ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «СИСТЕМА БАГАТОФАКТОРНОЇ АВТЕНТИФІКАЦІЇ ДЛЯ ДОСТУПУ ДО ПРИВАТНОЇ СЕРВЕРНОЇ ІНФРАСТРУКТУРИ»

на здобуття освітнього ступеня бакалавра
зі спеціальності 123 Комп'ютерна інженерія
(код, найменування спеціальності)
освітньо-професійної програми «Комп'ютерна інженерія»
(назва)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ *Назарій ОХРІМЕНКО*
(підпис) *Ім'я, ПРІЗВИЩЕ здобувача*

Виконав: здобувач вищої освіти гр. КІД-41
Назарій ОХРІМЕНКО
Ім'я, ПРІЗВИЩЕ

Керівник: Ігор БУЧЕНКО
науковий ступінь, вчене звання Ім'я, ПРІЗВИЩЕ

Рецензент: _____
науковий ступінь, вчене звання Ім'я, ПРІЗВИЩЕ

Київ 2026

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут Інформаційних технологій

Кафедра Комп'ютерної інженерії

Ступінь вищої освіти бакалавр

Спеціальність 123 Комп'ютерна інженерія

Освітньо-професійна програма «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Комп'ютерної інженерії
_____ Наталія ЛАЩЕВСЬКА
«_____» _____ 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Охріменко Назарій Олександрович
(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: «Система багатофакторної автентифікації для доступу до приватної серверної інфраструктури»
керівник кваліфікаційної роботи: Ігор БУЧЕНКО, старший викладач ,
(Ім'я, ПРІЗВИЩЕ, науковий ступінь, вчене звання)
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «20» лютого 2026р. №51
2. Строк подання кваліфікаційної роботи «14» травня 2026р.
3. Вихідні дані до кваліфікаційної роботи: науково-технічна література, стандарти та протоколи OAuth 2.0, OpenID Connect і SAML, принципи та алгоритми багатофакторної автентифікації (TOTP, HOTP, FIDO2), архітектура систем Identity and Access Management та платформи Authentik, контейнеризація та оркестрація сервісів Docker, методи захисту мережевих сервісів через Forward Authentication, аналіз продуктивності, масштабованості та безпеки серверної інфраструктури.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
    1. Технології автентифікації та контролю доступу в сучасних мережах
    2. Побудова системи автентифікації на базі Fedora Linux та Authentik
    3. Аналіз ефективності MFA та перспективи масштабування
5. Перелік ілюстративного матеріалу: *презентація*
    1. Мета, об'єкт, предмет дослідження
    2. Актуальність роботи
    3. Архітектура рішення (основа)
    4. Практичне середовище
    5. Розгортання системи
    6. Налаштування Authentik
    7. Реалізація MFA (TOTP)
    8. Алгоритм роботи системи
    9. Практична інтеграція

10. Результати тестування
11. Аналіз продуктивності
12. Оптимізація та масштабування
13. Висновки
14. Публікації та апробація роботи

6. Дата видачі завдання «20» лютого 2026р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Збір даних	22.02-20.03.2026	Виконано
2	Обробка матеріалу	20.03-30.03.2026	Виконано
3	Розробка теоретичної частини	31.03-10.04.2026	Виконано
4	Побудова системи автентифікації на базі Fedora Linux та Authentik	11.04-20.04.2026	Виконано
5	Висновки за результатами аналізу	21.04-26.04.2026	Виконано
6	Оформлення пояснювальної записки	26.04-28.04.2026	Виконано
7	Розробка презентації	29.04-01.05.2026	Виконано
8	Передзахист	08.05-11.05.2026	Виконано
9	Здача в деканат	08.05-14.05.2026	Виконано

Здобувач(ка) вищої освіти _____ Назарій ОХРІМЕНКО
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Керівник кваліфікаційної роботи _____ Ігор БУЧЕНКО
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 79 стор., 39 рис., 5 табл., 30 джерел.

Мета роботи – зменшення ризиків несанкціонованого доступу та спрощення контролю доступу за рахунок MFA, Zero Trust і Forward Authentication.

Об'єкт дослідження – процес забезпечення безпечного доступу до сервісів у приватній мережевій інфраструктурі.

Предмет дослідження – засоби та технології реалізації MFA, централізованого управління доступом і захисту веб-сервісів у контейнеризованому середовищі.

Короткий зміст роботи: У роботі представлено теоретичні основи сучасних систем автентифікації та контролю доступу, розглянуто обмеження парольних підходів, принципи MFA, алгоритми TOTP/HOTP і протоколи OAuth 2.0, OpenID Connect та SAML, проаналізовано архітектуру Identity Provider і механізм Forward Authentication як спосіб захисту сервісів без їх модифікації, а також підходи до інтеграції в контейнерні середовища з урахуванням масштабованості, ізоляції та аудиту подій безпеки.

Особливу увагу приділено практичній реалізації прототипу на базі Fedora Linux із використанням Authentik і Docker, включаючи розгортання компонентів, налаштування MFA на основі TOTP, інтеграцію тестового сервісу через Forward Auth і перевірку сценаріїв доступу, а також аналіз ефективності, виявлення вузьких місць, обґрунтування використання кешування та подальший перехід до архітектури Edge Security Gateway із централізованим контролем доступу, масштабуванням і інтеграцією у корпоративне середовище.

КЛЮЧОВІ СЛОВА: БАГАТОФАКТОРНА АВТЕНТИФІКАЦІЯ, MFA, AUTHENTIK, FORWARD AUTHENTICATION, ZERO TRUST, DOCKER, TOTP, ІДЕНТИФІКАЦІЯ, КОНТРОЛЬ ДОСТУПУ, EDGE SECURITY GATEWAY, КОНТЕЙНЕРИЗАЦІЯ, БЕЗПЕКА МЕРЕЖІ.

ЗМІСТ

ВСТУП 8
1. ТЕХНОЛОГІЇ АВТЕНТИФІКАЦІЇ ТА КОНТРОЛЮ ДОСТУПУ В СУЧАСНИХ МЕРЕЖАХ 9
1.1 Еволюція систем контролю доступу 9
1.2 Методи та фактори автентифікації 14
1.3 Протоколи федеративної ідентифікації 23
1.4 Платформи управління ідентифікацією (на прикладі Authentik) 28
1.5 Безпека приватної серверної інфраструктури 34
2. ПОБУДОВА СИСТЕМИ АВТЕНТИФІКАЦІЇ НА БАЗІ FEDORA LINUX ТА AUTHENTIK 40
2.1 Архітектура та вибір компонентів 40
2.2 Реалізація та конфігурація системи 47
2.3 Алгоритми та внутрішні механізми захисту 57
2.4 Практична інтеграція та валідація системи 62
3. АНАЛІЗ ЕФЕКТИВНОСТІ MFA ТА ПЕРСПЕКТИВИ МАСШТАБУВАННЯ 70
3.1 Аналіз споживання ресурсів та критичні вузькі місця 70
3.2 Впровадження на мережевий вузол (Edge Security Gateway) 77
3.3 Перспективи розвитку та інфраструктурні покращення 82
ВИСНОВКИ 86
ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ 87
Додаток А. Конфігурація системи MFA на базі Authentik (Docker Compose) 89
Додаток Б. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ (Презентація) 90

ВСТУП

Мережеві інфраструктури характеризуються зростанням кількості сервісів, користувачів і точок доступу, що підвищує вимоги до механізмів автентифікації та контролю доступу, при цьому традиційні підходи на основі паролів поступово втрачають ефективність через вразливість до підбору, фішингу та повторного використання облікових даних, що зумовлює перехід до стійкіших моделей безпеки, де перевірка базується на поєднанні кількох факторів і централізованому управлінні ідентифікацією.

Розвиток багатофакторної автентифікації дозволяє об’єднати фактори знання, володіння та властивості в єдину модель перевірки доступу, де одноразові паролі за алгоритмами TOTP або HOTP доповнюються підходами на основі апаратних ключів і стандартів FIDO2/WebAuthn, а також інтегруються з протоколами федеративної ідентифікації, такими як OAuth 2.0, OpenID Connect та SAML, що забезпечує єдиний вхід і взаємодію між сервісами.

Подальший розвиток отримують системи управління ідентифікацією та доступом, які реалізують централізований контроль політик безпеки, обробку запитів і керування сесіями, при цьому важливим стає підхід Forward Authentication, що дозволяє винести перевірку доступу за межі сервісів і застосовувати її як захисний шар, а інтеграція у контейнерні середовища підвищує гнучкість, ізольованість і масштабованість інфраструктури з одночасним аудитом подій.

У практичній частині передбачається реалізація прототипу системи багатофакторної автентифікації на базі Fedora Linux із використанням Authentik і Docker, де буде розгорнуто ізольований стек компонентів, налаштовано MFA на основі TOTP, реалізовано Forward Auth для захисту сервісів, виконано інтеграцію тестового веб-ресурсу та перевірку сценаріїв доступу, що дозволить оцінити роботу системи й сформувати основу для подальшого розширення до рівня мережевого шлюзу безпеки.

1. ТЕХНОЛОГІЇ АВТЕНТИФІКАЦІЇ ТА КОНТРОЛЮ ДОСТУПУ В СУЧАСНИХ МЕРЕЖАХ

1.1 Еволюція систем контролю доступу

Контроль доступу є одним із базових елементів забезпечення інформаційної безпеки, який сформувався разом із розвитком комп'ютерних мереж, хмарних технологій та розподілених систем, що безпосередньо вплинуло на підходи до автентифікації користувачів і визначення їх прав у межах інформаційного середовища. На ранніх етапах розвитку інформаційних систем достатнім вважався простий механізм ідентифікації, заснований на використанні логіну та пароля, однак із суттєвим зростанням кількості кіберзагроз, витоків даних та ускладненням інфраструктур подібні рішення втратили ефективність і вже не здатні забезпечити належний рівень захисту.

Еволюція систем контролю доступу відбувалася поступово та включала кілька ключових етапів, починаючи з локальних систем автентифікації, у яких кожен ресурс мав власну базу користувачів і не був інтегрований з іншими сервісами, що створювало значні труднощі в управлінні доступом і підтримці актуальності облікових даних. Подальший розвиток привів до появи централізованих систем, які дозволили об'єднати контроль доступу в межах організації та значно спростили адміністрування, після чого відбулося впровадження федеративних систем ідентифікації, що забезпечують доступ до різних сервісів через єдину точку входу, відому як Single Sign-On, що суттєво підвищило зручність користування та зменшило кількість облікових даних, які необхідно запам'ятовувати.

Наступним етапом розвитку стали адаптивні системи безпеки, які враховують не лише облікові дані користувача, а й контекст доступу, поведінкові характеристики та інші додаткові параметри, що дозволяє більш точно визначати

легітимність дій. У межах такого підходу особливої актуальності набуває концепція Zero Trust, відповідно до якої жоден користувач або пристрій не вважається довіреним за замовчуванням навіть у межах внутрішньої мережі, що зумовлює необхідність постійної перевірки автентичності та впровадження додаткових механізмів контролю доступу, оскільки наявність правильних облікових даних не гарантує, що доступ здійснюється саме уповноваженою особою.

Обмеження традиційної парольної автентифікації. Парольна автентифікація протягом тривалого часу залишалася основним методом захисту доступу до інформаційних систем, оскільки базувалася на простому та зрозумілому принципі підтвердження особи через введення секретної інформації, відомої лише користувачу, однак сучасні умови експлуатації виявили значну кількість недоліків цього підходу, що обмежують його ефективність у контексті сучасних кіберзагроз.

Однією з ключових проблем є низька стійкість до атак, оскільки паролі залишаються вразливими до методів перебору, зокрема brute-force атак, словникових атак, які використовують поширені комбінації, а також до фішингових атак, під час яких користувач вводить свої дані на підроблених ресурсах, що імітують легітимні сервіси, при цьому додаткову небезпеку становлять атаки типу credential stuffing, коли використовуються злиті бази облікових даних, отримані внаслідок попередніх витоків інформації, що робить повторне використання паролів особливо ризикованим [1].

Суттєвий вплив на рівень безпеки має людський фактор, оскільки користувачі часто використовують прості або повторювані паролі, зберігають їх у небезпечних місцях або не змінюють протягом тривалого часу, що знижує ефективність навіть формально складних політик безпеки. Додатковим недоліком є відсутність механізмів перевірки, які виходять за межі знання пароля, через що система не може підтвердити, що доступ здійснюється саме власником облікового запису, а не сторонньою особою, яка отримала ці дані (див. рис. 1.1) [1].

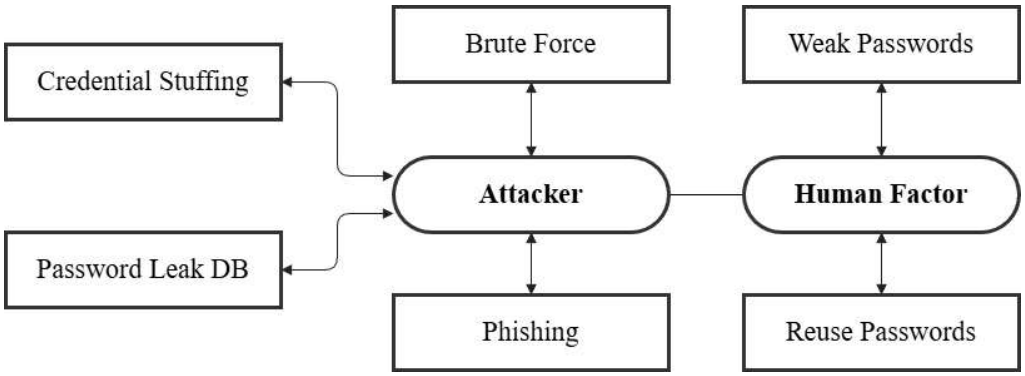


Рисунок 1.1 – Основні вразливості традиційної парольної автентифікації

У великих інфраструктурах виникають також проблеми масштабування, оскільки складно централізовано управляти обліковими даними, забезпечувати єдину політику безпеки та контролювати їх використання, що призводить до зростання навантаження на адміністраторів і підвищує ризик помилок. Сукупність цих факторів свідчить про те, що парольна автентифікація у чистому вигляді не відповідає сучасним вимогам безпеки і потребує доповнення більш надійними механізмами, зокрема багатофакторними підходами.

Концепція багатофакторної автентифікації (MFA). Обмеження традиційних методів автентифікації, зокрема використання лише паролів, обумовили розвиток більш надійних підходів до підтвердження особи користувача, одним із яких стала концепція багатофакторної автентифікації (Multi-Factor Authentication, MFA), що передбачає застосування кількох незалежних факторів для перевірки доступу. Такий підхід базується на ідеї, що одного джерела підтвердження недостатньо для забезпечення належного рівня безпеки, особливо в умовах зростання кількості атак, пов’язаних із компрометацією облікових даних.

У межах MFA використовуються різні типи факторів автентифікації, які доповнюють один одного та підвищують загальну стійкість системи до несанкціонованого доступу. Найбільш поширеним є фактор знання, який включає інформацію, відому лише користувачу, зокрема паролі, PIN-коди або відповіді на контрольні запитання, однак його недостатність у сучасних умовах зумовлює необхідність застосування додаткових механізмів. Фактор володіння передбачає використання фізичних або програмних засобів, таких як смартфони, апаратні

токени чи генератори одноразових паролів, які підтверджують, що користувач має доступ до певного пристрою, тоді як фактор властивості базується на унікальних біометричних характеристиках, включаючи відбитки пальців, розпізнавання обличчя або голосову ідентифікацію, що значно ускладнює можливість підробки [2].

Процес багатофакторної автентифікації реалізується у кілька послідовних етапів, де спочатку виконується введення базових облікових даних, після чого система ініціює додаткову перевірку за допомогою другого фактору, наприклад одноразового коду або підтвердження через мобільний додаток, а завершальним етапом стає прийняття рішення щодо надання або відмови в доступі на основі результатів перевірки. Така багаторівнева модель значно підвищує рівень захищеності, оскільки навіть у випадку компрометації одного з факторів зловмисник не отримує повного доступу до системи (див. рис. 1.2) [2].

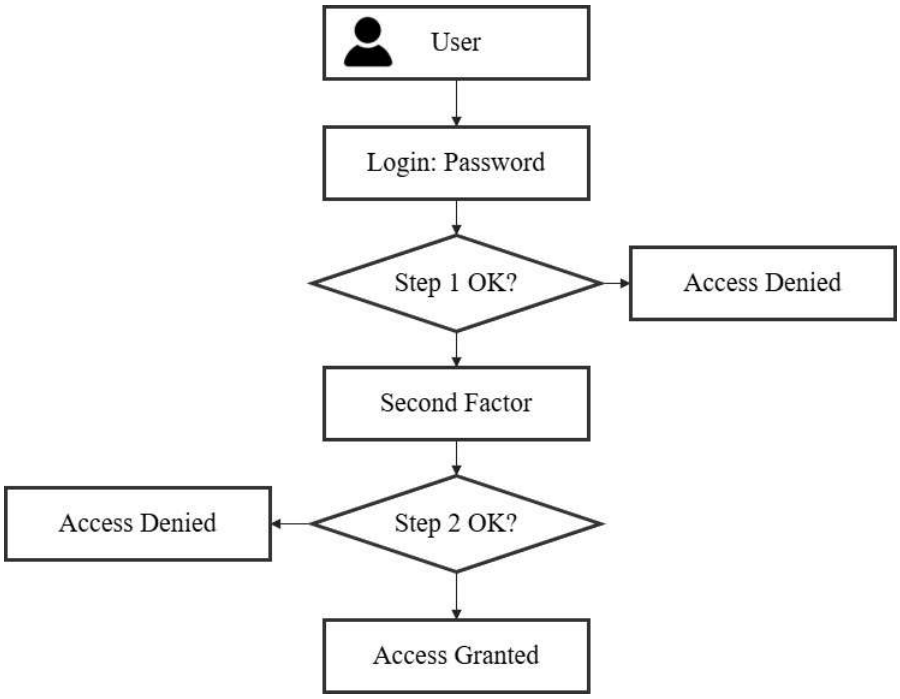


Рисунок 1.2 – Концепція багатофакторної автентифікації (MFA)

Застосування MFA забезпечує суттєве зниження ризику несанкціонованого доступу, підвищує загальний рівень безпеки інформаційних систем та дозволяє

інтегрувати додаткові механізми захисту без необхідності суттєвих змін у наявній інфраструктурі. Важливою перевагою є можливість використання цього підходу у різних типах сервісів, включаючи ті, що спочатку не підтримують багатофакторну автентифікацію, оскільки сучасні рішення дозволяють реалізувати її на рівні зовнішніх систем контролю доступу. Практичне значення такого підходу полягає у тому, що навіть у разі викрадення пароля зловмисник не зможе отримати доступ без другого фактору, що робить MFA одним із найбільш ефективних методів захисту, особливо у випадках віддаленого доступу або роботи з критично важливою інфраструктурою.

Роль Identity та Access Management (IAM) систем. Забезпечення ефективного контролю доступу у сучасних інформаційних системах неможливе без використання систем управління ідентифікацією та доступом (Identity and Access Management, IAM), які дозволяють централізовано керувати обліковими записами користувачів, їхніми правами та політиками безпеки в межах організації або окремої інфраструктури. Такий підхід значно спрощує адміністрування, підвищує рівень контролю та забезпечує узгодженість механізмів доступу до різних ресурсів.

Функціональність IAM-систем охоплює кілька взаємопов’язаних процесів, серед яких ідентифікація користувачів передбачає створення та ведення облікових записів із забезпеченням їх унікальності, що дозволяє однозначно визначити кожного суб’єкта доступу в системі. Автентифікація забезпечує перевірку достовірності користувача шляхом використання різних механізмів, включаючи багатофакторну автентифікацію, що підвищує рівень захищеності облікових записів. Авторизація визначає права доступу до ресурсів на основі встановлених політик, таких як рольова модель доступу або атрибутна модель, що дозволяє гнучко керувати дозволами залежно від контексту використання [3].

Важливою складовою є також аудит та моніторинг, які передбачають ведення журналів входів, фіксацію дій користувачів та аналіз підозрілої активності, що дозволяє своєчасно виявляти потенційні загрози та реагувати на інциденти безпеки. Централізований характер IAM-рішень сприяє зниженню адміністративного

навантаження, оскільки управління доступом здійснюється з єдиного місця, а також підвищує прозорість і контрольованість процесів безпеки.

У контексті сучасних технологій IAM-системи інтегруються з контейнерними середовищами та платформами розгортання, такими як Docker, що забезпечує можливість швидкого запуску сервісів автентифікації, їх масштабування відповідно до навантаження та ізоляцію компонентів системи. Подібні рішення можуть виступати як Identity Provider і забезпечувати єдину точку входу до різних сервісів, включаючи ті, що не мають власних механізмів автентифікації, що є особливо важливим для захисту різнорідної інфраструктури [3].

У межах приватних серверних середовищ використання IAM-систем дозволяє впровадити єдину політику доступу, забезпечити:

— захист внутрішніх ресурсів;

— контролювати доступ до адміністративних інтерфейсів;

— реалізувати централізовану модель безпеки через проксі-рівень.

Такий підхід формує основу для побудови комплексних систем захисту, які поєднують автентифікацію, авторизацію та контроль доступу, забезпечуючи високий рівень безпеки у сучасних інформаційних інфраструктурах.

1.2 Методи та фактори автентифікації

Автентифікація як процес підтвердження особи користувача перед наданням доступу до інформаційних ресурсів є невід’ємною складовою систем інформаційної безпеки, що набувала розвитку разом із ускладненням IT-інфраструктур, зростанням кількості кіберзагроз та підвищенням вимог до захисту даних. Класичні підходи, засновані на використанні одного фактора, поступово втратили ефективність, що зумовило перехід до багатофакторних моделей, у межах яких для перевірки користувача застосовується декілька незалежних ознак. Методи

автентифікації визначаються типом інформації або характеристик, які використовуються для ідентифікації, а їх класифікація традиційно ґрунтується на поділі на три базові категорії, до яких належать:

— фактори знання;

— володіння;

— властивості.

Поєднання яких формує основу сучасних систем захисту доступу.

Фактори знання, володіння та властивості (біометрія). Фактор знання передбачає використання інформації, відомої лише користувачу, зокрема:

— паролів;

— PIN-кодів;

— відповідей на секретні запитання.

Все це історично є найстарішим і найпростішим способом автентифікації, реалізація якого базується на здатності людини запам’ятовувати та відтворювати конфіденційні дані. Такий підхід характеризується залежністю від людського фактора, оскільки користувачі можуть використовувати прості або повторювані комбінації, зберігати їх у ненадійних місцях або ставати жертвами соціальної інженерії, що суттєво знижує рівень безпеки. Додатковою проблемою є можливість компрометації облікових даних унаслідок витоків інформації, що робить використання лише цього фактора недостатнім для сучасних умов, хоча він і надалі залишається базовим рівнем у більшості систем, доповнюючись іншими механізмами захисту [4].

Фактор володіння ґрунтується на наявності у користувача певного фізичного або віртуального об’єкта, який використовується для підтвердження доступу, що може включати [4]:

— смартфони;

— апаратні токени;

— смарт-карти;

— генератори одноразових паролів.

Остінні можуть бути реалізовані у вигляді мобільних додатків. Використання такого підходу значно підвищує рівень безпеки, оскільки навіть у випадку компрометації пароля зловмисник не зможе отримати доступ без наявності відповідного пристрою. Сучасні реалізації фактору володіння часто базуються на використанні одноразових кодів, які генеруються за допомогою спеціалізованих додатків після початкової прив’язки через QR-код, що забезпечує поєднання зручності використання та високого рівня захищеності (див. рис. 1.3).

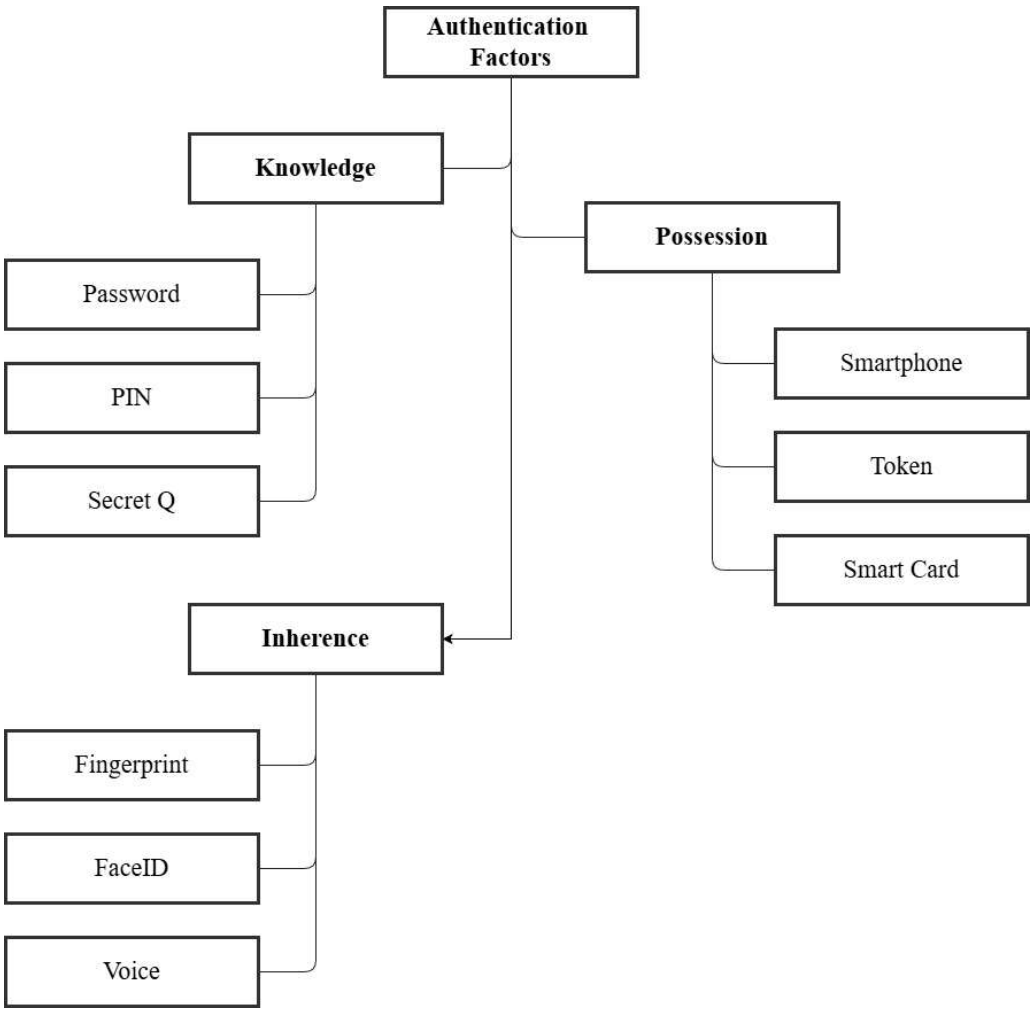


Рисунок 1.3 – Класифікація факторів автентифікації

Фактор властивості пов’язаний з використанням унікальних фізичних або поведінкових характеристик користувача, до яких належать:

— відбитки пальців;

— розпізнавання обличчя;

— голосова ідентифікація;

— поведінкові патерни.

Все це дозволяє визначити особу на основі індивідуальних особливостей. Такий підхід ускладнює можливість підробки або передачі даних іншій особі, однак супроводжується низкою обмежень, серед яких необхідність використання спеціалізованого обладнання, складність впровадження в окремих середовищах, а також ризики, пов’язані зі зберіганням біометричних даних, які у разі компрометації не можуть бути змінені, на відміну від паролів чи токенів (див. табл. 1.1).

Таблиця 1.1 – Порівняння методів автентифікації та їх характеристик

Тип фактору	Приклади	Переваги	Недоліки	Рівень безпеки
Знання	Пароль, PIN, секретні питання	Простота реалізації, не потребує додаткових пристроїв	Вразливість до фішингу, витоків, людський фактор	Низький
Володіння	Смартфон, OTP-додаток, токен	Захист навіть при компрометації пароля	Залежність від пристрою, можливість втрати	Середній
Властивості (біометрія)	Відбиток пальця, Face ID	Висока унікальність, складність підробки	Потреба в обладнанні, ризики конфіденційності	Високий
Комбінований (MFA)	Пароль та TOTP / ключ	Значне підвищення безпеки	Ускладнення користування	Дуже високий

Додатковим аспектом є питання конфіденційності, оскільки обробка біометричної інформації потребує дотримання підвищених вимог до захисту персональних даних [4].

У сучасних інформаційних системах ці фактори рідко використовуються ізольовано, оскільки комбінування декількох підходів дозволяє суттєво підвищити

рівень безпеки та зменшити ризик несанкціонованого доступу, формуючи багаторівневу модель захисту, яка може адаптуватися до різних сценаріїв використання. Така інтеграція факторів лежить в основі багатофакторної автентифікації, яка широко застосовується як у корпоративних, так і в приватних інфраструктурах, забезпечуючи баланс між зручністю користування та ефективністю захисту інформаційних ресурсів.

Принципи генерації одноразових паролів (TOTP/HOTP). Одним із найбільш поширених механізмів реалізації другого фактору автентифікації є використання одноразових паролів (One-Time Password, OTP), які забезпечують підвищений рівень захисту завдяки обмеженому терміну дії або одноразовому використанню кожного коду, що суттєво знижує ризик їх повторного застосування або компрометації. Основна ідея полягає у генерації унікального коду, який є дійсним лише протягом короткого проміжку часу або для одного сеансу автентифікації, що робить його малопридатним для перехоплення та подальшого використання зловмисниками.

Існує декілька основних алгоритмів генерації одноразових паролів, серед яких найбільш відомими є HMAC-based One-Time Password (HOTP) та Time-based One-Time Password (TOTP), кожен з яких має свої особливості реалізації та використання. Алгоритм HOTP базується на застосуванні секретного ключа, який спільно використовується клієнтом і сервером, а також лічильника, що змінюється при кожній генерації нового коду, унаслідок чого кожен наступний пароль формується на основі криптографічної функції з використанням попереднього значення лічильника. Такий підхід забезпечує унікальність кожного коду, однак потребує суворої синхронізації між клієнтською та серверною сторонами, оскільки розбіжність у значеннях лічильника може призвести до відмови в автентифікації, що ускладнює практичне використання в деяких сценаріях [5-6].

Розвитком цієї концепції став алгоритм TOTP, який використовує той самий принцип криптографічної генерації кодів, але замість лічильника застосовує поточний час як змінну величину, що дозволяє автоматично синхронізувати процес без необхідності відстеження кількості запитів. У такому випадку одноразовий

пароль генерується на основі секретного ключа та часового інтервалу, який зазвичай становить 30 секунд, після чого код стає недійсним і замінюється новим. Такий підхід значно спрощує використання, однак вимагає синхронізації системного часу між клієнтом і сервером, оскільки навіть незначні відхилення можуть призвести до некоректної перевірки коду [5].

Процес налаштування механізмів OTP зазвичай передбачає генерацію унікального секретного ключа, який передається користувачу у вигляді QR-коду, після чого цей ключ імпортується у мобільний додаток-генератор, що дозволяє автоматично створювати одноразові коди без необхідності зберігання їх на стороні сервера у відкритому вигляді. Під час автентифікації користувач вводить актуальний код, згенерований додатком, а система перевіряє його відповідність очікуваному значенню на основі тих самих параметрів, що забезпечує високий рівень захисту без передачі конфіденційної інформації.

Використання одноразових паролів має низку переваг, серед яких варто відзначити короткий термін дії кодів, що знижує ризик їх перехоплення, а також можливість ефективного захисту навіть у випадку компрометації основного пароля, оскільки доступ до системи потребує додаткового підтвердження. Водночас існують і певні обмеження, пов’язані з можливістю атак типу «людина посередині» за відсутності захищеного з’єднання, а також залежністю від коректної синхронізації часу у випадку використання TOTP (див. рис. 1.4) [5].

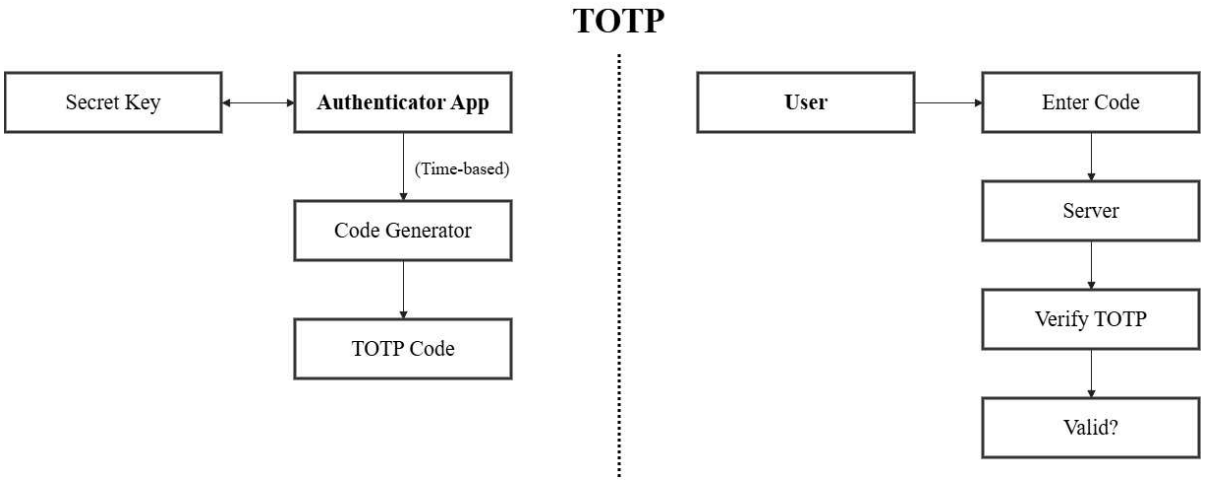


Рисунок 1.4 – Принцип генерації одноразового пароля (TOTP)

У сучасних системах управління доступом механізми OTP широко застосовуються як стандартний компонент багатофакторної автентифікації, забезпечуючи додатковий рівень захисту без суттєвого ускладнення користувацького досвіду, що робить їх ефективним рішенням для захисту як корпоративних, так і приватних інформаційних інфраструктур.

Апаратні ключі та протоколи FIDO2/WebAuthn. Подальший розвиток технологій автентифікації спрямований на усунення недоліків традиційних підходів, що призвело до появи апаратних ключів та сучасних протоколів безпарольного доступу, які забезпечують більш високий рівень безпеки та стійкість до поширених типів атак. Апаратні ключі автентифікації являють собою фізичні пристрої, що підключаються до комп'ютера або мобільного пристрою через інтерфейси USB, NFC або Bluetooth і використовуються для підтвердження особи користувача шляхом виконання криптографічних операцій у захищеному середовищі.

Особливістю таких пристроїв є зберігання приватних ключів у ізольованій області, що унеможливлює їх витік навіть у разі компрометації операційної системи або браузера, при цьому процес автентифікації передбачає фізичну взаємодію користувача з пристроєм, наприклад натискання кнопки або підтвердження через безконтактний інтерфейс, що додатково підвищує рівень захисту від автоматизованих атак. Завдяки відсутності необхідності введення паролів такі рішення значно знижують ризик фішингових атак, оскільки автентифікація прив'язується до конкретного домену і не може бути використана на підроблених ресурсах.

У межах сучасних стандартів широкого поширення набув протокол FIDO2, який дозволяє реалізувати безпарольну автентифікацію та базується на поєднанні двох основних компонентів, серед яких WebAuthn відповідає за взаємодію між браузером і пристроєм автентифікації, а CTAP (Client to Authenticator Protocol) забезпечує обмін даними між клієнтським пристроєм та апаратним ключем. Така архітектура дозволяє створити надійну систему, у якій під час реєстрації користувача генерується пара криптографічних ключів, публічний з яких

передається на сервер і зберігається для подальшої перевірки, тоді як приватний ключ залишається виключно на пристрої користувача і не покидає його меж [7].

Під час автентифікації система ініціює перевірку шляхом підпису певного виклику приватним ключем, після чого сервер перевіряє підпис за допомогою відповідного публічного ключа, що дозволяє підтвердити особу без передачі конфіденційних даних через мережу. Такий підхід забезпечує високий рівень криптографічного захисту, виключає можливість перехоплення облікових даних і значно підвищує стійкість до атак, пов'язаних із соціальною інженерією (див. рис. 1.5).

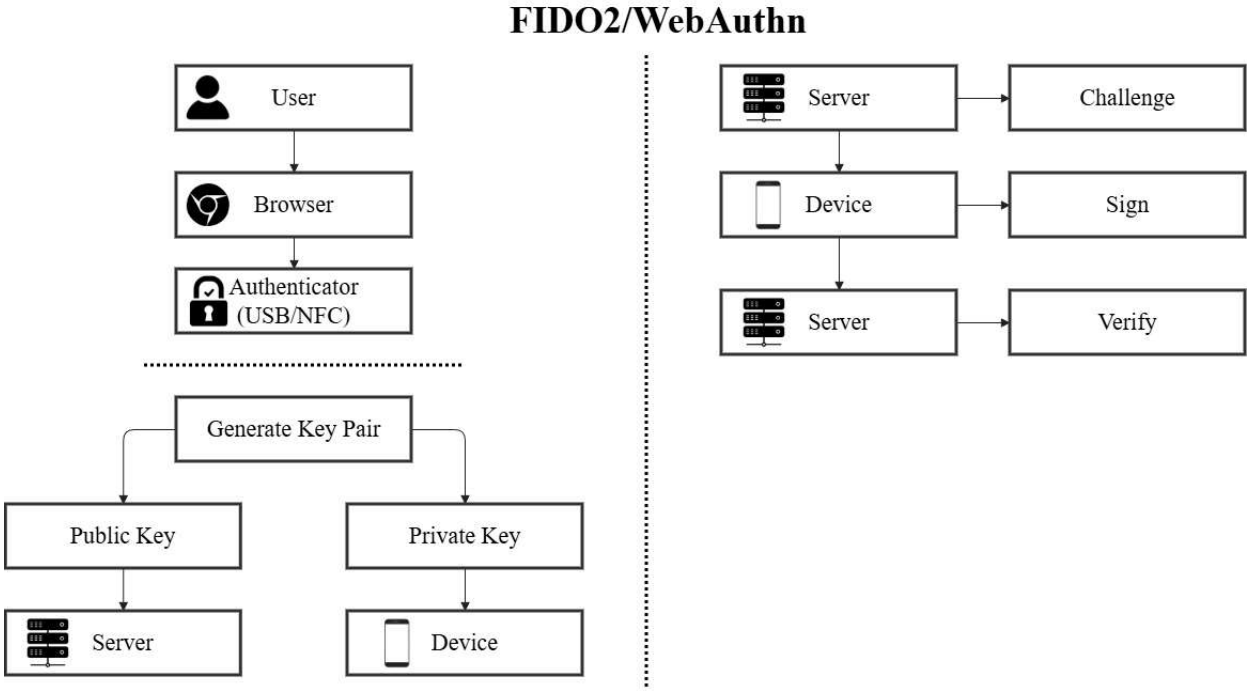


Рисунок 1.5 – Принцип роботи FIDO2/WebAuthn

Серед основних переваг використання FIDO2 та WebAuthn можна виділити відсутність необхідності використання паролів, що усуває один із найбільш вразливих елементів системи безпеки, захист від фішингових атак завдяки прив'язці автентифікації до конкретного ресурсу, а також зручність використання, оскільки процес підтвердження доступу зводиться до простої взаємодії з пристроєм. Інтеграція таких технологій у сучасні системи управління доступом

дозволяє реалізувати альтернативу традиційним методам, включаючи OTP, забезпечуючи більш високий рівень безпеки та покращений користувацький досвід (див. рис. 1.6) [7].

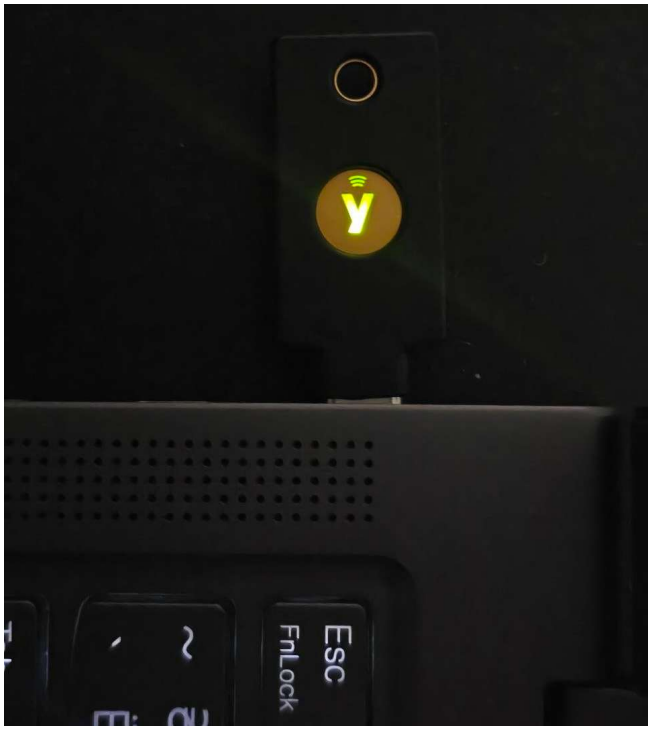


Рисунок 1.6 – Зовнішній вигляд апаратного ключа безпеки Yubikey

Застосування апаратних ключів та протоколів безпарольної автентифікації у поєднанні з централізованими системами управління доступом формує основу для побудови надійної та масштабованої інфраструктури, яка здатна ефективно протистояти сучасним загрозам і забезпечувати захист як у межах корпоративних мереж, так і в приватних середовищах, де необхідно контролювати доступ до критичних ресурсів.

1.3 Протоколи федеративної ідентифікації

В інформаційних система усе більшого поширення набуває концепція федеративної ідентифікації, яка дозволяє користувачам отримувати доступ до різних сервісів за допомогою єдиного облікового запису, що суттєво спрощує процес управління доступом, зменшує кількість необхідних облікових даних та підвищує загальний рівень безпеки інфраструктури. Такий підхід базується на розподілі ролей між кількома сторонами взаємодії, серед яких виділяються:

— користувач як власник ресурсів;

— постачальник ідентифікації;

— сервіс-провайдер.

Останній саме надає доступ до ресурсів і довіряє результатам перевірки, виконаної зовнішньою системою. У межах цієї моделі автентифікація виконується централізовано через відповідний провайдер ідентифікації, після чого інші сервіси приймають результат цієї перевірки та надають доступ без необхідності повторного введення облікових даних, що дозволяє реалізувати механізм єдиного входу, відомий як Single Sign-On.

Функціонування федеративної ідентифікації забезпечується за допомогою спеціалізованих протоколів, які регламентують обмін даними між компонентами системи та визначають правила взаємодії між учасниками. Найбільш поширеними серед них є OAuth 2.0, OpenID Connect та SAML, кожен із яких має свої особливості реалізації, сфери застосування та рівень складності інтеграції. Вибір конкретного протоколу залежить від вимог до системи, типу сервісів та рівня необхідної безпеки, однак загальний принцип залишається незмінним і полягає у відокремленні процесів автентифікації та доступу до ресурсів.

Архітектура та використання OAuth 2.0. Протокол OAuth 2.0 є відкритим стандартом авторизації, який дозволяє стороннім додаткам отримувати обмежений доступ до ресурсів користувача без передачі його конфіденційних облікових даних, що відрізняє його від традиційних підходів, у межах яких користувач змушений

був передавати логін і пароль кожному сервісу окремо. У цьому випадку реалізується механізм делегування доступу, коли користувач надає дозвіл додатку на виконання певних дій без розкриття своїх облікових даних, що значно підвищує рівень безпеки та контроль над використанням ресурсів. Архітектура OAuth 2.0 передбачає взаємодію кількох компонентів, серед яких користувач виступає як власник ресурсу, клієнтський додаток ініціює запит доступу, сервер авторизації відповідає за перевірку облікових даних і видачу токенів, а сервер ресурсів надає доступ до захищених даних на основі отриманого токена [8].

Типовий сценарій роботи передбачає, що користувач ініціює доступ до певного ресурсу через клієнтський додаток, після чого відбувається перенаправлення на сервер авторизації, де здійснюється автентифікація та підтвердження прав доступу, далі сервер повертає клієнту спеціальний код, який обмінюється на токен доступу, що використовується для звернення до ресурсу. У процесі можуть застосовуватися різні типи токенів, серед яких "access token" використовується безпосередньо для доступу до ресурсів, а "refresh token" дозволяє отримувати нові токени без повторної автентифікації, що підвищує зручність використання та оптимізує роботу системи (див. рис. 1.7).

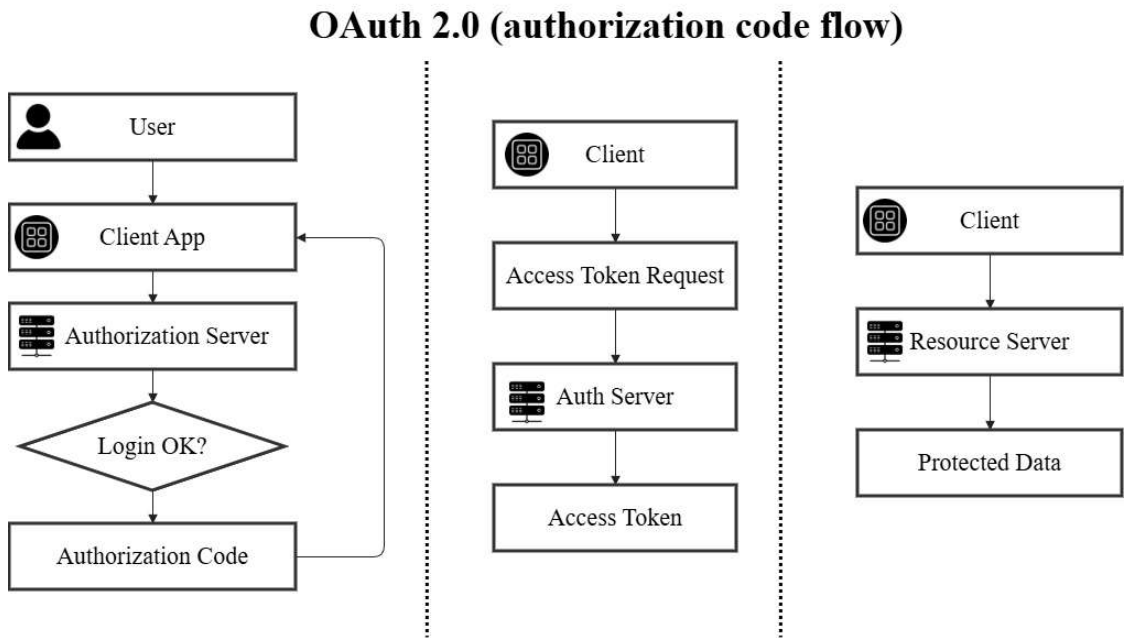


Рисунок 1.7 – Потік авторизації OAuth 2.0 (authorization code flow)

Використання OAuth 2.0 забезпечує гнучке управління правами доступу, підвищує безпеку за рахунок відсутності передачі паролів між сервісами та підтримує широкий спектр сценаріїв, включаючи веб-додатки, мобільні платформи та API, однак сам по собі цей протокол не призначений для автентифікації користувача, оскільки не відповідає на питання щодо його ідентичності, а лише визначає, які дії дозволено виконувати. Для розширення функціональності та реалізації повноцінної автентифікації використовується протокол OpenID Connect, який базується на OAuth 2.0 та додає механізми ідентифікації користувача через використання спеціальних токенів, що містять інформацію про особу [8].

Протокол OpenID Connect (OIDC). OpenID Connect є розширенням протоколу OAuth 2.0, яке доповнює його механізмами автентифікації користувачів і дозволяє не лише делегувати доступ до ресурсів, але й достовірно визначати особу користувача в межах інформаційної системи, у результаті чого формується повноцінна модель ідентифікації та авторизації, що широко застосовується у сучасних веб-додатках і вважається стандартом для реалізації єдиного входу. Архітектурно така модель базується на використанні ідентифікаційного токена, який передається після успішної автентифікації та містить необхідну інформацію про користувача, включаючи унікальний ідентифікатор, параметри сеансу та часові обмеження дії, при цьому сам токен формується у форматі JSON Web Token, що забезпечує компактність, цілісність і можливість криптографічної перевірки.

Процес взаємодії між компонентами системи передбачає, що користувач звертається до клієнтського додатку, після чого виконується перенаправлення до провайдера ідентифікації, де відбувається перевірка облікових даних, і після успішного проходження автентифікації формується відповідь, яка містить як ID токен, так і токен доступу, далі клієнтський додаток перевіряє отримані дані, валідуючи підпис і термін дії токена, після чого надає доступ до необхідних ресурсів без повторного введення облікових даних. Така модель забезпечує стандартизований підхід до автентифікації, дозволяє інтегрувати різні сервіси в єдину екосистему та підтримує широкий спектр сценаріїв, включаючи веб-додатки, мобільні застосунки та односторінкові інтерфейси [9].

Застосування OpenID Connect у практичних системах охоплює різноманітні варіанти використання, серед яких:

1. авторизація через сторонні сервіси;
2. інтеграція корпоративних порталів;
3. взаємодія між компонентами мікросервісних архітектур.

В останньому саме централізований провайдер ідентифікації забезпечує єдину точку входу для всіх ресурсів, що особливо актуально у випадках побудови захищених інфраструктур із великою кількістю розподілених сервісів [9].

Механізми SAML для корпоративного середовища. Поряд із сучасними протоколами, орієнтованими на веб-середовище, значне місце в корпоративних системах займає стандарт SAML, який використовується для обміну даними автентифікації та авторизації між різними доменами безпеки і базується на передачі структурованих повідомлень у форматі XML, що дозволяє реалізувати складні сценарії інтеграції між організаційними системами. Основна ідея такого підходу полягає у формуванні спеціального твердження про автентифікацію, яке створюється провайдером ідентифікації після перевірки користувача та передається сервісу, що надає доступ до ресурсів, у результаті чого останній приймає рішення про надання доступу на основі отриманої інформації без необхідності повторної перевірки облікових даних.

Типовий процес взаємодії передбачає, що користувач ініціює доступ до певного сервісу, після чого відбувається перенаправлення до провайдера ідентифікації, де виконується автентифікація, далі формується SAML-твердження, яке містить інформацію про користувача, його атрибути та факт успішного входу, після чого це твердження передається сервісу, який перевіряє його цілісність і достовірність за допомогою цифрового підпису та надає доступ до ресурсів. Використання цифрових підписів і стандартизованих форматів обміну даними забезпечує високий рівень безпеки та дозволяє інтегрувати систему з корпоративними каталогами, такими як Active Directory або LDAP, що робить цей підхід особливо актуальним для великих організацій (див. рис. 1.8) [10].

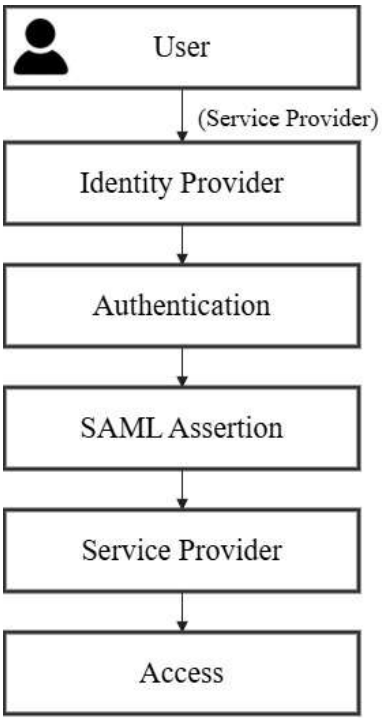


Рисунок 1.8 – Обмін SAML-assertion

Переваги SAML проявляються у можливості централізованого управління доступом, підтримці механізму єдиного входу та сумісності з корпоративними середовищами, однак складність реалізації, необхідність роботи з XML-структурами та менш зручна інтеграція з сучасними веб-технологіями обмежують його використання у нових системах, де перевага надається більш легким і гнучким рішенням. У результаті формується певний розподіл сфер застосування, де OAuth 2.0 забезпечує механізми делегування доступу, OpenID Connect доповнює його можливостями автентифікації та є базовим стандартом для сучасних веб-додатків, а SAML зберігає актуальність у корпоративному середовищі з високими вимогами до безпеки та інтеграції [10].

У контексті побудови систем багатофакторної автентифікації та централізованого контролю доступу використання цих протоколів дозволяє забезпечити узгоджену взаємодію між компонентами інфраструктури, реалізувати єдину точку входу до ресурсів та створити масштабовану архітектуру, здатну адаптуватися до різних сценаріїв використання, включаючи захист внутрішніх

сервісів, інтеграцію з зовнішніми системами та впровадження додаткових механізмів перевірки особи користувача.

1.4 Платформи управління ідентифікацією (на прикладі Authentik)

Платформи управління ідентифікацією, які реалізуються у вигляді Identity Providers, формують централізований механізм контролю доступу до ресурсів, забезпечуючи єдину точку перевірки користувачів та інтеграцію різних протоколів і методів автентифікації в межах однієї архітектури, що дозволяє уніфікувати політики безпеки та спростити адміністрування доступу до розподілених сервісів. Такі системи виступають посередником між користувачем і цільовими ресурсами, виконуючи:

1. перевірку облікових даних;
2. застосування багатофакторної автентифікації;
3. формування результату у вигляді токенів або сесій.

Саме останні надалі використовуються для доступу до сервісів без повторної перевірки, що створює основу для централізованої моделі безпеки.

Одним із сучасних прикладів таких платформ є Authentik, який являє собою гнучке рішення з відкритим кодом, орієнтоване на побудову систем автентифікації та управління доступом у різних середовищах, включаючи приватні серверні інфраструктури, де існує потреба у впровадженні єдиної політики доступу для великої кількості різнорідних сервісів, серед яких можуть бути як сучасні веб-додатки, так і застарілі системи без вбудованих механізмів захисту. Використання подібної платформи дозволяє:

1. централізувати процес автентифікації;
2. реалізувати багатофакторний захист;
3. інтегрувати різні сервіси.

У випалку останнього – незалежно від їхньої архітектури або способу розгортання.

Огляд архітектури сучасних Identity Providers. Архітектура сучасних Identity Providers будується за модульним принципом, що забезпечує гнучкість налаштування, масштабованість і можливість адаптації до різних сценаріїв використання, при цьому одним із основних компонентів є сервер автентифікації, який виконує перевірку облікових даних користувача та реалізує логіку автентифікаційних процесів, включаючи використання декількох факторів, обробку політик доступу та прийняття рішення щодо надання або відмови у доступі. Паралельно з цим функціонує підсистема зберігання даних, що містить інформацію про користувачів, їхні ролі, налаштування багатофакторної автентифікації, а також криптографічні ключі та інші службові дані, необхідні для коректної роботи системи.

Важливим елементом виступає механізм генерації та обробки токенів, який забезпечує створення сесійних ідентифікаторів або структурованих токенів доступу, що використовуються для взаємодії між компонентами системи, дозволяючи уникнути повторної автентифікації при кожному зверненні до ресурсу, а також інтерфейс управління, який надає можливість адміністраторам виконувати налаштування користувачів, політик доступу та інтеграцій із зовнішніми сервісами. У межах такої архітектури взаємодія компонентів реалізується таким чином, що при зверненні до захищеного ресурсу відбувається перенаправлення до провайдера ідентифікації, де виконується автентифікація користувача, після чого формується токен або сесія, і лише після цього надається доступ до цільового сервісу.

Сучасні платформи цього класу підтримують широкий набір стандартів і протоколів, серед яких OpenID Connect, OAuth 2.0 та SAML, що забезпечує сумісність із різними типами додатків і дозволяє інтегрувати як хмарні, так і локальні сервіси, включаючи середовища контейнеризації. У цьому контексті Authentik надає додаткові можливості, серед яких варто відзначити підтримку гнучких сценаріїв автентифікації у вигляді керованих потоків, що дозволяє адаптувати процес перевірки користувача під конкретні вимоги безпеки, а також реалізацію механізму Forward Authentication, який дозволяє впроваджувати захист навіть для тих сервісів, що не мають власної системи автентифікації [11].

Застосування такого підходу є особливо актуальним у приватних інфраструктурах, де часто використовуються різнорідні сервіси, включаючи застарілі рішення, доступ до яких необхідно обмежити без внесення змін у їхній вихідний код або архітектуру, у результаті чого централізований провайдер ідентифікації дозволяє накладати додатковий рівень захисту, забезпечуючи перевірку користувача перед доступом до ресурсу. Інтеграція з контейнерними середовищами, зокрема з використанням Docker, дозволяє розгортати такі системи як ізольовані сервіси, що спрощує їх масштабування, оновлення та управління, а також забезпечує гнучкість у побудові інфраструктури.

Концепція Forward Authentication для захисту legacy-сервісів. Однією з найбільш проблемних особливостей сучасних IT-інфраструктур залишається наявність застарілих сервісів, які не підтримують сучасні протоколи автентифікації, не мають вбудованих механізмів багатофакторного захисту та часто не підлягають модернізації через обмеження архітектури або пропрієтарний характер реалізації, у результаті чого виникає необхідність у впровадженні зовнішніх механізмів контролю доступу, здатних забезпечити належний рівень безпеки без внесення змін у самі сервіси. У такому контексті застосовується концепція Forward Authentication, яка базується на перехопленні запитів користувача до ресурсу за допомогою проміжного компонента, що виконує функцію проксі та здійснює перевірку автентифікації перед передачею запиту до цільового сервісу.

У межах цієї моделі користувач спочатку звертається до ресурсу, після чого запит обробляється реверс-проксі, який перед виконанням основного запиту звертається до зовнішнього провайдера ідентифікації для перевірки доступу, у разі відсутності автентифікації виконується перенаправлення на сторінку входу, де здійснюється перевірка облікових даних із можливим використанням багатофакторної автентифікації, після чого формується сесія або видається токен доступу, і лише після цього запит передається до цільового сервісу, що дозволяє реалізувати додатковий рівень захисту без змін у самому ресурсі. Такий підхід забезпечує централізацію контролю доступу, дозволяє застосовувати єдині

політики безпеки для різних сервісів та розширює можливості інтеграції навіть для тих систем, які спочатку не передбачали механізмів автентифікації (див. рис. 1.9) [12].

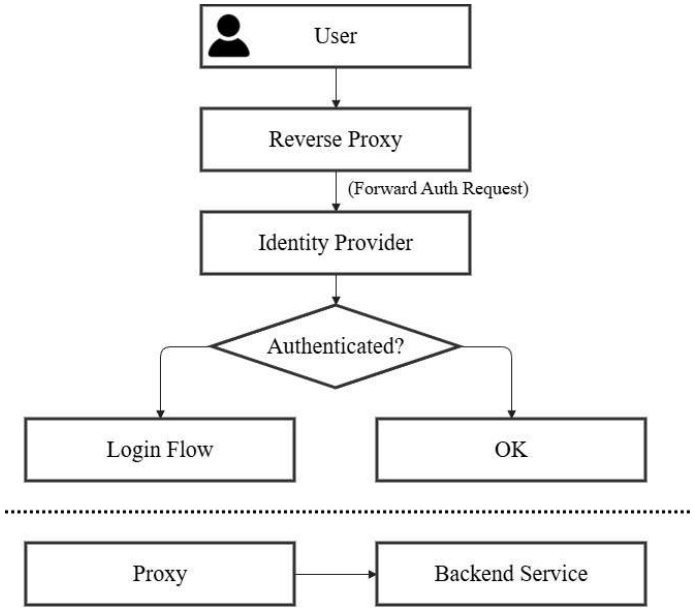


Рисунок 1.9 – Принцип роботи Forward Authentication

Практична реалізація цієї концепції передбачає використання провайдера ідентифікації як центрального елемента, який відповідає за перевірку користувача, та реверс-проксі, який виконує маршрутизацію запитів і забезпечує взаємодію між клієнтом і сервісами, при цьому у ролі проксі можуть використовуватися рішення на кшталт Nginx або Traefik, що дозволяють інтегрувати механізм Forward Authentication у різні середовища без суттєвих змін у структурі інфраструктури. У результаті формується модель, у якій будь-який ресурс, включаючи веб-сайти, адміністративні інтерфейси мережевого обладнання або API, може бути захищений шляхом додавання зовнішнього рівня автентифікації, що мінімізує ризики несанкціонованого доступу навіть у випадках, коли сам сервіс не має вбудованих засобів захисту (див. рис. 1.10) [12].

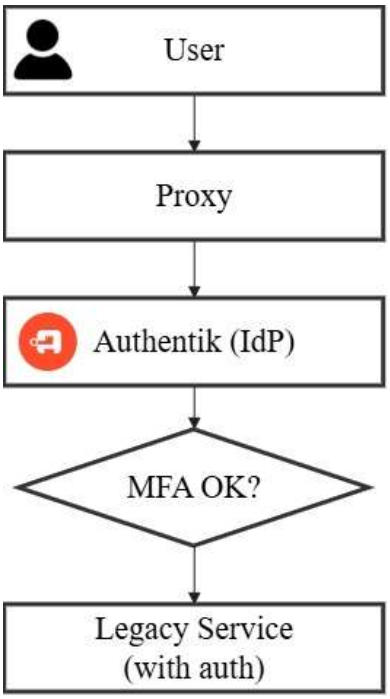


Рисунок 1.10 – Схема захисту legacy-сервісу через Forward Auth

Значення такого підходу полягає у можливості впровадження багатофакторної автентифікації для будь-якого ресурсу незалежно від його технічних обмежень, що дозволяє створити єдину точку контролю доступу та забезпечити узгоджену політику безпеки в межах усієї інфраструктури, при цьому навіть у випадку компрометації одного з факторів доступ до системи залишається обмеженим за рахунок додаткових перевірок.

Інтеграція провайдерів з контейнерними середовищами. Паралельно з розвитком підходів до автентифікації відбувається активне впровадження технологій контейнеризації, які змінюють принципи розгортання та управління сервісами, включаючи системи ідентифікації, оскільки використання контейнерів дозволяє створювати ізольовані середовища виконання, спрощувати процес інсталяції та забезпечувати гнучкість у масштабуванні компонентів. У такому середовищі платформи управління ідентифікацією можуть розгортатися у вигляді набору взаємопов’язаних контейнерів, що включають сервер автентифікації, компоненти обробки задач, базу даних і кеш-систему, які разом формують єдину систему.

Інтеграція з контейнерними середовищами дозволяє організувати взаємодію між сервісами через внутрішні мережі, використовуючи DNS-імена контейнерів і стандартизовані протоколи обміну даними, у результаті чого система автентифікації може функціонувати як частина загальної інфраструктури або як окремий сервіс, що обслуговує зовнішні ресурси. У такій моделі реверс-проксі також інтегрується у контейнерне середовище, приймає запити користувачів, передає їх до провайдера ідентифікації для перевірки та здійснює маршрутизацію до відповідних сервісів, забезпечуючи централізований контроль доступу (див. рис. 1.11) [13].

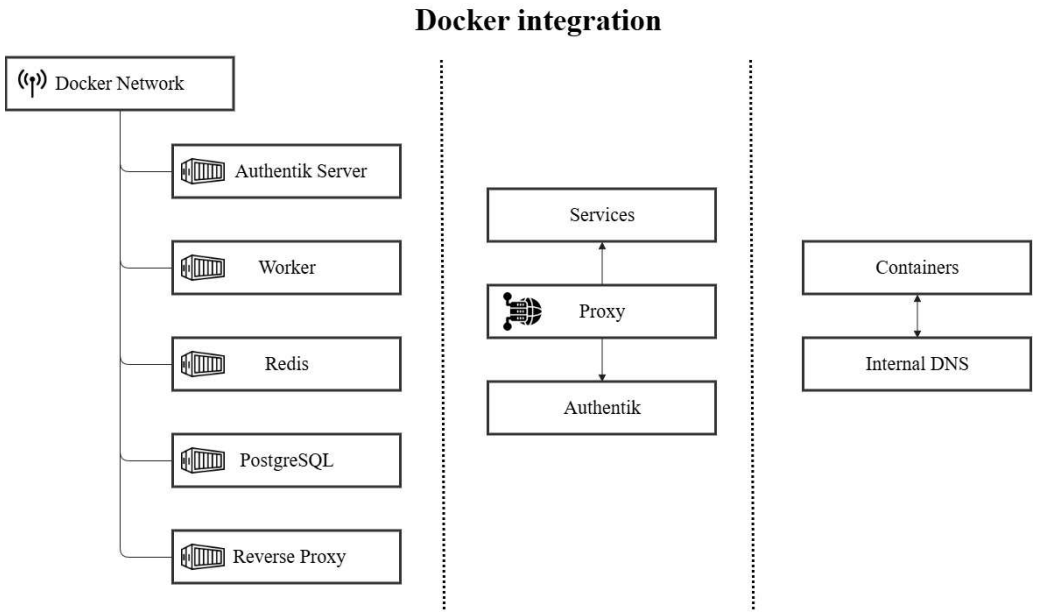


Рисунок 1.11 – Інтеграція IdP у Docker-середовище

Використання контейнеризації надає низку переваг, серед яких варто відзначити гнучкість конфігурації, можливість швидкого розгортання, спрощення тестування та перенесення системи між різними середовищами, що дозволяє реалізувати ідентичну інфраструктуру як на локальних пристроях, так і на серверних платформах без зміни логіки роботи. Такий підхід забезпечує універсальність рішення, у межах якого система автентифікації може бути розгорнута на будь-якому пристрої, взаємодіяти з внутрішніми або зовнішніми

сервісами та забезпечувати захист доступу за допомогою багатофакторної автентифікації [13].

У підсумку поєднання концепції Forward Authentication із можливостями контейнеризації формує ефективну модель захисту, яка дозволяє забезпечити контроль доступу до будь-яких ресурсів, включаючи застарілі системи, без необхідності їх модифікації, при цьому централізована система управління ідентифікацією забезпечує узгодженість політик безпеки, масштабованість і адаптивність до сучасних вимог, що робить таке рішення придатним для використання у приватних серверних інфраструктурах різного рівня складності.

1.5 Безпека приватної серверної інфраструктури

Безпека приватної серверної інфраструктури формується як необхідна умова стабільного функціонування будь-якого середовища, у якому відбувається обробка даних, керування ресурсами та взаємодія між сервісами, причому навіть найпростіша інфраструктура, що складається з одного маршрутизатора та кількох внутрішніх сервісів, уже містить потенційні вразливості через обмежені можливості вбудованих механізмів захисту та відсутність сучасних засобів автентифікації, унаслідок чого виникає потреба в застосуванні додаткових підходів до забезпечення безпеки, які виходять за межі стандартних налаштувань обладнання та базових політик доступу.

Приватні інфраструктури часто стають об’єктом атак не через свою масштабність, а через недооцінку ризиків, пов’язаних із внутрішніми ресурсами, які традиційно вважаються довіреними, однак на практиці можуть бути скомпрометовані як через зовнішні канали доступу, так і через внутрішні загрози, включаючи некоректну конфігурацію, використання застарілого програмного забезпечення або слабкі механізми автентифікації, у результаті чого забезпечення безпеки потребує комплексного підходу, що охоплює контроль точок доступу,

управління сесіями користувачів, а також постійний моніторинг подій і аналіз активності.

Захист адміністративних інтерфейсів мережевого обладнання. Адміністративні інтерфейси мережевого обладнання виступають одними з найбільш критичних елементів інфраструктури, оскільки доступ до них фактично означає повний контроль над мережею, включаючи:

1. можливість зміни конфігурації;
2. перехоплення трафіку;
3. повного блокування роботи сервісів.

Тобто значна кількість таких інтерфейсів залишається недостатньо захищеною через використання стандартних облікових даних, відсутність обмежень доступу або відкритий доступ із зовнішніх мереж.

Основні загрози в цьому контексті включають атаки перебору паролів, використання злитих облікових даних, експлуатацію вразливостей веб-інтерфейсів обладнання, а також несанкціонований доступ через відкриті порти, що робить необхідним впровадження обмежень доступу на мережевому рівні, зокрема шляхом дозволу підключення лише з довірених IP-адрес або використання захищених каналів зв’язку, таких як VPN, що дозволяє ізолювати адміністративні інтерфейси від прямого доступу з інтернету [14].

Додатковим рівнем захисту виступає сегментація мережі, яка дозволяє відокремити критичні компоненти інфраструктури від загальнодоступних ресурсів, обмежуючи можливість горизонтального переміщення зловмисника в разі компрометації одного з вузлів, при цьому ефективність такого підходу значно зростає у поєднанні з централізованими механізмами автентифікації, що забезпечують єдину політику доступу для всіх компонентів системи.

Важливим аспектом є впровадження багатофакторної автентифікації навіть для тих систем, які не підтримують її на рівні вбудованого функціоналу, що досягається шляхом використання зовнішніх сервісів автентифікації та проксі-рівня, який перехоплює запити до ресурсу та виконує перевірку доступу перед передачею запиту до цільового сервісу, у результаті чого доступ стає можливим

лише після успішного проходження всіх етапів перевірки, включаючи введення пароля та підтвердження другого фактору [14].

Використання реверс-проксі в цьому процесі дозволяє створити єдину точку входу до адміністративних інтерфейсів, через яку проходить увесь трафік, що забезпечує централізований контроль доступу та можливість інтеграції додаткових механізмів захисту без необхідності змінювати самі сервіси або мережеве обладнання, унаслідок чого навіть застарілі системи отримують додатковий рівень безпеки.

Управління сесіями та токенами доступу становить ще один критичний аспект безпеки, оскільки після успішної автентифікації система повинна забезпечити контроль подальшої взаємодії користувача з ресурсом, використовуючи сесійні ідентифікатори або токени, які підтверджують автентичність запитів, при цьому важливо враховувати обмеження часу дії таких токенів, їх захищене зберігання та механізми відкликання у разі підозрілої активності [14].

Некоректна реалізація сесій може призвести до атак типу перехоплення сесії або повторного використання токенів, що робить необхідним застосування захищених протоколів передачі даних, регулярне оновлення токенів і використання додаткових параметрів перевірки, таких як прив’язка до IP-адреси або пристрою, що ускладнює несанкціоноване використання облікових даних.

Моніторинг подій безпеки та аудит входів дозволяє своєчасно виявляти підозрілу активність і реагувати на потенційні загрози, забезпечуючи збереження інформації про всі спроби доступу, зміни конфігурації та інші критичні події, що відбуваються в системі, при цьому аналіз журналів дозволяє виявляти аномалії, такі як часті невдалі спроби входу, доступ із незвичних локацій або нетипова поведінка користувачів [14].

Ефективний аудит передбачає не лише збереження даних, а й їхню обробку з використанням систем аналізу, що дозволяє автоматизувати виявлення загроз і мінімізувати час реагування на інциденти, у результаті чого підвищується

загальний рівень захищеності інфраструктури та зменшується ймовірність успішної атаки.

Управління сесіями та токенами доступу. Управління сесіями та токенами доступу формується як невід'ємна частина механізмів контролю доступу після успішного проходження автентифікації, оскільки саме на цьому етапі визначається подальша взаємодія користувача із системою без необхідності повторного введення облікових даних при кожному запиті, унаслідок чого створюється сесія як тимчасовий контекст взаємодії, що ідентифікує користувача та зберігає інформацію про стан його автентифікації протягом певного періоду часу.

Сесія у цьому контексті являє собою механізм, який дозволяє системі розпізнавати користувача між окремими запитами, забезпечуючи безперервність роботи, водночас токени доступу виступають інструментом підтвердження прав користувача, причому існує декілька типів таких токенів, серед яких session cookie, що зберігаються у браузері та використовуються у веб-додатках, а також JSON Web Token, які містять структуровану інформацію про користувача, підписуються криптографічно та широко застосовуються у взаємодії з API [15].

Окрему увагу приділено "access token", який використовується для безпосереднього доступу до ресурсів, та "refresh token", призначеному для оновлення "access token" без повторної автентифікації, що дозволяє зберігати баланс між безпекою та зручністю використання, однак наявність таких механізмів супроводжується низкою ризиків, серед яких [15]:

1. перехоплення сесії;
2. повторне використання токенів;
3. надмірно тривалий час їхньої дії;
4. відсутність прив'язки до конкретного пристрою/мережевого контексту.

Загроза викрадення сесії може реалізовуватися через атаки типу "session hijacking", коли зловмисник отримує ідентифікатор сесії та використовує його для несанкціонованого доступу, унаслідок чого виникає потреба в обмеженні часу життя сесій шляхом встановлення короткого значення часу дії та автоматичного

завершення сесії у разі тривалої неактивності, що значно знижує ризик використання викрадених даних [15].

Додатковим рівнем захисту виступає використання захищених cookie з атрибутами Secure та HttpOnly, які обмежують доступ до них з боку клієнтських скриптів і забезпечують передачу виключно через захищені канали зв'язку, що ускладнює перехоплення або модифікацію даних під час передачі, при цьому доцільним є застосування прив'язки сесії до контексту, включаючи IP-адресу, User-Agent або інші параметри, які дозволяють виявляти аномальні зміни в поведінці користувача [15].

Ротація токенів забезпечує регулярне оновлення ключових елементів автентифікації, унаслідок чого старі токени втрачають свою актуальність і не можуть бути використані повторно, що підвищує стійкість системи до атак, пов'язаних із витоком даних, при цьому інвалідація токенів дозволяє негайно припинити доступ у разі виявлення підозрілої активності або завершення сесії.

Додатковий рівень довіри до сесії досягається шляхом інтеграції багатофакторної автентифікації, яка може вимагати повторного підтвердження особи користувача при виконанні критичних дій або при зміні контексту доступу, що дозволяє реалізувати адаптивні механізми безпеки, орієнтовані на оцінку ризиків у реальному часі, у результаті чого навіть у разі компрометації первинних облікових даних доступ до системи залишається обмеженим (див. рис. 1.12).

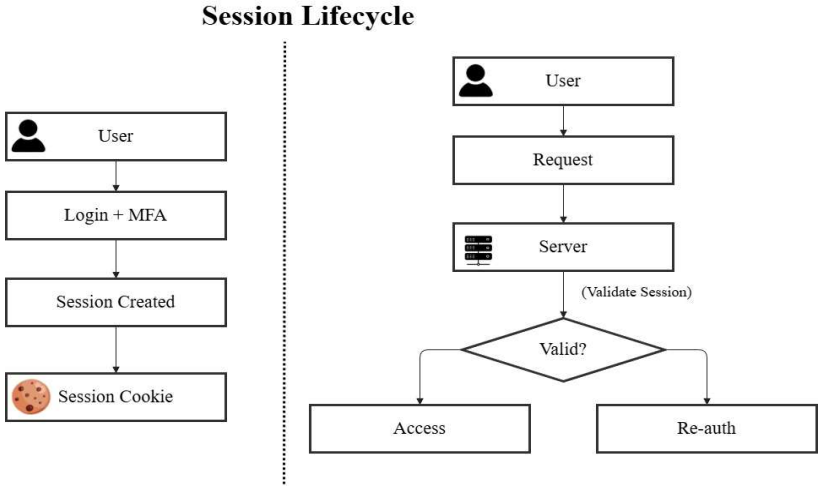


Рисунок 1.12 – Життєвий цикл сесії користувача

У практичній реалізації після проходження багатофакторної перевірки формується сесійний ідентифікатор або cookie, який перевіряється при кожному запиті до захищеного ресурсу, причому доступ надається лише за умови валідності цієї сесії, а будь-яке відхилення від очікуваного стану призводить до необхідності повторної автентифікації, що створює безперервний контроль доступу на всіх етапах взаємодії.

Моніторинг подій безпеки та аудит входів. Моніторинг подій безпеки та аудит входів доповнює механізми управління сесіями, забезпечуючи можливість відстеження дій користувачів і виявлення потенційних загроз, оскільки фіксація подій, таких як успішні та невдалі спроби входу, використання багатофакторної автентифікації, зміни налаштувань або доступ до критичних ресурсів, дозволяє формувати повну картину активності в системі.

Журнали подій містять інформацію про час входу, IP-адресу, тип пристрою та інші параметри, що використовуються для подальшого аналізу інцидентів і проведення аудиту безпеки, унаслідок чого стає можливим виявлення підозрілих дій, таких як масові спроби входу або доступ із нетипових локацій, що може свідчити про спробу компрометації облікового запису [16].

Аналіз подій може здійснюватися як після виникнення інциденту, так і в проактивному режимі, спрямованому на попередження атак, використовуючи методи виявлення аномалій, кореляції подій або інтеграцію із спеціалізованими системами аналізу, що дозволяє автоматизувати процес реагування та мінімізувати вплив потенційних загроз.

Аудит доступу передбачає [16]:

1. Регулярну перевірку прав користувачів.
2. Аналіз історії входів.
3. Виявлення порушень політик безпеки.

Все це забезпечує відповідність встановленим вимогам і дозволяє своєчасно виявляти слабкі місця в системі, при цьому інтеграція з системами логування та моніторингу розширює можливості контролю та підвищує рівень прозорості всієї інфраструктури.

2. ПОБУДОВА СИСТЕМИ АВТЕНТИФІКАЦІЇ НА БАЗІ FEDORA LINUX ТА AUTHENTIK

2.1 Архітектура та вибір компонентів

У контексті побудови прототипу системи багатофакторної автентифікації, реалізованої у нативному Unix-середовищі без використання віртуалізації, де основна увага зосереджується на формуванні максимально наближених до реальних умов функціонування середовища, здатного забезпечити прямий доступ до ресурсів операційної системи Linux і водночас виключити вплив додаткових накладних витрат, характерних для гіпервізорів на кшталт VirtualBox або підсистем типу WSL, що потенційно здатні спотворювати результати досліджень продуктивності та мережевої взаємодії, особливо в контексті аналізу поведінки контейнеризованих сервісів під навантаженням. Обрана архітектура базується на мікросервісному підході із використанням контейнеризації Docker, де кожен компонент виконує чітко визначену функцію, що відповідає сучасним принципам побудови систем контролю доступу та забезпечує гнучкість, масштабованість і ізольованість окремих функціональних модулів.

У рамках цієї архітектури основними компонентами виступають Authentik Server як центральний елемент керування та адміністративного інтерфейсу, Authentik Worker як виконавець фонових задач, PostgreSQL як постійне сховище даних, що забезпечує збереження конфігурацій, політик доступу та облікових записів, а також Authentik Outpost, який виконує функцію проксі-рівня для перехоплення та обробки запитів користувачів. Додатково інтегровано тестовий сервіс "whoami", який використовується як цільовий ресурс для перевірки коректності роботи механізмів автентифікації, дозволяючи наочно відстежувати передані заголовки запитів, ідентифікаційні дані та результати доступу. Такий підхід дозволяє логічно розділити відповідальність між компонентами,

забезпечуючи незалежність їх функціонування, що є типовою характеристикою сучасних мікросервісних систем, а також створює передумови для подальшого масштабування та інтеграції в складні інфраструктурні рішення (див. рис. 2.1).

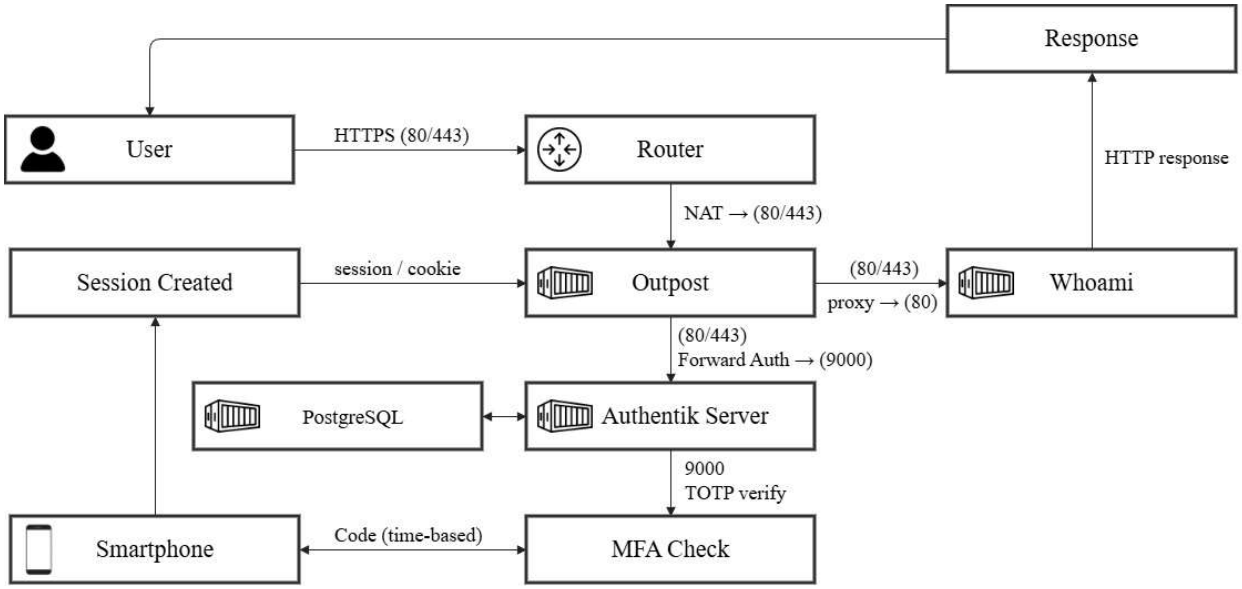


Рисунок 2.1 – Архітектура системи MFA на базі Authentik

Апаратне середовище. У якості апаратної платформи обрано ноутбук Lenovo ThinkPad X1 Yoga Gen 8, який функціонує під керуванням операційної системи Fedora Workstation 43, що належить до сучасних дистрибутивів сімейства Linux і забезпечує актуальне ядро, оптимізоване для роботи з контейнерними технологіями, де відсутність додаткових шарів віртуалізації дозволяє отримати достовірні результати використання ресурсів системи, включаючи оперативну пам’ять, процесорний час та дискові операції. Такий підхід дозволяє уникнути спотворення показників продуктивності, що могло б виникати у випадку використання віртуальних машин або проміжних середовищ, де мережеві та обчислювальні ресурси проходять через додаткові рівні абстракції [17].

Апаратна конфігурація ThinkPad X1 Yoga Gen 8 включає багатоядерний процесор сімейства Intel Core i5-1335U, що забезпечує достатній рівень паралелізму для обробки контейнеризованих сервісів, оперативну пам’ять обсягом 16 ГБ, яка дозволяє одночасно виконувати кілька контейнерів без критичного

впливу на стабільність системи, а також швидкісний SSD-накопичувач обсягом до 2 ТБ, що забезпечує низькі затримки при роботі з базою даних PostgreSQL та файловою системою контейнерів. Така конфігурація створює умови для моделювання поведінки невеликого серверного вузла, що є важливим аспектом при дослідженні систем автентифікації, орієнтованих на подальше впровадження в інфраструктурі типу Edge Gateway (див. рис. 2.2).



Рисунок 2.2 – Апаратне середовище на базі Lenovo ThinkPad X1 Yoga Gen 8

Важливим аспектом є також нативна підтримка Docker Engine у Fedora, яка базується на можливостях ядра Linux, включаючи "namespaces" та "cgroups", що дозволяє ефективно ізолювати процеси контейнерів та керувати їх ресурсами, формуючи середовище, максимально наближене до серверного розгортання без необхідності використання додаткових платформ віртуалізації.

Базова топологія. Базова топологія системи реалізована в межах ізольованої віртуальної мережі Docker з умовною назвою "authentik_default", у якій усі компоненти взаємодіють між собою через внутрішні DNS-імена, що дозволяє забезпечити логічну ізоляцію від зовнішнього середовища та водночас спростити адресацію сервісів без жорсткої прив’язки до IP-адрес. Така модель організації мережі формує основу для стабільної взаємодії компонентів, де кожен контейнер ідентифікується за своїм іменем, наприклад "server", "whoami" або "postgresql", а маршрутизація запитів відбувається на рівні внутрішнього DNS Docker, що забезпечує гнучкість при зміні мережевих параметрів хост-системи.

У структурі системи виділяється кілька ключових компонентів, взаємодія між якими формує цілісну архітектуру. Authentik Server функціонує як центральний елемент керування, який надає адміністративний інтерфейс та реалізує логіку автентифікації і політик доступу, у той час як Authentik Worker виконує обробку фонових задач, пов’язаних із перевіркою запитів, обробкою подій та іншими асинхронними процесами. Постійне зберігання даних забезпечується за рахунок використання PostgreSQL, що працює на стандартному порту 5432 та виконує роль персистентного сховища, у якому зберігаються облікові записи користувачів, конфігурації системи, політики доступу та частково сесійні дані, що є критично важливим для збереження стану системи після перезапуску контейнерів.

Окремим компонентом виступає Outpost, який працює як проксі-рівень і є точкою входу для користувача, приймаючи HTTP-запити на порту 8080, проброшеному на хост-систему, та виконуючи перевірку автентифікації перед передачею запиту до цільового сервісу. Додатково інтегровано тестовий сервіс whoami, який використовується для демонстрації роботи системи багатофакторної автентифікації, функціонуючи як внутрішній ресурс у Docker-мережі без прямого доступу ззовні, що дозволяє перевірити механізм захисту сервісів, які не мають власної реалізації автентифікації.

Зовнішня взаємодія з системою здійснюється через IP-адресу хост-системи, наприклад "192.168.0.21:8080", де користувач формує запит до Outpost, після чого

відбувається внутрішня обробка в межах Docker-мережі. Outpost виконує перевірку наявності сесії шляхом звернення до Authentik Server за внутрішнім DNS-іменем "server:9000", де здійснюється аналіз валідності сесійних даних та політик доступу із залученням PostgreSQL як джерела постійного зберігання. У разі успішної автентифікації запит перенаправляється до цільового сервісу через внутрішній виклик whoami:80, що повністю ізолює цей сервіс від прямого зовнішнього доступу та забезпечує його захист через проксі-рівень.

Особливістю такої топології є використання механізму Host Port Mapping, де окремі порти контейнерів проброшуються на хост для забезпечення доступу з локальної мережі, зокрема порт 9000 для адміністративного інтерфейсу, 8080 для проксі-рівня та 8081 або внутрішній порт 80 для тестового сервісу, який у даній конфігурації доступний лише через Outpost. Така організація дозволяє чітко розмежувати зовнішній та внутрішній трафік, де сервіси, що не мають проброшених портів, залишаються недоступними напряму, що підвищує загальний рівень безпеки системи.

Важливою характеристикою є також незалежність внутрішньої адресації від IP-адреси хост-системи, оскільки використання DNS-імен контейнерів дозволяє змінювати мережеві параметри без необхідності внесення змін у конфігурацію сервісів, що є суттєвою перевагою при подальшому масштабуванні або міграції системи. PostgreSQL у цій архітектурі забезпечує сталість даних, завдяки чому всі налаштування, облікові записи та політики доступу зберігаються незалежно від життєвого циклу контейнерів, формуючи основу для надійного функціонування системи автентифікації (див. рис. 2.3).

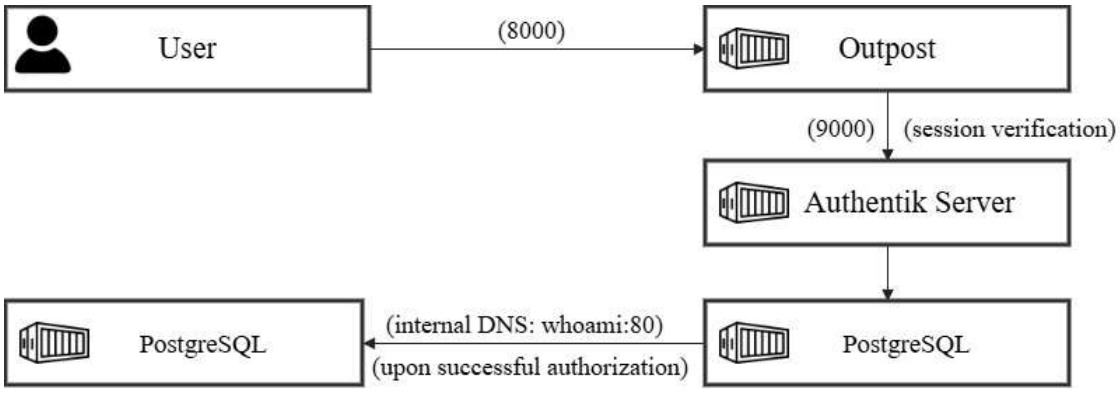


Рисунок 2.3 – Базова топологія мережі з MFA

Принцип Forward Auth. Механізм Forward Auth є фундаментальним елементом архітектури, що реалізує централізований контроль доступу до веб-сервісів шляхом винесення логіки автентифікації на окремий проксі-рівень, де Outpost виступає посередником між користувачем та цільовим ресурсом і здійснює перевірку кожного HTTP-запиту перед його передачею далі. У такій моделі будь-який запит спочатку потрапляє до Outpost, який аналізує наявність сесійного cookie та визначає, чи є користувач вже автентифікованим і чи має відповідні права доступу.

У випадку, коли валідна сесія відсутня або втратила чинність, виконується перенаправлення користувача на сторінку автентифікації, де відбувається введення облікових даних та подальша перевірка другого фактора, після чого формується нова сесія і користувачу видається відповідний cookie. Після успішного проходження автентифікації Outpost отримує підтвердження від Authentik Server через внутрішній запит до "server:9000" та починає пропускати запити користувача до цільового сервісу без додаткових перевірок до завершення сесії.

Логіка роботи механізму може бути представлена у вигляді послідовного алгоритму, де першим етапом є отримання запиту від користувача на адресу типу "192.168.0.21:8080", після чого виконується перевірка сесії через внутрішній сервіс автентифікації, далі за умови наявності валідної сесії відбувається перенаправлення запиту до внутрішнього ресурсу "whoami:80", а у випадку

відсутності сесії здійснюється редирект на сторінку логіну з подальшим проходженням повного циклу автентифікації (див. рис. 2.4).

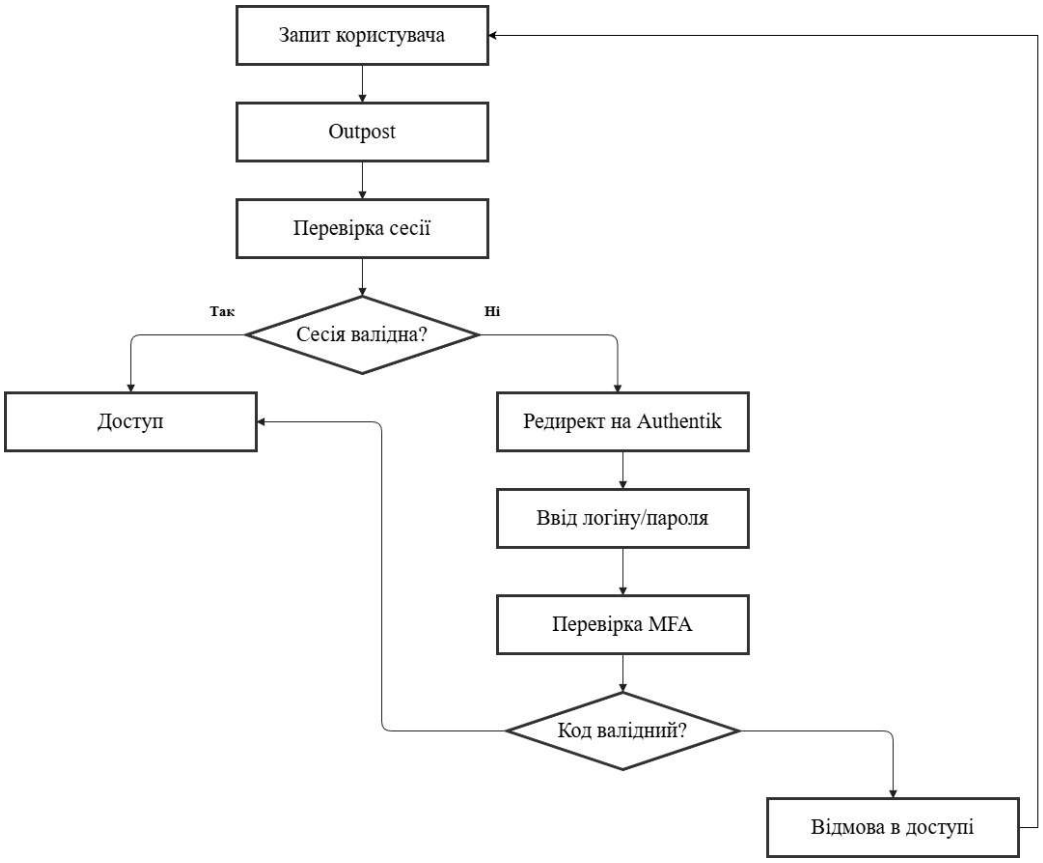


Рисунок 2.4 – Концепція Forward Auth у системі

Перевага такого підходу полягає у централізації механізму безпеки, де всі правила доступу, перевірка облікових даних та управління сесіями зосереджені в одному компоненті, що спрощує адміністрування та підвищує контрольованість системи, одночасно забезпечуючи прозорість для кінцевих сервісів, які не потребують власної реалізації автентифікації. Гнучкість цієї моделі дозволяє інтегрувати практично будь-який HTTP-сервіс у захищене середовище без змін у його внутрішній логіці, що створює основу для побудови масштабованих інфраструктурних рішень, орієнтованих на концепцію централізованого контролю доступу та подальшого переходу до архітектури мережевих шлюзів.

2.2 Реалізація та конфігурація системи

Практична реалізація прототипу системи багатофакторної автентифікації базується на поєднанні нативного середовища Fedora Linux та контейнеризованої архітектури, що дозволяє відтворити ізольовану, але максимально наближену до реальних умов інфраструктуру. У межах даного підходу кожен функціональний компонент системи розгортається у вигляді окремого контейнера, що забезпечує логічне розділення сервісів, контроль залежностей та спрощення подальшого масштабування. Важливим аспектом реалізації виступає свідоме виключення механізму кешування на базі Redis на етапі прототипування, оскільки така конфігурація дозволяє дослідити поведінку системи за умов відсутності зовнішнього кешу та оцінити вплив цього фактора на стабільність і використання ресурсів.

Розгортання Docker-стеку. Розгортання системи виконується із використанням технології контейнеризації Docker та інструменту оркестрації Docker Compose, що забезпечує декларативний підхід до опису всієї інфраструктури у вигляді конфігураційного файлу. Підготовка середовища передбачає наявність встановленого Docker Engine та Docker Compose у середовищі Fedora Workstation 43, що функціонує на апаратній платформі Lenovo ThinkPad X1 Yoga Gen 8, завдяки чому забезпечується повна сумісність із можливостями ядра Linux та коректна робота контейнерів без додаткових шарів віртуалізації [18].

Ключовим елементом конфігурації виступає файл змінних середовища .env, у якому зосереджені критично важливі параметри безпеки та налаштувань системи. До таких параметрів належать криптографічний ключ "AUTHENTIK_SECRET_KEY", що використовується для підпису токенів і забезпечення цілісності сесій, а також облікові дані для доступу до бази даних PostgreSQL, включаючи пароль користувача. Генерація секретного ключа здійснюється за допомогою криптографічно стійких генераторів випадкових чисел,

наприклад через утиліти типу OpenSSL, що виключає можливість його передбачення або підбору.

Після формування конфігураційних файлів процес запуску системи виконується командою "docker compose up -d", яка ініціалізує створення ізольованої мережі "authentik_default", розгортає всі визначені контейнери та запускає їх у фоновому режимі. У результаті формується повноцінний стек, що включає сервер Authentik, воркер для обробки фонових задач, базу даних PostgreSQL та проксі-компонент Outpost. Додатково інтегрується тестовий сервіс whoami, який використовується для перевірки коректності роботи механізмів автентифікації (див. табл. 2.1).

Таблиця 2.1 – Конфігурація Docker-стеку та сервісів Authentik

Компонент	Порт	Основні змінні	Призначення
Authentik Server	9000	AUTHENTIK_SECRET_KEY	Центральний сервер автентифікації
Authentik Worker	—	DB, SECRET_KEY	Обробка фонових задач
PostgreSQL	5432	POSTGRES_USER, PASSWORD	База даних
Outpost (Proxy)	80/443	AUTHENTIK_HOST	Forward Auth
Whoami	80	—	Тестовий сервіс

Свідоме виключення Redis із конфігурації дозволяє спростити початкову архітектуру, а також створює умови для дослідження навантаження на оперативну пам’ять, оскільки всі тимчасові стани та сесії обробляються безпосередньо в межах Python-процесів контейнерів Authentik, що у подальшому дає можливість виявити потенційні проблеми продуктивності (див. рис. 2.5).

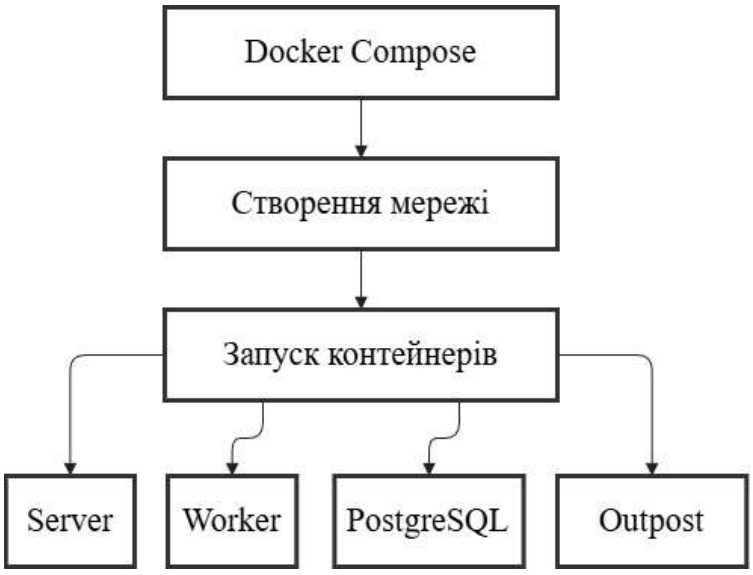


Рисунок 2.5 – Послідовність розгортання Docker-стеку Authentik

Конфігурація компонентів Authentik. Після запуску контейнерів формується етап логічної конфігурації компонентів системи, у межах якого визначається функціональна роль кожного сервісу та їх взаємодія між собою в межах єдиної архітектури. Центральним елементом виступає Authentik Server, що надає веб-інтерфейс адміністратора через порт 9000 та забезпечує керування користувачами, політиками доступу і сценаріями автентифікації, водночас обробка HTTP-запитів, пов’язаних із процесом ідентифікації, відбувається саме на його рівні, що дозволяє централізувати логіку контролю доступу (див. рис. 2.6) [19].

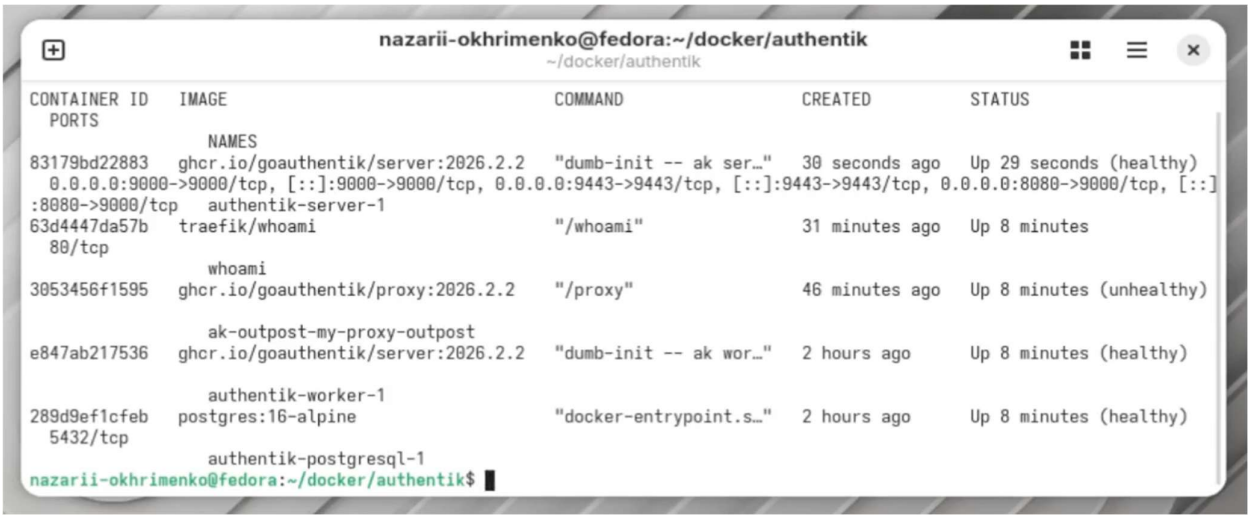


Рисунок 2.6 – Вивід терміналу при розгортанні Docker Compose

Паралельно функціонує Authentik Worker, який виконує фонові задачі, пов’язані з обробкою автентифікаційних подій, генерацією токенів, виконанням політик та обробкою черг повідомлень, причому асинхронна природа цього компонента дозволяє розвантажити основний сервер і забезпечити більш стабільну обробку великої кількості запитів, що особливо важливо при масштабуванні системи або збільшенні навантаження [19].

У межах конфігурації також використовується PostgreSQL як постійне сховище даних, де зберігаються облікові записи користувачів, налаштування багатофакторної автентифікації, політики доступу та частина інформації про сесії, завдяки чому забезпечується збереження стану системи між перезапусками контейнерів, а також цілісність усієї конфігурації, оскільки відсутність такого сховища призвела б до втрати всіх налаштувань і фактичної непридатності системи для практичного використання [19].

Взаємодія компонентів організована таким чином, що користувацький запит спочатку надходить до проксі-рівня Outpost, після чого передається на Authentik Server для перевірки автентифікаційних даних, а у разі необхідності обробки додаткових операцій залучається Worker, який виконує асинхронні задачі, при цьому всі критичні дані зберігаються у PostgreSQL, що формує цілісну та узгоджену систему керування доступом, побудовану за принципами розподіленої мікросервісної архітектури.

Створення MFA-потоку (TOTP). Реалізація багатофакторної автентифікації у системі базується на використанні механізму одноразових паролів, що генеруються на основі часу, який інтегрується у загальний процес автентифікації через налаштування відповідного Authentication Flow (див. рис. 2.7).

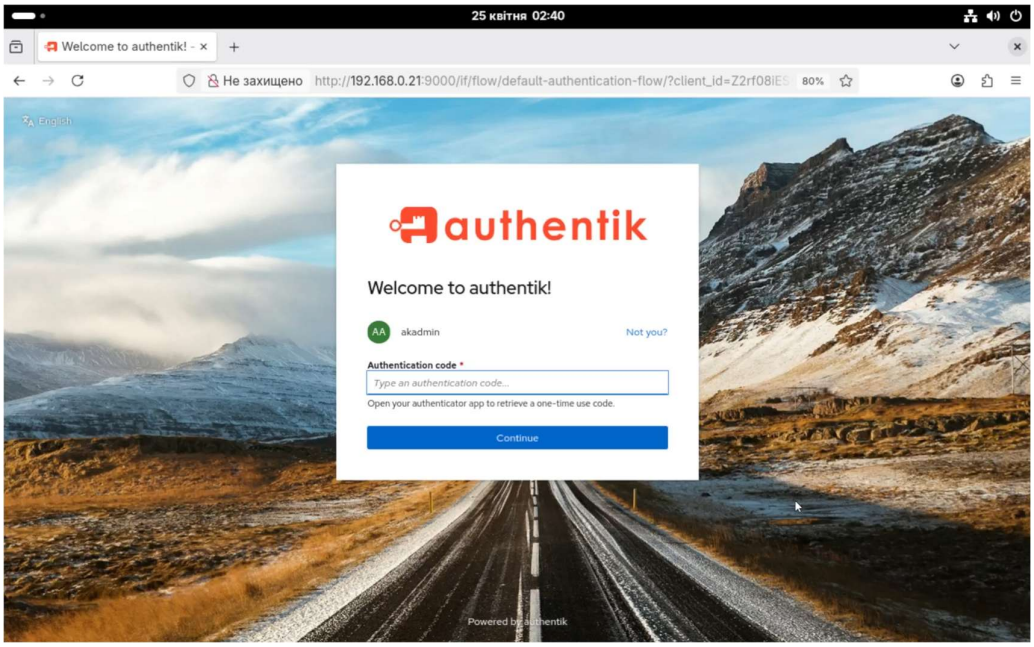


Рисунок 2.7 – Веб-інтерфейс Authentik (панель адміністратора)

У межах цієї моделі визначається послідовність дій, яку повинен виконати користувач для отримання доступу, де перевірка TOTP виступає як додатковий етап після введення основних облікових даних (див. рис. 2.8).

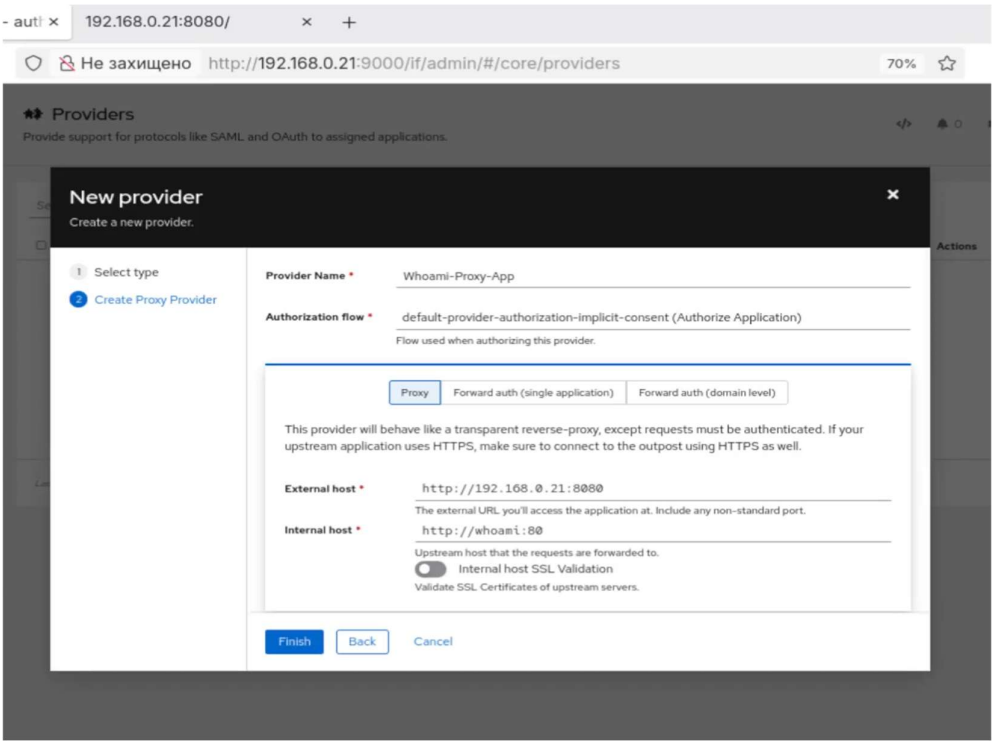


Рисунок 2.8 – Налаштування провайдера автентифікації

Процес починається з реєстрації пристрою автентифікації, під час якої користувач після входу до системи отримує згенерований QR-код, що містить секретний ключ, призначений для генерації одноразових кодів (див. рис. 2.9).

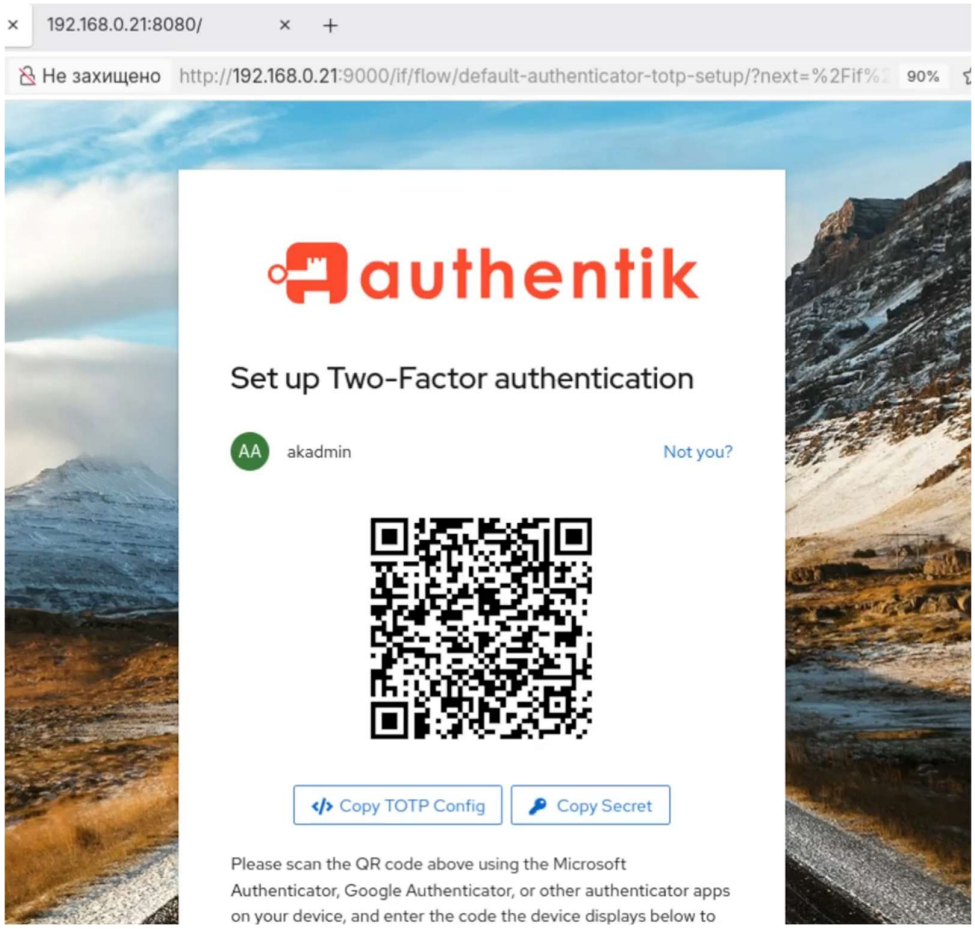


Рисунок 2.9 – Генерація QR-коду для TOTP

Після сканування цього коду за допомогою мобільного додатку-аутентифікатора формується прив’язка пристрою до облікового запису, а сам додаток починає генерувати шестизначні коди з обмеженим часом дії (див. рис. 2.10).

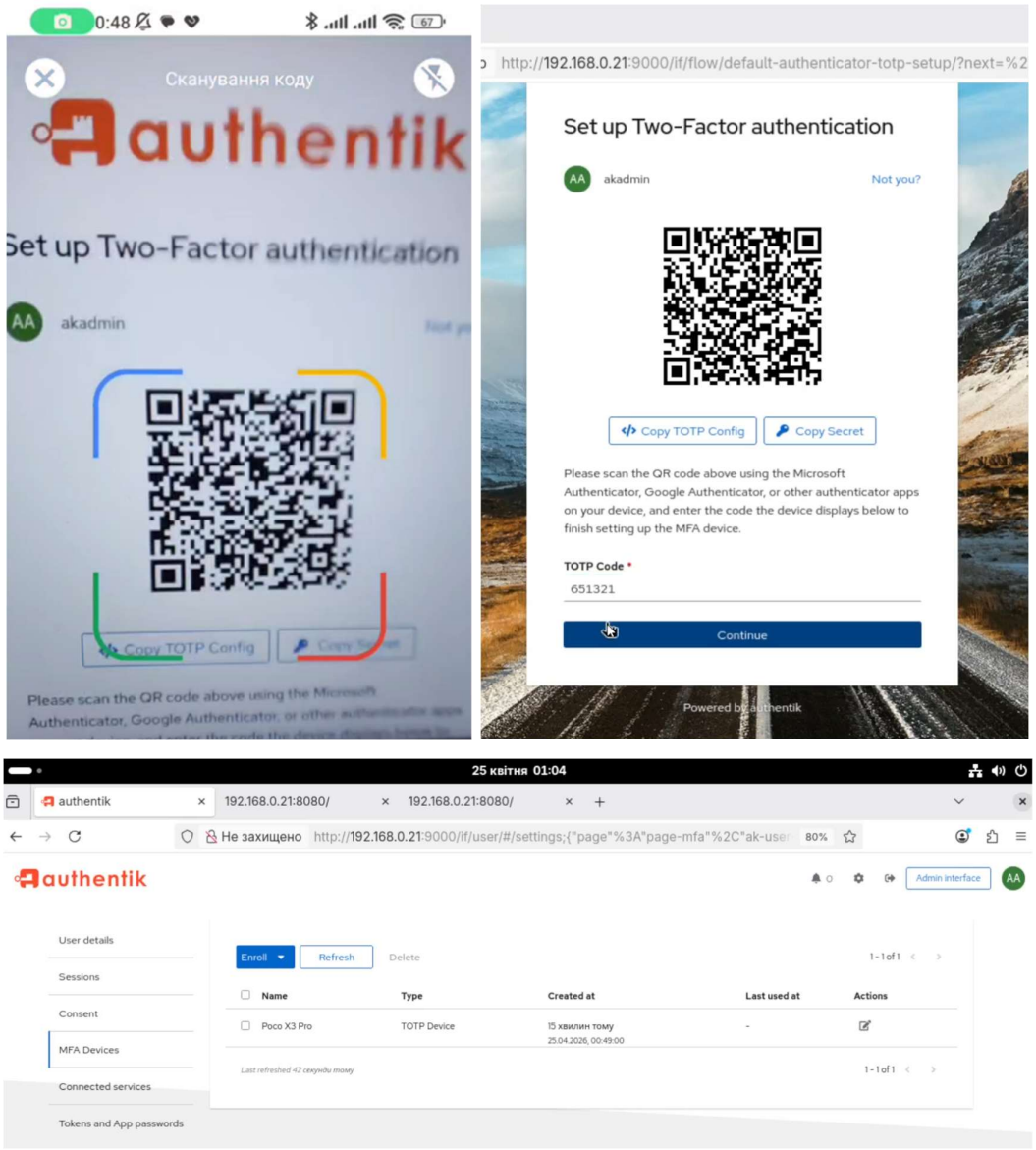


Рисунок 2.10 – Прив’язка сервісу Authentik до пристрою в Google Authenticator

Подальший процес автентифікації відбувається у два етапи, де спочатку здійснюється перевірка логіна та пароля, після чого у разі успішної первинної ідентифікації система переходить до етапу перевірки TOTP-коду, введеного користувачем, і лише у випадку відповідності очікуваному значенню доступ надається, тоді як будь-яке відхилення призводить до відмови в доступі, що значно підвищує рівень захисту навіть у випадку компрометації пароля.

Особливу увагу необхідно приділити фактору синхронізації часу, оскільки алгоритм TOTP базується на поточному часовому інтервалі, а отже точність

системного годинника як на стороні сервера, так і на стороні клієнтського пристрою визначає коректність генерації та перевірки кодів, при цьому навіть незначне відхилення може призвести до систематичних помилок автентифікації, що підкреслює необхідність використання механізмів синхронізації часу, таких як NTP [20].

Мережева інтеграція та DNS Docker. У межах реалізації контейнеризованої архітектури особлива увага приділяється організації мережевої взаємодії між компонентами системи, оскільки традиційний підхід, що базується на використанні статичних IP-адрес, створює низку обмежень, пов’язаних із залежністю від конкретної мережевої конфігурації, зміною адрес при перезапуску сервісів та ускладненням процесу масштабування, тоді як використання вбудованого механізму DNS у Docker дозволяє усунути ці недоліки за рахунок динамічної резолюції імен контейнерів у межах ізольованої мережі.

У створеній мережі "authentik_default" кожен контейнер автоматично отримує власне DNS-ім’я, яке відповідає імені сервісу, визначеному у конфігурації Docker Compose, у результаті чого взаємодія між компонентами відбувається не через IP-адреси, а через логічні імена, такі як "server", "postgresql" або "whoami", що формує стабільний і передбачуваний механізм адресації незалежно від змін у мережевому середовищі. Звернення до сервісів набуває вигляду внутрішніх HTTP-запитів, наприклад доступ до сервера автентифікації здійснюється через адресу "http://server:9000", тоді як взаємодія з тестовим сервісом реалізується через "http://whoami:80", а підключення до бази даних виконується через "postgresql:5432", що дозволяє забезпечити ізольовану та керовану комунікацію всередині контейнерної інфраструктури (див. рис. 2.11) [21].

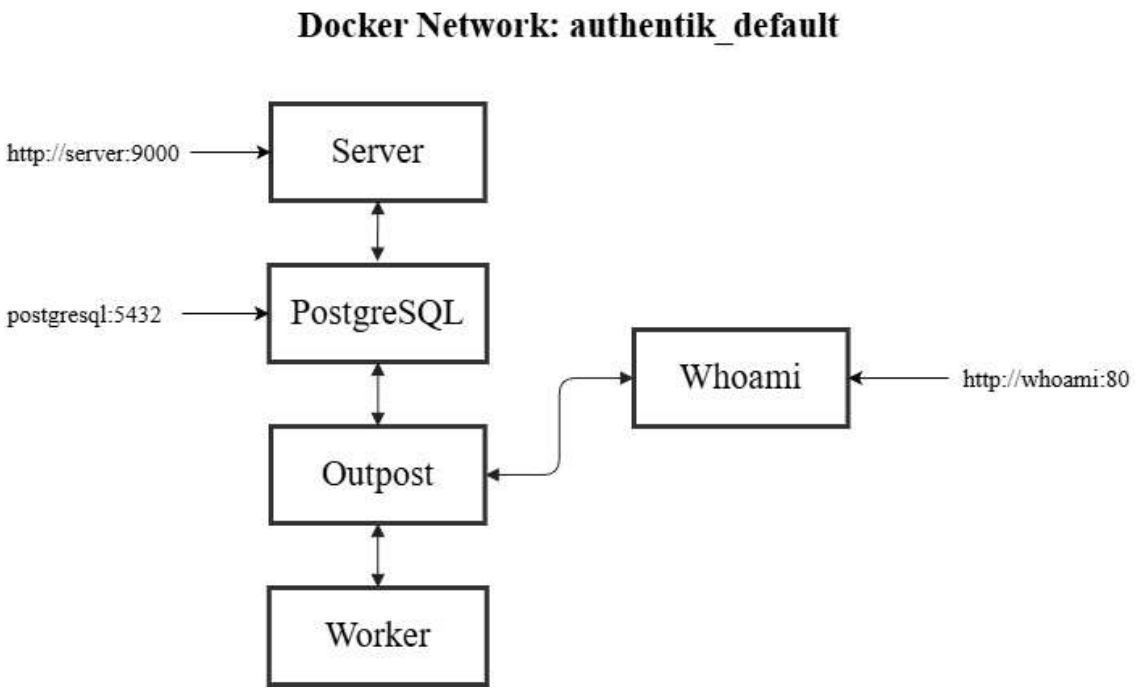


Рисунок 2.11 – Внутрішня DNS-резолюція Docker

У практичному сценарії взаємодії Outpost, який виступає точкою входу для користувацьких запитів, формує внутрішні запити до Authentik Server за допомогою DNS-імені, перевіряючи наявність валідної сесії через адресу "server:9000", після чого у разі успішної автентифікації виконується перенаправлення трафіку до цільового сервісу, наприклад whoami:80, що відбувається повністю всередині Docker-мережі без залучення зовнішніх IP-адрес, а така модель взаємодії дозволяє досягти високого рівня ізоляції та безпеки.

Інтеграція додаткових сервісів у мережу може здійснюватися як на етапі визначення конфігурації Docker Compose, так і динамічно за допомогою команди підключення контейнера до мережі, що забезпечує гнучкість у керуванні інфраструктурою, наприклад додавання тестового сервісу whoami до мережі може бути виконане окремо без необхідності повного перезапуску всієї системи, водночас можливість видалення або повторного розгортання окремих компонентів не впливає на інші сервіси, оскільки вони продовжують взаємодіяти через стабільні DNS-імена (див. рис. 2.12).

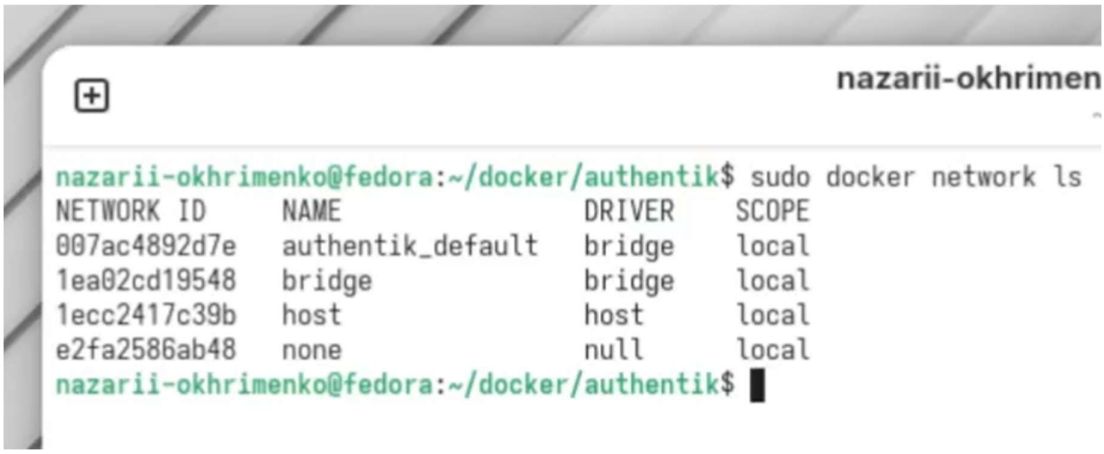


Рисунок 2.12 – Конфігурація мережі контейнерів

Використання Docker DNS усуває жорстку прив’язку до IP-адрес, що значно підвищує переносимість системи та спрощує її масштабування, оскільки додавання нових сервісів або зміна топології мережі не потребує ручного оновлення конфігурацій адресації, а така модель створює основу для подальшого переходу до більш складних рішень, зокрема використання реверс-проксі з підтримкою повноцінних доменних імен, що відповідають стандарту FQDN [21].

У підсумку реалізація мережевої інтеграції на базі Docker DNS дозволяє сформувати ізольовану, гнучку та масштабовану інфраструктуру, у межах якої всі компоненти системи багатофакторної автентифікації взаємодіють через внутрішні імена, використовуючи PostgreSQL як надійне сховище даних та підтримуючи механізм MFA на основі TOTP, при цьому свідоме виключення Redis на етапі прототипування не лише спрощує початкову конфігурацію, але й створює передумови для виявлення обмежень, пов’язаних із обробкою сесій у пам’яті, що формує основу для подальшого аналізу ефективності системи.

2.3 Алгоритми та внутрішні механізми захисту

У межах побудованої системи багатофакторної автентифікації необхідним є розгляд внутрішньої логіки функціонування, яка визначає поведінку компонентів у процесі обробки запитів у реальному часі, включаючи:

1. механізми прийняття рішень щодо доступу;
2. перевірку автентифікаційних даних;
3. керування станами користувацьких сесій.

При цьому на відміну від попередніх підрозділів, де основна увага приділялася архітектурі та конфігурації, у даному випадку акцент зміщується на алгоритмічний рівень, що описує послідовність дій системи при взаємодії з користувачем та забезпеченні захисту ресурсів.

Алгоритм перехоплення запиту (Forward Auth). Механізм Forward Auth реалізує концепцію зовнішньої перевірки доступу, за якої цільовий веб-сервіс не виконує автентифікацію самостійно, а делегує її спеціалізованому компоненту, яким виступає Outpost, що функціонує як проміжний проксі-рівень і перехоплює всі HTTP-запити перед їх передачею до захищеного ресурсу, у результаті чого формується централізована модель контролю доступу.

Послідовність обробки запиту починається з моменту звернення користувача до ресурсу через зовнішню адресу хоста, наприклад за допомогою URL із відповідним портом, після чого запит надходить до Outpost, який виступає єдиною точкою входу і ініціює перевірку наявності сесійного cookie, що містить ідентифікатор активної сесії, далі відбувається формування внутрішнього запиту до Authentik Server через Docker DNS, наприклад за адресою server:9000, де виконується перевірка валідності цієї сесії та відповідності політикам доступу [22].

У випадку відсутності сесії або її невалідності Outpost формує HTTP-відповідь із перенаправленням користувача на сторінку автентифікації, що знаходиться на сервері Authentik, де необхідно пройти процедуру входу, після успішного завершення якої створюється нова сесія та видається відповідний cookie,

який надалі використовується для ідентифікації користувача, тоді як при наявності валідної сесії запит безперешкодно передається до цільового сервісу через внутрішню адресу, наприклад whoami:80, що відбувається без додаткової участі користувача.

Така модель дозволяє відокремити логіку безпеки від функціональної частини сервісів, оскільки самі сервіси не потребують реалізації механізмів автентифікації, а весь контроль доступу централізовано обробляється на рівні проксі, що забезпечує гнучкість у зміні політик та спрощує інтеграцію нових сервісів у захищене середовище, при цьому концепція Zero Trust реалізується через необхідність перевірки кожного запиту незалежно від його джерела або попереднього стану сесії (див. рис. 2.13).

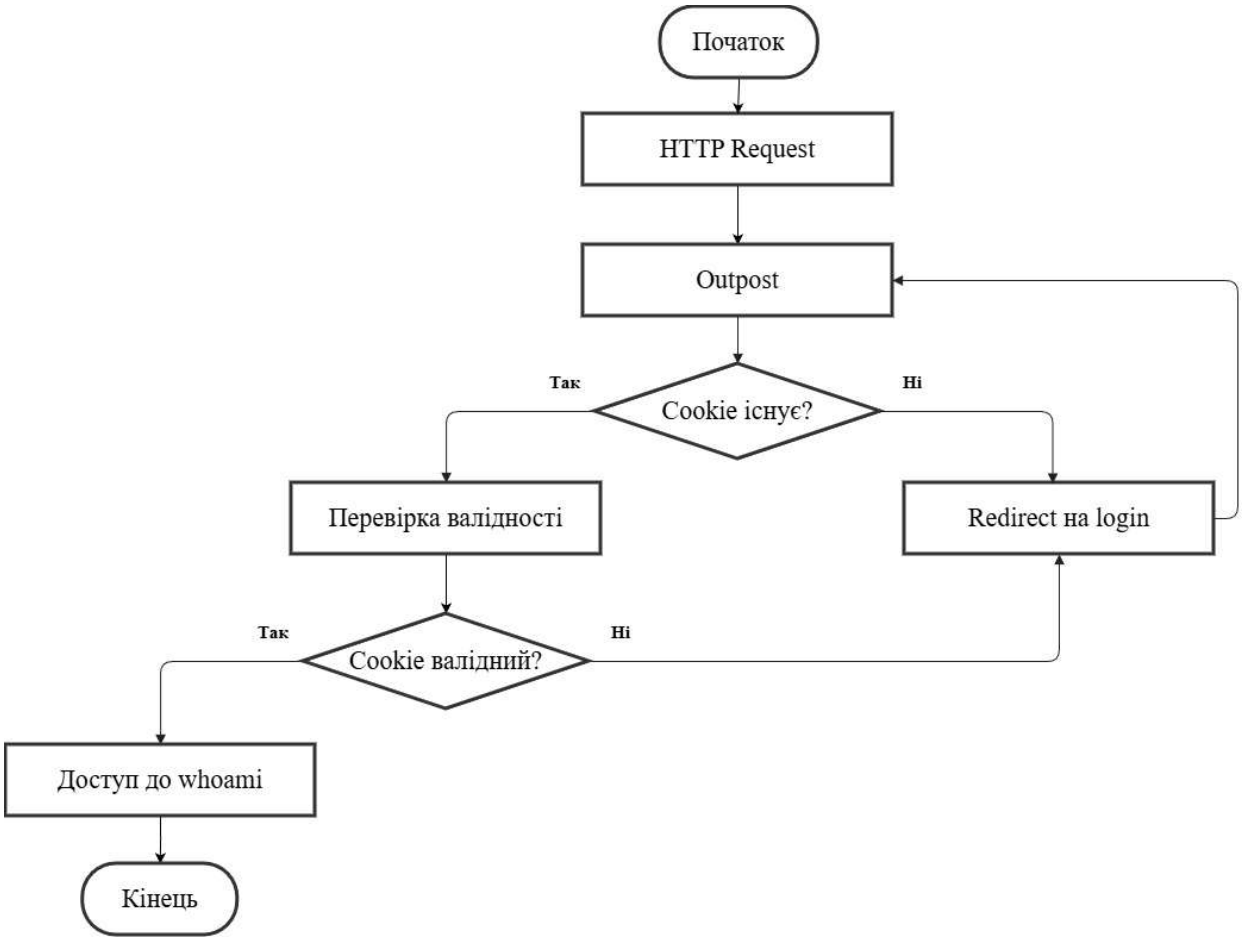


Рисунок 2.13 – Алгоритм перехоплення HTTP-запиту (Forward Auth)

Двофакторна перевірка (MFA, TOTP). Реалізація багатофакторної автентифікації у системі базується на класичній моделі «знання зі володінням», де першим фактором виступає пароль користувача, а другим — одноразовий динамічний код, згенерований мобільним пристроєм, наприклад смартфоном Poco X3 Pro із використанням додатку Google Authenticator.

Алгоритм перевірки починається з введення облікових даних на сторінці логіну, після чого пароль проходить процедуру хешування та порівняння з відповідним значенням, збереженим у базі даних PostgreSQL, де у разі невідповідності доступ негайно блокується без переходу до наступного етапу. У випадку успішної перевірки активується додатковий етап автентифікації — TOTP Stage, який вимагає введення шестизначного коду, що генерується на стороні клієнта.

Принцип роботи TOTP базується на використанні спільного секретного ключа, який генерується сервером під час реєстрації пристрою та передається користувачу у вигляді QR-коду, після чого зберігається як у системі Authentik, так і у додатку аутентифікації. Генерація одноразового коду здійснюється шляхом застосування криптографічної функції HMAC до поточного часу, поділеного на інтервали тривалістю 30 секунд, що забезпечує постійну зміну коду та його обмежений термін дії [23].

Алгоритмічно процес виглядає як послідовність перевірок, де після успішного введення пароля система очікує TOTP-код, перевіряє його валідність з урахуванням часових інтервалів, і у разі збігу дозволяє завершення автентифікації, інакше повертає помилку та пропонує повторне введення. Важливим аспектом функціонування такого механізму виступає синхронізація часу між сервером і клієнтом, що реалізується через протокол NTP, оскільки навіть незначне відхилення системного часу може призвести до невідповідності кодів і, як наслідок, відмови у доступі (див. рис. 2.14) [23].

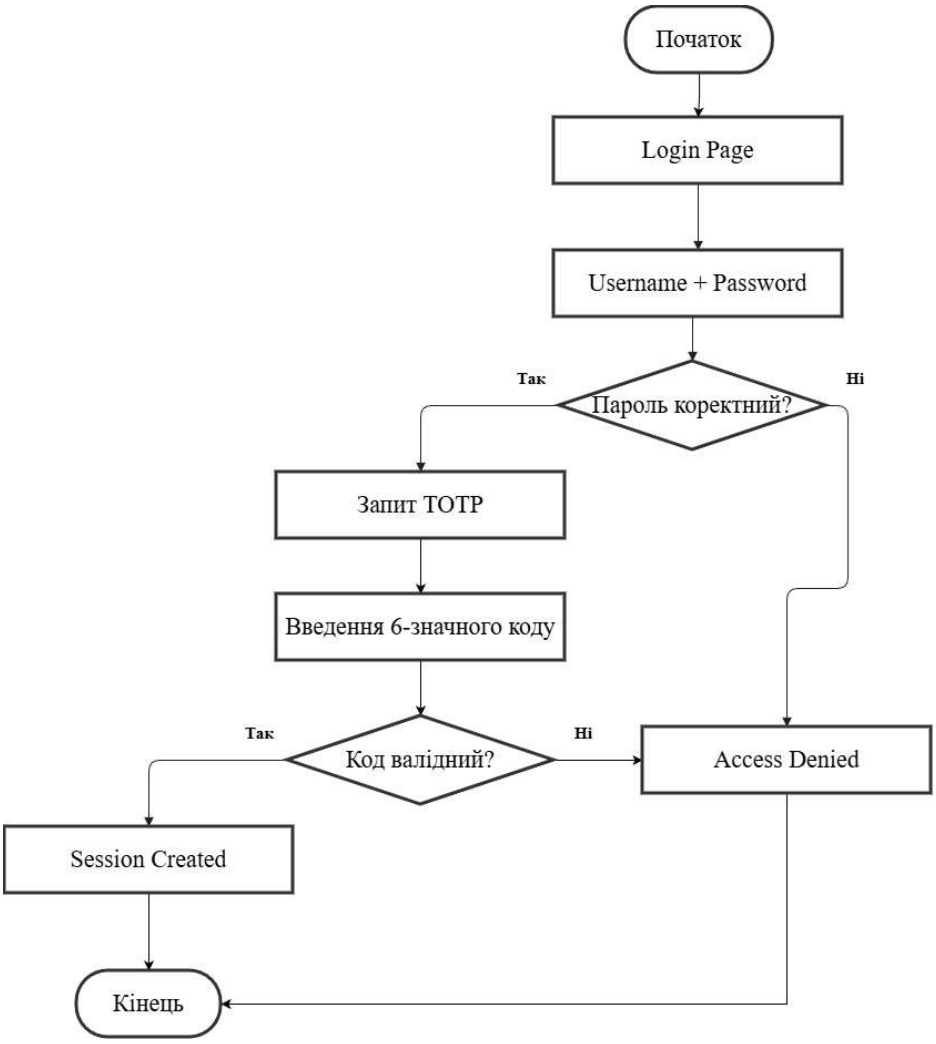


Рисунок 2.14 – Алгоритм двофакторної перевірки (MFA, TOTP)

Застосування MFA суттєво підвищує рівень безпеки системи, оскільки компрометація пароля не дає можливості отримати доступ без другого фактора, а короткий життєвий цикл TOTP-кодів мінімізує ризики повторного використання або перехоплення.

Керування сесіями. Після успішного проходження обох етапів автентифікації система переходить до створення сесії, що дозволяє уникнути необхідності повторного введення облікових даних при кожному запиті та забезпечує безперервність взаємодії користувача з ресурсами. У цьому процесі генерується унікальний ідентифікатор сесії, який передається клієнту у вигляді cookie, що зберігається у браузері та використовується для подальшої ідентифікації.

Подальша робота системи передбачає перевірку цього cookie при кожному HTTP-запиті, де Authentik Outpost отримує ідентифікатор сесії, виконує внутрішній запит до серверного компонента за адресою "http://server:9000" і визначає, чи є сесія дійсною, не простроченою та такою, що відповідає політикам доступу. У разі позитивного результату доступ до ресурсу надається без повторної автентифікації, тоді як у випадку відсутності або завершення терміну дії сесії користувач перенаправляється на сторінку логіну.

Особливістю реалізації виступає відсутність компонента Redis у поточній конфігурації, що призводить до зберігання частини сесійних даних безпосередньо у пам’яті Python-процесів серверного та воркер-компонентів, а також частково у базі даних. Такий підхід створює додаткове навантаження на оперативну пам’ять і збільшує час обробки запитів, особливо при зростанні кількості активних користувачів, що у довготривалій перспективі може призводити до деградації продуктивності.

Життєвий цикл сесії визначається параметрами часу життя (TTL), після завершення якого відбувається автоматичне завершення сесії та видалення відповідних даних, що підвищує рівень безпеки за рахунок обмеження часу дії доступу. Додаткові механізми захисту включають використання HTTPOnly та Secure cookie, які обмежують доступ до сесійних даних зі сторони клієнтських скриптів і забезпечують передачу лише через захищені канали (див. рис. 2.15).

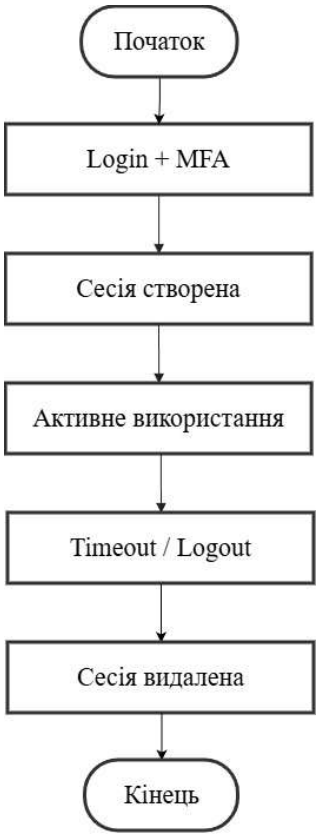


Рисунок 2.15 – Алгоритм життєвого циклу сесії

Сукупність описаних механізмів формує завершену модель автентифікації, де Forward Auth забезпечує централізований контроль доступу, TOTP реалізує багатофакторну перевірку, а система керування сесіями оптимізує взаємодію користувача з ресурсами, при цьому відсутність кешуючого шару створює передумови для подальшого аналізу ефективності та необхідності оптимізації архітектури при переході до промислового використання.

2.4 Практична інтеграція та валідація системи

Завершальний етап побудови прототипу системи багатофакторної автентифікації передбачає практичну інтеграцію цільового сервісу та перевірку коректності функціонування всієї архітектури в умовах, максимально наближених

до реального використання, де основною метою виступає підтвердження працездатності механізму Forward Auth, демонстрація повного циклу MFA-автентифікації та аналіз поведінки системи при різних сценаріях доступу, включаючи як успішні, так і відхилені спроби входу, що дозволяє оцінити не лише функціональність, але й надійність реалізованого підходу до контролю доступу.

У якості тестового ресурсу використовується легковаговий сервіс "whoami", який призначений для відображення інформації про HTTP-запити, включаючи:

— заголовки;

— IP-адресу клієнта;

— інші параметри запиту.

Все це робить його зручним інструментом для візуального підтвердження факту проходження автентифікації та передачі трафіку через проксі-рівень, при цьому відсутність у нього власної логіки автентифікації дозволяє чітко розмежувати відповідальність між системою захисту та цільовим сервісом [29].

Підключення Whoami сервісу. Інтеграція сервісу whoami реалізується шляхом розгортання відповідного контейнера на базі образу "traefik/whoami" у середовищі Docker з подальшим підключенням до ізольованої мережі "authentik_default", що забезпечує можливість взаємодії з іншими компонентами системи через внутрішні DNS-імена без необхідності використання зовнішніх IP-адрес або відкриття додаткових портів на хості. Такий підхід дозволяє зберегти ізоляцію сервісу, оскільки контейнер не експонує власний порт напряму в зовнішню мережу, а доступ до нього здійснюється виключно через проксі-компонент Outpost.

У процесі інтеграції контейнер додається до конфігураційного файлу Docker Compose або підключається до мережі за допомогою команди "docker network connect", після чого стає доступним за внутрішнім DNS-іменем "http://whoami:80", яке використовується Outpost для маршрутизації трафіку після успішної автентифікації користувача. Вхідною точкою для клієнта виступає адреса з проброшеним портом проксі, наприклад "http://192.168.0.21:8080", де Outpost приймає HTTP-запити, виконує перевірку сесії через сервер Authentik та у разі

успішного результату перенаправляє запит до внутрішнього сервісу (див. рис. 2.16).

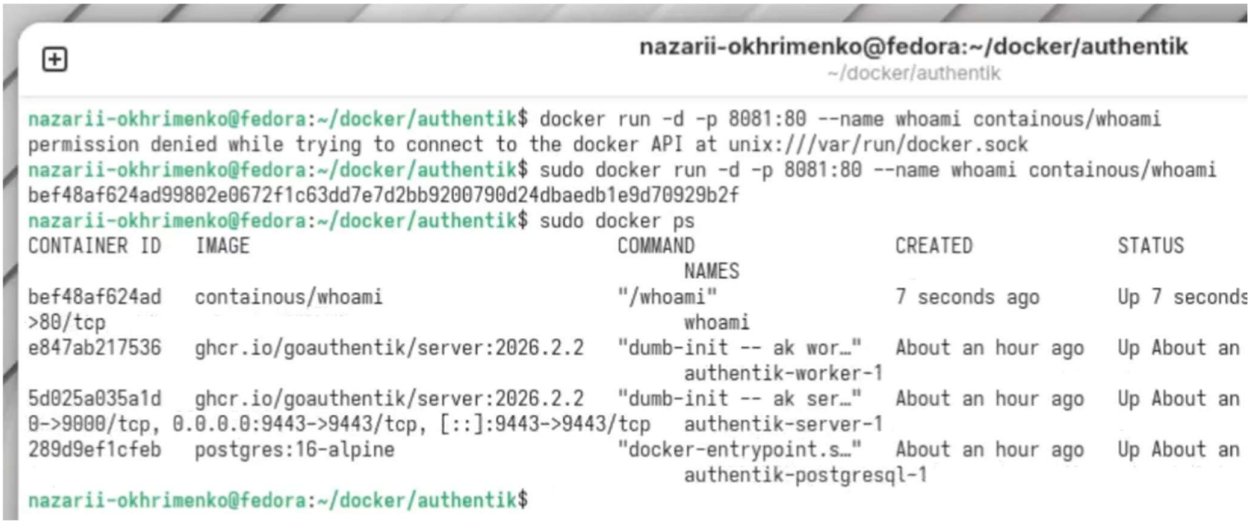


Рисунок 2.16 – Розгортання Whoami сервісу у Docker

Логіка взаємодії в межах системи формується як послідовний ланцюг обробки запиту, де клієнт ініціює HTTP-звернення до проксі, проксі звертається до серверного компонента для перевірки автентифікації, після чого у разі позитивного результату здійснюється внутрішній виклик до сервісу "whoami", який формує відповідь і повертає її користувачу, при цьому сам сервіс залишається повністю ізольованим від зовнішнього середовища та не містить жодних механізмів контролю доступу.

Така архітектура дозволяє продемонструвати принцип незалежності цільового ресурсу від системи автентифікації, де будь-який HTTP-сервіс може бути інтегрований у захищений контур без необхідності модифікації його коду або додавання власних механізмів безпеки, що відповідає концепції "Security as a Layer", у межах якої захист реалізується як окремий інфраструктурний рівень.

Тестування сценаріїв доступу. Після завершення інтеграції цільового сервісу whoami у середовище системи автентифікації було проведено серію тестових сценаріїв, спрямованих на перевірку коректності функціонування всіх компонентів

у реальних умовах експлуатації, де оцінюється поведінка системи при різних типах доступу, включаючи:

1. неавторизовані запити;
2. успішну автентифікацію з використанням багатофакторної перевірки;
3. помилки введення другого фактора.

У першому сценарії розглядається ситуація неавторизованого доступу, за якої користувач не має активної сесії та намагається отримати доступ до ресурсу через точку входу, представлену адресою проксі, після ініціації HTTP-запиту система перехоплює звернення на рівні Outpost і виконує перевірку наявності cookie з ідентифікатором сесії, відсутність якого призводить до автоматичного блокування доступу до сервісу та формування відповіді з редиректом на сторінку автентифікації Authentik, у результаті чого користувач не отримує доступ до whoami і змушений пройти процедуру входу (див. рис. 2.17).

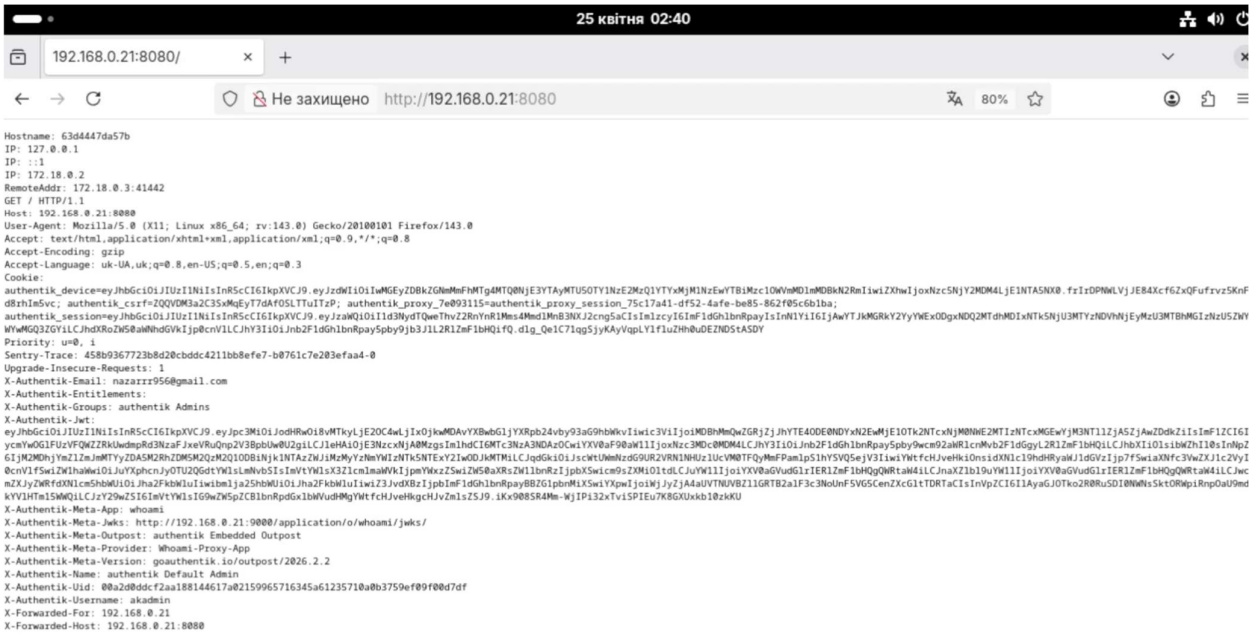


Рисунок 2.17 – Відображення HTTP-запиту у Whoami

Другий сценарій передбачає успішну автентифікацію з використанням MFA, де користувач вводить коректні облікові дані та проходить обидва етапи перевірки, після введення логіну і пароля активується другий фактор, який реалізується через

одноразовий код, згенерований мобільним додатком Google Authenticator, після чого у разі правильного введення коду створюється сесія, генерується cookie та виконується автоматичний редирект назад до цільового ресурсу, де відображається інформація про HTTP-запит, що підтверджує успішне проходження всіх етапів автентифікації та коректну маршрутизацію трафіку через проксі (див. рис. 2.18).

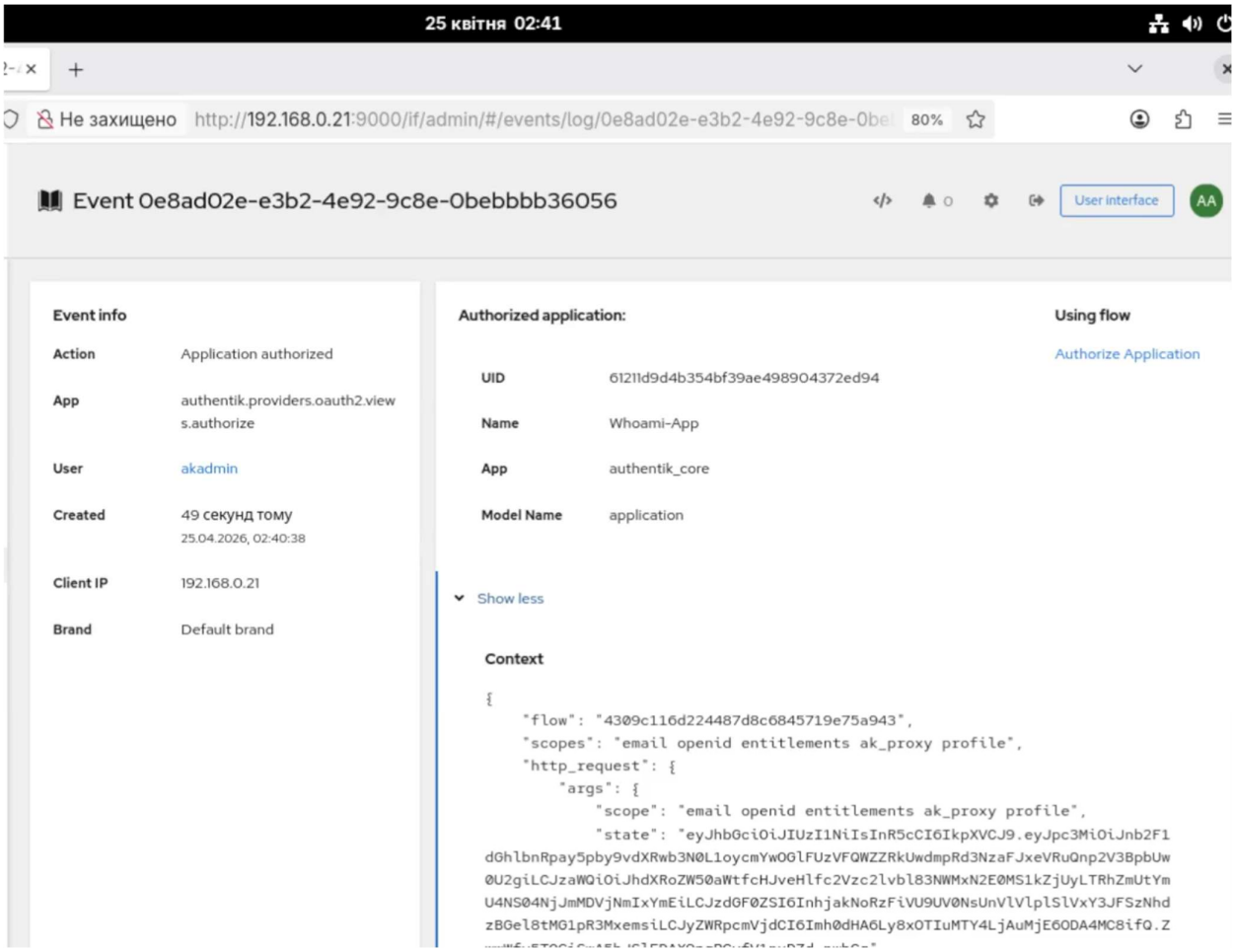


Рисунок 2.18 – Успішна автентифікація та доступ до сервісу

Третій сценарій демонструє поведінку системи у випадку помилки другого фактора, коли користувач вводить неправильний TOTP-код після успішного проходження першого етапу, у такій ситуації система відхиляє спробу автентифікації, не створює сесію та залишає користувача на етапі введення одноразового коду, забезпечуючи можливість повторної спроби введення, при цьому доступ до ресурсу повністю блокується до моменту введення валідного

значення, що підтверджує коректність реалізації механізму багатофакторної перевірки (див. рис. 2.19).

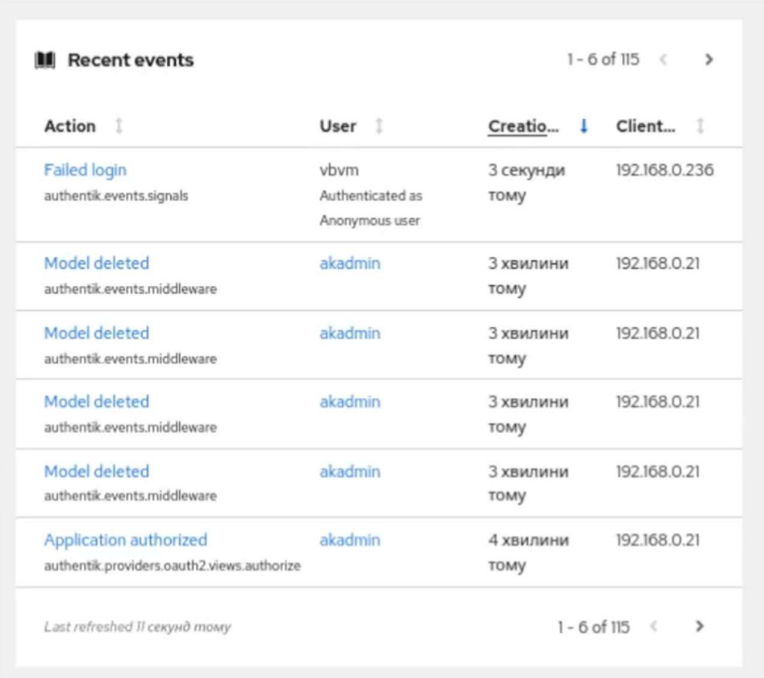


Рисунок 2.19 – Помилка введення TOTP-коду

Для аналізу внутрішніх процесів обробки запитів використовується перегляд логів, який може здійснюватися як через інструменти Docker, наприклад команду "docker compose logs -f outpost", так і через вбудований веб-інтерфейс Authentik, де відображаються події системи у вигляді журналу, що включає інформацію про успішні та невдалі спроби входу, створення сесій, HTTP-відповіді з кодами 200, 401 або 403, а також деталі перевірки політик доступу, що дозволяє відстежити повний життєвий цикл кожного запиту (див. табл. 2.2).

Таблиця 2.2 – Сценарії тестування доступу та результати перевірки MFA

Сценарій	Умови	Поведінка системи	Результат
Неавторизований доступ	Немає сесії	Редирект на Authentik	Доступ заборонено
Успішна MFA	Валідні логін + TOTP	Створення сесії, редирект	Доступ дозволено
Помилка пароля	Невірний пароль	Відмова на першому етапі	Доступ заборонено
Невірний TOTP	Невалідний код	Повторне введення	Доступ заборонено
Повторний доступ	Є cookie	Прямий доступ	Доступ дозволено

Аналіз журналів дає можливість ідентифікувати конкретні сценарії використання системи, включаючи спроби введення некоректних даних, відсутність сесії або помилки валідації, що забезпечує додатковий рівень контролю та спрощує процес налагодження, оскільки вся інформація про взаємодію користувача з системою зберігається централізовано та доступна через адміністративний інтерфейс (див. рис. 2.20).

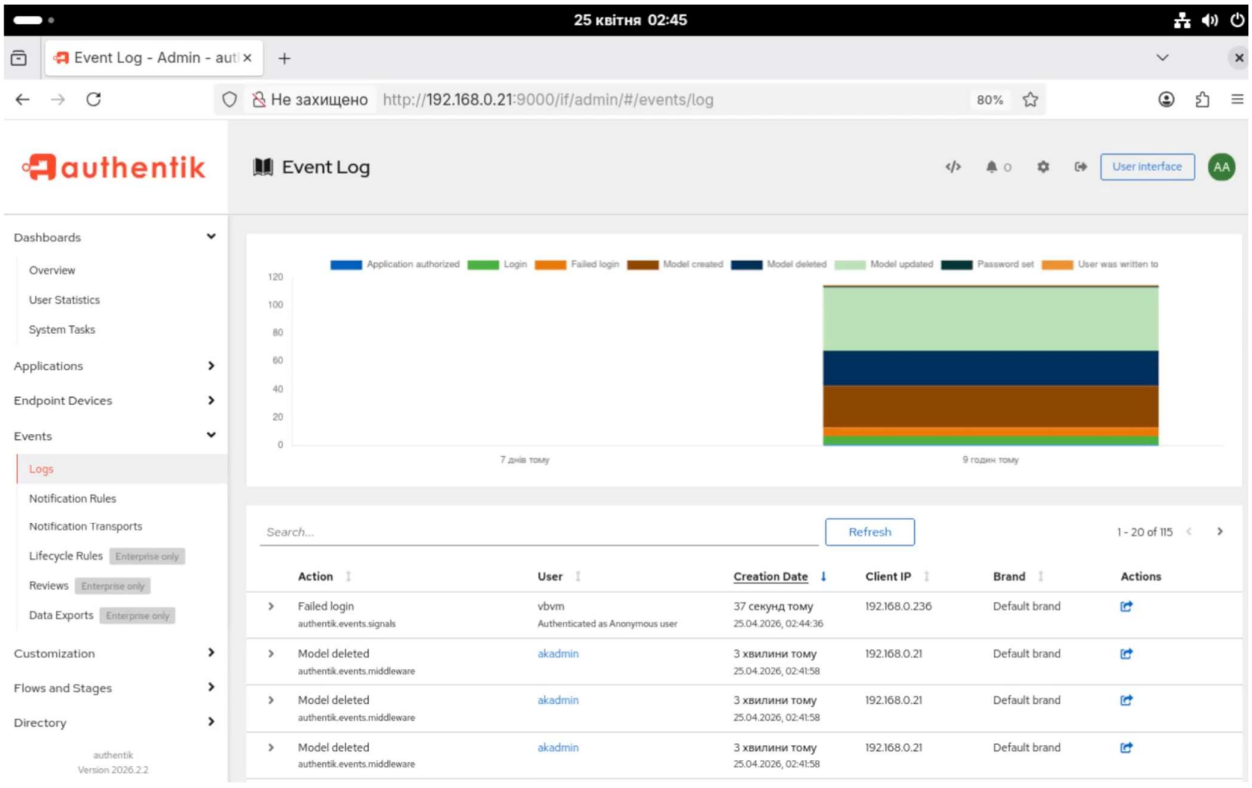


Рисунок 2.20 – Логи автентифікації у веб-інтерфейсі Authentik

У ході тестування встановлено, що система коректно блокує всі неавторизовані запити, забезпечує стабільну роботу механізму MFA та автоматично відновлює доступ після успішної автентифікації за рахунок використання сесій, що покращує зручність користування, однак відсутність кешуючого шару на базі Redis призводить до можливих затримок при обробці запитів за умов підвищеного навантаження, що проявляється у збільшенні часу відповіді системи.

Практична перевірка MFA включає використання мобільного пристрою, зокрема смартфона Xiaomi Poco X3 Pro, на якому встановлено додаток для генерації одноразових кодів, процес взаємодії передбачає сканування QR-коду, згенерованого системою, після чого пристрій починає генерувати шестизначні коди з інтервалом у 30 секунд, які вводяться у форму автентифікації для підтвердження особи користувача, що дозволяє реалізувати практичний і зручний механізм другого фактора без необхідності використання спеціалізованого обладнання (див. рис. 2.21).

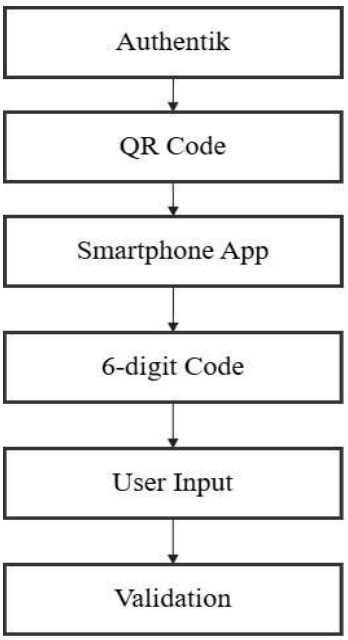


Рисунок 2.21 – Повний цикл доступу користувача (end-to-end)

3. АНАЛІЗ ЕФЕКТИВНОСТІ MFA ТА ПЕРСПЕКТИВИ МАСШТАБУВАННЯ

3.1 Аналіз споживання ресурсів та критичні вузькі місця

Аналіз споживання ресурсів та критичні вузькі місця розглядає поведінку прототипу системи багатофакторної автентифікації в умовах реального навантаження та дозволяє оцінити ефективність використання апаратних ресурсів у нативному середовищі Fedora Linux, розгорнутому на фізичному пристрої без застосування віртуалізації, де відсутність гіпервізорів усуває додаткові накладні витрати і забезпечує більш точне відображення характеристик використання CPU, RAM та дискової підсистеми, при цьому основна увага зосереджується на поведінці контейнерів Authentik Server та Worker, які виконують обробку автентифікаційних запитів, генерацію токенів і підтримку сесій, а також на взаємодії з базою даних PostgreSQL у межах контейнеризованої архітектури, де кожен компонент функціонує ізольовано, але логічно залежить від інших, унаслідок чого навіть незначні зміни конфігурації, зокрема виключення кешуючого шару, призводять до помітних змін у поведінці всієї системи, що дозволяє виявити слабкі місця та сформувати обґрунтовані висновки щодо подальшої оптимізації.

Проблема витоку пам'яті (Memory Leak). Проблема витоку пам'яті (Memory Leak) проявляється під час тривалого функціонування системи та зростання кількості автентифікаційних запитів, коли спостерігається поступове збільшення використання оперативної пам'яті контейнерами Authentik Server та Worker, що пов'язано з особливостями обробки сесійних даних і тимчасових станів у середовищі без Redis, де всі проміжні результати, включаючи токени доступу, інформацію про активні сесії, результати перевірки прав доступу та інші об'єкти стану, зберігаються безпосередньо в пам'яті Python-процесів, при цьому механізм автоматичного звільнення пам'яті, реалізований через garbage collection, не встигає

ефективно очищати невикористані об’єкти за умов високої інтенсивності запитів, унаслідок чого накопичення даних призводить до зростання споживання RAM, що має кумулятивний характер і не зменшується навіть після завершення частини сесій, оскільки очищення відбувається із затримками або неповністю [25].

Подальше збільшення навантаження формує ситуацію, в якій кожна нова сесія додає нові об’єкти до пам’яті, що ускладнює роботу збирача сміття та підвищує навантаження на CPU, оскільки процес очищення починає вимагати більше обчислювальних ресурсів, що, у свою чергу, впливає на загальну продуктивність системи, спричиняючи збільшення часу відповіді на HTTP-запити та погіршення швидкодії сервісів, при цьому за критичних значень використання пам’яті можливе завершення контейнерів через механізм Out Of Memory, що призводить до втрати доступності сервісу та необхідності його повторного запуску, така поведінка не є класичним програмним дефектом у вигляді помилки в коді, а скоріше наслідком архітектурного рішення, де відсутній спеціалізований кешуючий рівень, призначений для обробки короткоживучих даних (див. рис. 3.1).

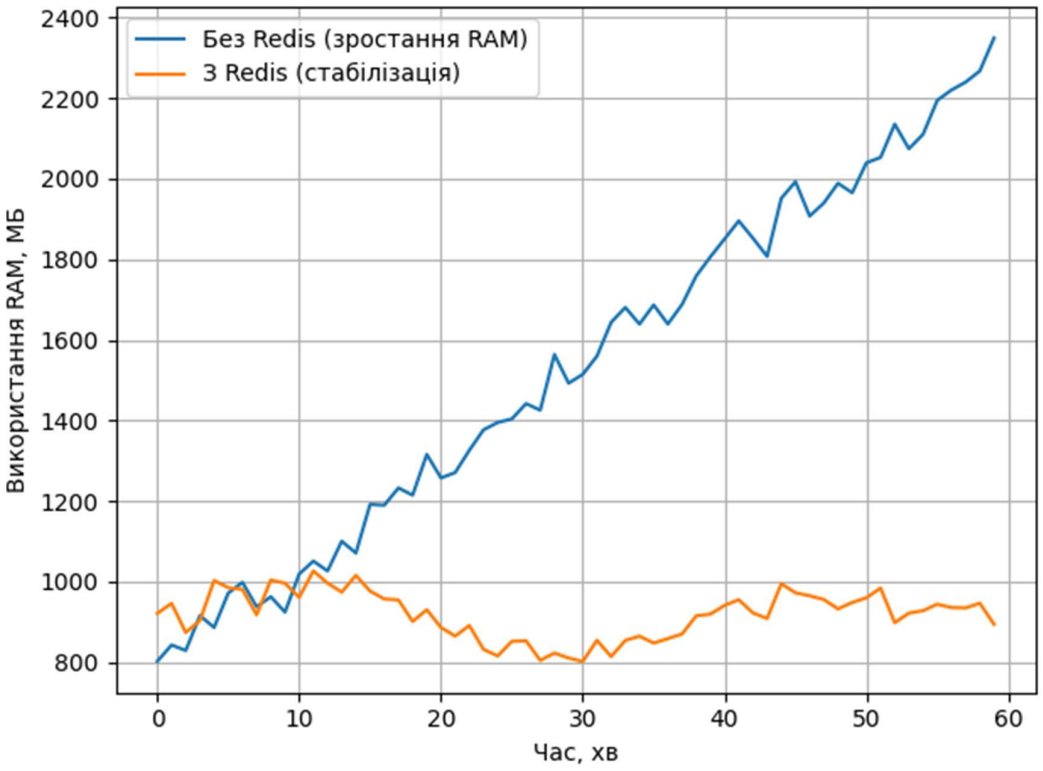


Рисунок 3.1 – Динаміка використання оперативної пам’яті (memory leak)

Аналіз внутрішньої логіки показує, що у стандартній конфігурації Authentik Redis використовується як швидке "in-memory" сховище для сесій, черг задач та тимчасових станів, що дозволяє винести навантаження з основних компонентів та оптимізувати обробку великої кількості запитів, однак у досліджуваному прототипі його виключення створює ситуацію, в якій серверні процеси змушені виконувати функції, для яких вони не оптимізовані, що призводить до неефективного використання ресурсів та появи ефекту, який інтерпретується як витік пам’яті, хоча фактично йдеться про накопичення невивільнених об’єктів у рамках нормальної, але перевантаженої логіки виконання (див. табл. 3.1) [25].

Таблиця 3.1 – Показники використання ресурсів системи під навантаженням

Стан системи	CPU (%)	RAM (МБ)	Час відповіді (мс)	Примітка
Без Redis (початок)	15–25	~800	~120	Стабільна робота
Без Redis (навантаження)	40–70	>2000	~300–500	Зростання пам’яті
Після оптимізації (Redis)	20–35	~900	~100–150	Стабілізація
Пікове навантаження (Redis)	50–65	~1200	~200	Контрольоване

Практичні спостереження підтверджують наявність проблеми через стабільне зростання споживання RAM у процесі тривалого тестування, відсутність повернення до початкового рівня після завершення частини сесій, поступове збільшення часу відповіді системи та підвищення навантаження на процесор, що особливо помітно при інтенсивному використанні MFA, де кожен етап автентифікації створює додаткові об’єкти стану, які залишаються в пам’яті довше, ніж очікується, у результаті чого система демонструє деградацію продуктивності, яка з часом посилюється (див. рис. 3.2).

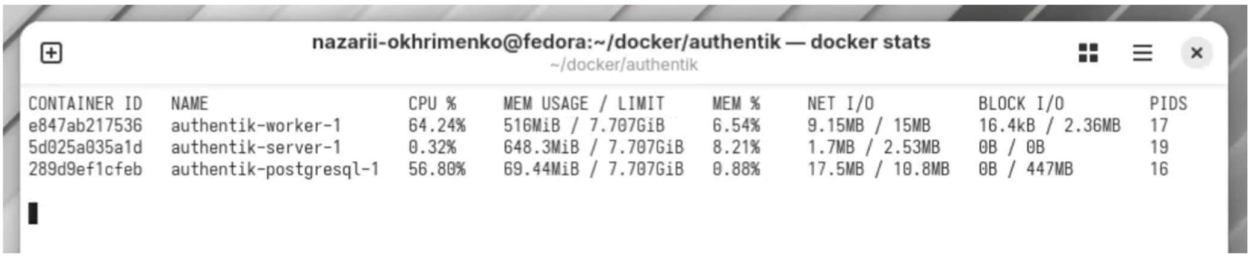


Рисунок 3.2 – Вплив відсутності кешування на продуктивність

Отримані результати дозволяють зробити висновок про те, що відсутність Redis у контейнерному стеку є критичним фактором, який безпосередньо впливає на стабільність роботи системи, оскільки саме цей компонент забезпечує ефективне кешування сесійних даних, розвантаження основних сервісів та оптимізацію обробки запитів, а його виключення, хоча й спрощує архітектуру на етапі прототипування, створює штучні обмеження, що проявляються у вигляді перевантаження пам’яті та зниження загальної продуктивності, таким чином дослідження підтверджує необхідність використання кешуючого шару як невід’ємної частини системи багатофакторної автентифікації при переході до реального середовища експлуатації.

Рекомендації з оптимізації. Рекомендації з оптимізації формуються на основі проведеного аналізу поведінки системи під навантаженням і дозволяють окреслити практичні кроки для підвищення стабільності, продуктивності та придатності до промислового використання, при цьому ключовим напрямом виступає впровадження додаткового кешуючого шару, який усуває виявлені обмеження архітектури без Redis і переводить обробку тимчасових даних у спеціалізоване середовище, де доступ до сесій, токенів та проміжних станів здійснюється значно швидше та ефективніше, унаслідок чого зменшується навантаження на Python-процеси та знижується інтенсивність використання оперативної пам’яті, що раніше накопичувалась у вигляді об’єктів стану всередині контейнерів "server" та "worker", при цьому сама логіка взаємодії компонентів змінюється з моделі, де обробка стану відбувається безпосередньо в пам’яті додатку, на модель із проміжним кешуючим рівнем, який перехоплює значну частину запитів і дозволяє серверу виконувати лише керуючі функції, тоді як "worker" отримує вже оптимізовані задачі [24].

У такій конфігурації клієнтський запит проходить через сервер, після чого звернення до тимчасових даних відбувається не в оперативній пам’яті процесу, а у швидкому "in-memory" сховищі, що знижує затримки доступу та стабілізує використання ресурсів, при цьому спостерігається зменшення часу відповіді системи, більш передбачувана поведінка при пікових навантаженнях та усунення поступового накопичення даних у RAM, яке раніше призводило до деградації продуктивності, додатково відкривається можливість горизонтального масштабування, оскільки декілька worker-процесів можуть одночасно працювати з єдиним кешуючим середовищем без дублювання станів у пам’яті кожного окремого контейнера (див. рис. 3.3).

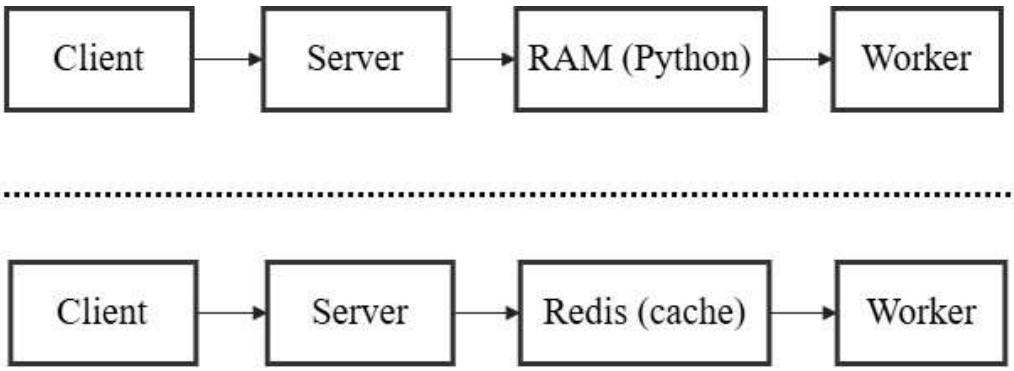


Рисунок 3.3 – Метод роботи системи з Redis та без нього

Подальша оптимізація стосується обмеження ресурсів на рівні контейнеризації, де використання механізмів Docker дозволяє задавати чіткі межі споживання оперативної пам’яті та процесорного часу для кожного сервісу, що запобігає неконтрольованому зростанню навантаження та підвищує передбачуваність роботи системи в умовах багатокористувацького доступу, при цьому моніторинг стану контейнерів через інструменти на кшталт "docker stats" або аналіз логів дозволяє оперативно виявляти аномалії, пов’язані зі зростанням споживання ресурсів, затримками відповіді або помилками обробки запитів, що створює основу для проактивного управління системою.

Окремим аспектом виступає структурна оптимізація архітектури, де доцільним є розділення компонентів за ролями на різні вузли інфраструктури,

унаслідок чого база даних PostgreSQL виноситься на окремий сервер для забезпечення стабільного зберігання даних та зменшення конкуренції за ресурси, кешуючий сервіс розгортається як незалежний компонент із високою швидкодією доступу до пам’яті, а worker-процеси масштабуються окремо залежно від навантаження, що дозволяє гнучко адаптувати систему до змін інтенсивності запитів без впливу на інші елементи, така модель відповідає принципам сучасної мікросервісної архітектури та забезпечує більш високий рівень відмовостійкості і керованості.

NTP-синхронізація. NTP-синхронізація розглядається як критично важливий аспект стабільності роботи механізму багатофакторної автентифікації, оскільки алгоритм TOTP безпосередньо залежить від точності системного часу як на стороні сервера, так і на стороні клієнтського пристрою, де генерація одноразових кодів відбувається на основі спільного секретного ключа та поточного часового інтервалу, що зазвичай становить 30 секунд, при цьому навіть незначні відхилення годинника призводять до розсинхронізації обчислюваних значень і, як наслідок, до відмови у автентифікації, що проявляється у вигляді помилок валідації коду навіть за умови його правильного введення користувачем.

Підтримка синхронізованого часу досягається через використання протоколу NTP, який забезпечує регулярне оновлення системного годинника на сервері та клієнтських пристроях, унаслідок чого гарантується узгодженість часових інтервалів, необхідних для коректної роботи алгоритму TOTP, при цьому важливим є не лише початкове налаштування синхронізації, але й постійний контроль її актуальності, оскільки дрейф системного часу може накопичуватись поступово та залишатися непомітним до моменту виникнення помилок автентифікації [26].

В умовах контейнеризованого середовища точність часу визначається системним годинником хост-операційної системи, тому стабільність NTP-синхронізації на рівні Fedora Linux безпосередньо впливає на всі запущені контейнери, що робить цей аспект частиною загальної інфраструктурної надійності, додатково враховується синхронізація часу на мобільних пристроях, які використовуються для генерації TOTP-кодів, оскільки навіть при коректно

налаштованому сервері некоректний час на смартфоні призводить до неможливості проходження другого фактора (див. рис. 3.4).

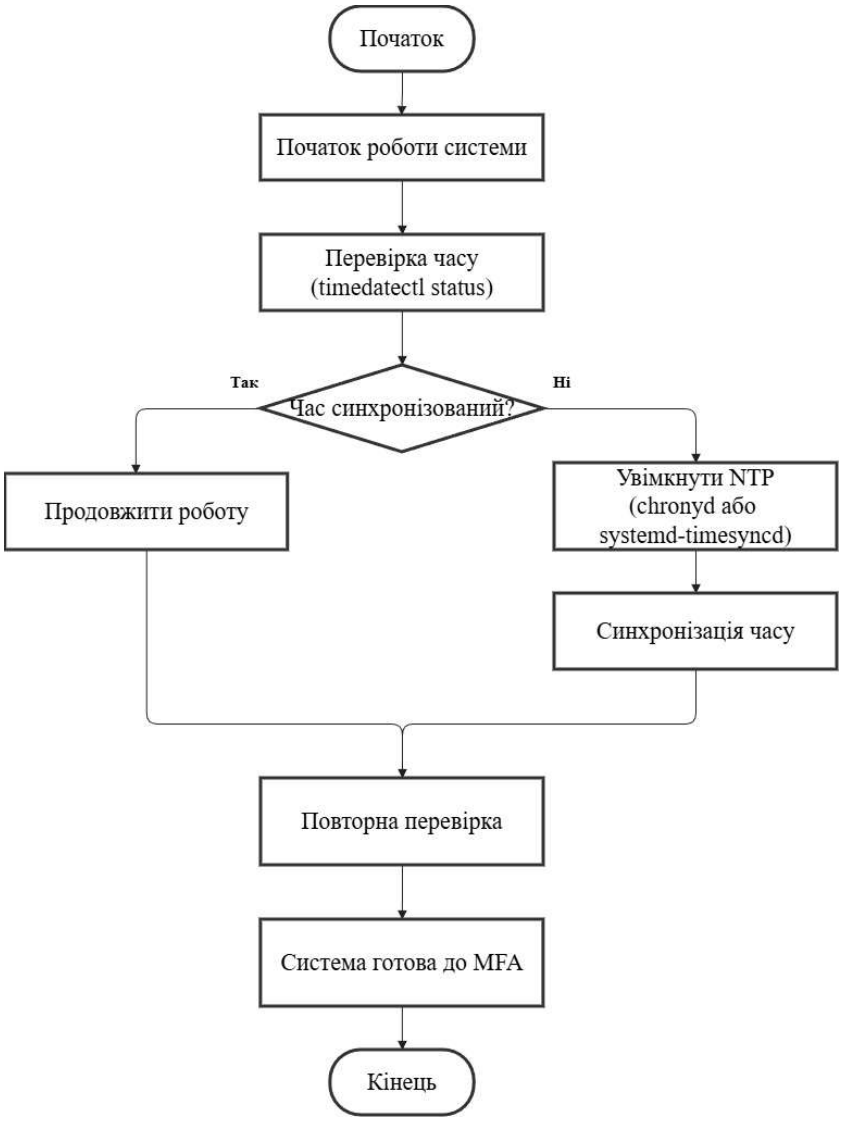


Рисунок 3.4 – Метод NTP-синхронізації в системі

З урахуванням виявлених залежностей можна сформулювати висновок про необхідність інтеграції механізмів контролю часу в загальну систему моніторингу, що дозволяє своєчасно виявляти відхилення та запобігати відмовам автентифікації, при цьому стабільна NTP-синхронізація у поєднанні з кешуванням та оптимізованою архітектурою формує основу для надійної роботи системи багатофакторної автентифікації в умовах реального навантаження.

3.2 Впровадження на мережевий вузол (Edge Security Gateway)

Впровадження на мережевий вузол (Edge Security Gateway) описує перехід від лабораторного прототипу, розгорнутого на ноутбуці з Fedora Linux, до архітектури, орієнтованої на реальну експлуатацію в межах мережевої інфраструктури, де система багатофакторної автентифікації використовується як централізований механізм контролю доступу до внутрішніх ресурсів, при цьому початкова реалізація, що функціонує у вигляді ізольованого стенду, демонструє коректність алгоритмів автентифікації, але не забезпечує необхідного рівня доступності, відмовостійкості та інтеграції з мережею, унаслідок чого виникає необхідність перенесення логіки захисту на окремий вузол, який виступає проміжною ланкою між зовнішнім середовищем і внутрішніми сервісами, де весь вхідний трафік проходить через контрольований шлюз і підлягає обов’язковій перевірці.

У такій моделі змінюється сама концепція застосування системи, оскільки замість захисту окремого веб-сервісу реалізується підхід, за якого весь сегмент мережі або набір сервісів опиняється за єдиною точкою контролю доступу, що дозволяє централізувати політики безпеки, уніфікувати механізми автентифікації та мінімізувати ризики несанкціонованого доступу, при цьому користувацький запит незалежно від цільового ресурсу спочатку потрапляє на шлюз, де виконується перевірка сесії, ініціюється процес автентифікації через Authentik, після чого за умови успішного проходження MFA здійснюється перенаправлення до внутрішнього сервісу, а у випадку відмови доступ блокується на рівні шлюзу без передачі запиту вглиб мережі, що формує модель ізоляції внутрішніх ресурсів від прямого зовнішнього доступу.

Вибір цільової платформи. Вибір цільової платформи визначає технічну основу для розгортання системи у ролі мережевого шлюзу, де поточна реалізація на ноутбуці Lenovo ThinkPad X1 Yoga Gen 8 розглядається як тестове середовище, придатне для досліджень і демонстрації функціональності, але обмежене з точки

зору безперервної роботи, масштабованості та інтеграції в корпоративну або приватну інфраструктуру, унаслідок чого для реального використання доцільно застосовувати окремий серверний вузол, який функціонує постійно та забезпечує обробку мережевого трафіку в режимі 24/7, при цьому можливі різні варіанти реалізації, включаючи фізичний сервер, віртуальну машину або спеціалізований вузол у DMZ-зоні, де встановлюється Docker-середовище для розгортання всіх компонентів системи.

Апаратні вимоги. Перехід від локального прототипу до повноцінного мережевого вузла рівня Edge Gateway природним чином супроводжується зростанням вимог до апаратних ресурсів, оскільки змінюється сама модель навантаження, де система перестає обслуговувати окремі тестові запити та переходить у режим постійної обробки трафіку від багатьох користувачів одночасно, включаючи зберігання сесій, виконання криптографічних операцій, перевірку багатофакторної автентифікації та обробку HTTP-запитів через проксі-рівень.

У такій моделі навантаження оперативна пам’ять використовується не лише для базових процесів контейнерів, а й для зберігання тимчасових даних сесій, кешу та внутрішніх станів автентифікації, що особливо критично за відсутності оптимізованого кешування або при високій інтенсивності запитів, у результаті чого мінімально доцільний обсяг оперативної пам’яті формується на рівні 4 ГБ, тоді як більш стабільна конфігурація починається від 8 ГБ, що дозволяє уникнути деградації продуктивності при пікових навантаженнях.

Обчислювальні ресурси процесора також набувають більшого значення, оскільки виконуються операції хешування паролів, генерації та перевірки TOTP-кодів, обробки HTTPS-з’єднань та логіки маршрутизації, що формує потребу у багатоядерній архітектурі з мінімально двома ядрами, тоді як конфігурація з чотирма ядрами забезпечує значно кращу стабільність роботи системи при паралельному обслуговуванні декількох клієнтів.

Підсистема зберігання даних повинна базуватися на SSD-накопичувачах, оскільки база даних PostgreSQL постійно виконує операції читання та запису,

включаючи роботу з обліковими записами, політиками доступу та сесіями, а також логування подій, що при використанні повільних носіїв призводить до затримок у відповіді системи та зниження загальної продуктивності.

Мережеві характеристики також набувають критичного значення, оскільки вузол Edge Gateway функціонує як центральна точка обробки трафіку, що передбачає стабільне мережеве з'єднання з пропускною здатністю на рівні Gigabit Ethernet, де затримки або нестабільність каналу можуть безпосередньо впливати на швидкість автентифікації та доступ до сервісів.

У такій архітектурі також формується залежність від використання кешуючих механізмів, зокрема Redis, який забезпечує швидкий доступ до тимчасових даних і знижує навантаження на основні процеси, що дозволяє стабілізувати використання оперативної пам'яті та зменшити затримки при обробці запитів, тоді як без такого шару значна частина навантаження переноситься на Python-процеси, що призводить до накопичення даних у пам'яті та потенційної деградації продуктивності (див. рис. 3.5).

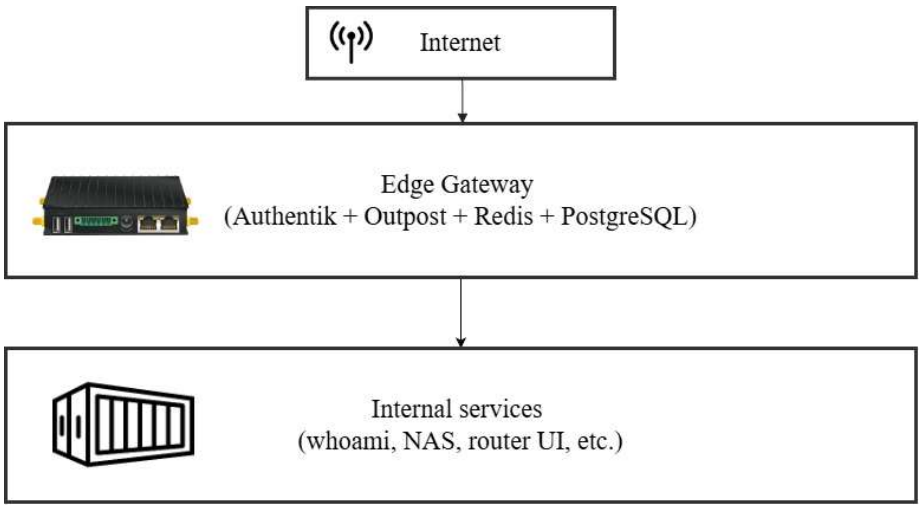


Рисунок 3.5 – Схема розміщення компонентів у мережі (DMZ)

Обмеження бюджетних маршрутизаторів класу SOHO стають очевидними у такому контексті, оскільки подібні пристрої зазвичай мають від 128 до 512 МБ оперативної пам'яті та побудовані на ARM-процесорах низької продуктивності, що

не дозволяє ефективно виконувати контейнеризовані навантаження, обробляти криптографічні операції та підтримувати бази даних, у результаті чого використання подібного обладнання для реалізації Edge Gateway є технічно невиправданим.

Більш доцільними платформами виступають серверні рішення або наближені до них конфігурації, включаючи:

— домашні сервери типу homelab;

— компактні системи класу Intel NUC;

— віртуальні сервери VPS/VDS;

— повноцінні корпоративні сервери.

Всі вони, той чи іншою мірою, забезпечують необхідний баланс між продуктивністю, енергоефективністю та масштабованістю (див. табл. 3.2).

Таблиця 3.2 – Порівняння архітектур: локальний прототип vs Edge Gateway

Критерій	Локальний прототип	Edge Security Gateway
Централізація	Часткова	Повна
Масштабованість	Обмежена	Висока
Безпека	Базова MFA	Zero Trust
Контроль доступу	Локальний	Централізований
Видимість сервісів	Прямий доступ	Приховані за шлюзом
Аудит	Обмежений	Централізований
Зручність	Середня	Висока (SSO)

Таким чином апаратна платформа для розгортання системи на рівні Edge Gateway повинна розглядатися як окремий інфраструктурний компонент, орієнтований на безперервну роботу, обробку значної кількості запитів та підтримку сучасних механізмів безпеки, що формує основу для стабільного функціонування всієї системи контролю доступу.

Переваги централізації (Zero Trust модель). Перехід до архітектури централізованого контролю доступу усуває основні недоліки децентралізованих моделей безпеки, де кожен сервіс функціонує із власною системою автентифікації, окремими політиками доступу та різними підходами до захисту, що призводить до фрагментації безпекового середовища та підвищення ризиків, пов’язаних із

людським фактором, такими як використання слабких паролів або відсутність багатофакторної автентифікації.

У централізованій моделі весь трафік проходить через єдину точку контролю — Edge Gateway, де виконується перевірка автентифікації та авторизації перед наданням доступу до внутрішніх ресурсів, що формує чітку схему взаємодії, в якій користувач ініціює запит до шлюзу, після чого система перевіряє наявність валідної сесії, у разі її відсутності виконується процедура MFA, і лише після успішного проходження всіх етапів доступ перенаправляється до відповідного внутрішнього сервісу, тоді як при невдачі запит блокується.

Така модель відповідає концепції Zero Trust, яка базується на принципі повної відсутності довіри за замовчуванням, де кожен запит незалежно від джерела повинен бути перевірений, автентифікований та авторизований, при цьому не враховується розташування користувача, тип мережі або пристрою, що забезпечує універсальний підхід до безпеки [27].

Централізація забезпечує єдину точку контролю доступу, де всі політики зосереджені в одному місці, що значно спрощує адміністрування системи та дозволяє швидко змінювати правила доступу без необхідності модифікації кожного окремого сервісу, додатково формується централізоване логування подій, що спрощує аудит та аналіз безпекових інцидентів.

Внутрішні сервіси у такій моделі ізольовані від прямого доступу з мережі, оскільки всі запити проходять через шлюз, що дозволяє використовувати навіть застарілі або небезпечні з точки зору автентифікації сервіси, обгортаючи їх у захисний шар без зміни їхньої внутрішньої логіки, що зменшує площу атаки та підвищує загальний рівень безпеки інфраструктури.

Архітектурно змінюється і сама модель доступу до ресурсів, де замість прямого звернення до кожного сервісу формується єдиний вхідний канал, через який проходять усі запити, що дозволяє приховати внутрішню структуру мережі та зробити її менш доступною для потенційного зловмисника.

Додатковою перевагою виступає можливість повторного використання автентифікації, де після створення сесії користувач отримує доступ до декількох

сервісів без повторного введення облікових даних, що формує поведінку, подібну до Single Sign-On, та підвищує зручність використання системи [27].

Гнучкість політик доступу також значно розширюється, оскільки стає можливим реалізувати доступ на основі ролей, IP-адрес, часових обмежень або інших параметрів, а також інтегрувати систему з зовнішніми каталогами користувачів, такими як LDAP або Active Directory, що забезпечує єдине джерело ідентифікації та централізоване управління користувачами [27].

Узагальнюючи, перехід від лабораторного прототипу до моделі Edge Security Gateway формує принципово новий рівень використання системи, де вона виступає не окремим компонентом, а інфраструктурним елементом безпеки, який забезпечує централізований контроль доступу, масштабованість та відповідність сучасним підходам до захисту інформаційних систем, що дозволяє розглядати її як повноцінний інструмент реалізації моделі Zero Trust у межах мережевої інфраструктури.

3.3 Перспективи розвитку та інфраструктурні покращення

Подальший розвиток реалізованого прототипу системи багатофакторної автентифікації логічно переходить у площину формування повноцінної інфраструктурної моделі, яка відповідає сучасним вимогам безпеки, масштабованості та керованості, при цьому лабораторна реалізація, виконана у нативному середовищі Linux із використанням контейнеризації, виступає як базовий етап, що дозволив виявити як сильні сторони архітектури, так і її обмеження, пов’язані з продуктивністю, мережею взаємодією та організацією доступу.

Розвиток системи у цьому напрямі передбачає поступову трансформацію від ізольованого середовища до інтегрованого інфраструктурного рішення, де всі компоненти функціонують як частина єдиного контуру безпеки, а доступ до

ресурсів організовується через централізовані механізми автентифікації та маршрутизації, що дозволяє забезпечити узгодженість політик доступу, зменшити складність адміністрування та підвищити загальний рівень захисту.

Такий підхід також дозволяє перейти від використання технічних адрес та портів до логічно структурованої моделі, де кожен сервіс має власне місце у загальній архітектурі, а взаємодія між компонентами відбувається через стандартизовані механізми, що відповідають практикам сучасних корпоративних середовищ [30].

Перехід на Traefik Proxy. У поточній архітектурі доступ до сервісів організований через використання IP-адрес та портів із застосуванням компонента Outpost, що забезпечує базову функціональність проксіювання та перевірки доступу, однак така модель залишається обмеженою з точки зору масштабованості, зручності використання та відповідності сучасним стандартам безпеки, у зв’язку з чим виникає необхідність переходу до більш універсального рішення на рівні мережевої маршрутизації.

У якості такого рішення розглядається Traefik Proxy, який виступає центральним маршрутизатором HTTPS-трафіку та забезпечує обробку вхідних запитів із подальшим перенаправленням до відповідних сервісів на основі визначених правил, при цьому взаємодія будується таким чином, що користувач звертається до доменного імені, після чого запит проходить через Traefik, де застосовуються правила маршрутизації, TLS-шифрування та механізми інтеграції з системою автентифікації [28].

Замість використання IP-адрес формату 192.168.x.x формується доменна модель із застосуванням FQDN, де кожен сервіс отримує зрозуміле ім’я, наприклад "https://auth.example.local" або "https://service.example.local", що не лише підвищує зручність доступу, але й забезпечує відповідність корпоративним стандартам адресації та спрощує подальше масштабування системи.

Інтеграція із Let's Encrypt дозволяє автоматизувати процес отримання та оновлення SSL-сертифікатів, у результаті чого всі з’єднання переходять у захищений режим HTTPS без необхідності ручного налаштування, що є критично

важливим для систем автентифікації, оскільки передача облікових даних та токенів повинна відбуватися виключно у зашифрованому вигляді, при цьому відсутність HTTPS у сучасних браузерах призводить до обмежень функціональності, включаючи блокування доступу до буфера обміну або некоректну роботу політик безпеки.

Архітектурна схема взаємодії при цьому трансформується у більш структуровану модель, де Traefik приймає запит, виконує TLS-термінацію, після чого передає його до системи автентифікації через механізм Forward Auth, і лише після успішної перевірки доступу перенаправляє запит до внутрішнього сервісу, що дозволяє чітко розділити рівні маршрутизації та безпеки [28].

Додатковою перевагою виступає підтримка динамічної конфігурації, де Traefik інтегрується з Docker та автоматично виявляє нові контейнери, використовуючи механізм "labels" для визначення правил маршрутизації, у результаті чого додавання нового сервісу не потребує зміни центральної конфігурації, а здійснюється шляхом опису параметрів безпосередньо у контейнері [28].

У поєднанні з механізмом Forward Auth система дозволяє реалізувати прозорий контроль доступу, де кожен запит перевіряється через Authentik, і лише після успішної автентифікації користувач отримує доступ до ресурсу, що забезпечує єдину модель безпеки для всіх сервісів.

Масштабування системи. Після впровадження базової інфраструктури наступним логічним етапом стає розширення можливостей системи та її інтеграція у корпоративне середовище, де виникає потреба централізованого керування користувачами, підвищення рівня безпеки та забезпечення стабільної роботи під зростаючим навантаженням, при цьому одним із напрямів виступає інтеграція з каталогами користувачів на кшталт LDAP або Active Directory, що дозволяє об’єднати всі облікові записи в єдину систему, усунути дублювання акаунтів, централізувати управління правами доступу та забезпечити відповідність корпоративним стандартам безпеки, оскільки автентифікація починає спиратися на вже існуючу інфраструктуру ідентифікації, а не на ізольовані локальні бази.

Паралельно з цим доцільним напрямом розвитку стає впровадження сучасніших механізмів багатофакторної автентифікації, зокрема стандарту FIDO2, який базується на використанні апаратних ключів безпеки, наприклад пристроїв типу YubiKey, де на відміну від TOTP відсутня передача одноразових кодів і використовується криптографічний виклик-відповідь, прив’язаний до конкретного домену, що значно знижує ризики фішингу та перехоплення даних, при цьому процес автентифікації відбувається через генерацію унікального криптографічного запиту, підтвердження користувачем фізичною взаємодією з пристроєм та перевірку підпису на серверній стороні, що формує більш стійку модель захисту порівняно з класичними методами.

Окрему увагу займає горизонтальне масштабування, яке стає необхідним у разі збільшення кількості користувачів і запитів, оскільки архітектура системи дозволяє додавати додаткові worker-контейнери для обробки фонових задач, розгортати Redis у вигляді окремого або кластеризованого сервісу для ефективного кешування сесій та тимчасових даних, а також масштабувати PostgreSQL через реплікацію або розподілення навантаження, завдяки чому досягається баланс між продуктивністю та відмовостійкістю, при цьому запити користувачів можуть оброблятися паралельно кількома компонентами без втрати узгодженості станів (див. рис. 3.6).

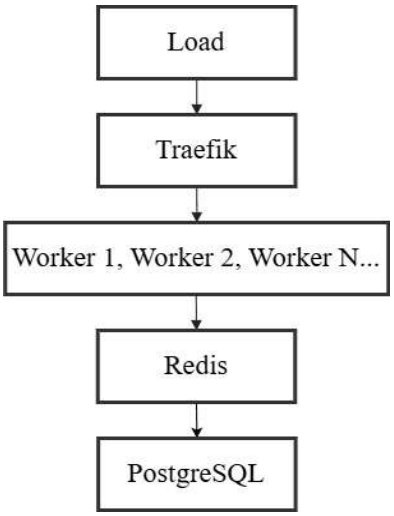


Рисунок 3.6 – Горизонтальне масштабування worker-компонентів

ВИСНОВКИ

Реалізація системи багатофакторної автентифікації на базі Fedora Linux та Authentik продемонструвала можливість побудови ізольованої та керованої архітектури з використанням контейнеризації Docker, де кожен компонент функціонує у власному середовищі, а взаємодія між ними забезпечується через внутрішню мережу та DNS-механізми, при цьому обрана топологія з використанням Forward Auth дозволяє відокремити логіку безпеки від прикладних сервісів і централізувати контроль доступу без модифікації цільових ресурсів.

Конфігурація компонентів системи та створення MFA-потоку на основі TOTP підтвердили коректність роботи механізмів автентифікації, де поєднання перевірки пароля та одноразового коду забезпечує підвищений рівень захисту, а використання контейнерної інфраструктури спрощує розгортання та повторюваність середовища, при цьому внутрішні алгоритми обробки запитів, асинхронна робота worker-процесів і механізми керування сесіями формують цілісну модель взаємодії користувача з системою.

Практична інтеграція тестового сервісу та перевірка сценаріїв доступу підтвердили працездатність підходу, де неавторизовані запити блокуються, коректна автентифікація з MFA надає доступ, а помилки перевірки призводять до відмови, при цьому використання діагностичного сервісу дозволило наочно оцінити поведінку системи та ефективність механізму Forward Auth.

Аналіз ефективності виявив обмеження через відсутність кешування, що проявляється у зростанні споживання пам’яті та зниженні продуктивності, що обґрунтовує використання Redis, а також важливість синхронізації часу для стабільності TOTP, подальший розвиток передбачає перенесення архітектури на рівень мережевого вузла з централізованим контролем доступу за моделлю Zero Trust, впровадження реверс-проксі з HTTPS, масштабування компонентів та інтеграцію з корпоративними сервісами, що формує повноцінне інфраструктурне рішення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Password security: past, present, future. National Institute of Standards and Technology. URL: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-63b.pdf (дата звернення: 25.02.2026).
2. Multi-Factor Authentication (MFA). Microsoft. URL: https://learn.microsoft.com/en-us/azure/active-directory/authentication/concept-mfa-howitworks (дата звернення: 28.02.2026).
3. What is IAM? IBM. URL: https://www.ibm.com/topics/identity-access-management (дата звернення: 03.03.2026).
4. Authentication factors explained. Okta. URL: https://www.okta.com/identity-101/authentication-factors/ (дата звернення: 06.03.2026).
5. TOTP: Time-Based One-Time Password Algorithm. Internet Engineering Task Force. URL: https://datatracker.ietf.org/doc/html/rfc6238 (дата звернення: 10.03.2026).
6. HOTP Algorithm Specification. Internet Engineering Task Force. URL: https://datatracker.ietf.org/doc/html/rfc4226 (дата звернення: 13.03.2026).
7. FIDO2 Overview. FIDO Alliance. URL: https://fidoalliance.org/fido2/ (дата звернення: 17.03.2026).
8. OAuth 2.0 Framework. Internet Engineering Task Force. URL: https://datatracker.ietf.org/doc/html/rfc6749 (дата звернення: 20.03.2026).
9. OpenID Connect Core 1.0. OpenID Foundation. URL: https://openid.net/specs/openid-connect-core-1_0.html (дата звернення: 23.03.2026).
10. SAML Technical Overview. OASIS. URL: https://docs.oasis-open.org/security/saml/v2.0/ (дата звернення: 27.03.2026).
11. Identity Providers architecture. Auth0. URL: https://auth0.com/docs/get-started/identity-fundamentals (дата звернення: 30.03.2026).
12. Forward Authentication concept. Traefik Labs. URL: https://doc.traefik.io/traefik/middlewares/http/forwardauth/ (дата звернення: 03.04.2026).
13. Docker networking basics. Docker Inc.. URL: https://docs.docker.com/network/ (дата звернення: 06.04.2026). [1.4.3]
14. Securing network devices. Cisco. URL: https://www.cisco.com/c/en/us/products/security/index.html (дата звернення: 10.04.2026).
15. Session management best practices. OWASP Foundation. URL: https://owasp.org/www-project-cheat-sheets/cheatsheets/Session_Management_Cheat_Sheet.html (дата звернення: 13.04.2026).
16. Logging and monitoring. OWASP Foundation. URL: https://owasp.org/www-project-top-ten/ (дата звернення: 16.04.2026).
17. Fedora Linux documentation. Fedora Project. URL: https://docs.fedoraproject.org/ (дата звернення: 20.04.2026).

18. Docker Compose overview. Docker Inc.. URL: https://docs.docker.com/compose/ (дата звернення: 23.04.2026).
19. Authentik documentation. Authentik. URL: https://goauthentik.io/docs/ (дата звернення: 25.04.2026).
20. Google Authenticator usage. Google. URL: https://support.google.com/accounts/answer/1066447 (дата звернення: 28.04.2026).
21. Docker DNS resolution. Docker Inc.. URL: https://docs.docker.com/config/containers/container-networking/ (дата звернення: 01.05.2026).
22. Reverse proxy patterns. NGINX. URL: https://www.nginx.com/resources/glossary/reverse-proxy-server/ (дата звернення: 02.05.2026).
23. MFA implementation patterns. Okta. URL: https://developer.okta.com/docs/guides/mfa/ (дата звернення: 02.05.2026).
24. Redis in-memory database. Redis Ltd.. URL: https://redis.io/docs/ (дата звернення: 03.05.2026).
25. Python memory management. Python Software Foundation. URL: https://docs.python.org/3/c-api/memory.html (дата звернення: 03.05.2026).
26. NTP protocol overview. Internet Engineering Task Force. URL: https://datatracker.ietf.org/doc/html/rfc5905 (дата звернення: 04.05.2026).
27. Zero Trust Architecture. National Institute of Standards and Technology. URL: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-207.pdf (дата звернення: 05.05.2026).
28. Traefik Proxy documentation. Traefik Labs. URL: https://doc.traefik.io/traefik/ (дата звернення: 05.05.2026).
29. Охріменко Н. О., Бученко І. А. Система багатофакторної автентифікації для доступу до приватної серверної інфраструктури на основі контейнеризованого сервісу. «Сучасний стан та перспективи розвитку IoT» : VII Науково-техн. конф., м. Київ, 20 квітня 2026 р. С.254-257.
30. Охріменко Н. О., Бученко І. А. Система централізованого контролю доступу з використанням федеративної ідентифікації та Forward Authentication у приватній інфраструктурі. «Застосування програмного забезпечення в ІКТ» : VII Всеукр. науково-техн. конф., м. Київ, 23 квітня 2026 р. (подано до публікації).

Додаток А

Конфігурація системи MFA на базі Authentik (Docker Compose)

```
services:
  postgresql:
    env_file:
    - .env
    environment:
      POSTGRES_DB: ${PG_DB:-authentik}
      POSTGRES_PASSWORD: ${PG_PASS:?database password required}
      POSTGRES_USER: ${PG_USER:-authentik}
    healthcheck:
      interval: 30s
      retries: 5
      start_period: 20s
      test:
      - CMD-SHELL
      - pg_isready -d $${POSTGRES_DB} -U $${POSTGRES_USER}
      timeout: 5s
    image: docker.io/library/postgres:16-alpine
    restart: unless-stopped
    volumes:
    - database:/var/lib/postgresql/data
  server:
    command: server
    depends_on:
      postgresql:
        condition: service_healthy
    env_file:
    - .env
    environment:
      AUTHENTIK_POSTGRESQL__HOST: postgresql
      AUTHENTIK_POSTGRESQL__NAME: ${PG_DB:-authentik}
      AUTHENTIK_POSTGRESQL__PASSWORD: ${PG_PASS}
      AUTHENTIK_POSTGRESQL__USER: ${PG_USER:-authentik}
      AUTHENTIK_SECRET_KEY: ${AUTHENTIK_SECRET_KEY:?secret key required}
    image: ${AUTHENTIK_IMAGE:-ghcr.io/goauthentik/server}:${AUTHENTIK_TAG:-
2026.2.2}
    ports:
    - ${COMPOSE_PORT_HTTP:-9000}:9000
    - ${COMPOSE_PORT_HTTPS:-9443}:9443
    - 8080:9000
    restart: unless-stopped
    shm_size: 512mb
    volumes:
    - ./data:/data
    - ./custom-templates:/templates
  worker:
    command: worker
    depends_on:
      postgresql:
        condition: service_healthy
    env_file:
    - .env
    environment:
      AUTHENTIK_POSTGRESQL__HOST: postgresql
      AUTHENTIK_POSTGRESQL__NAME: ${PG_DB:-authentik}
      AUTHENTIK_POSTGRESQL__PASSWORD: ${PG_PASS}
      AUTHENTIK_POSTGRESQL__USER: ${PG_USER:-authentik}
      AUTHENTIK_SECRET_KEY: ${AUTHENTIK_SECRET_KEY:?secret key required}
```

Продовження додаток А

Конфігурація системи MFA на базі Authentik (Docker Compose)

```
    image: ${AUTHENTIK_IMAGE:-ghcr.io/goauthentik/server}:${AUTHENTIK_TAG:-
2026.2.2}
    restart: unless-stopped
    shm_size: 512mb
    user: root
    volumes:
    - /var/run/docker.sock:/var/run/docker.sock
    - ./data:/data
    - ./certs:/certs
    - ./custom-templates:/templates
  whoami:
    image: traefik/whoami
    container_name: whoami
    restart: unless-stopped
    networks:
      - default
volumes:
  database:
    driver: local
```

Додаток Б

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

(Презентація)

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
Кафедра комп'ютерної інженерії



КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня бакалавра

НА ТЕМУ: «СИСТЕМА БАГАТОФАКТОРНОЇ АВТЕНТИФІКАЦІЇ ДЛЯ ДОСТУПУ ДО ПРИВАТНОЇ СЕРВЕРНОЇ ІНФРАСТРУКТУРИ»

Виконав студент групи КІД-41
Охріменко Назарій Олександрович
Керівник кваліфікаційної роботи:
Бученко Ігор Анатолійович,
старший викладач кафедри КІ

Київ – 2026

Мета, об’єкт, предмет дослідження

Мета роботи – зменшення ризиків несанкціонованого доступу та спрощення контролю доступу за рахунок MFA, Zero Trust і Forward Authentication

Об’єкт дослідження – процес забезпечення безпечного доступу до сервісів у приватній мережевій інфраструктурі

Предмет дослідження – засоби та технології реалізації MFA, централізованого управління доступом і захисту веб-сервісів у контейнеризованому середовищі

Актуальність роботи

У роботі представлено теоретичні основи сучасних систем автентифікації та контролю доступу, розглянуто обмеження парольних підходів, принципи MFA, алгоритми TOTP/HOTP і протоколи OAuth 2.0, OpenID Connect та SAML, проаналізовано архітектуру Identity Provider і механізм Forward Authentication як спосіб захисту сервісів без їх модифікації, а також підходи до інтеграції в контейнерні середовища з урахуванням масштабованості, ізоляції та аудиту подій безпеки.

Особливу увагу приділено практичній реалізації прототипу на базі Fedora Linux із використанням Authentik і Docker, включаючи розгортання компонентів, налаштування MFA на основі TOTP, інтеграцію тестового сервісу через Forward Auth і перевірку сценаріїв доступу, а також аналіз ефективності, виявлення вузьких місць, обґрунтування використання кешування та подальший перехід до архітектури Edge Security Gateway із централізованим контролем доступу, масштабуванням і інтеграцією у корпоративне середовище.

Архітектура рішення (основа)

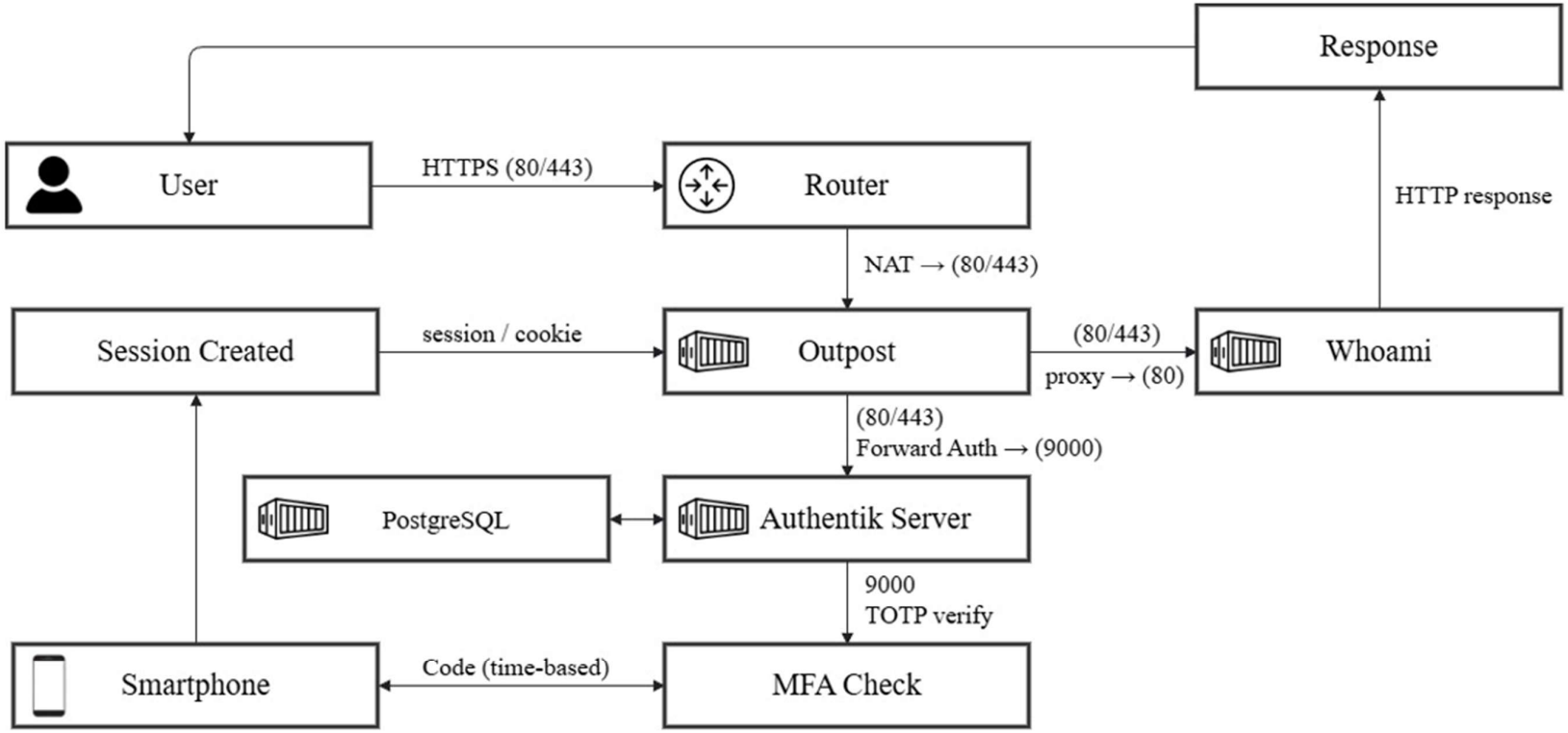


Рис. 2.1 – Архітектура системи MFA на базі Authentik

Архітектура рішення (основа)

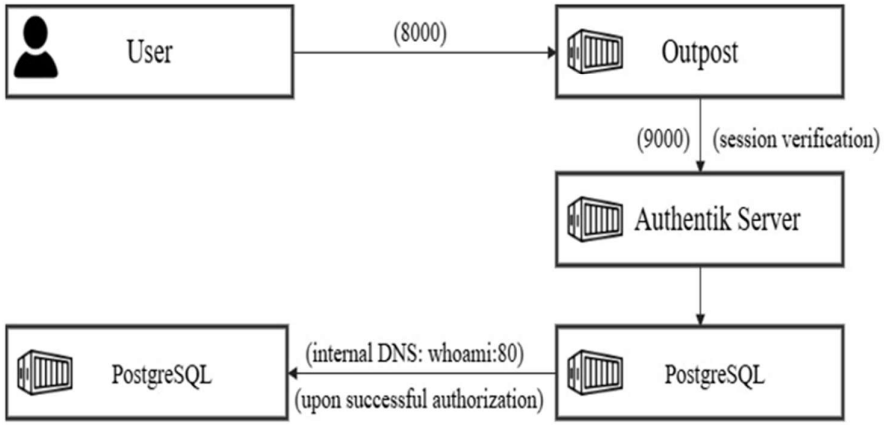


Рис. 2.3 – Базова топологія мережі з MFA

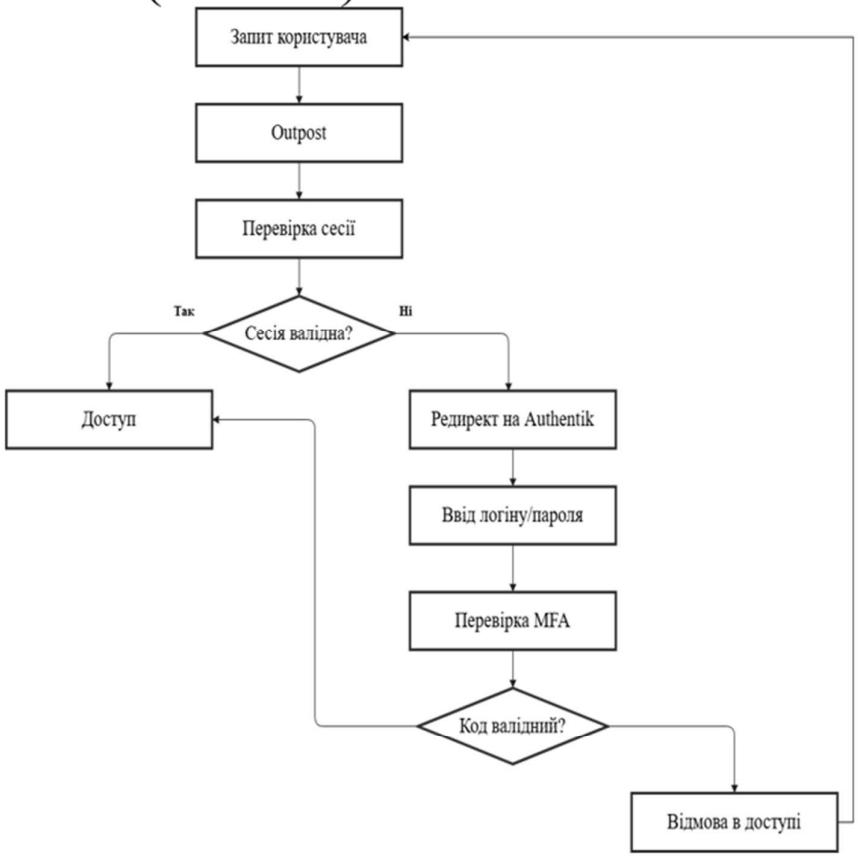


Рис. 2.4 – Концепція Forward Auth у системі

Практичне середовище

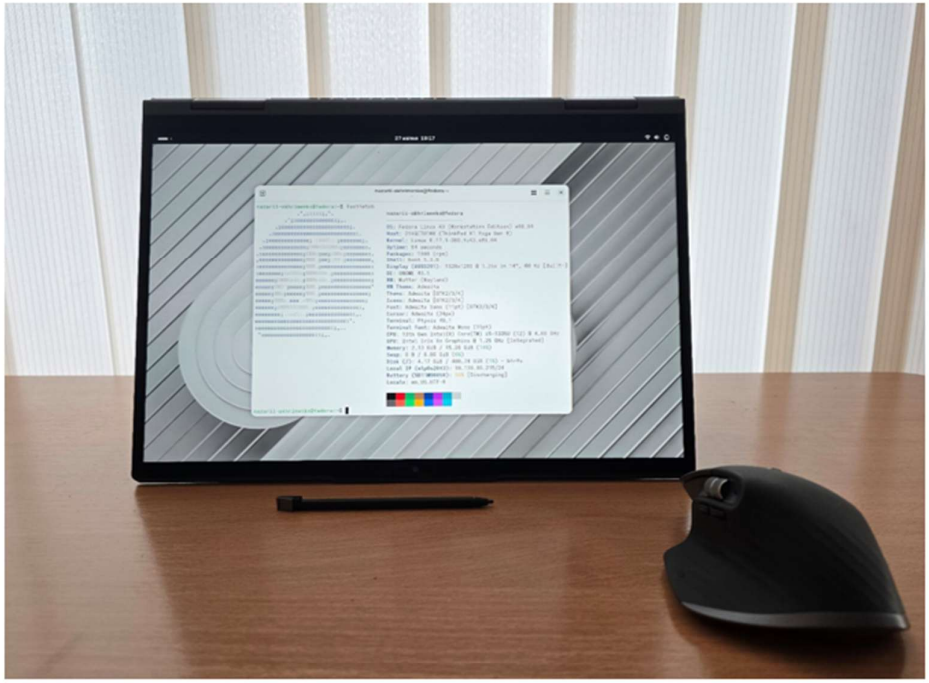


Рис. 2.2 – Апаратне середовище на базі Lenovo ThinkPad X1 Yoga Gen 8

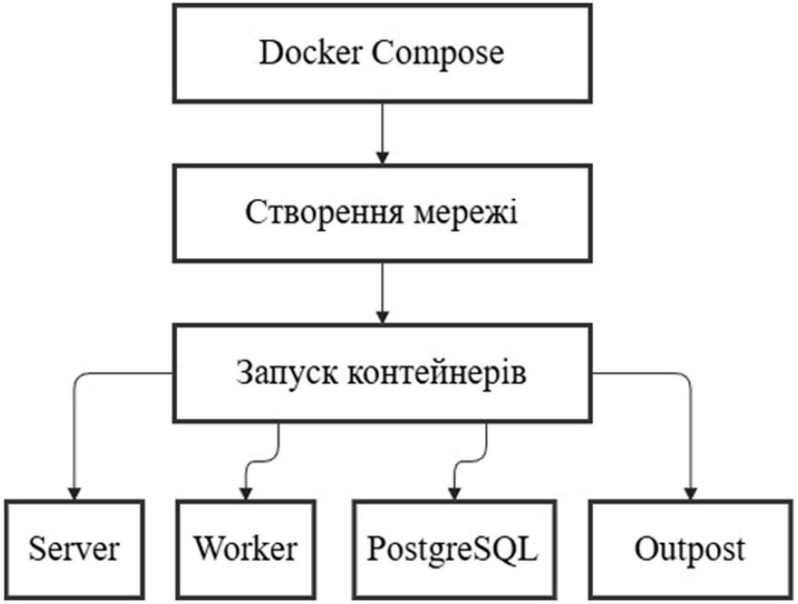


Рис. 2.5 – Послідовність розгортання Docker-стеку Authentik

Розгортання системи



Рис. 2.6 – Вивід терміналу при розгортанні Docker Compose

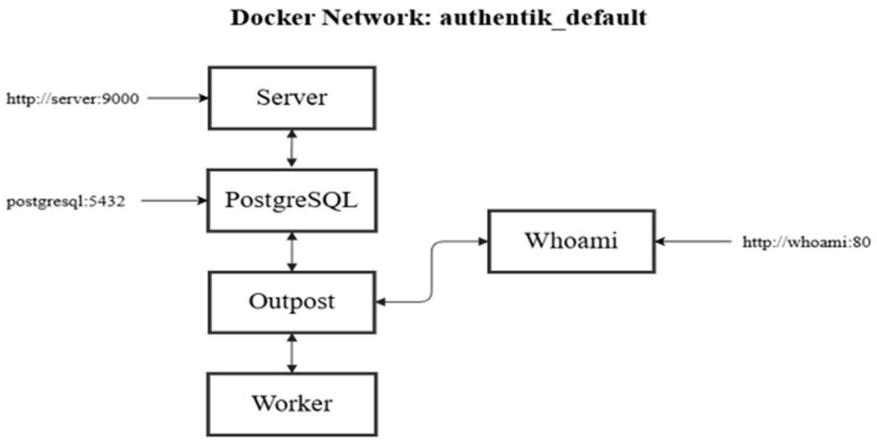


Рис. 2.11 – Внутрішня DNS-резолюція Docker

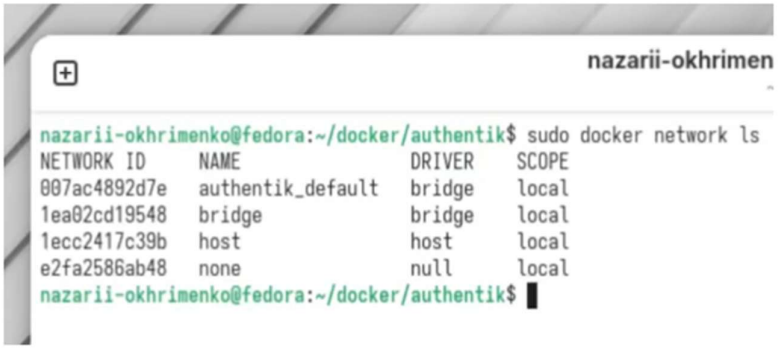


Рис. 2.12 – Конфігурація мережі контейнерів

Налаштування Authentik

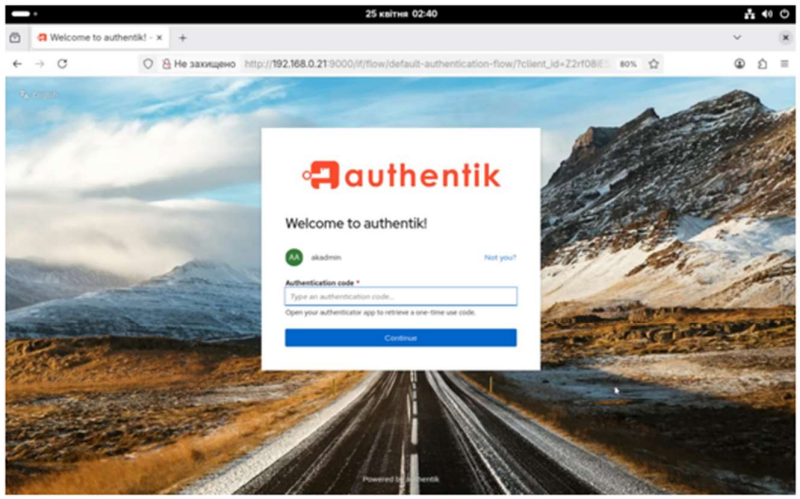


Рис. 2.7 – Веб-інтерфейс Authentik (панель адміністратора)

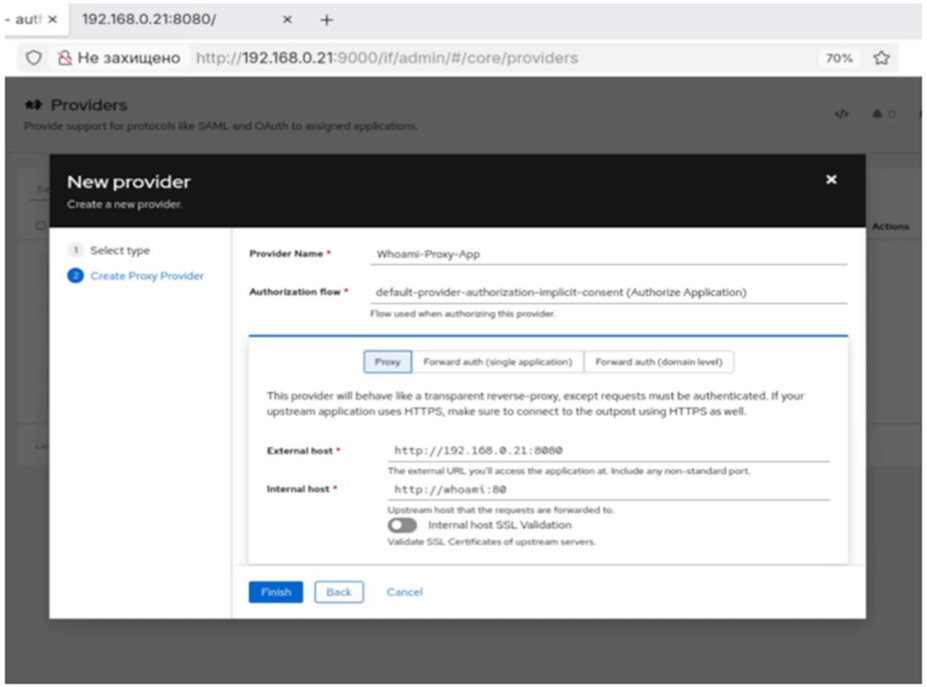


Рис. 2.8 – Налаштування провайдера автентифікації

Реалізація MFA (TOTP)

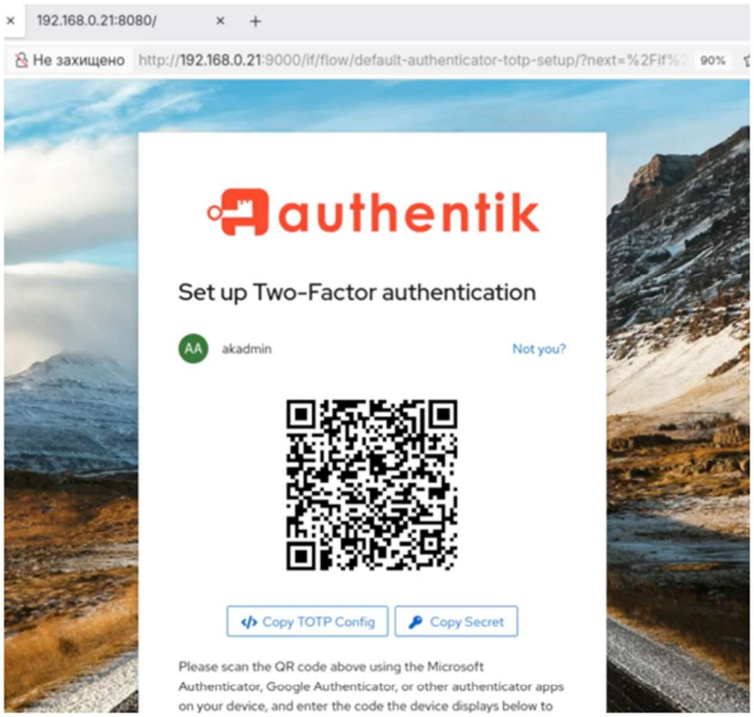


Рис. 2.9 – Генерація QR-коду для TOTP

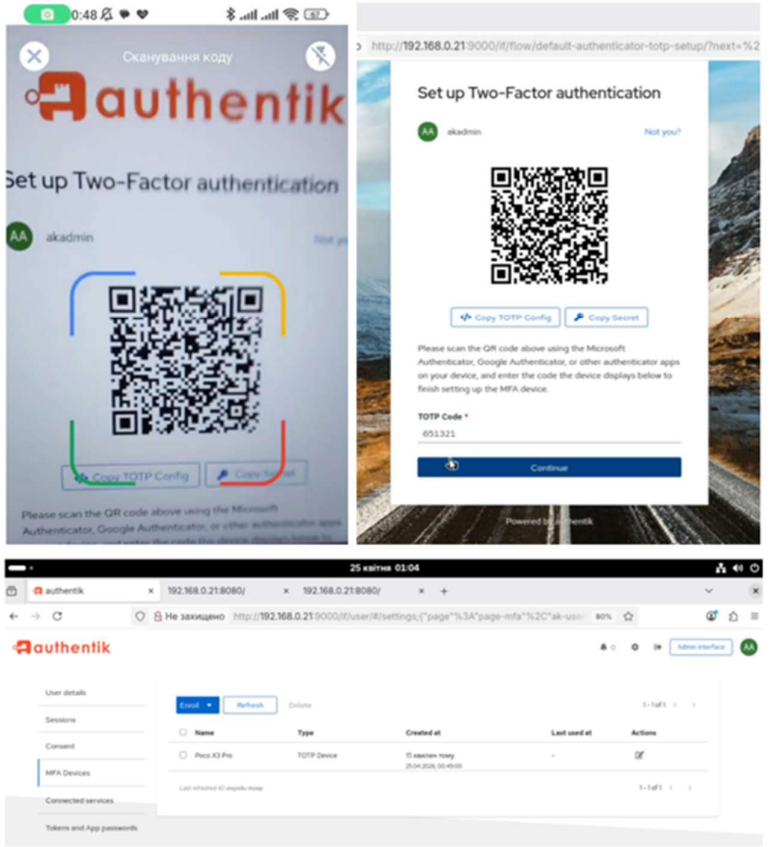


Рис. 2.10 – Прив'язка сервісу Authentik до пристрою в Google Authenticator

Алгоритм роботи системи

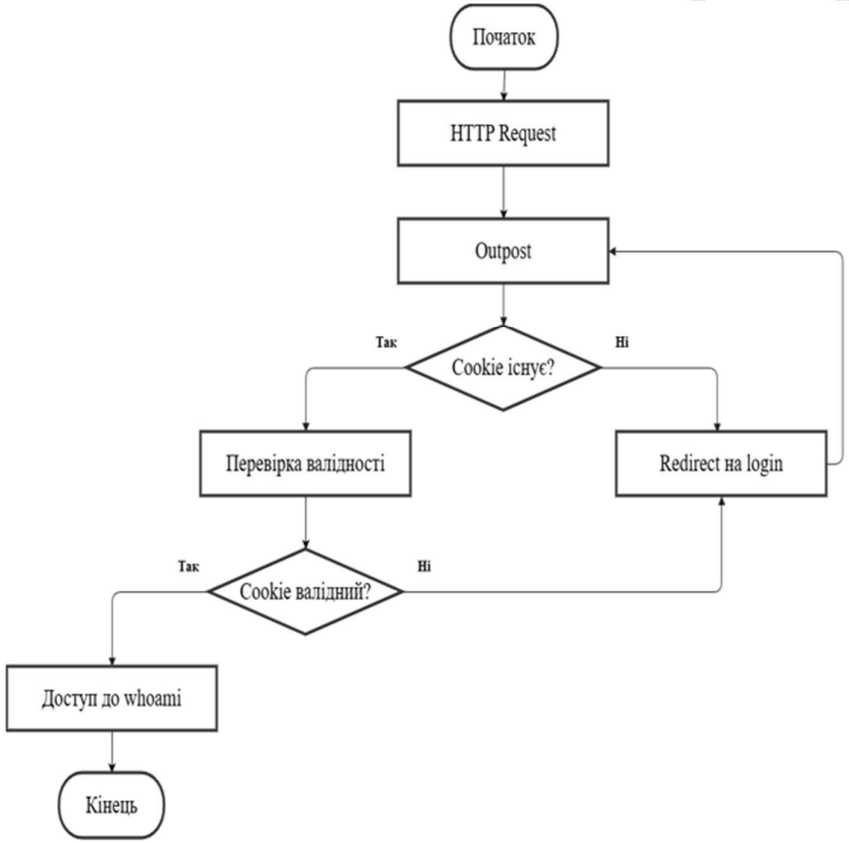


Рис. 2.13 – Алгоритм перехоплення HTTP-запиту (Forward Auth)

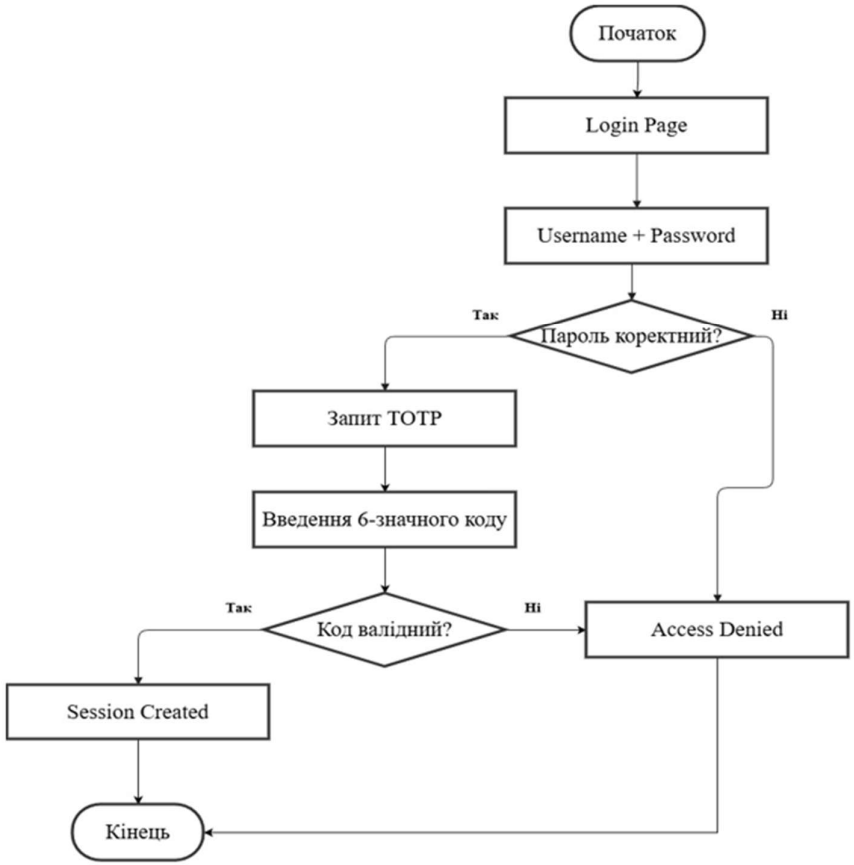


Рис. 2.14 – Алгоритм двофакторної перевірки (MFA, TOTP)

Практична інтеграція



Рис. 2.16 – Розгортання Whoami сервісу у Docker

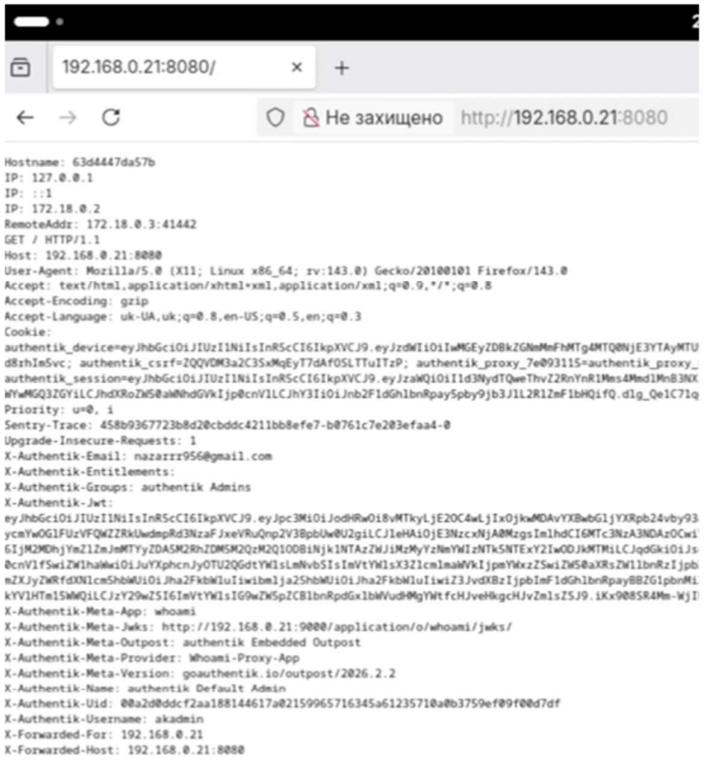


Рис. 2.17 – Відображення HTTP-запиту у Whoami

Результати тестування

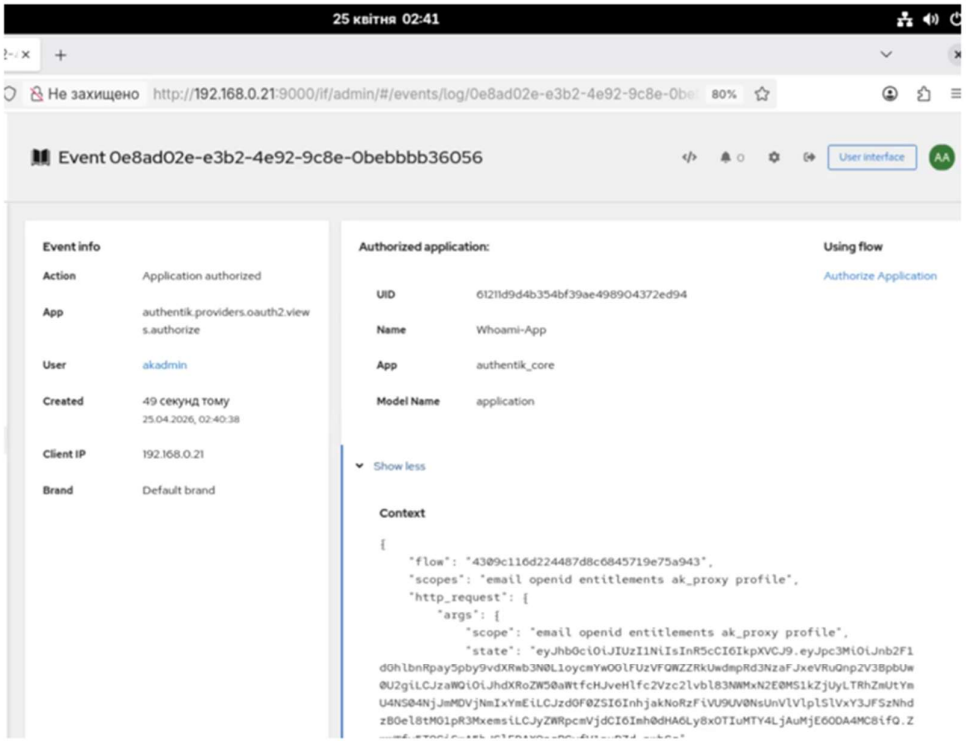


Рис. 2.18 – Успішна автентифікація та доступ до сервісу

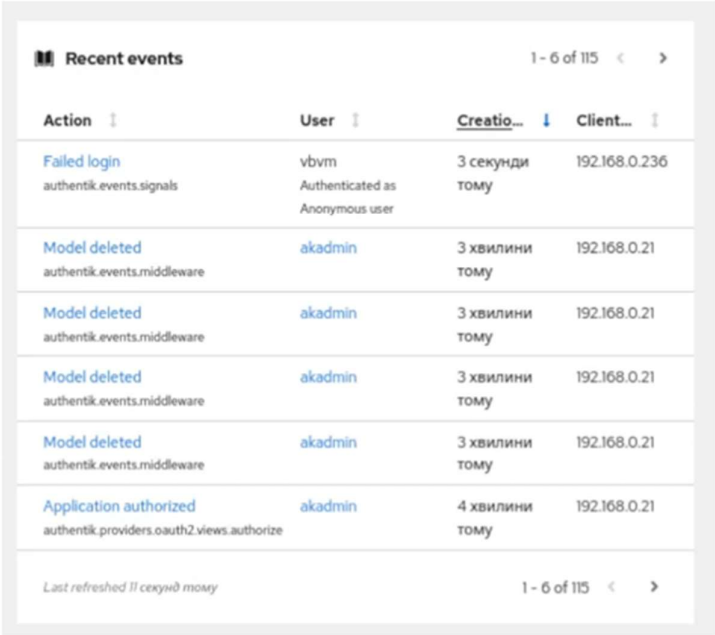


Рис. 2.19 – Помилка введення TOTP-коду

Результати тестування

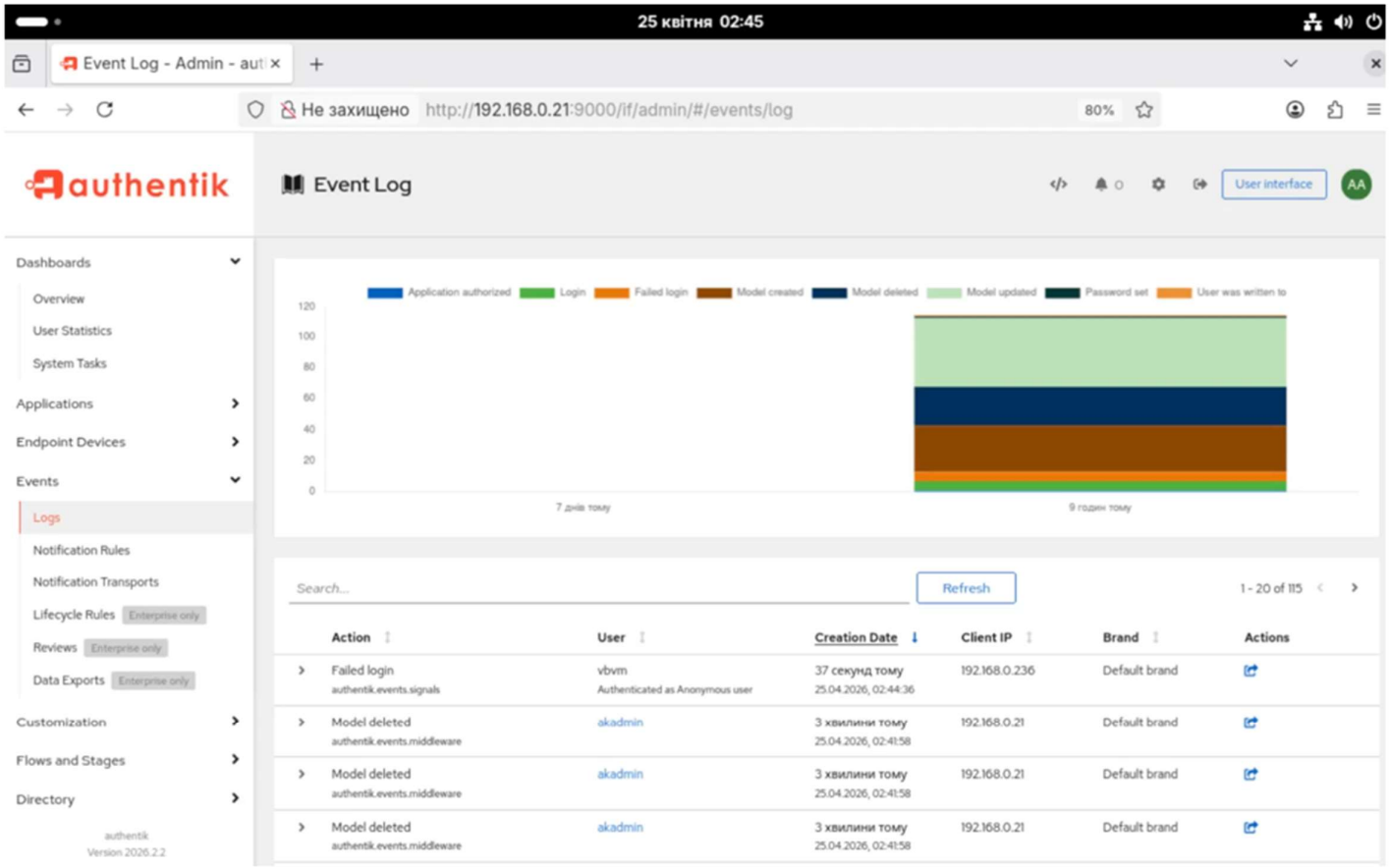


Рис. 2.20 – Логи автентифікації у веб-інтерфейсі Authentik

Аналіз продуктивності

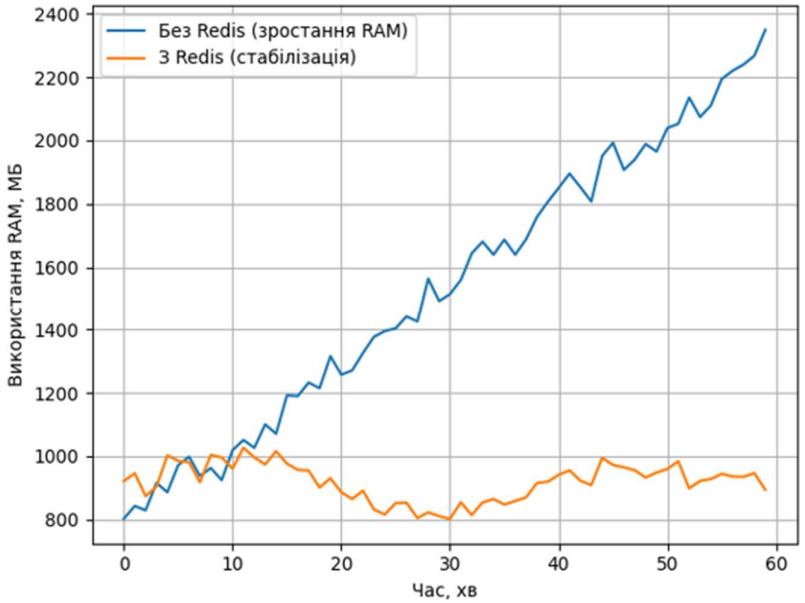


Рис. 3.1 – Динаміка використання оперативної пам'яті (memory leak)

Табл. 3.1 – Показники використання ресурсів системи під навантаженням

Стан системи	CPU (%)	RAM (МБ)	Час відповіді (мс)	Примітка
Без Redis (початок)	15–25	~800	~120	Стабільна робота
Без Redis (навантаження)	40–70	>2000	~300–500	Зростання пам'яті
Після оптимізації (Redis)	20–35	~900	~100–150	Стабілізація
Пікове навантаження (Redis)	50–65	~1200	~200	Контрольоване

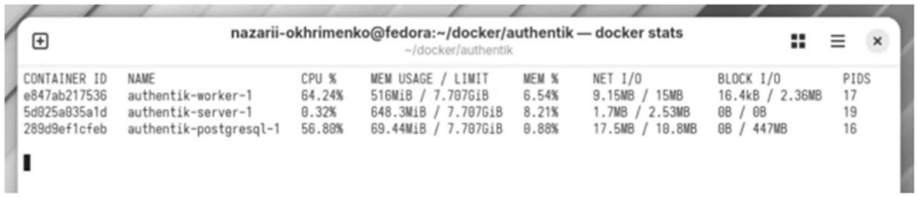


Рис. 3.2 – Вплив відсутності кешування на продуктивність

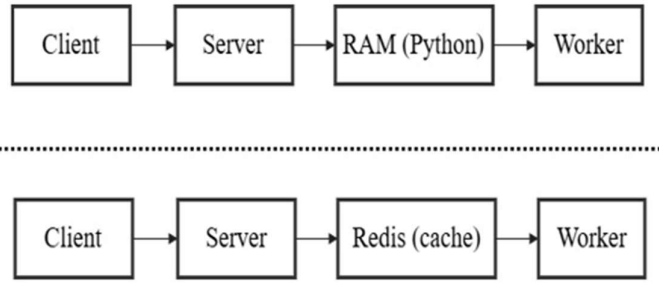


Рис. 3.3 – Метод роботи системи з Redis та без нього

Оптимізація та масштабування

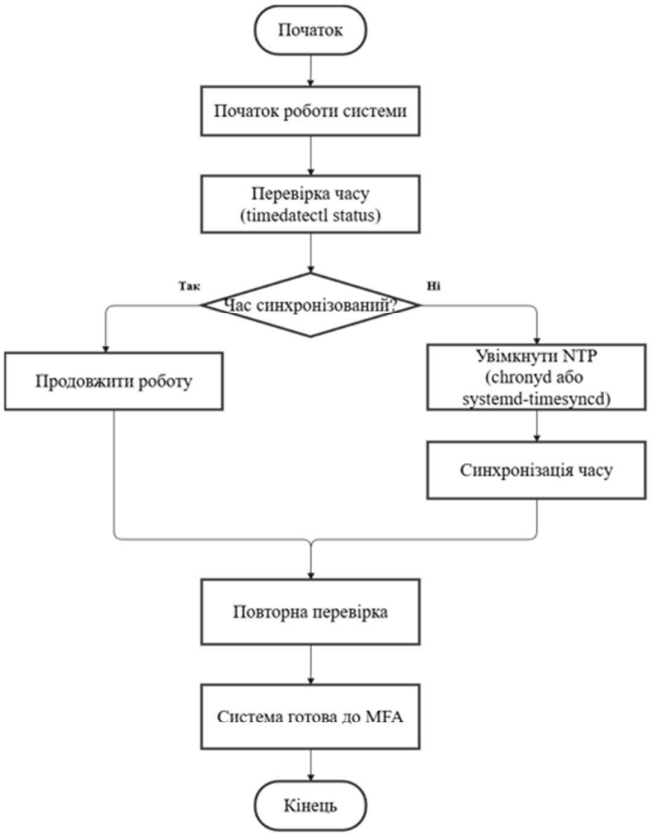


Рис. 3.4 – Метод NTP-синхронізації в системі

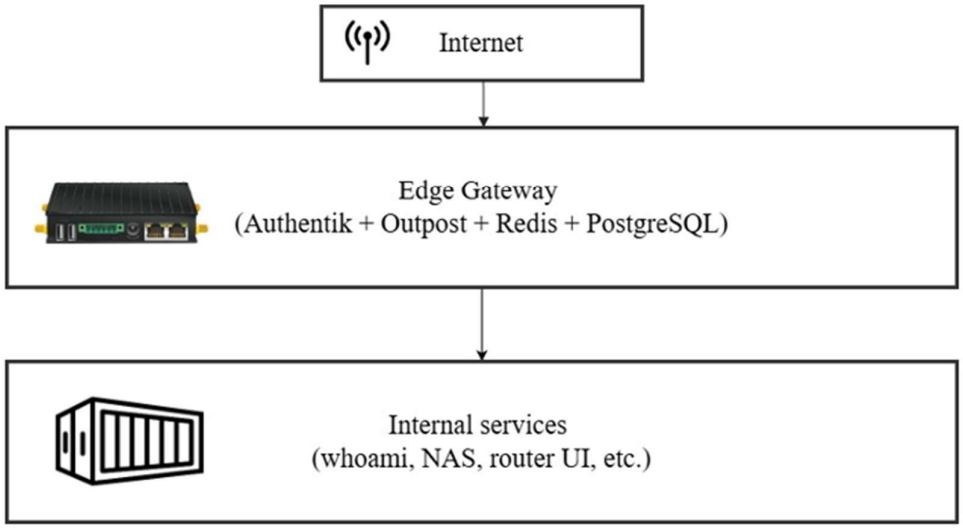


Рис. 3.5 – Схема розміщення компонентів у мережі (DMZ)

Висновки

Реалізація системи MFA на базі Fedora Linux та Authentik підтвердила можливість побудови ізольованої контейнерної архітектури з централізованим контролем доступу. Використання Docker та підходу Forward Auth дозволило відокремити механізми безпеки від прикладних сервісів, забезпечивши керованість, масштабованість і відсутність необхідності змін у цільових ресурсах. Налаштування MFA на основі TOTP продемонструвало коректну роботу автентифікації, де поєднання пароля та одноразового коду підвищує рівень захисту, а інтеграція тестового сервісу підтвердила ефективність блокування неавторизованих запитів і надання доступу лише після успішної перевірки.

Аналіз ефективності виявив залежність продуктивності від наявності кешування та точності системного часу, що обґрунтовує використання Redis і NTP-синхронізації як обов'язкових компонентів. Подальший розвиток передбачає масштабування архітектури до рівня мережевого вузла (Edge Gateway), впровадження HTTPS через реверс-проксі, а також інтеграцію з корпоративними сервісами, що дозволяє реалізувати повноцінну модель централізованого контролю доступу за принципами Zero Trust.

Результати впровадження

1. Охріменко Н. О., Бученко І. А. Система багатофакторної автентифікації для доступу до приватної серверної інфраструктури на основі контейнеризованого сервісу. «Сучасний стан та перспективи розвитку IoT» : VII Науково-техн. конф., м. Київ, 20 квітня 2026 р. С.254-257.
2. Охріменко Н. О., Бученко І. А. Система централізованого контролю доступу з використанням федеративної ідентифікації та Forward Authentication у приватній інфраструктурі. «Застосування програмного забезпечення в ІКТ» : VII Всеукр. науково-техн. конф., м. Київ, 23 квітня 2026 р. (подано до публікації).