, а довіряє вже перевіреним даним, що підвищує безпеку та спрощує реалізацію.

./nginx/nginx.conf:/etc/nginx/nginx.conf:ro: Монтування лише для читання (суфікс:ro) запобігає випадковій модифікації всередині контейнера

./certbot/conf:/etc/letsencrypt: Сертифікати та конфігурація поновлення Let's Encrypt

./certbot/www:/var/www/certbot: Каталог відповідей на запит ACME для перевірки сертифіката

Основний процес nginx працює на передньому плані, тоді як фоновий цикл надсилає сигнал перезавантаження кожні 6 годин, щоб отримати оновлені сертифікати без простою.

Критично важливо для поновлення сертифіката - має бути доступним через HTTP на порту 80.

Періодичне перезавантаження отримує поновлені сертифікати без переривання з'єднання. Процес перезавантаження Nginx породжує нові робочі процеси з оновленою конфігурацією, водночас коректно завершуючи старі робочі процеси після виконання запитів у процесі, забезпечуючи нульовий час простою при ротації сертифікатів.

Зіставлення портів:

- 80:80 - HTTP для викликів ACME та перенаправлень HTTPS
- 443:443 - HTTPS для всього трафіку програм

ACME Challenge Location — критично важливо розмістити перед HTTPS redirect. Let's Encrypt робить HTTP запити до /.well-known/acme-challenge/ для валідації домену.

Використовує Let's Encrypt сертифікати (fullchain.pem + privkey.pem). Протоколи обмежені до TLSv1.2/1.3, вимкнено слабкі cipher suites. Це забезпечує сучасний безпечний TLS.

```
openvpn:
  image: kylemanna/openvpn:latest
  container_name: openvpn
  cap_add:
    - NET_ADMIN
  ports:
    - "1194:1194/udp"
  volumes:
    - openvpn_pki:/etc/openvpn
  environment:
    - OVPN_SERVER=10.8.0.0/24
    - OVPN_PROTO=udp
  restart: unless-stopped
  networks:
    - vpn_net
```

Лістинг 2.5 – Контейнер openvpn

Контейнер використовує образ kylemanna/openvpn, який упаковує серверне програмне забезпечення OpenVPN разом з інструментарієм EasyRSA для управління PKI та допоміжними скриптами для генерації конфігурації. Директива cap_add надає можливості NET_ADMIN, необхідні для операцій VPN - ця можливість Linux дозволяє контейнеру створювати та налаштовувати віртуальні мережеві інтерфейси, змінювати таблиці маршрутизації, встановлювати правила NAT iptables та керувати мережевими просторами імен.

Без цих привілеїв OpenVPN не може створити пристрій tun або налаштувати пересилання пакетів, необхідне для функціональності тунелювання.

Зіставлення портів явно визначає протокол UDP, використовуючи позначення косої риски 1194:1194/udp, оскільки Docker за замовчуванням використовує TCP без суфікса. OpenVPN підтримує транспортування як TCP, так і UDP, але UDP є переважною для VPN-трафіку через менші накладні витрати та кращі характеристики продуктивності. Тунелювання TCP-поверх TCP створює проблематичні взаємодії, коли втрата пакетів викликає повторні передачі як на тунельному, так і на прикладному рівнях, що призводить до експоненціального відкату та значного зниження пропускної здатності.

Іменований том openvpn_pki, змонтований до /etc/openvpn всередині контейнера, забезпечує постійне сховище для всієї структури каталогів PKI та файлів конфігурації сервера. Цей том містить кореневий сертифікат та закритий ключ центру сертифікації, статичний ключ TLS-auth для додаткового рівня автентифікації, список відкликаних сертифікатів та всі видані клієнтські сертифікати з відповідними закритими ключами. Перед початком роботи контейнер необхідно ініціалізувати за допомогою скрипта ovpn_genconfig. Цей скрипт створює базову конфігурацію сервера, визначаючи підмережу, налаштування портів та протоколів, алгоритм шифрування, директиви маршрутизації та вибір DNS-сервера.

Змінна OVPN_SERVER визначає підмережу VPN, виділену підключеним клієнтам, у нотації CIDR – підмережа 10.8.0.0/24 надає 254 корисні IP-адреси від 10.8.0.1 до 10.8.0.254, причому сервер зазвичай використовує 10.8.0.1 як шлюз.

Перед першим запуском контейнер потребує ініціалізації за допомогою допоміжного скрипта ovpn_genconfig, який генерує базову конфігурацію сервера,

включаючи визначення підмережі, специфікації портів та протоколів, вибір алгоритму шифрування, директиви маршрутизації та призначення DNS-сервера. Цей скрипт створює файл ovpn_env.sh, який зберігає змінні конфігурації як експортовані дані оболонки, та файл openvpn.conf, що містить директиви сервера OpenVPN.

Пакети даних від клієнтів надходять зашифрованими за допомогою узгодженого набору шифрів – зазвичай AES-256-GCM, що забезпечує автентифіковане шифрування з надійним захистом конфіденційності та цілісності. Сервер розшифровує пакети, перевіряє адреси призначення та пересилає їх відповідно до правил маршрутизації. Для трафіку, що спрямовується в Інтернет, сервер виконує NAT джерела, замінюючи VPN-адресу клієнта публічною IP-адресою сервера перед пересиланням до інтернет-шлюзу. Зворотний трафік йде зворотним шляхом – сервер отримує відповіді, шукає відповідний VPN-тунель на основі відстеженого стану з'єднання, шифрує пакети та передає їх назад клієнту.

Модель PKI забезпечує надійну автентифікацію без передачі пароля через мережу. Кожен клієнт має унікальну пару сертифіката та закритого ключа – закритий ключ ніколи не залишає клієнтський пристрій, а сертифікат підписується CA, що підтверджує його легітимність. Сервер перевіряє підписи сертифікатів за допомогою довіреного відкритого ключа CA, гарантуючи, що лише авторизовані клієнти з належним чином підписаними сертифікатами можуть пройти автентифікацію.

Конфігурація впроваджує низку провідних практик з кібербезпеки, які включають використання TLS-аутентифікації для посилення рівня автентифікації та забезпечення захисту від атак типу «відмова в обслуговуванні»

(DoS). У ній застосовуються надійні набори шифрів, що виключають використання застарілих алгоритмів, а також автентифікація, заснована на сертифікатах, яка ефективно запобігає можливості атак, пов'язаних із перехопленням або підбором паролів. Періодична ротація короткотривалих сеансових ключів сприяє більшому рівню безпеки, а сувора перевірка сертифікатів слугує надійним захистом від атак посередника. Комплексне поєднання взаємної автентифікації через TLS, шифрованих каналів передачі даних і автентифікації пакетів формує багаторівневий захист, здатний протистояти множинним загрозам, спрямованим на інфраструктуру VPN.

```
vpn-admin:
  build: ./vpn-admin
  container_name: vpn-admin
  depends_on:
    - openvpn
    - db
    - authelia
    - redis
  environment:
    -
DATABASE_URL=postgresql://vpnuser:strongpassword@db:5432/vpndb
    - OPENVPN_PKI_PATH=/etc/openvpn
    - OVPN_REMOTE=${OVPN_REMOTE}
    - AUTHELIA_URL=http://authelia:9091
    - JWT_SECRET=${JWT_SECRET}
    - SERVER_ADDRESS=${SERVER_ADDRESS}
  volumes:
    - openvpn_pki:/etc/openvpn
    - ./vpn-admin/data:/data
    - /var/run/docker.sock:/var/run/docker.sock
  ports:
    - "5000:5000"
  networks:
    - vpn_net
  restart: unless-stopped
```

Лістинг 2.6 – Контейнер vpn-admin

У представленій архітектурі контейнер vpn-admin, реалізований на Python із використанням Flask, виконує роль центрального керуючого компонента, який координує взаємодію між користувачами, системою автентифікації, VPN-сервером та інфраструктурою Docker. Його функція виходить за межі звичайного веб-додатка, оскільки він фактично виступає як рівень orchestration (керуюча площина), що забезпечує автоматизацію процесів створення, управління та видачі клієнтських конфігурацій VPN.

Ключовим архітектурним рішенням є використання Docker API через сокет /var/run/docker.sock, змонтований у контейнер. Це дозволяє Python-додатку безпосередньо взаємодіяти з Docker-демоном хоста, зокрема виконувати команди всередині інших контейнерів. На практиці це реалізується через виклик команди docker exec для контейнера OpenVPN, де запускається утиліта EasyRSA для генерації клієнтських сертифікатів. Таким чином, Python не виконує криптографічні операції самостійно, а делегує їх спеціалізованому контейнеру OpenVPN, який виступає як центр сертифікації (PKI). Це забезпечує розділення відповідальностей: OpenVPN відповідає за криптографію та безпеку, тоді як Python відповідає за логіку управління.

Другим важливим механізмом взаємодії є використання спільного іменованого тому openvpn_pki, який монтується одночасно в контейнери OpenVPN і vpn-admin. Завдяки цьому всі згенеровані сертифікати, ключі та інші файли PKI автоматично стають доступними Python-додатку без необхідності створення окремого API або мережевої взаємодії. Python читає ці файли безпосередньо з файлової системи, обробляє їх та використовує для формування готових клієнтських конфігурацій. Такий підхід реалізує патерн спільного стану

(shared state) через volume і значно спрощує архітектуру, хоча й вимагає довіри між контейнерами.

Контекст збірки: build: ./vpn-admin вказує Docker Compose збирати образ з Dockerfile в каталозі vpn-admin, а не витягувати попередньо зібраний образ

Монтування кількох томів:

- openvpn_pki:/etc/openvpn: Спільний том з контейнером OpenVPN, що забезпечує доступ для читання сертифікатів PKI
- ./vpn-admin/data:/data: Прив'язка монтування для згенерованих Файли конфігурації .ovpn
- /var/run/docker.sock:/var/run/docker.sock: Надає сокет демона Docker, що дозволяє контейнеру виконувати команди docker exec для генерації сертифіката всередині контейнера OpenVPN

Сервіс з'єднує автентифікацію Authelia, каталог користувачів LLDAP, інфраструктуру OpenVPN PKI та кінцевих користувачів, яким потрібен доступ до VPN. Застосунок автоматично генерує клієнтські сертифікати на вимогу та збирає персоналізовані файли конфігурації OpenVPN, вбудовуючи всі необхідні криптографічні матеріали в один портативний файл.

Сервіс створюється з користувацького Dockerfile в каталозі vpn-admin, а не з використанням попередньо створеного образу, що дозволяє налаштовувати залежності Python та код застосунку. Кілька монтувань томів забезпечують доступ до різних компонентів системи — том з назвою openvpn_pki спільно використовується лише для читання з контейнером OpenVPN, надаючи доступ до центру сертифікації, виданих сертифікатів та закритих ключів, що зберігаються в структурі каталогів PKI.

Сервіс підключається до мережі мосту vpn_net, що дозволяє здійснювати зв'язок з іншими контейнерами через виявлення служб на основі DNS. Ім'я хоста контейнера vpn-admin автоматично розпізнається в мережі, що дозволяє Nginx проксіювати запити, а Authelia пересилати автентифіковані запити.

Програма Flask складається з кількох модулів Python, що реалізують різні аспекти функціональності. Модуль config централізує зчитування змінних середовища та налаштування шляхів, забезпечуючи єдине джерело достовірної інформації про налаштування. Модуль бази даних визначає моделі SQLAlchemy, що представляють користувачів VPN, записи підключень та журнали аудиту – ці моделі автоматично генерують відповідні таблиці PostgreSQL під час першого запуску. Модуль auth реалізує декоратори для захисту маршрутів, перевіряючи заголовки Authelia та витягуючи контекст автентифікованого користувача.

Основний клас OpenVPNPManager інкапсулює всі операції PKI та логіку створення файлів конфігурації. Після ініціалізації він перевіряє наявність необхідних каталогів та зберігає посилання на шляхи PKI, розташування інструментарію EasyRSA та каталог виводу конфігурації клієнта. Метод generate_client_config збирає повні файли ovpn, зчитуючи публічний сертифікат центру сертифікації, сертифікат клієнта та закритий ключ з каталогу PKI, а також ключ TLS-auth для додаткового рівня безпеки. Він використовує регулярні вирази для вилучення сертифікатів у форматі PEM з файлів, що містять метадані та коментарі.

Метод створює файли конфігурації, вбудовуючи всі криптографічні матеріали в теги у стилі XML, що усуває необхідність в окремих файлах сертифікатів - користувачі отримують один автономний файл ovpn. Конфігурація включає адресу сервера, протокол, порт, параметри шифрування та директиви

маршрутизації, які вказують клієнту направляти весь трафік через тунель VPN. Метод create_client_cert автоматизує генерацію сертифікатів, виконуючи команди docker exec для запуску easyrsa build-client-full всередині контейнера OpenVPN. Цей підхід працює, оскільки контейнери використовують спільний том openvpn_pki - сертифікати, згенеровані в одному контейнері, одразу відображаються в іншому.

Сервіс надає доступ до кількох кінцевих точок HTTP для різних операцій. Коренева кінцева точка перенаправляє на панель інструментів, забезпечуючи логічну точку входу. Маршрут панелі інструментів обслуговує HTML-інтерфейс користувача, де автентифіковані користувачі бачать інформацію про свій профіль та кнопку завантаження. Кінцева точка перевірки справності повертає JSON, що вказує на стан служби та підключення до бази даних – корисно для систем моніторингу та балансувальників навантаження, яким потрібно перевірити працездатність служби.

Кінцева точка завантаження конфігурації реалізує основну функціональність – при виклику вона витягує ім'я користувача із заголовків, що пересилаються Nginx, запитує базу даних на наявність існуючих записів користувачів або створює нові записи, перевіряє активність облікового запису користувача, перевіряє наявність існуючих сертифікатів у каталозі PKI, автоматично генерує сертифікати, якщо вони відсутні, викликаючи create_client_cert, ненадовго чекає на синхронізацію файлової системи, генерує файл конфігурації ovpn, оновлює позначку часу останнього завантаження в базі даних, реєструє дію в журналі аудиту та повертає файл як вкладення для завантаження з відповідним типом MIME.

Кінцева точка API інформації про користувача витягує контекст автентифікації із заголовків HTTP, які Nginx пересилає після перевірки Authelia. Він зчитує заголовки Remote-User, Remote-Email, Remote-Name та Remote-Groups, повертаючи цю інформацію у форматі JSON для відображення JavaScript на панелі інструментів. Такий підхід усуває необхідність для програми Flask реалізовувати власну систему автентифікації – вона просто довіряє заголовкам від Nginx, які зовнішні клієнти не можуть підробити, оскільки вони встановлюються внутрішньо зворотним проксі-сервером після перевірки Authelia.

Кінцеві точки адміністратора, захищених як автентифікацією, так і членством у групі адміністратора, надають адміністративні можливості. Кінцева точка списку користувачів запитує всі записи користувачів VPN з PostgreSQL, повертаючи вичерпну інформацію, включаючи позначки часу створення, час останнього завантаження та активний стан.

Додаток довіряє заголовкам автентифікації від Nginx, які є безпечними, оскільки зовнішні клієнти не можуть їх підробити - Nginx встановлює ці заголовки внутрішньо після перевірки Authelia та видаляє будь-які заголовки, які клієнти намагаються вставити.

```
db:
  image: postgres:15
  container_name: vpn-db
  environment:
    - POSTGRES_USER=vpnuser
    - POSTGRES_PASSWORD=Yourpassword
    - POSTGRES_DB=vpndb
  volumes:
    - postgres_data:/var/lib/postgresql/data
  networks:
    - vpn_net
  restart: unless-stopped
```

Лістинг 2.7 – Контейнер db

PostgreSQL служить постійною реляційною базою даних для програми адміністрування VPN, зберігаючи записи користувачів, журнали підключень та журнали аудиту. На відміну від SQLite, який Authelia використовує для свого внутрішнього стану, PostgreSQL забезпечує відповідність ACID, підтримку одночасного доступу та кращу продуктивність для вимог програми Flask.

Вибір версії: postgres:15 вказує на PostgreSQL 15.x – останню стабільну основну версію на момент розгортання. Закріплення основної версії запобігає неочікуваним критичним змінам від автоматичних оновлень, водночас дозволяючи встановлювати незначні виправлення безпеки через оновлення образів Docker.

Змінні середовища – Початкове налаштування бази даних:

- POSTGRES_USER=vpnuser – Створює обліковий запис суперкористувача для програми. Програма Flask підключається, використовуючи це ім'я користувача в рядку підключення DATABASE_URL.
- POSTGRES_DB=vpndb – Створює початкову базу даних з цим ім’ям.
- POSTGRES_PASSWORD=yourpassword – Встановлює пароль для облікового запису користувача.

```
certbot:
  image: certbot/certbot
  volumes:
    - ./certbot/conf:/etc/letsencrypt
    - ./certbot/www:/var/www/certbot
  entrypoint: "/bin/sh -c 'trap exit TERM; while :; do certbot
renew; sleep 12h & wait $${!}; done;'"
```

Лістинг 2.8 – Контейнер certbot

- `./certbot/conf:/etc/letsencrypt` - Постійне сховище для сертифікатів, облікових даних облікового запису та конфігурації поновлення.

./certbot/www:/var/www/certbot - Спільний каталог з Nginx для перевірки ACME HTTP-01. Коли Let's Encrypt перевіряє право власності на домен, він запитує http://domain/.well-known/acme-challenge/token - Certbot записує відповідь на запит до цього каталогу, а Nginx його обслуговує.

Сертифікати Let's Encrypt дійсні протягом 90 днів, що вимагає частого поновлення. Сервіс працює як демон, який перевіряє термін дії сертифіката кожні 12 годин і автоматично поновлює сертифікати протягом 30 днів після закінчення терміну дії. Сервер перевірки Let's Encrypt повинен звертатися до порту 80 на вашому сервері, щоб перевірити право власності на домен через HTTP-01

Перед розгортанням Certbot необхідно налаштувати дійсне доменне ім'я, яке вказуватиме на публічну IP-адресу сервера. Let's Encrypt робить перевірку права власності на домен через спеціальний протокол ACME (Automated Certificate Management Environment). В даному випадку зареєстрований домен був на DuckDNS, тепер домен вказує на публічну IP-адресу сервера:

corporate-vpn.duckdns.org → 109.197.217.223.

Загальний сценарій роботи системи виглядає наступним чином: користувач проходить автентифікацію через Authelia, після чого звертається до веб-інтерфейсу vpn-admin. Python-додаток отримує ідентифікаційні дані з HTTP-заголовків, перевіряє наявність відповідного користувача в базі даних і сертифіката у PKI. Якщо сертифікат відсутній, додаток через Docker API викликає генерацію сертифіката в контейнері OpenVPN, після чого зчитує

створені файли зі спільного тому, формує конфігураційний файл .ovpn та надає його користувачу для завантаження.

### 2.4 Мережева архітектура

Мережна архітектура системи побудована за принципом багаторівневого доступу з використанням контейнеризації Docker та ізольованої мережі типу bridge (vpn_net). Зовнішній користувач взаємодіє із системою через Інтернет, звертаючись до публічної IP-адреси сервера, після чого трафік надходить до зворотного проксі-сервера Nginx, який обробляє HTTP та HTTPS-запити та виконує TLS-термінацію. Перед передачею запиту до внутрішніх сервісів Nginx інтегрується з системою автентифікації Authelia, яка забезпечує перевірку користувачів із застосуванням багатофакторної автентифікації.

Authelia взаємодіє з LDAP-каталогом LLDAP, що виступає централізованим сховищем облікових записів користувачів і груп, а також використовує Redis для зберігання сесій, що дозволяє зберігати стан автентифікації між запитами.

Після успішної автентифікації запити передаються до сервісу vpn-admin, реалізованого на Python (Flask), який виконує функції управління VPN-користувачами, генерації сертифікатів та конфігураційних файлів. Даний сервіс взаємодіє з контейнером OpenVPN через Docker API, виконуючи команди всередині контейнера та використовуючи спільний том для доступу до інфраструктури відкритих ключів (PKI). Контейнер OpenVPN виступає як VPN-шлюз, приймає підключення клієнтів через UDP-порт 1194, створює захищений тунель і призначає клієнтам IP-адреси з приватної підмережі 10.8.0.0/24.

Після встановлення з’єднання трафік маршрутизується через VPN-шлюз до внутрішньої корпоративної мережі, де розташовані захищені ресурси, доступ до яких можливий виключно через VPN, що забезпечує ізоляцію та підвищений рівень безпеки всієї системи.

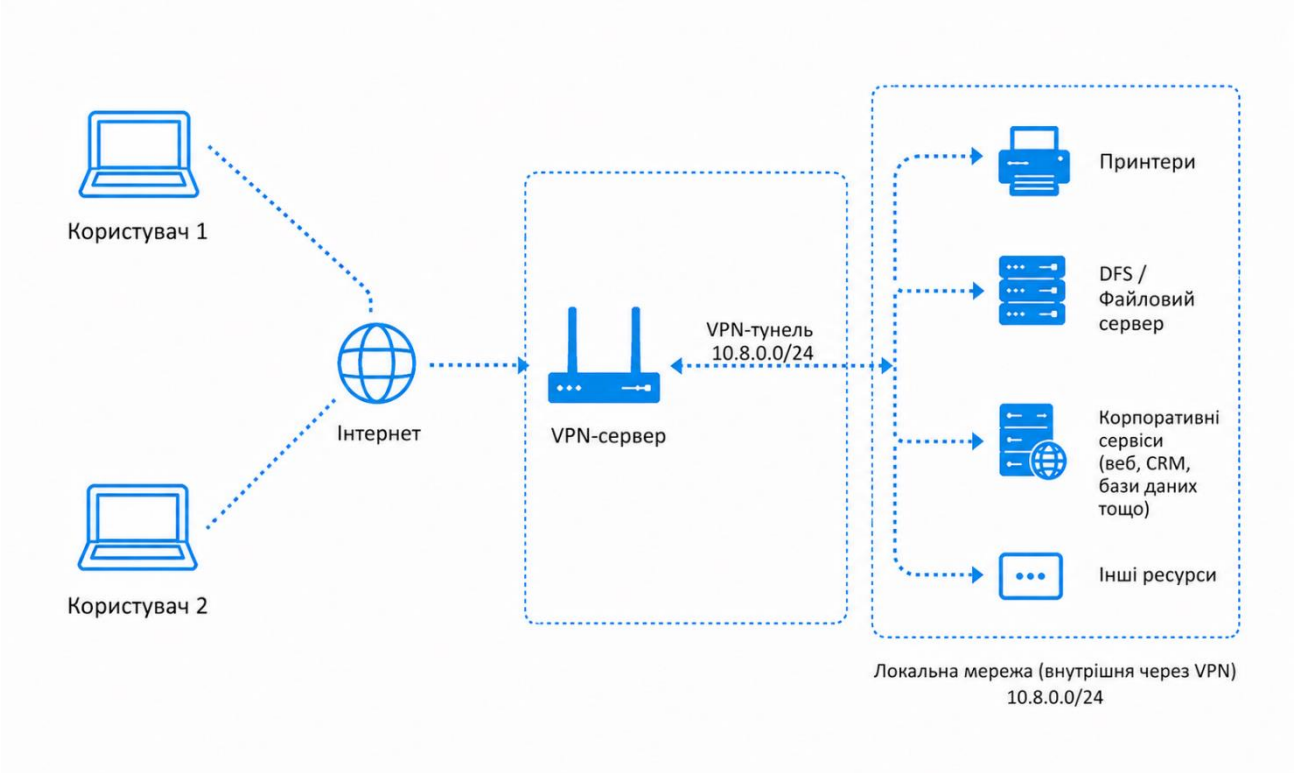


Рисунок 2.2 – Схема мережевої топології

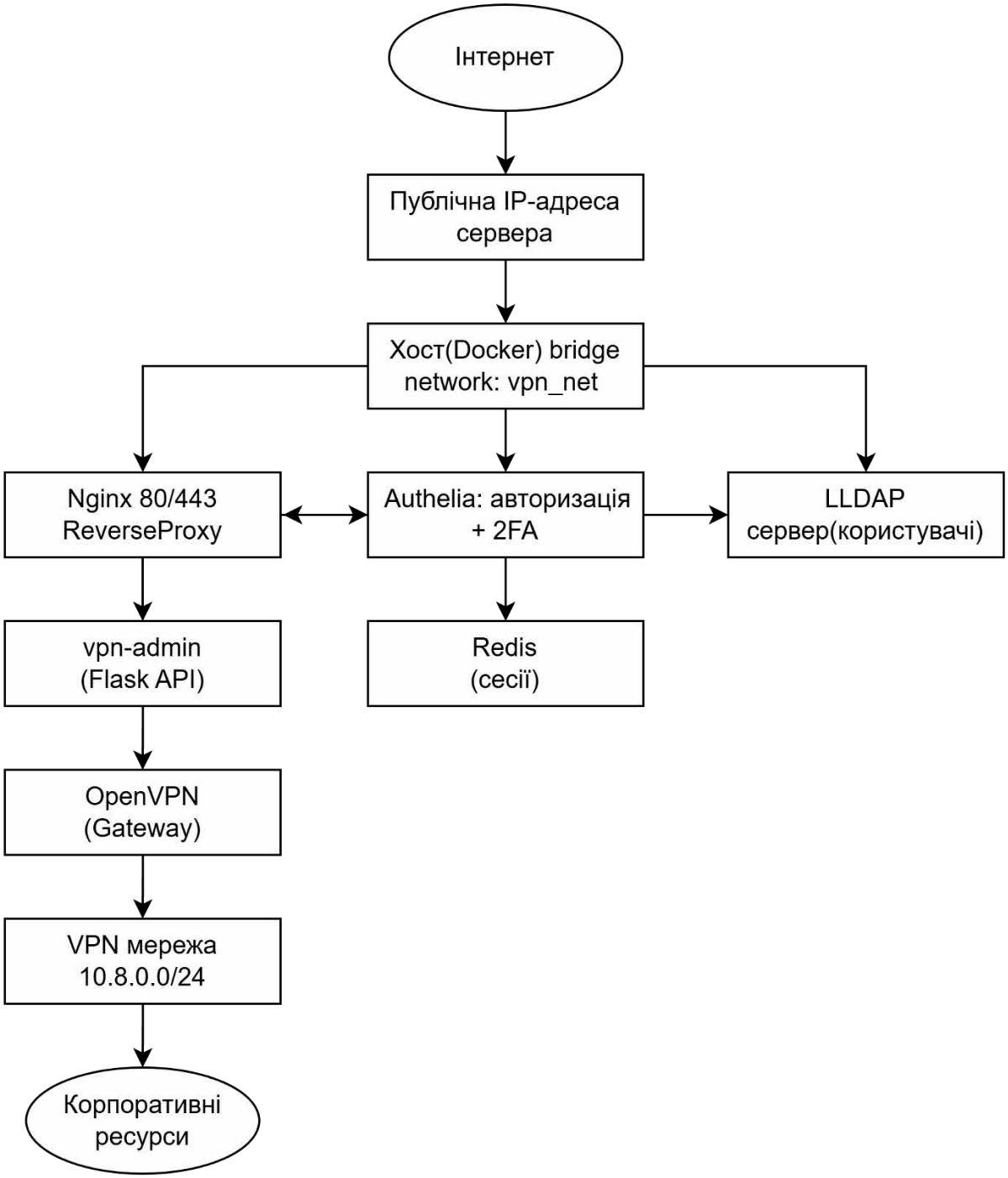


Рисунок 2.3 – Блок-схема мережевої архітектури

# 3 ТЕСТУВАННЯ ПРАЦЕСДАТНОСТІ ПРОГРАМИ

## 3.1 Запуск та тестування функцій апаратно-програмного комплекса

Запуск всіх сервісів складається з двох основних кроків.

Перший крок - це отримати SSL сертифікат. Спочатку потрібно розгорнути контейнер з nginx, дальше одноразово запускаємо контейнер з certbot, щоб отримати сертифікат для нашого домена.

```
docker compose up -d nginx

docker compose run --rm certbot certonly --webroot --webroot-
path=/var/www/certbot --email ki22-a.mykhailovskyi@nubip.edu.ua --
agree-tos --no-eff-email -d corporate-vpn.duckdns.org
```

Лістинг 3.1 - Отримання SSL сертифіката

Другий крок - це ініціалізація OpenVpn PKI. Щоб це виконати розгортаємо контейнер в якому генеруємо базову конфігурацію OpenVpn. Створюємо структуру папок для PKI, тоді генеруємо кореневий сертифікат, який буде підписувати всі інші сертифікати. Виконуємо генерацію серверного сертифікату і приватного ключа сервера. Генеруємо Diffle-Hell параметри і TLS-auth ключ для додаткового захисту. Обов`язково створюємо список відкликаних сертифікатів, після цих дій запускаємо всі наші сервіси.

```
docker run --rm -it -v openvpn_authelia_openvpn_pki:/etc/openvpn
kylemanna/openvpn bash
ovpn_genconfig -u udp://corporate-vpn.duckdns.org
easyrsa init-pki
easyrsa build-ca nopass
easyrsa build-server-full corporate-vpn.duckdns.org nopass
easyrsa gen-dh
```

```
openvpn --genkey --secret /etc/openvpn/pki/ta.key
easyrsa gen-crl
exit
docker compose up -d
```

Лістинг 3.2 – Ініціалізація OpenVpn PKI

Тест №1 Додати нового користувача в LLDAP

Заходимо на http://localhost:17170/ проходимо авторизацію для адміністратора. Вибираємо пунк Users, нажимаємо створити користувача і вказуємо його дані.

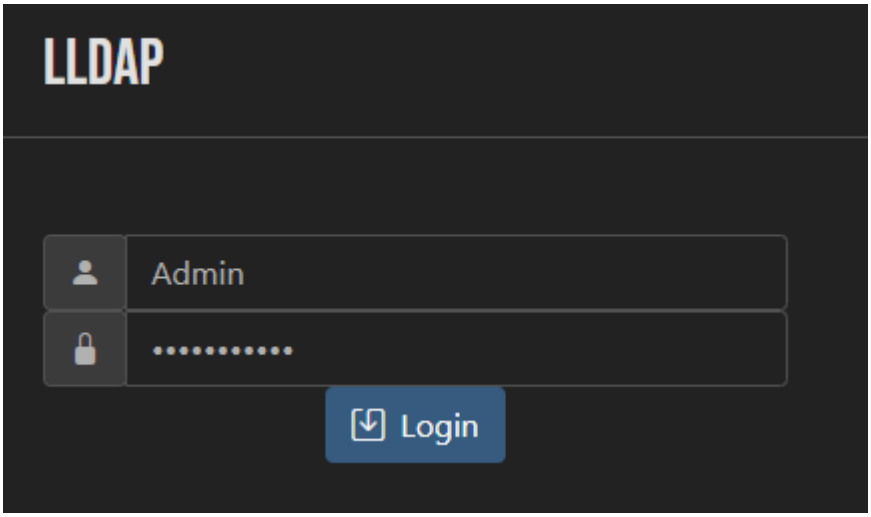


Рисунок 3.1 – Авторизація в LLDAP

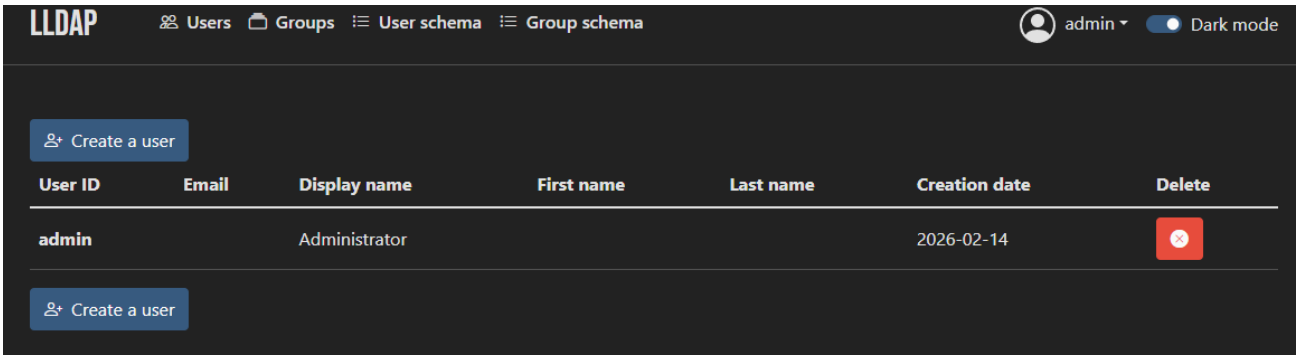


Рисунок 3.2 – Список користувачів у LLDAP

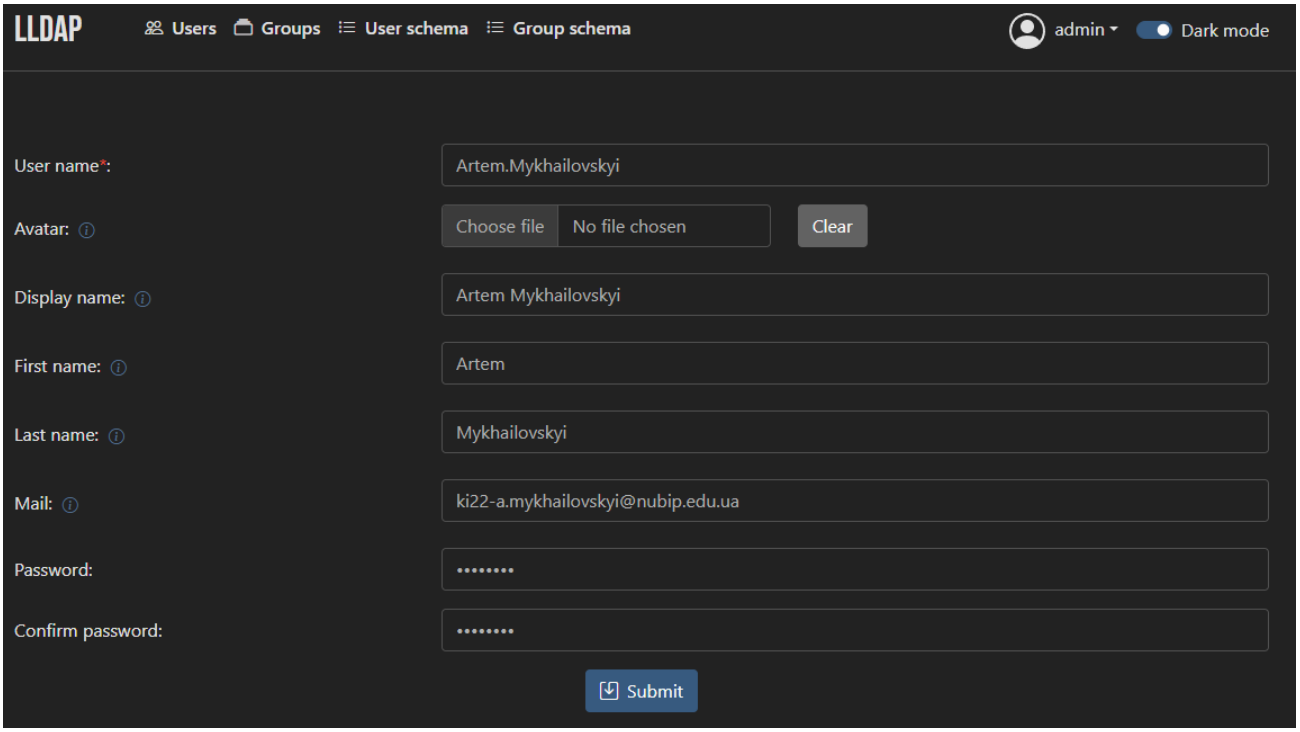


Рисунок 3.3 – Заповнення даних нового користувача

Тепер появився новий користувач, вибираємо його і призначаємо групу vpn-users, щоб він міг проходити авторизацію для отримання конфігураційного файла.

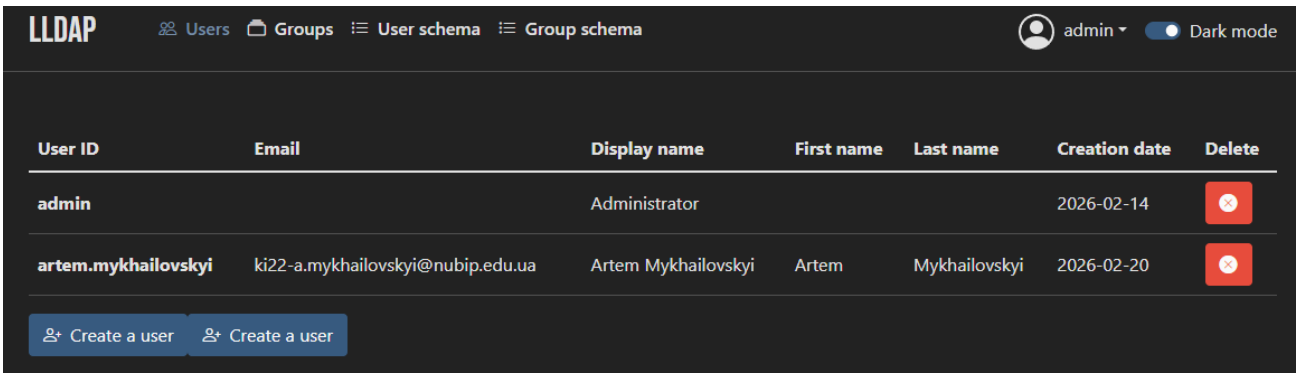


Рисунок 3.4 – Створений новий користувач

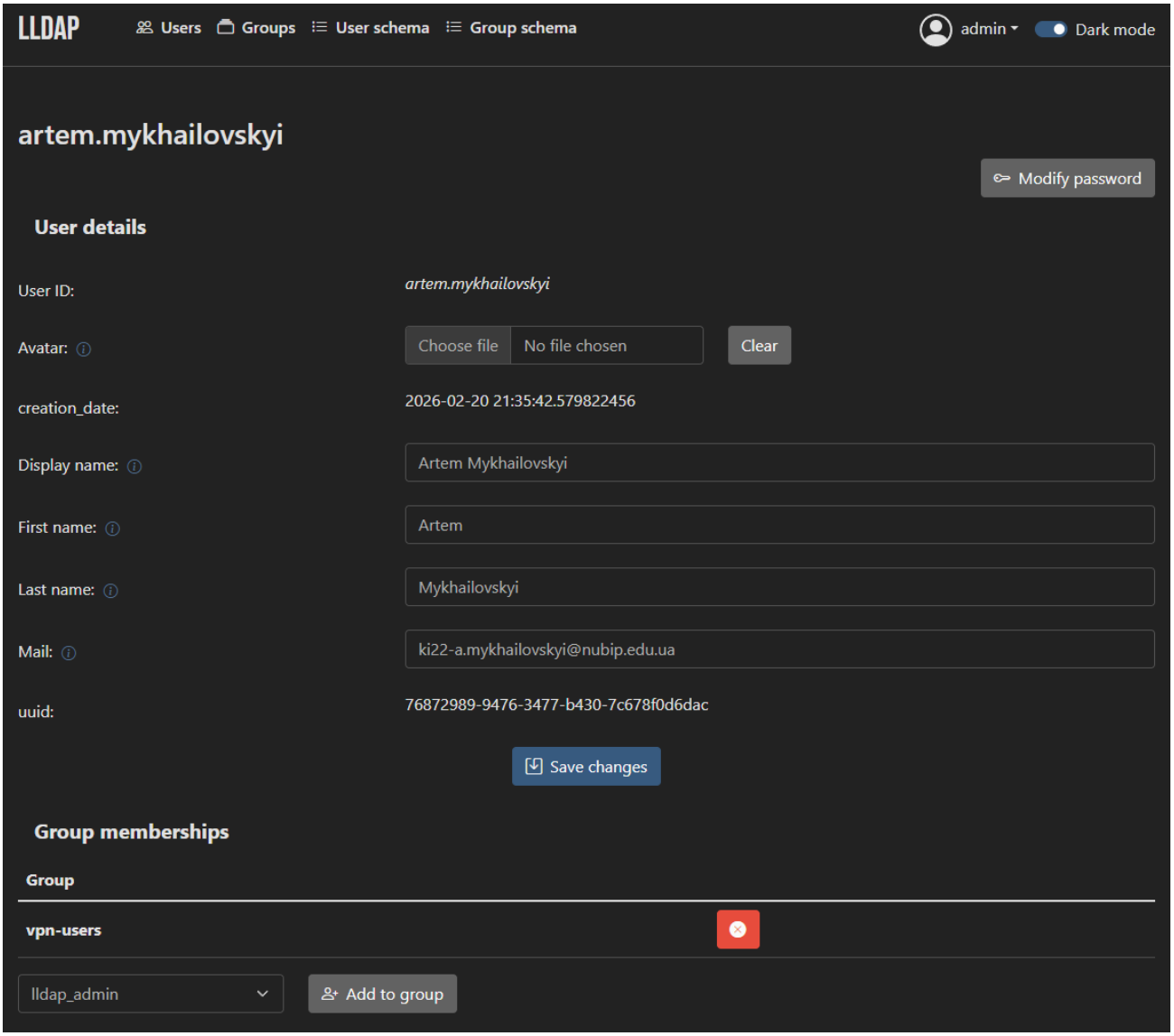


Рисунок 3.5 – Призначення групи vpn-users

Для перевірки правильності створення користувача була зроблена спроба зайти на сайт, пройти авторизацію і 2 MFA, як результат зміг скачати конфігураційний файл.

Тест №2 Користувач має скачати конфігураційний файл, щоб підключитись до VPN.

Потрібно зайти на сайт https://corporate-vpn.duckdns.org для проходження авторизації і 2 MFA. Щоб пройти перший раз 2MFA потрібно його зареєструвати, для цього ми використаємо одноразовий пароль в Google/Microsoft authenticator.

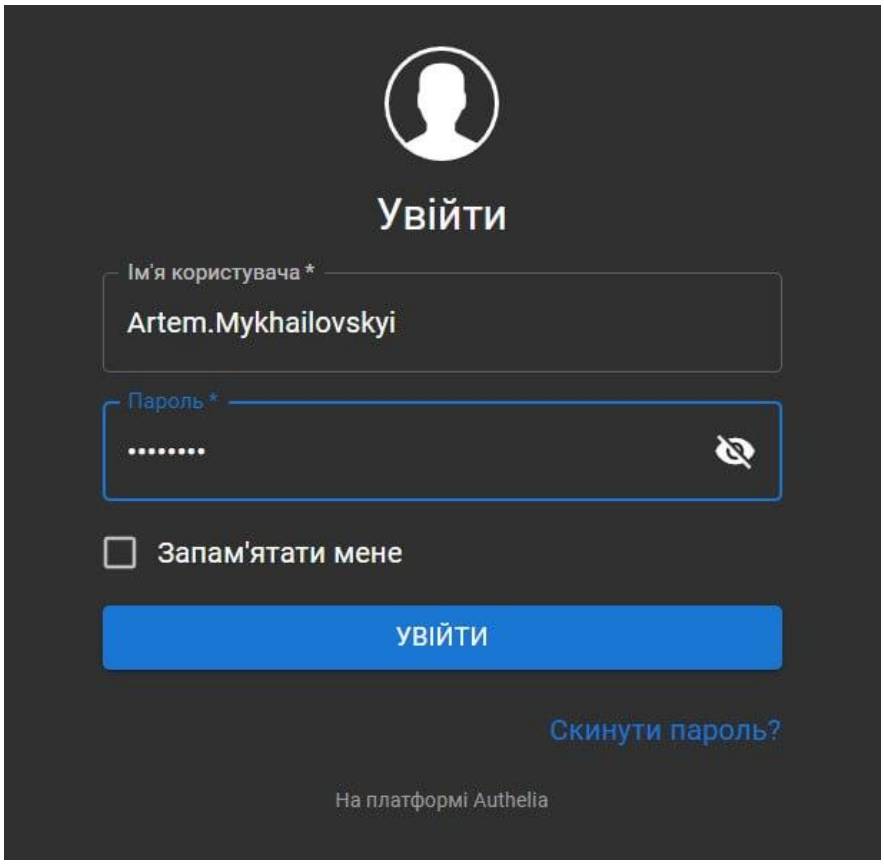


Рисунок 3.6 – Проходження авторизації на сайті

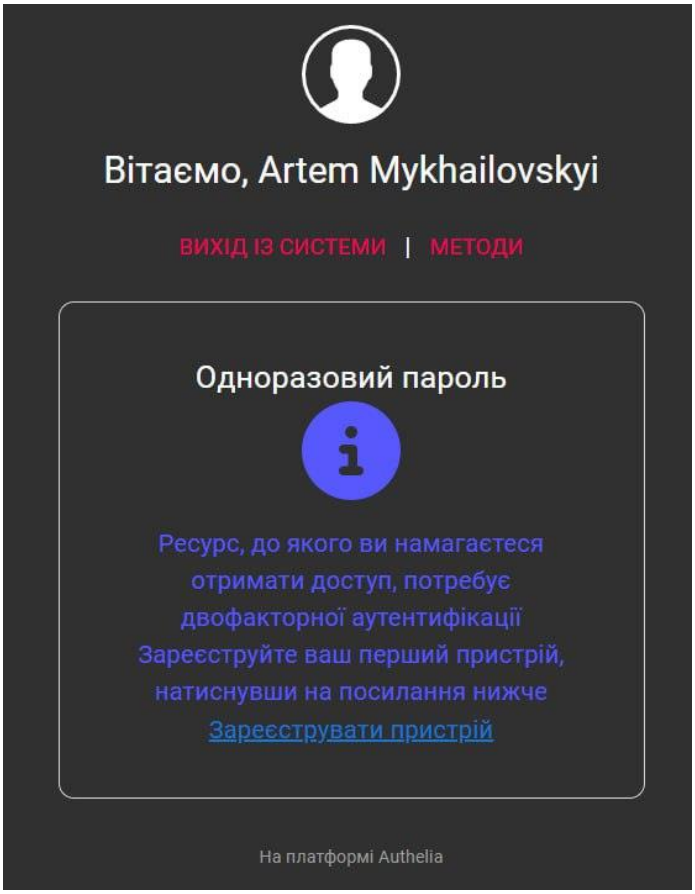


Рисунок 3.7 – Пропонує зареєструвати двофакторну аутентифікацію

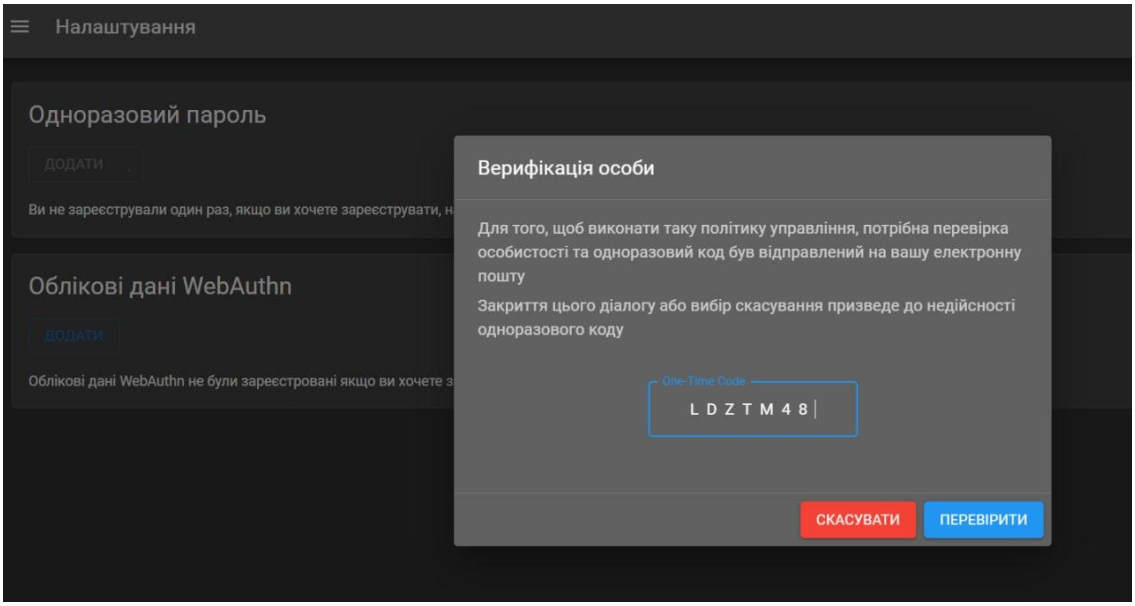


Рисунок 3.8 – Підтвердження особи через електрону пошту

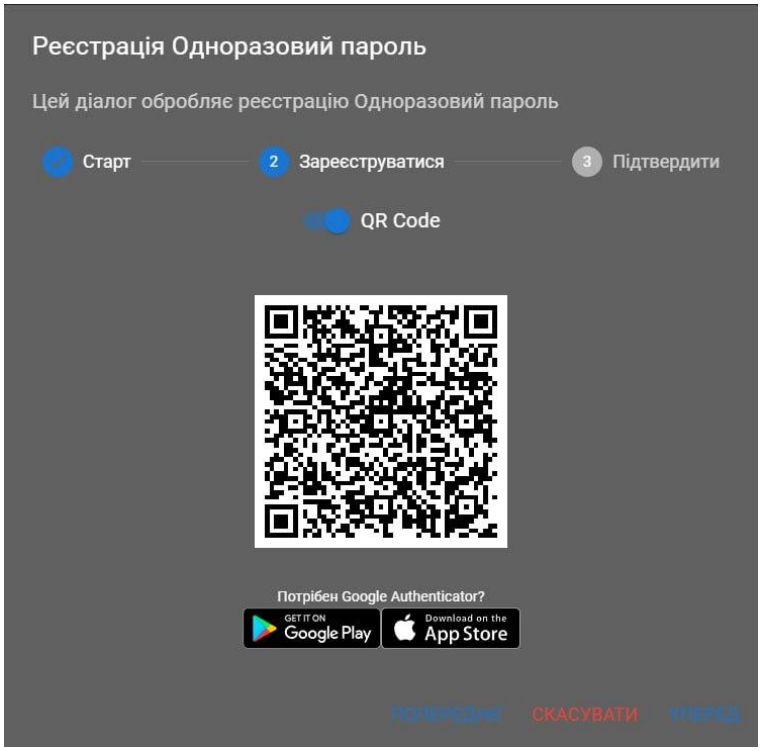


Рисунок 3.9 – QR код для прив`язки Google/Microsoft authenticator

Тепер можна побачити що в нас є прив`язана двофакторна аутентифікація і ми можемо повноцінно зайти на сайт. На dashboard є інструкція в якій детально розписано як підключитись до OpenVPN connect.

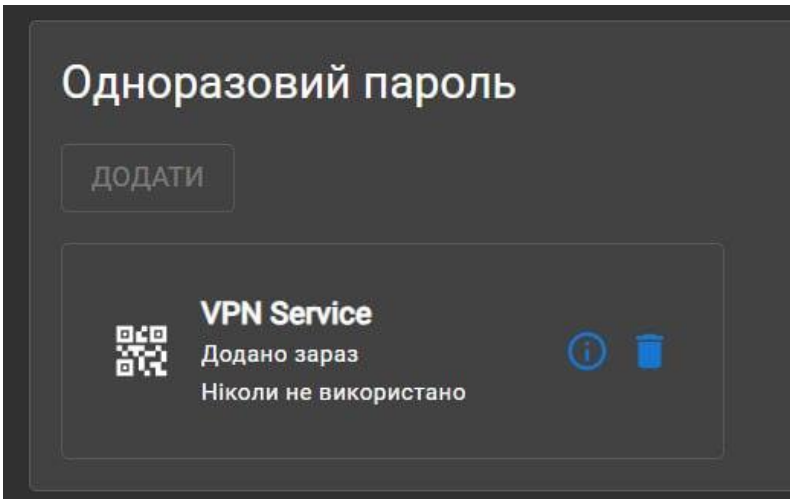


Рисунок 3.10 – Прив`язаний Microsoft authenticator

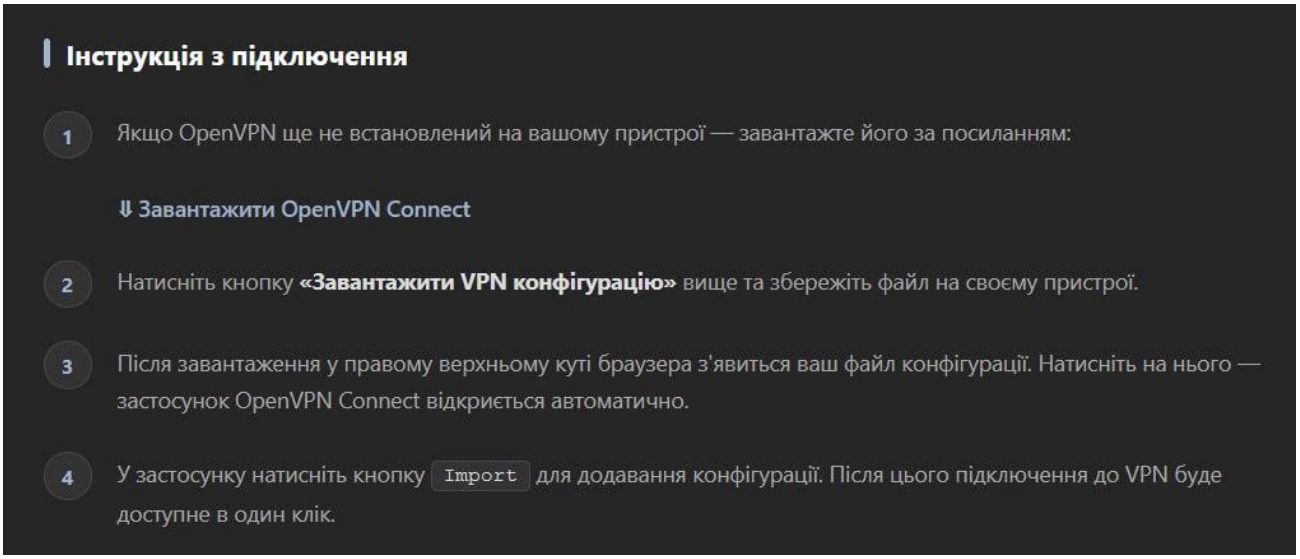


Рисунок 3.11 – Інструкція для користувача

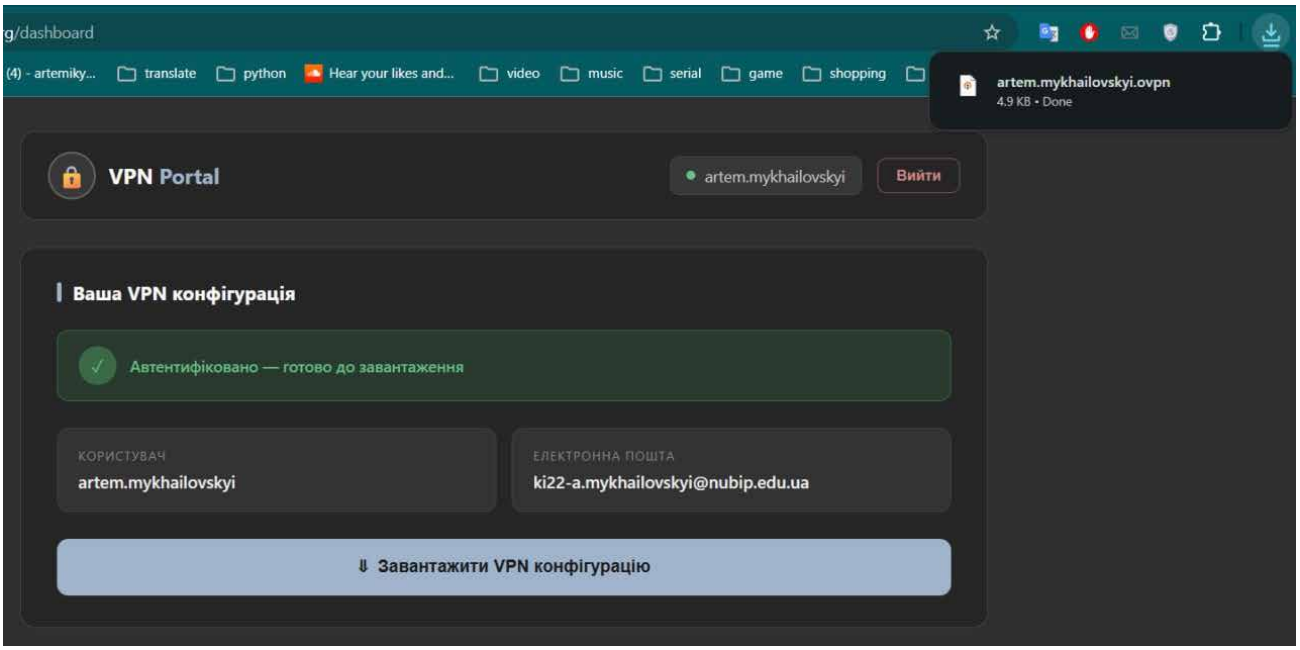


Рисунок 3.12 – Завантаження конфігураційного файла

Після того як ми завантажили конфігураційний файл собі на пк і встановили OpenVPN Connect, можна імпортувати його в додаток, щоб підключитись.

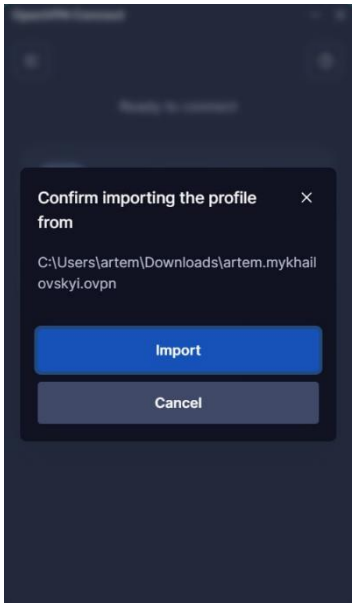


Рисунок 3.13 – Імпортування файла в OpenVPN Connect

Рисунок 3.14 – Успішне з’єднання з VPN сервером

Для перевірки правильності під’єднання до VPN сервера потрібно перевірити чи співпадає глобальний IP-адрес клієнта і сервера, як результат вони одинакові.

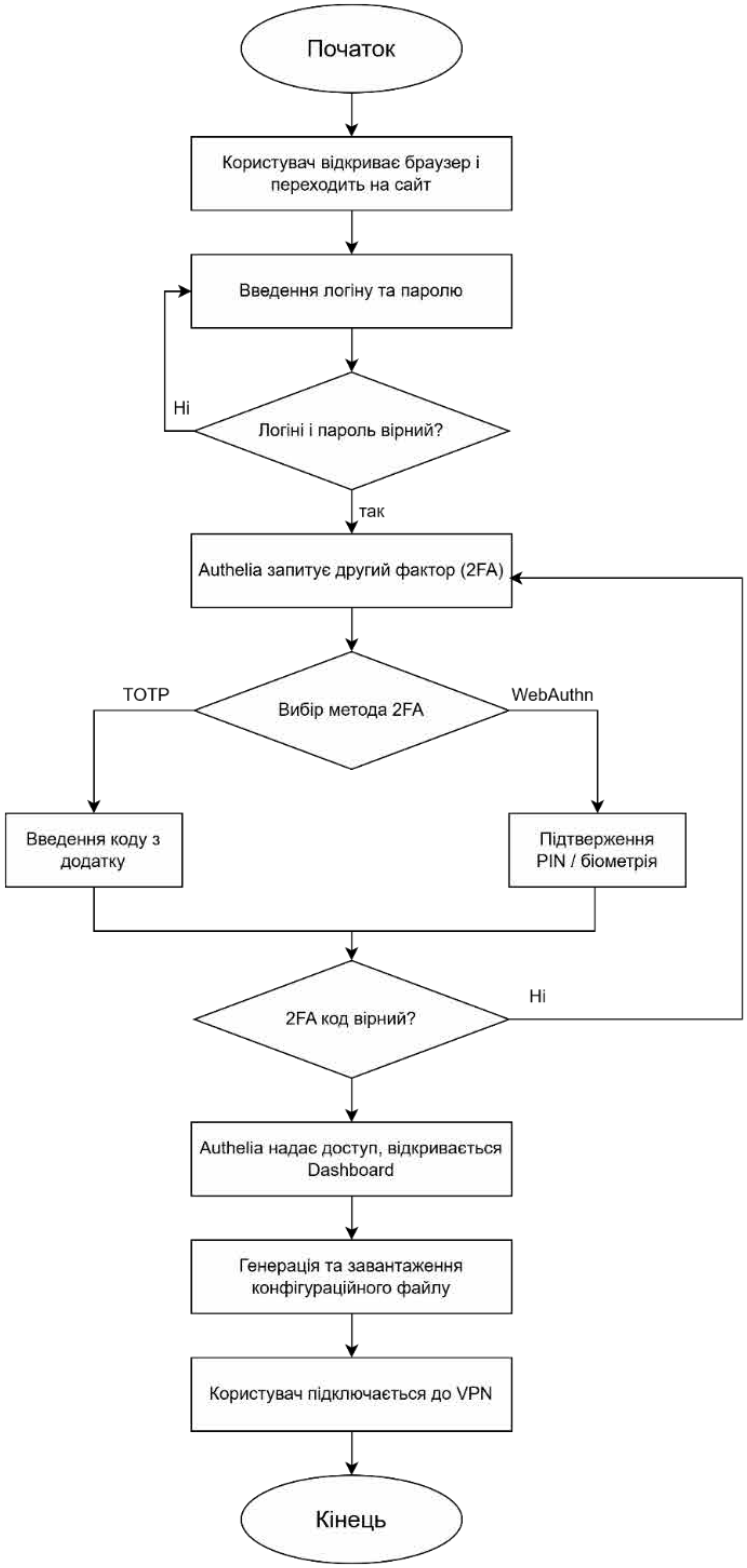


Рисунок 2.15 – Блок-схема підключення користувача до VPN

Тест №3 Потрібно перевірити можливість зареєструвати інший метод двофакторної аутентифікації.

Проходимо авторизацію користувача і нажимаємо кнопку <МЕТОДИ>. Вибираємо додати дані WebAuthn і підтверджуємо особу через електронну пошту.

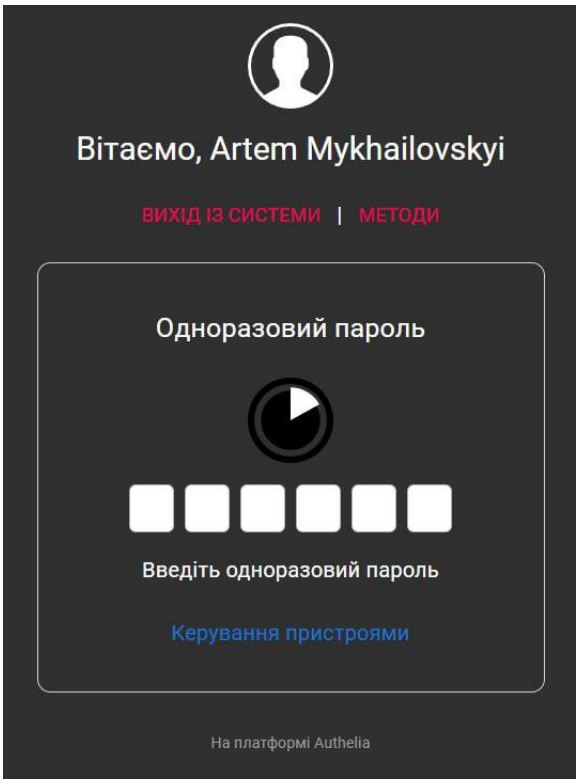


Рисунок 3.16 – Підтвердження двофакторної аутентифікації

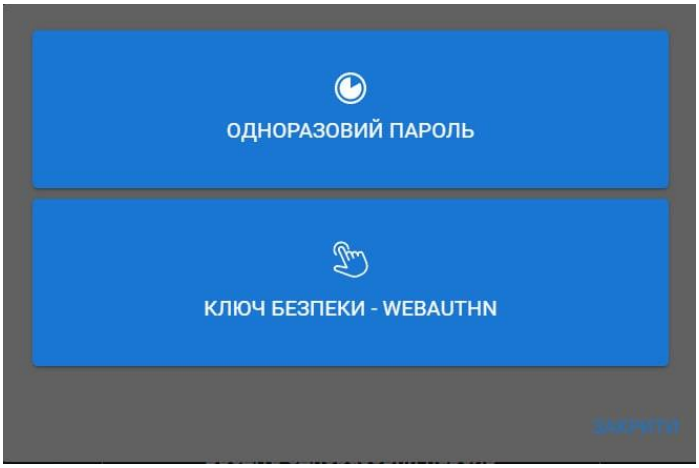


Рисунок 3.17 – Вибір двофакторної аутентифікації

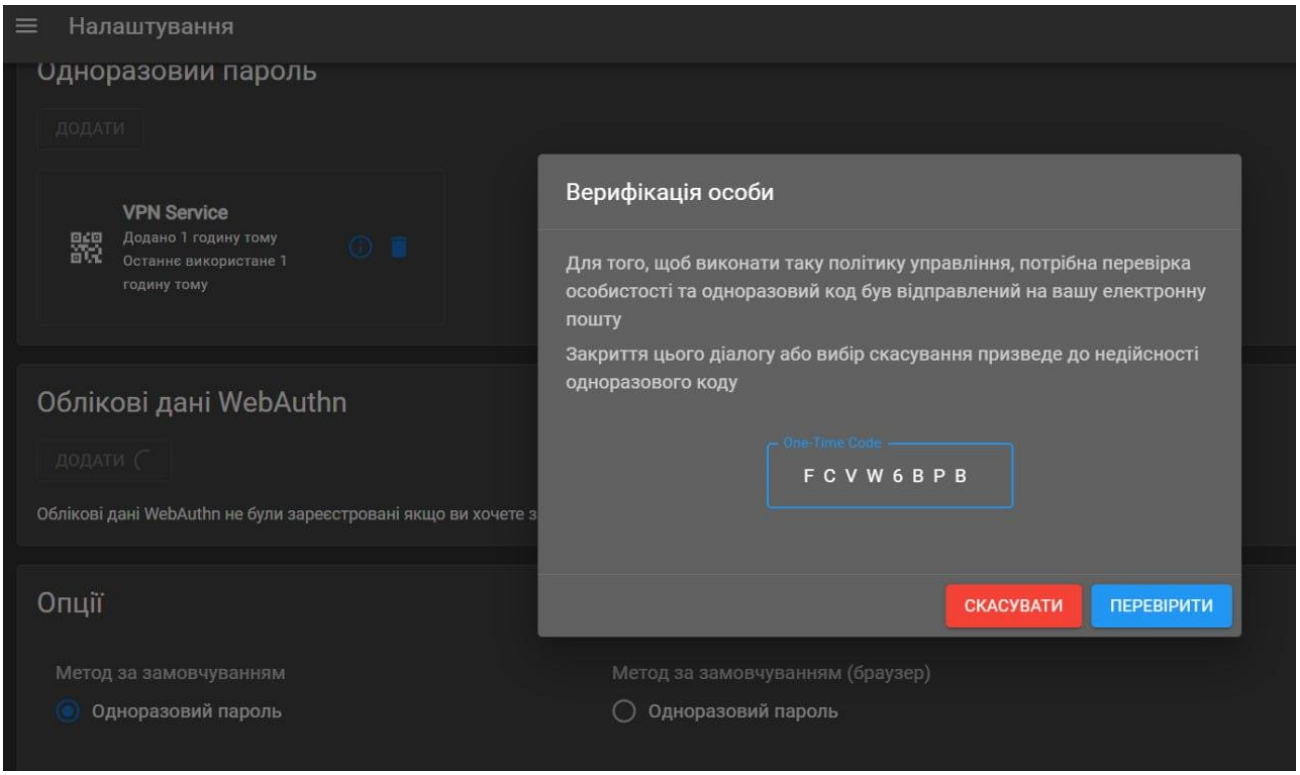


Рисунок 3.18 – Додавання нової аутентифікації і підтвердження через пошту особи

Тепер ми можемо додати апаратні ключі або біометрію.

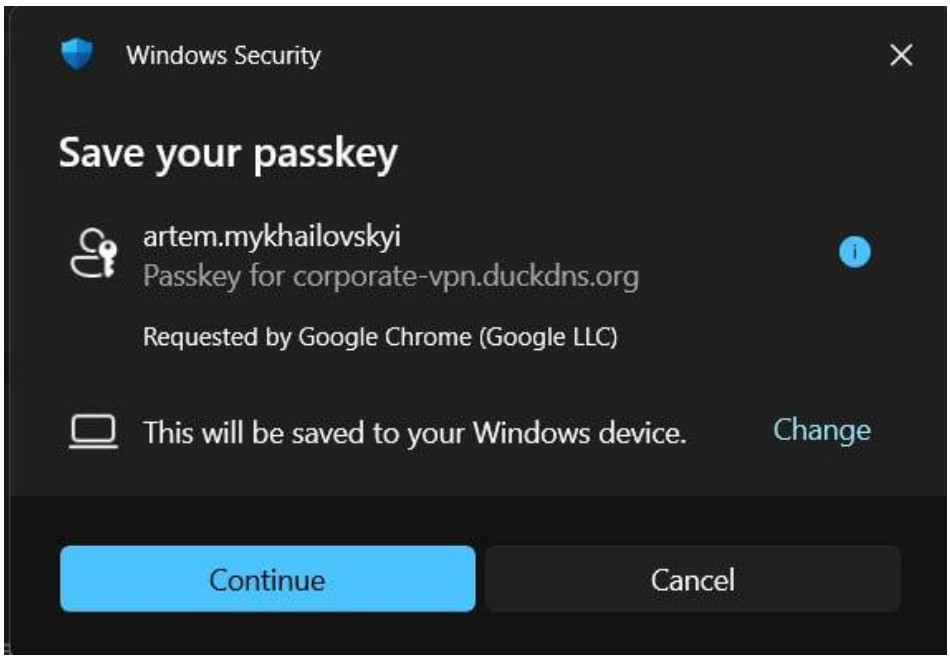


Рисунок 3.19 – Збереження ключів

Для перевірки правильності реєстрування двофакторної аутентифікації Webauthn була зроблена спроба зайти на dashboard, як результат користувач зміг завантажити конфігураційний файл.

Тест №4 Чи має можливість відновити свій пароль користувач, якщо не пам’ятає його?

Вибираємо скинути пароль і вказуємо ім’я користувача. На пошту користувач отримує повідомлення в якому, він підтвердить те що хоче змінити пароль.

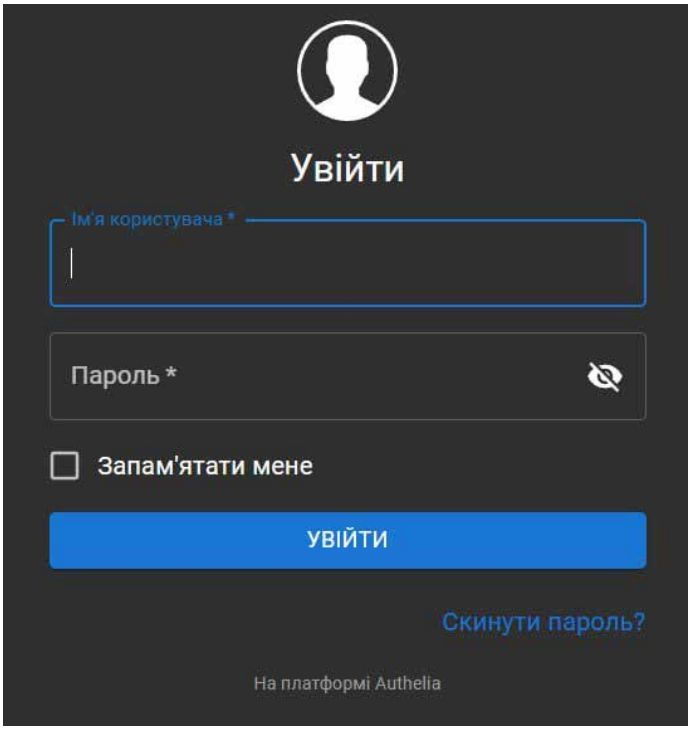


Рисунок 3.20 – Користувач забув свій пароль

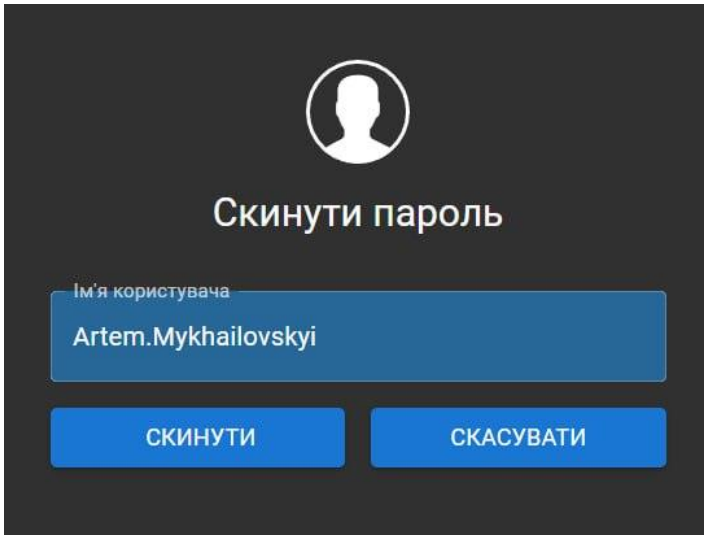


Рисунок 3.21 – Вказують ім’я користувача для зміни пароля

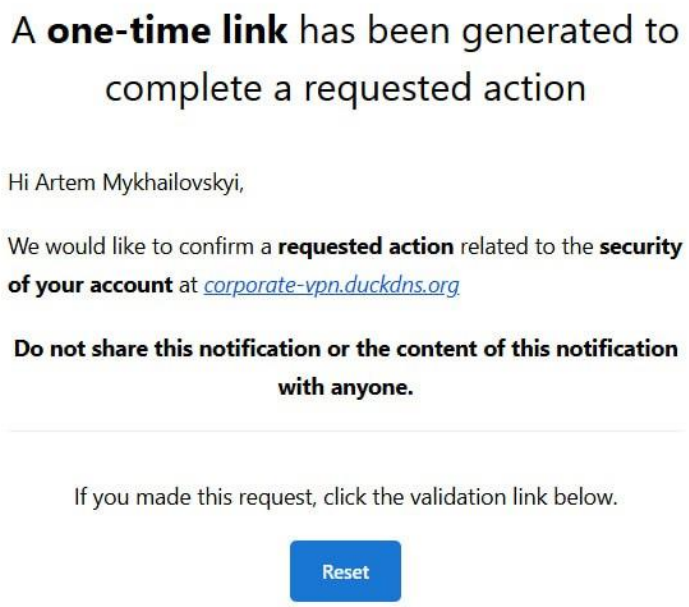


Рисунок 3.22 – Підтвердження на електронній пошті

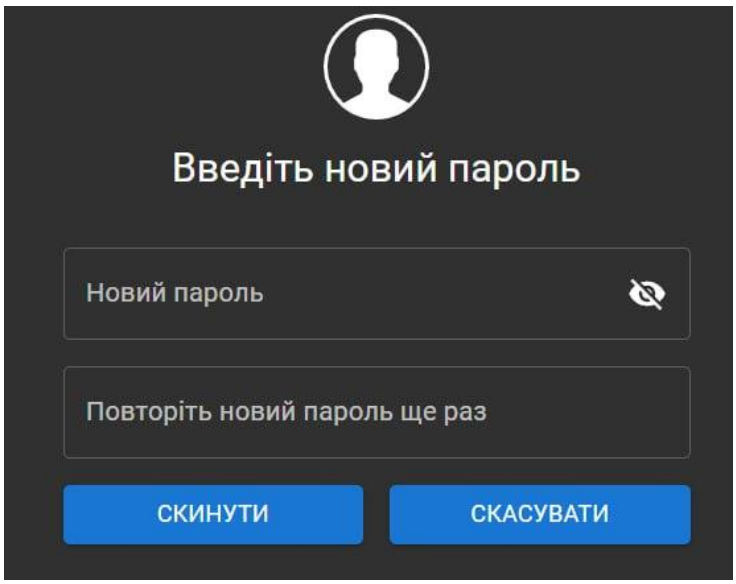


Рисунок 3.23 – Вводим новий пароль

Для перевірки правильності нового пароля була зроблена спроба авторизуватись, як результат користувач зміг її пройти.

## 3.2 Апаратні вимоги до серверного обладнання та тестування продуктивності

Правильний вибір апаратного забезпечення для VPN-сервера корпоративного рівня є критично важливим етапом проєктування системи, оскільки недостатня продуктивність обладнання безпосередньо впливає на якість з'єднання, затримки та стабільність роботи для всіх підключених користувачів. На відміну від звичайних веб-серверів, VPN-сервер виконує криптографічні операції шифрування та дешифрування для кожного пакета даних кожного підключеного користувача в режимі реального часу, що висуває специфічні вимоги до апаратної складової.

Центральний процесор є найважливішим компонентом VPN-сервера, оскільки саме на нього припадає основне навантаження з шифрування трафіку. При використанні алгоритму AES-256-GCM, який застосовується в даній системі, вирішальне значення має підтримка процесором набору інструкцій AES-NI (Advanced Encryption Standard New Instructions). Апаратне прискорення AES дозволяє виконувати операції шифрування на рівні залізного блоку процесора, що в 5-10 разів швидше порівняно з програмною реалізацією і суттєво знижує загальне навантаження на CPU.

Для системи з підтримкою до 50 одночасних VPN-підключень мінімальною прийнятною конфігурацією є 4-ядерний процесор з тактовою частотою не менше 2.5 ГГц з підтримкою AES-NI. Проте враховуючи, що сервер одночасно обслуговує кілька Docker-контейнерів - OpenVPN, Nginx, Authelia, LLDAP, Redis та vpn-admin - рекомендованою конфігурацією є 6–8 ядер з тактовою частотою від 3.0 ГГц**.** Така конфігурація дозволяє виділити окремі ядра для OpenVPN, інші для допоміжних сервісів та залишити резерв для пікових навантажень. Слід

зазначити, що OpenVPN у базовій конфігурації є однопотоковим застосунком, тому висока тактова частота одного ядра важливіша за загальну кількість ядер.

Оперативна пам'ять для VPN-сервера розраховується виходячи із споживання кожного Docker-контейнера у робочому режимі. Контейнер OpenVPN при 50 активних підключеннях споживає від 512 МБ до 1 ГБ RAM залежно від обсягу трафіку та розміру буферів. Authelia разом з Redis займають близько 256–384 МБ, LLDAP як сервіс на Rust споживає менше 64 МБ, Nginx - близько 64–128 МБ, vpn-admin на Flask - до 256 МБ. З урахуванням операційної системи та системних процесів загальне споживання становить приблизно 3–4 ГБ при повному навантаженні.

Виходячи з цих розрахунків, мінімальним обсягом оперативної пам'яті є 4 ГБ, однак такий обсяг не залишає резерву для пікових навантажень та системних процесів. Рекомендованим обсягом є 8 ГБ RAM**,** що забезпечує комфортну роботу всіх контейнерів з запасом для кешування та тимчасових пікових навантажень. Тип пам'яті DDR4 або DDR5 суттєво не впливає на продуктивність VPN-системи, проте швидша пам'ять позитивно позначається на загальній чутливості системи. Використання пам'яті з підтримкою ECC (Error Correcting Code) є бажаним для серверного середовища, оскільки знижує ризик апаратних помилок при тривалій безперервній роботі.

Для VPN-сервера тип накопичувача відіграє меншу роль порівняно з CPU та RAM, оскільки основна операційна активність системи зосереджена в пам'яті та мережевому стеку. Проте використання SSD-накопичувача є обов'язковим - він забезпечує швидке завантаження контейнерів після перезапуску, швидкий доступ до PKI-сховища при генерації сертифікатів та низькі затримки при записі

журналів. Механічні HDD-накопичувачі є неприйнятними через високі затримки операцій введення-виведення.

Щодо обсягу: операційна система займає близько 5-8 ГБ, Docker-образи всіх контейнерів - близько 2-3 ГБ, PKI-інфраструктура з сертифікатами для 50 користувачів - менше 100 МБ, системні журнали при активному логуванні - до 5 ГБ на місяць. Таким чином, мінімальний обсяг накопичувача становить 20 ГБ**, а** рекомендованим є SSD обсягом від 50 ГБ, що забезпечує достатній простір для журналів, резервних копій конфігурацій та можливого розширення системи.

Мережевий канал є найбільш критичним обмеженням для VPN-сервера, розгорнутого на домашньому або малому офісному підключенні. При розрахунку необхідної пропускної здатності слід виходити з того, що кожен активний VPN-користувач споживає в середньому від 1 до 10 Мбіт/с залежно від характеру роботи. Для корпоративного використання з переважно офісними задачами (документи, електронна пошта, корпоративні сервіси) середнє споживання становить близько 3-5 Мбіт/с на користувача.

При 50 одночасних підключеннях з середнім споживанням 5 Мбіт/с сумарне навантаження на канал становитиме 250 Мбіт/с. З урахуванням накладних витрат на VPN-інкапсуляцію (близько 10-15%) реальна потреба в пропускній здатності каналу становить **не** менше 300 Мбіт/с симетричного з'єднання. Мережевий інтерфейс сервера має підтримувати швидкість 1 Гбіт/с, що є стандартом для сучасного серверного обладнання та залишає запас для зростання навантаження.

Тестування продуктивності

Тестування розробленої VPN-системи проводилось з метою перевірки відповідності системи заявленим технічним вимогам щодо підтримки до 50 одночасних корпоративних підключень. Для проведення тестування було обрано утиліту iperf3 - стандартний інструмент вимірювання пропускної здатності мережі, який широко використовується в інженерній практиці та академічних дослідженнях для оцінки продуктивності мережевих систем. Утиліта iperf3 дозволяє генерувати керований мережевий трафік із заданими параметрами швидкості, тривалості та кількості паралельних потоків, що робить її оптимальним інструментом для імітації навантаження від визначеної кількості одночасних користувачів.

Тестове середовище складалось з двох вузлів. Першим вузлом виступав сервер - домашній ПК з процесором AMD Ryzen 7 7800X3D та 32 ГБ оперативної пам'яті DDR5, на якому розгорнуто всі Docker-контейнери системи. Другим вузлом виступав клієнтський ноутбук з встановленим OpenVPN GUI, підключений до VPN-сервера через публічну мережу інтернет. Утиліту iperf3 було встановлено на обох вузлах: на серверному вузлі — всередині Docker-контейнера OpenVPN за допомогою пакетного менеджера Alpine Linux командою apk add iperf3, на клієнтському вузлі - на операційній системі Windows через менеджер пакетів winget командою winget install iperf3.

Серверна частина iperf3 запускалась всередині контейнера OpenVPN командою iperf3 -s -B 0.0.0.0 -p 5201, що забезпечувало прослуховування вхідних з'єднань на всіх мережевих інтерфейсах контейнера включно з VPN-інтерфейсом tun0 з адресою 192.168.255.1. Паралельно на серверному вузлі в окремому терміналі виконувалась команда docker stats --no-stream для знімання показників

споживання ресурсів всіма контейнерами системи в реальному часі під час навантажувального тесту.

Першим етапом тестування було вимірювання пропускної здатності VPN-тунелю для встановлення базових показників продуктивності системи. Після підключення клієнтського ноутбука до VPN через OpenVPN GUI та отримання віртуальної адреси 192.168.255.6 у підмережі 192.168.255.0/24 на клієнтському вузлі виконувалась команда iperf3 -c 192.168.255.1 -t 30 -P 4, що запускала тест тривалістю 30 секунд з використанням 4 паралельних TCP-потоків. Параметр -P 4 було обрано для першого тесту щоб отримати базові показники пропускної здатності без надмірного навантаження.

За результатами базового тестування середня пропускна здатність VPN-тунелю склала 85.7 Мбіт/с у напрямку клієнт-сервер та 84.3 Мбіт/с у зворотному напрямку сервер-клієнт. Кількість повторних передач пакетів (параметр Retr) склала лише 3 пакети за весь час тестування, що складає менше 0.002% від загального обсягу переданих даних і свідчить про виняткову стабільність з'єднання та відсутність суттєвих втрат пакетів у VPN-тунелі. Затримка (RTT) через VPN-тунель вимірювалась командою ping 192.168.255.1 -n 100 і становила в середньому 8 мс, що є цілком прийнятним показником для корпоративного використання та не створює відчутних затримок при роботі з корпоративними застосунками.

Основним етапом тестування була імітація одночасного навантаження від 50 корпоративних користувачів. Оскільки розгортання 50 фізичних клієнтських машин є практично неможливим в умовах розробки індивідуального проєкту, було застосовано методологію імітаційного тестування з використанням

параметра -P 50 утиліти iperf3, який створює 50 паралельних TCP-потоків в межах одного VPN-з'єднання. З точки зору навантаження на механізм шифрування OpenVPN та мережевий стек сервера 50 паралельних потоків є функціонально еквівалентними до 50 окремих клієнтів з активним трафіком, оскільки кожен потік вимагає окремих операцій шифрування та дешифрування пакетів.

На клієнтському ноутбуці виконувалась команда iperf3 -c 192.168.255.1 -t 30 -P 50 -p 5201, що запускала навантажувальний тест тривалістю 30 секунд з 50 паралельними потоками. Параметри тесту розраховувались виходячи з типового споживання корпоративного користувача - в середньому 1.5-3 Мбіт/с при роботі з офісними застосунками, електронною поштою та внутрішніми веб-сервісами, що в сумі для 50 користувачів дає сумарне навантаження 75-150 Мбіт/с.

За результатами навантажувального тестування сумарна пропускна здатність при 50 паралельних потоках склала 85.7 Мбіт/с на стороні відправника та 84.3 Мбіт/с на стороні отримувача, що підтверджує здатність системи обслуговувати заявлену кількість користувачів з достатньою пропускною здатністю. Середня швидкість на один потік становила 1.71 Мбіт/с, що відповідає типовому споживанню корпоративного користувача при роботі з офісними застосунками. Кількість повторних передач за весь час навантажувального тесту склала лише 3 пакети, що підтверджує стабільність з'єднання навіть при значному навантаженні.

Паралельно з навантажувальним тестуванням на серверному вузлі виконувався моніторинг споживання ресурсів всіма Docker-контейнерами

системи за допомогою команди docker stats. Результати моніторингу зафіксовано на рисунку нижче.

| CONTAINER ID | NAME | CPU % | MEM USAGE / LIMIT | MEM % | NET I/O | BLOCK I/O | PIDS |
|---|---|---|---|---|---|---|---|
| 13841c170db4 | openvpn_authelia-certbot-1 | 0.00% | 2.352MiB / 15.26GiB | 0.02% | 872B / 126B | 0B / 4.56MB | 2 |
| d15b05fc81ec | nginx | 0.02% | 3.754MiB / 15.26GiB | 0.02% | 5.54MB / 6.32MB | 135kB / 4.1kB | 4 |
| 3ad0643219e1 | vpn-admin | 0.10% | 103.5MiB / 15.26GiB | 0.67% | 17.2kB / 31.9kB | 26.4MB / 922kB | 3 |
| ebc8a5547da8 | authelia | 0.01% | 28.79MiB / 15.26GiB | 0.18% | 467kB / 4.69MB | 40.1MB / 4.1kB | 14 |
| f4bc6c555091 | redis | 0.09% | 5.195MiB / 15.26GiB | 0.03% | 54.4kB / 86.6kB | 111kB / 4.1kB | 6 |
| 737ff5e349db | vpn-db | 0.00% | 29.41MiB / 15.26GiB | 0.19% | 16.7kB / 9.11kB | 19.6MB / 65.1MB | 8 |
| bdab5b2f4d8c | openvpn | 9.14% | 214.7MiB / 15.26GiB | 1.37% | 4.64GB / 4.71GB | 11MB / 2.27MB | 5 |
| d49f9d1aef5a | lldap | 0.01% | 4.75MiB / 15.26GiB | 0.03% | 84.7kB / 3.84MB | 30.3MB / 1.18MB | 4 |

Рисунок 3.24 – Результати команди docker stats

Як видно з таблиці, найбільше навантаження під час тесту припадало на контейнер OpenVPN - 9,14% CPU, що є очікуваним результатом, оскільки саме цей компонент виконує криптографічні операції шифрування та дешифрування для кожного пакета даних.

# ВИСНОВКИ

У результаті виконання роботи було розроблено автоматизований комплекс для безпечного користування корпоративними ресурсами.

Виконано ознайомлення з предметною областю корпоративних VPN рішень та систем двофакторної автентифікації, було визначено необхідний фунціонал програмного продукту і вибрано мову та середовище програмування. Розроблено алгоритм програми, визначено її сервіси, і як результат було розроблено новий апаратно-програмний комплекс. Проведено детальне тестування працездатності програми, як результат було встановлено, що програма виконує свій заявлений функціонал.

З даної роботи можна виділити такі переваги як, підвищений рівень безпеки для компанії, автоматизація процесів, мікросервісна архітектура, зручний користувацький інтерфейс.

Отже, даний програмний застосунок забезпечує повнофункціональну систему безпечного віддаленого доступу на основі протоколу OpenVPN.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. Welcome to Python.org [Електронний ресурс] / Режим доступу: https://www.python.org/
2. Розгортання Docker [Електронний ресурс] / Режим доступу: https://www.docker.com/
3. Інтеграція Authelia [Електронний ресурс] / Режим доступу: https://www.authelia.com/
4. Використання OpenVPN [Електронний ресурс] / Режим доступу: https://openvpn.net/
5. Документація Nginx [Електронний ресурс] / Режим доступу: https://nginx.org/
6. Сертифікати SSL [Електронний ресурс] / Режим доступу: https://certbot.eff.org/
7. Розуміння LDAP-протокол [Електронний ресурс] / Режим доступу: https://habr.com/ru/articles/538662/
8. Реєстрація домена [Електронний ресурс]. — Режим доступу: https://www.duckdns.org/domains
9. Документація PostgreSQL [Електронний ресурс] / Режим доступу: https://www.postgresql.org/docs/
10. Бібліотеки Python [Електронний ресурс] / Режим доступу: https://www.aspose.org/uk/
11. Документація Redis [Електронний ресурс] / Режим доступу: https://redis.io/docs/latest/

12. Підручник по JavaScript [Електронний ресурс] / Режим доступу: https://uk.javascript.info/

13. Як працює VPN [Електронний ресурс] / Режим доступу: https://www.cloud4y.ru/blog/what-is-a-vpn-its-features-and-configuration/

14. Порівняння протоколів [Електронний ресурс] / Режим доступу: https://www.vpnunlimited.com/ua/help/vpn-protocols/comparison?srsltid=AfmBOoqd7faLsNw7wUPXYEYGv0XPHHLSLY8O9CZmQwCMQ-obuZGrmOkE/

15. Пилгрим Марк. Погружение в Python 3 (Dive into Python 3 на русском) / Пилгрим Марк. – М. : Самиздат. – 2009. – 148 с.

16. Чан Уэсли Дж. Python: создание приложений / Чан Уэсли Дж. – 3-е изд. – М. : Вильямс, 2015. – 816 с.