Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Факультет інформатики
Кафедра інформатики

**Кваліфікаційна робота**

освітній ступінь – магістр

на тему: **“Розробка масштабованої SaaS-платформи для валідації електронних адрес в реальному часі”**

Виконав: студент 2-го року навчання
освітньої програми «Комп’ютерні науки»,
спеціальності 122 Комп’ютерні науки
Ярошепта Богдан Павлович
Керівник: Нагірна А. М.,
кандидат фіз.-мат. наук, доцент
Рецензент: Афонін А.О.,
кандидат фізико-математичних наук, доцент

Кваліфікаційна робота захищена
з оцінкою _____
Секретар ЕК _____
«____» _____ 20____ р.

Київ 2025

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Факультет інформатики

Кафедра інформатики

ЗАТВЕРДЖУЮ

Зав. кафедри інформатики,

доцент, к.ф-м.н.,

С. С. Гороховський

_____

(підпис)

„____”_____2025 р.

**ІНДИВІДУАЛЬНЕ ЗАВДАННЯ**

на дипломну роботу

студенту 2-го курсу, факультету інформатики

Ярошепті Богдану Павловичу

Розробити Масштабований застосунок для валідації електронних адрес в реальному часі

Зміст ТЧ до магістерської роботи:

Зміст

Анотація

Вступ

1 Аналіз предметної області

2 Архітектура та технології розробки

3 Модуль валідації адрес електронної пошти

4 Реалізація платформи

5 Оцінка ефективності та можливості розвитку

Висновки

Список літератури

Додатки

Дата видачі „___” _____ 2024 р.  Керівник _____

(підпис)

Завдання отримав _____

(підпис)

**Графік підготовки кваліфікаційної роботи до захисту**

Графік узгоджено «_____» _____ 2024 р.

| № п/п | Назва етапу дипломного проекту (роботи) | Термін виконання етапу | Примітка |
|---|---|---|---|
| 1. | Отримання завдання на дипломну роботу | 01.10.2024 | |
| 2. | Огляд технічної літератури за темою роботи | 15.10.2024 | |
| 3. | Аналіз існуючих рішень та формулювання вимог до платформи | 01.11.2024 | |
| 4. | Розробка загальної архітектури SaaS-платформи | 15.11.2024 | |
| 5. | Реалізація MVP-версії серверної частини (NestJS, Supabase, Kafka, RabbitMQ) | 10.12.2024 | |
| 6. | Реалізація модуля валідації електронних адрес (SMTP, DNS, MX, ML) | 05.01.2025 | |
| 7. | Інтеграція черг повідомлень, кешування, захисту | 25.01.2025 | |
| 8. | Реалізація клієнтської частини (React, SSE, Dashboard) | 15.02.2025 | |
| 9. | Проведення тестування, моніторингу та оптимізації | 10.03.2025 | |
| 10. | Написання пояснювальної записки | 15.04.2025 | |

| | | | |
|---|---|---|---|
| 11. | Створення слайдів для доповіді та чорнової версії доповіді | 25.05.2025 | |
| 12. | Попередній захист, отримання рецензій, аналіз зауважень | 30.05.2025 | |
| 13. | Остаточне оформлення роботи та презентації | 05.06.2025 | |
| 14. | Захист магістерської роботи (проекту) | 11-12.06.2025 | |

# ЗМІСТ

Стор.

# АНОТАЦІЯ

У магістерській роботі представлено процес розробки масштабованої SaaS-платформи для валідації електронних адрес у реальному часі. У ході роботи було спроєктовано мікросервісну архітектуру, реалізовано основний API на базі NestJS, а також окремий сервіс валідації із підтримкою асинхронної обробки через черги повідомлень (Kafka , RabbitMQ) [1][2][3]. Для забезпечення оновлень у реальному часі використано технологію Server-Sent Events [4]. Також реалізовано веб-застосунок на React.js для завантаження файлів зі списками електронних адрес, перегляду статусу перевірок та аналітики.

Забезпечено підтримку синтаксичної, MX- і SMTP-верифікації , з урахуванням обмежень поштових серверів [5][6]. Особливу увагу приділено масштабуванню, безпеці даних, ізоляції сервісів та логуванню процесів перевірки. Результати тестування підтверджують ефективність запропонованого рішення для обробки великих обсягів даних у реальному часі.

# ВСТУП

У сучасних умовах електронна пошта залишається основним інструментом комунікації в бізнес сфері. Вона широко використовується для надсилання інформаційних, транзакційних і рекламних повідомлень. Проте ефективність розсилок залежить від якості списків адрес: недійсні чи тимчасові емейли спричиняють високий відсоток недоставки, блокування доменів і зниження репутації відправника.

Зважаючи на ці ризики, перевірка валідності адрес електронної пошти стала важливою задачею як для прикладної інженерії, так і для наукових досліджень. У технічній літературі та практиці поширеними є підходи, що включають синтаксичну перевірку згідно зі стандартами RFC 5322, аналіз DNS та MX-записів доменів, а також перевірку наявності адрес через SMTP-емуляцію [7][6].

Крім того, сучасні загрози, зокрема масова генерація фейкових акаунтів і автоматизовані атаки на реєстраційні форми, вимагають більш глибокого інтелектуального аналізу імен користувачів у структурі електронних адрес. Це, в свою чергу, актуалізує потребу в створенні масштабованої та адаптивної платформи валідації, яка б дозволяла виявляти недійсні адреси на всіх рівнях перевірки.

У зв’язку з цим, доцільним є розробка сучасної SaaS-платформи, яка об’єднуватиме класичні мережеві підходи до перевірки електронних адрес із методами машинного навчання, підтримкою подієвої архітектури та асинхронної обробки запитів. Така система має працювати в реальному часі, бути стійкою до навантажень і легко масштабуватись.

У межах роботи запропоновано та реалізовано багаторівневий алгоритм валідації електронних адрес, який поєднує класичні методи

перевірки з інтелектуальними підходами на основі нейромережевих моделей.

Робота складається з п’яти розділів. У першому подано огляд методів валідації електронних адрес. У другому - сформульовано технічні вимоги та спроєктовано архітектуру платформи. У третьому - розглянуто реалізацію алгоритму перевірки. Четвертий розділ присвячено технічній реалізації. П’ятий - оцінці ефективності та перспективам розвитку системи.

**Об’єкт дослідження:** процес валідації електронних адрес у SaaS-середовищі.

**Предмет дослідження:** архітектура, алгоритми і технології перевірки валідності електронних адрес у реальному часі на основі мікросервісного підходу.

**Мета роботи:** розробити масштабовану платформу для автоматизованої валідації електронних адрес з використанням сучасних вебтехнологій, черг повідомлень і засобів інтелектуального аналізу.

**Постановка задачі:**

1. Провести огляд сучасних методів перевірки валідності електронних адрес та аналіз існуючих сервісів.
2. Сформулювати технічні вимоги до архітектури і функціональності платформи.
3. Спроєктувати мікросервісну архітектуру з підтримкою масштабування та асинхронної обробки.
4. Реалізувати багаторівневий алгоритм валідації, що включає перевірку синтаксису, DNS, SMTP та машинне навчання[6].
5. Здійснити тестування продуктивності, точності перевірки та виявити потенційні вузькі місця.
6. Запропонувати напрями розвитку системи та інтеграції в зовнішні сервіси.

# РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

## *1.1 Проблематика валідації електронних адрес у веб-застосунках*

У сучасних веб-застосунках електронна адреса є одним з основних ідентифікаторів користувача. Вона широко використовується для реєстрації, авторизації, відновлення доступу, розсилки повідомлень, маркетингових кампаній і транзакційних операцій. Водночас саме на цьому етапі системи часто стикаються з проблемами низької якості або підроблених контактних даних, що призводить до фінансових втрат, технічного навантаження та порушення вимог безпеки.

Класична перевірка електронної адреси у веб-застосунку зазвичай обмежується валідацією синтаксису (тобто формального формату адреси) за регулярним виразом. Такий підхід дозволяє відсіяти очевидно некоректні введення, однак не гарантує фактичної доступності адреси або її реального існування. Наприклад, адреса на домені, який не обслуговується поштовим сервером або є тимчасовим (disposable), успішно пройде синтаксичну перевірку, але не буде придатною для доставки повідомлень.

Серед основних проблем, пов’язаних із валідацією електронних адрес у веб-застосунках, варто виокремити наступні:

– Фальшиві або одноразові адреси. Багато користувачів свідомо вводять тимчасові адреси електронної пошти (через сервіси типу TempMail або GuerrillaMail), щоб уникнути подальших контактів із сервісом [8][9];
– Некоректні домени. Домени можуть бути неактивними, мати відсутні MX-записи або не мати поштових серверів взагалі;

- SMTP-пастки [10]. У ряді випадків поштові сервери навмисно вводять механізми, які завжди відповідають позитивно на перевірку адреси, навіть якщо вона не існує (так звані catch-all домени);
- Зловживання “keysmash”-реєстрацією. Автоматично згенеровані акаунти із випадковими іменами часто використовуються в бот-атаках або для масової реєстрації.

Ще однією проблемою є обмеження сторонніх SMTP-серверів , які можуть заблокувати занадто часті запити на перевірку або вважати такі запити за спам [6]. Це створює складність у реалізації прямої перевірки наявності адреси через емульований SMTP-діалог [6].

Із огляду на вищезазначене, актуальним є розробка багаторівневих систем валідації, які не лише аналізують синтаксис, а й перевіряють адресу на наявність у базах тимчасових сервісів, існування поштових DNS-записів, відповіді SMTP-серверів та ознаки автоматичної генерації. Такий комплексний підхід значно підвищує якість контактної інформації в базі користувачів та знижує ризики, пов’язані з невалідними або несправжніми електронними адресами.

## *1.2 Огляд існуючих рішень*

Ринок маркетингу через електронну пошту та масових розсилок активно зростає, що зумовило появу великої кількості сервісів, орієнтованих на валідацію електронних адрес з метою підвищення точності доставки повідомлень та очищення баз контактів. У цьому підрозділі розглянуто найпопулярніші з них - ZeroBounce , NeverBounce , Hunter, EmailListVerify, MailboxValidator та інші - з точки зору їхніх функцій, переваг і обмежень [11][12][13][14].

**ZeroBounce** - один із найбільш розвинених SaaS-сервісів для очищення списків електронних адрес. Він пропонує:

- перевірку формату та домену;
- фільтрацію одноразових, спам-пасток, abuse-адрес;
- перевірку SMTP-доступності [6];
- розпізнавання catch-all доменів;
- оцінку активності адреси на основі власних баз;

ZeroBounce надає API для інтеграції з CRM-системами та можливість масового завантаження файлів. Недоліком є обмежена прозорість алгоритмів перевірки та висока вартість при великій кількості запитів.

Сервіс **NeverBounce** спеціалізується на швидкій перевірці списків контактів. Основні можливості:

- валідація синтаксису, доменів, SMTP-серверів;
- розподіл адрес за категоріями: valid, invalid, catch-all, unknown;
- інтеграції з поширеними маркетинговими платформами (Mailchimp, HubSpot тощо);
- підтримка реального часу через REST API [15].

Платформа позиціонується як рішення “під ключ”, однак користувач не має контролю над логікою валідації, а також обмежений у кастомізації.

**Hunter** - сервіс, більш орієнтований на пошук адрес електронної пошти та перевірку достовірності корпоративних адрес. Його можливості включають:

- перевірку існування домену та MX-записів;
- базову SMTP-перевірку;
- виявлення ролей (admin@, info@, sales@) та одноразових адрес;
- API для перевірки у реальному часі.

Hunter менш гнучкий у плані глибокої технічної перевірки, проте широко використовується для побудови B2B-баз завдяки інтеграції з LinkedIn і CRM.

Також, потрібно звернути увагу на менш популярні сервіси з їх перевагами та недоліками:

- EmailListVerify - сервіс з багаторівневою перевіркою: DNS, SMTP, одноразові домени, спам-пастки. Має високу швидкість обробки, однак його SMTP-перевірка менш стабільна для catch-all доменів.
- MailboxValidator - пропонує як онлайн-перевірку, так і API. Відрізняється простотою, однак поступається конкурентам у точності.
- Clearout, Verifalia, BriteVerify - мають схожі функції, однак відрізняються в моделі ціноутворення та стабільності SMTP-перевірки.

Попри високу популярність і широкий функціонал, існуючі сервіси мають ряд спільних обмежень:

- Закритість алгоритмів. Валідаційна логіка часто є непрозорою і недоступною для адаптації до конкретних кейсів.
- Обмеження в API. У деяких сервісів відсутня можливість детального налаштування запитів або обробки в реальному часі.
- Висока вартість при великому обсязі перевірок, що робить такі сервіси недоступними для невеликих стартапів.
- Відсутність або слабка підтримка інтелектуального аналізу username-частини адреси, наприклад, виявлення keysmash або автоматичних акаунтів.

Аналіз ринку показує, що хоча сучасні сервіси забезпечують базові рівні перевірки електронних адрес, більшість із них не надає можливості гнучкого налаштування логіки валідації, не дозволяє глибоко адаптувати механізми перевірки під специфіку окремого бізнес-процесу або інтегруватися в існуючі високонавантажені системи з урахуванням реального часу. Це створює потребу у розробці масштабованої та адаптивної SaaS-платформи, яка б об'єднувала багаторівневу перевірку,

сучасну архітектуру, можливість інтеграції через API та швидку обробку запитів. Саме таку платформу реалізовано в межах даної роботи.

### *1.3 Основні вимоги до сучасної SaaS-платформи*

Розробка сучасної SaaS-платформи передбачає врахування як технічних, так і функціональних вимог, які забезпечують надійність, масштабованість, доступність і зручність використання сервісу. У контексті валідації електронних адрес такі вимоги стають ще більш критичними через потребу в реальному часі обробляти великі обсяги запитів, дотримуватися високих стандартів безпеки та забезпечувати стабільну роботу в умовах змінного навантаження.

До основних функціональних характеристик сучасної SaaS-платформи для валідації електронних адрес належать:

- Підтримка багаторівневої перевірки електронних адрес, включно з:
    1) валідацією формату;
    2) перевіркою на одноразовість (disposable email);
    3) DNS/MX-перевірками;
    4) визначенням наявності SMTP-серверів;
    5) емульованим SMTP-діалогом;
    6) семантичною оцінкою username-частини.
- Пакетна і поодинока перевірка - можливість перевіряти як великі списки адрес, так і окремі електронні скриньки через API або через інтерфейс користувача.
- Підтримка перевірки в реальному часі - особливо актуальна для інтеграцій з формами реєстрації, CRM та маркетинговими платформами, які фокусуються на електронних адресах.

- API-доступ для сторонніх систем з документованим REST/JSON-інтерфейсом.
- Інтерфейс користувача для завантаження файлів, перегляду результатів, аналітики й завантаження очищених списків.

Окрім базового функціоналу, платформа повинна відповідати низці нефункціональних вимог, які визначають її якість як SaaS-продукту:

- Масштабованість. Система повинна обробляти десятки тисяч запитів щодня без деградації продуктивності. Це передбачає підтримку горизонтального масштабування сервісів та черг повідомлень.
- Надійність і відмовостійкість. Платформа повинна продовжувати працювати навіть при часткових збоях сервісів або тимчасовій недоступності зовнішніх SMTP-серверів.
- Продуктивність. Середній час відповіді повинен залишатися низьким, незалежно від обсягу оброблюваних даних. Потрібна оптимізація SMTP-сесій, кешування DNS та MX-результатів.
- Безпека. Необхідне шифрування трафіку (HTTPS/TLS), контроль доступу, захист API-ключів, обмеження за IP-адресами, а також відповідність вимогам до захисту персональних даних (наприклад, GDPR).
- Моніторинг і логування. Повинна бути реалізована система спостереження за роботою компонентів платформи (наприклад, через Prometheus, Grafana , Loki), що дозволяє своєчасно реагувати на помилки [16][17][18].
- Кросплатформеність. Платформа має бути доступною на різних пристроях через веб-інтерфейс без втрати функціональності.
- Інтернаціоналізація (i18n). Підтримка декількох мов інтерфейсу для користувачів з різних країн.

У контексті SaaS-моделі також важливими є:

- Модель підписки або “pay-as-you-go” формат. Гнучка тарифікація залежно від кількості перевірок або запитів.
- Інтеграція з платіжними системами. Підтримка оплати через Stripe, PayPal тощо.
- Маркетингова аналітика. Підготовка до SEO-оптимізації та залучення трафіку через Google Search.
- Аналітика використання. Відстеження поведінки користувачів, популярності API та завантажених обсягів.

***Висновки до розділу 1***

У цьому розділі було проаналізовано предметну область валідації електронних адрес у контексті сучасних веб-застосунків. Встановлено, що ефективна перевірка електронних адрес є критично важливою для забезпечення якості взаємодії з користувачами, зниження показника недоставленої пошти, боротьби зі спамом та запобігання шахрайству. Проблематика полягає в обмеженості стандартних механізмів перевірки, зростанні обсягів даних і потребі у швидкій та точній обробці запитів у реальному часі.

Проведено огляд популярних рішень на ринку (ZeroBounce, NeverBounce, Hunter.io та ін.), які демонструють високий рівень інтеграції та базовий набір перевірок, проте часто мають обмеження щодо кастомізації, адаптивності до нестандартних сценаріїв використання або прозорості логіки роботи [19].

На основі аналізу сформульовано основні вимоги до сучасної SaaS-платформи валідації електронних адрес: багаторівнева система перевірки (синтаксична, DNS, SMTP, нейронна оцінка), підтримка масштабованості

та асинхронної обробки, гнучке API, безпечне зберігання та обробка даних, інтеграція в CI/CD-процеси клієнтів і доступна аналітика результатів.

Таким чином, було обґрунтовано доцільність створення нової платформи з орієнтацією на гнучкість, точність перевірки та можливість масштабування відповідно до навантаження.

# РОЗДІЛ 2. АРХІТЕКТУРА ТА ТЕХНОЛОГІЇ РОЗРОБКИ

## *2.1 Загальна архітектура платформи*

Розроблена SaaS-платформа для валідації електронних адрес у реальному часі базується на сучасній мікросервісній архітектурі, яка забезпечує незалежність компонентів, масштабованість, спрощене розгортання та гнучкість у подальшому розвитку системи. Уся система умовно поділяється на п’ять основних частин: Frontend, Backend, черга повідомлень (RabbitMQ), подієва шина (Kafka) та Сховище даних (Storage).

Клієнтська частина реалізована на основі React.js, що забезпечує інтерактивний, швидкий та адаптивний веб-інтерфейс. Основні функції клієнта:

- Завантаження списків електронних адрес для перевірки (через форму або CSV-файл).
- Візуалізація статусів перевірки у реальному часі за допомогою Server-Sent Events (SSE).
- Показ результатів перевірки, включно з деталізацією причин неправильності (наприклад: “некоректний формат”, “одноразовий домен”, “неіснуюча електронна адреса”).
- Можливість експорту перевіреного списку та аналітики у форматі CSV.
- Управління підпискою користувача, балансом запитів та API-ключами.

Серверна логіка побудована на NestJS - прогресивному TypeScript-фреймворку, що базується на принципах модульності та використання декораторів. Основні компоненти Backend:

- REST API для обробки запитів з клієнтської частини та сторонніх систем.
- Валідаційний модуль, який ініціює процес перевірки електронних адрес (синтаксична перевірка, DNS, SMTP, нейромережевий аналіз).
- SSE-шлюз (модуль сповіщень), який транслює події зміни статусу перевірки безпосередньо у браузер користувача.
- Сервіс білінгу, що відповідає за облік перевірок, тарифікацію та інтеграцію з платіжною системою.

Для асинхронної обробки запитів до валідації використовується RabbitMQ, що дозволяє:

- Зменшити навантаження на основний сервер у години пік.
- Розподіляти навантаження між окремими сервісами-виконувачами.
- Забезпечити масштабування через горизонтальне додавання виконувачів.

Черга працює за принципом “публікатор-споживач” (producer-consumer), де бекенд додає електронну адресу до черги, а окремі валідаційні виконувачі здійснюють перевірку та надсилають результати назад.

Всі події результатів надсилаються Kafka, з якої потім сервіс сповіщень розподіляє користувачам через SSE з’єднання.

Основна база даних реалізована за допомогою PostgreSQL, що забезпечує зберігання:

- списків електронних адрес, наданих користувачем;
- результатів перевірок (із повною розшифровкою причин);
- метаінформації про сесії, запити, користувачів, тарифні плани;
- журналу подій для аналітики та зручного аудиту.

Також використовується Supabase як бекенд-платформа з готовими механізмами автентифікації, ролей доступу, збереження файлів, та зручним RESTful API-інтерфейсом для клієнта [15].

Таким чином, архітектура платформи побудована за принципами розділення відповідальностей, асинхронної обробки, гнучкої інтеграції та масштабованості, що дозволяє обробляти великі обсяги запитів у реальному часі з мінімальними затримками та високою стабільністю.

## *2.2 Вибір технологій*

Для реалізації масштабованої SaaS-платформи з високою продуктивністю та підтримкою обробки даних у реальному часі було обрано сучасний набір технологій, що поєднує веб- і сервер-інструменти, черги повідомлень, хмарну інфраструктуру та інструменти контейнеризації.

Клієнтську частину платформи реалізовано з використанням React.js. Ця бібліотека дозволяє створювати інтерфейс користувача з високою інтерактивністю, швидкою реакцією на події та ефективним оновленням стану. Основні причини вибору React:

- Компонентна структура, що спрощує підтримку та масштабування інтерфейсу;
- Інтеграція з Server-Sent Events (SSE) для отримання результатів перевірки в реальному часі;
- Підтримка сучасних інструментів стилізації (TailwindCSS) та керування станом (React Query, Zustand).

Серверну частину реалізовано на базі NestJS - TypeScript-фреймворку, що поєднує зручність Express і модульність Angular. NestJS був обраний завдяки:

- Зручній реалізації REST API та SSE;
- Підтримці мікросервісної архітектури;
- Простій інтеграції з чергами RabbitMQ і Kafka;

– Високій читабельності коду та модульності, що спрощує розширення системи.

Supabase використовується як BaaS (backend-as-a-service) рішення, що забезпечує:

– PostgreSQL-базу даних із вбудованим REST-інтерфейсом;
– Авторизацію користувачів (email, OAuth, magic link);
– Зберігання файлів результатів перевірки у Supabase Storage;
– Гнучке керування правами доступу через Row Level Security (RLS).

Supabase значно скорочує час розробки типових функціональностей і дозволяє зосередитися на бізнес-логіці платформи.

Для обробки електронних адрес у фоновому режимі використовується RabbitMQ - брокер повідомлень, що реалізує модель публікатор-споживач (publisher-subscriber). Його застосування забезпечує:

– Асинхронну постановку задач валідації у чергу;
– Можливість масштабування через додавання кількох worker-процесів;
– Надійність доставки повідомлень і повторну обробку у разі помилок;

Для забезпечення потокової обробки даних, журналювання змін та потенційного розширення функціоналу в напрямку аналітики в реальному часі, інтегровано Apache Kafka. Її застосування дозволяє:

– Буферизувати великі обсяги подій із різних частин системи;
– Обробляти повідомлення незалежно в кількох модулях;
– Підключати аналітичні або ML-сервіси без втручання в основну логіку.
– Організовувати realtime-оновлення статусу валідацій для користувача (сповіщення про кроки та результат валідації)

Kafka є додатковим рівнем надійності та масштабованості, що особливо важливо при розширенні системи.

Контейнеризація реалізована за допомогою Docker, що дає змогу:

- Створювати ізольовані середовища для кожного сервісу;
- Уніфікувати процеси локальної розробки та розгортання в хмарі;
- Створювати складові середовища через Docker Compose (frontend, backend, брокери);
- Готувати основу для CI/CD-процесів[20].

Хмарна інфраструктура базується на Amazon Web Services. Зокрема, використовуються:

- EC2 - для хостингу серверної частини та сервісів перевірки;
- S3 - для зберігання файлів і результатів;
- Route 53 - для DNS-керування;
- CloudFront - для кешування та швидкої доставки frontend-контенту;
- Certificate Manager - для автоматичного керування SSL-сертифікатами.

AWS забезпечує надійність, масштабованість і безпеку, необхідні для комерційної SaaS-платформи [21].

Обраний технологічний стек є гнучким, розширюваним і оптимізованим під високонавантажені системи, що дає змогу платформі не лише ефективно виконувати перевірку електронних адрес у реальному часі, але й масштабуватись відповідно до зростання кількості користувачів.

### *2.3 Архітектура мікросервісів та комунікація між ними*

У рамках побудови SaaS-платформи для валідації електронних адрес у реальному часі було реалізовано мікросервісну архітектуру, яка дозволяє забезпечити гнучкість, масштабованість та розділення відповідальностей між логічними компонентами системи. Кожен мікросервіс виконує чітко визначену роль і взаємодіє з іншими сервісами через асинхронні канали обміну повідомленнями.

Основними компонентами архітектури є п’ять сервісів: backend, validator, notifications, RabbitMQ та Kafka. Така структура дозволяє організувати роботу системи за принципом подієвої взаємодії (event-driven) з мінімальними затримками, гарантованою доставкою повідомлень і високою відмовостійкістю (рисунок 2.1).

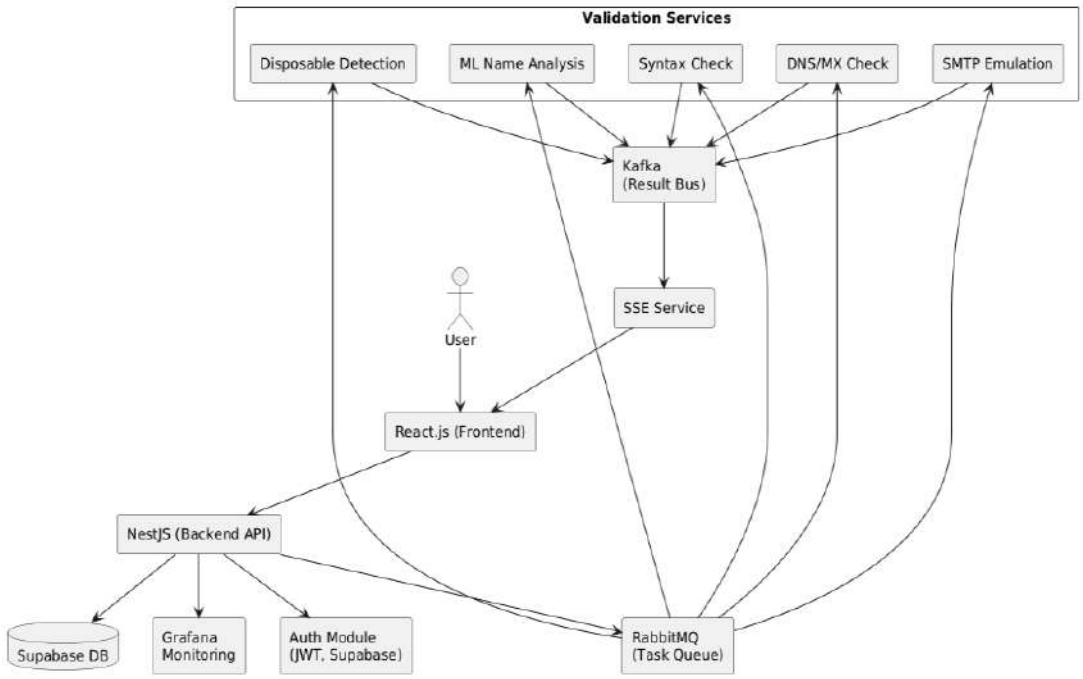


Рисунок 2.1 - Схема взаємодії мікросервісів

Центральний API-сервер, реалізований на NestJS, приймає вхідні запити від користувача. Його основні функції:

1. Прийом даних для валідації (через HTTP POST або Web UI);
2. Створення завдань на валідацію та постановка їх у чергу RabbitMQ;
3. Збереження інформації про запити користувачів, тарифні плани, результати перевірок;

RabbitMQ виступає як брокер черг повідомлень, забезпечуючи асинхронну передачу завдань від Backend до Validator. Переваги RabbitMQ у цьому контексті:

1. Забезпечення надійної доставки (ack/retry);
2. Можливість пріоритезації задач;
3. Буферизація під навантаженням;
4. Простота масштабування через додавання черг або каналів.

Validator - ключовий сервіс, який виконує реальну перевірку електронних адрес. Він реалізований як набір виконувачів, кожен із яких:

1. Підключається до RabbitMQ як споживач (consumer);
2. Виконує перевірку адреси на декількох рівнях:
   — синтаксична перевірка (RFC 5322);
   — перевірка одноразових доменів (через локальний список);
   — DNS/MX-запити;
   — перевірка наявності SMTP-сервера;
   — емуляція SMTP-діалогу;
   — ML-аналіз username (визначення ймовірності “keysmash”);
3. Формує результат перевірки;
4. Публікує події з результатами у Kafka для подальшої обробки.

Kafka - сервіс подієвої комунікації, який виступає як подієва шина (event bus) для всіх оновлень, пов’язаних з перевіркою. Validator публікує події в тему (наприклад, “email-validations”), які потім споживають інші сервіси. Основні переваги Kafka:

1. Потокова обробка великої кількості повідомлень;
2. Висока швидкодія та збереження порядку подій;
3. Масштабованість (можна додати кілька consumers для однієї теми).

Notifications - сервіс, що відповідає за доставку результатів валідації до користувача в реальному часі. Він:

1. Підписується на відповідні Kafka-топіки;
2. Слухає події, що містять результат перевірки;
3. Через відкриті SSE-з’єднання (Server-Sent Events) надсилає відповідь у браузер користувача;

4. Забезпечує мультикористувацьке обслуговування з підтримкою одночасних сесій.

Послідовність взаємодіїі описаних раніше мікросервісів виглядає наступним чином:

1. Користувач надсилає список електронних адрес для перевірки через Frontend.
2. Backend приймає запит, створює задачу для кожної адреси та ставить їх у чергу RabbitMQ.
3. Один або кілька інстансів Validator вибирають задачі з черги, проводять усі рівні перевірки та формують результат.
4. Під час та після перевірки Validator надсилає подію з результатами у Kafka.
5. Сервіс Notifications миттєво отримує ці події, зіставляє їх з активними SSE-сесіями користувачів і надсилає результат назад на клієнтську частину.

Запропонована архітектура є оптимальним варіантом для реалізації застосунку, основними перевагами є:

– Чітке розділення обов’язків: кожен сервіс виконує конкретну функцію, що полегшує тестування, налагодження та оновлення.
– Горизонтальне масштабування: усі сервісні компоненти можна масштабувати незалежно. Наприклад, Validator можна запускати в кількох інстансах паралельно для прискорення обробки.
– Асинхронна обробка: RabbitMQ дозволяє уникнути блокувань при високих навантаженнях і забезпечує балансування навантаження.
– Подієва архітектура: Kafka забезпечує централізовану доставку результатів до кількох підписників, відкриваючи можливості для аналітики, логування або алертингу.

– Доставка результатів у реальному часі: через Notifications та SSE користувач отримує оновлення миттєво, без необхідності ручного оновлення сторінки чи запитів до API.

Архітектура мікросервісів у даній платформі забезпечує надійність, модульність, масштабованість і можливість подальшого розвитку - наприклад, додавання нових типів перевірок, зберігання історії, аналітики чи інтеграції з CRM-системами без порушення існуючої логіки.

### *2.4 Безпека та масштабованість (Auth, HTTPS, Cloud, VPC, TLS)*

У сучасній SaaS-платформі захист даних користувача, стійкість до збоїв та здатність системи обробляти зростаючі обсяги трафіку є ключовими аспектами якості та довіри до продукту. Розроблена платформа для валідації електронних адрес побудована з урахуванням принципів безпеки за замовчуванням (secure by design) та масштабованості за замовчуванням (scale by design). У цьому підрозділі розглянуто реалізовані механізми безпеки та технології, що забезпечують горизонтальне та вертикальне масштабування системи.

У платформі реалізовано централізовану систему аутентифікації користувачів на основі Supabase Auth, яка підтримує:

– реєстрацію та вхід через email/password;
– підтвердження адреси через верифікаційне посилання (magic link);
– підтримку OAuth2 (Google, GitHub тощо);
– керування ролями та правами доступу через RLS (Row Level Security) на рівні бази даних PostgreSQL.

Це забезпечує захист ресурсів на рівні як API, так і даних, навіть у разі прямого доступу до бази.

Уся комунікація між клієнтом, API, брокерами та базою даних відбувається виключно через зашифровані канали (HTTPS/TLS 1.2/1.3):

- SSL-сертифікати автоматично керуються через AWS Certificate Manager;
- Frontend обслуговується через CloudFront з примусовим шифруванням та політиками HSTS;
- API з боку серверної частини захищено reverse proxy з підтримкою TLS termination.

Завдяки цьому забезпечується захист від атак типу man-in-the-middle (MITM) та витоку даних при передачі.

Платформа розгортається в хмарному середовищі Amazon Web Services (AWS), що забезпечує:

- високу доступність (high availability);
- резервування на рівні обчислювальних ресурсів і сховищ;
- гнучке масштабування через Auto Scaling Groups;
- відновлення після збоїв завдяки використанню декількох Availability Zones.

Це дозволяє гарантувати стабільну роботу сервісу навіть за зростання кількості користувачів або при несправності окремих компонентів.

Для ізоляції та захисту внутрішньої інфраструктури використовується VPC (Virtual Private Cloud), яка дозволяє:

- розділити публічні та приватні підмережі;
- обмежити доступ до RabbitMQ, Kafka, бази даних та мікросервісів виключно з авторизованих маршрутів;
- використовувати NAT/Internet Gateway для вихідного трафіку без відкриття вхідних портів;
- застосовувати Security Groups і Network ACLs для контролю трафіку між компонентами.

VPC є основою для створення ізольованого та безпечного середовища виконання усіх сервісів платформи.

Для забезпечення ефективної обробки запитів у режимі реального часу реалізовано підтримку горизонтального масштабування критичних сервісів:

- Validator може запускатися у кількох інстансах, кожен з яких паралельно обробляє задачі з RabbitMQ;
- Notifications може масштабуватись при зростанні кількості одночасних SSE-з'єднань;
- Kafka та RabbitMQ підтримують кластеризацію для розширення пропускної здатності;
- Backend може масштабуватись через ECS, EC2 або Kubernetes з балансуванням навантаження (ALB/ELB).

Також реалізовано кешування DNS/MX-запитів та повторне використання SMTP-з'єднань для зменшення часу обробки та навантаження на мережу.

Усі API-запити захищені JWT-токенами, які перевіряються сервером на кожен запит, застосовано rate limiting та throttling для захисту від зловживань та DoS-атак, введено облік запитів користувача (billing-aware architecture), що дозволяє поєднати безпеку із контролем ресурсів.

***Висновки до розділу 2***

У другому розділі було представлено детальний опис архітектури та обґрунтування вибору технологій для реалізації масштабованої SaaS-платформи валідації електронних адрес у реальному часі. На основі аналізу функціональних вимог побудовано мікросервісну архітектуру, що забезпечує незалежність окремих компонентів, гнучкість масштабування та високу стійкість до збоїв.

Обґрунтовано використання сучасного стеку технологій:

- React - для створення швидкого та інтерактивного користувацького інтерфейсу;
- NestJS - як основа для реалізації модульного backend-застосунку;
- RabbitMQ - для постановки задач в чергу з можливістю контролю навантаження;
- Kafka - для потокової доставки оновлень та подій;
- Supabase - як зручне рішення для автентифікації та зберігання даних;
- Docker та AWS - для контейнеризації, автоматизованого розгортання та масштабування у хмарному середовищі.

Описано схему взаємодії між основними сервісами: backend ставить задачі у RabbitMQ, validator обробляє їх та публікує результати через Kafka, які, у свою чергу, надсилаються сервісом сповіщень через SSE користувачам.

Окрема увага приділена питанням безпеки та масштабованості. Платформа реалізує автентифікацію, авторизацію, HTTPS-з’єднання, шифрування даних у транзиті (TLS), ізоляцію компонентів у межах VPC та готовність до автоматичного масштабування.

Таким чином, розроблена архітектура і технологічна основа повністю відповідають сучасним вимогам до високонавантажених, безпечних та розширюваних SaaS-рішень.

# РОЗДІЛ 3. МОДУЛЬ ВАЛІДАЦІЇ ЕЛЕКТРОННИХ АДРЕС

## *3.1 Класифікація типів валідації*

Процес валідації електронних адрес у розробленій платформі реалізовано як послідовність логічно впорядкованих етапів, кожен з яких виконує окрему перевірку і дає власний внесок у підсумкову оцінку достовірності адреси. Такий підхід дозволяє зменшити кількість запитів до зовнішніх сервісів, оптимізувати продуктивність системи та покращити якість класифікації.

Перший етап - перевірка правильності формального запису електронної адреси відповідно до стандарту RFC 5322:

- наявність символу @ та доменної частини;
- допустимі символи в local-part;
- відсутність неприпустимих символів або пробілів.

Це дозволяє відразу відфільтрувати некоректні або порожні значення без потреби звернення до зовнішніх ресурсів.

Наступним кроком є перевірка наявності DNS-записів для домену:

- A/AAAA-записи підтверджують існування домену;
- MX-записи визначають поштові сервери, які можуть обробляти кореспонденцію.

У разі відсутності записів або помилки отримання результату адреса визнається недійсною.

Після цього відбувається поглиблена валідація MX-записів, що включає:

- наявність валідного сервера;
- доступність MX-адрес для SMTP-з’єднання;
- перевірку пріоритетів і конфігурацій.

Цей етап забезпечує впевненість у тому, що поштовий домен технічно здатен приймати листи.

На наступному етапі виконується емуляція SMTP-сесії:

– встановлення з’єднання з сервером;
– виконання команд MAIL FROM / RCPT TO;
– аналіз відповіді (коди 250, 550, 451 тощо).

Це дає змогу з високою ймовірністю визначити, чи дійсно поштову скриньку створено. Проте деякі сервери можуть повертати позитивну відповідь для будь-якої адреси (див. нижче).

Також, необхідно взяти до уваги Catch-all (універсальні) домени, які приймають повідомлення для будь-якого локального імені (local-part). Така поведінка робить неможливим визначення достовірності конкретної адреси через SMTP.

Метод виявлення catch-all:

– виконання SMTP-запитів до випадково згенерованих адрес на тому ж домені;
– аналіз і порівняння відповідей з реальною адресою;
– у разі збігу - домен вважається catch-all, а адреса позначається як “невизначена”.

Останній етап перевірки - інтелектуальний аналіз імені користувача (до символу @) з використанням нейронної мережі, натренованої для виявлення шаблонів “keysmash”-подібних імен.

Типові приклади:

– ajsdg8723@gmail.com
– qweasdzxc1989@mail.com
– xjfhqoiweuh@randomhost.com

Для виявлення таких підозрілих адрес використано модель на базі SklearnClassifier SVC, що аналізує послідовність символів у username:

– визначає відсутність лексичної/грамотної структури;

– розраховує ймовірність того, що адреса згенерована автоматично;
– класифікує її як потенційно фейкову.

Цей етап особливо важливий для захисту від бот-реєстрацій та фільтрації неякісних контактів у B2C/маркетингових платформах.

Комбіноване використання класичних методів (RFC, DNS, SMTP) із сучасними ML-підходами дозволяє:

– забезпечити високу точність оцінки валідності електронних адрес;
– виявити нетипові шаблони, які складно ідентифікувати вручну;
– підвищити загальну якість контактної бази;
– адаптувати перевірку до зловживань і динамічних змін у поведінці користувачів.

Такий поетапний процес валідації реалізовано у валідаційному модулі платформи як незалежну, масштабовану службу, яка обробляє запити у фоновому режимі, забезпечуючи швидку та надійну перевірку кожної адреси.

### *3.2 Алгоритми перевірки: структурні правила, SMTP-сесії, таймаути*

У модулі валідації електронних адрес реалізовано набір алгоритмів, які виконуються послідовно та забезпечують перевірку адрес на відповідність формальним правилам, технічну доступність доменів і скриньок, а також поведінковий аналіз. Кожен з алгоритмів розроблений з урахуванням практичних обмежень (час, мережеві помилки, доступність SMTP-серверів) і має механізми відновлення після помилок або недоступності зовнішніх ресурсів.

На першому етапі перевіряється формат адреси електронної пошти відповідно до вимог стандарту RFC 5322. Для цього використовуються регулярні вирази, які враховують:

– допустимі символи у local-part та domain-part;
– довжину обох частин;
– наявність лише одного символу @;
– відсутність пробілів, лапок, дужок тощо.

Цей етап реалізується у вигляді детермінованого алгоритму, що дозволяє швидко відсіяти очевидно помилкові адреси до звернення до зовнішніх систем.

Далі відбувається перевірка наявності та коректності DNS- та MX-записів:

– Розв’язання доменного імені (domain-part) через getaddrinfo або аналоги;
– Пошук MX-записів через DNS-запити;
– Перевірка IP-адрес поштових серверів та відкритості порту 25;
– У разі відсутності MX-запису — перевірка A/AAAA-записів як fallback.

Алгоритм кешує результати запитів на заданий час (наприклад, 300 секунд), що дозволяє знизити навантаження на DNS-сервери при масовій перевірці.

На цьому етапі застосовується алгоритм, що імітує процес доставки листа без його фактичної відправки. Суть полягає у встановленні TCP-з’єднання з SMTP-сервером і виконанні комбінації SMTP-команд:

– HELO / EHLO — ініціалізація сесії;
– MAIL FROM:<valid@sender.com> — імітація відправника;
– RCPT TO:<target@domain.com> — основна команда для перевірки адреси;
– Аналіз відповіді сервера (250 OK, 550 No such user, 451 Temporary failure тощо);
– Закриття сесії через QUIT.

У процесі реалізовано логіку повторних спроб для тимчасових помилок (timeouts, 4xx-коди) та аналізу catch-all-сценаріїв через генерацію фіктивних адрес.

Оскільки багато SMTP-серверів мають механізми rate limiting або штучне затягування відповіді (до 30+ секунд), реалізовано спеціальні обмеження для кожного запиту:

– TCP timeout (наприклад, 5 секунд) - максимальний час встановлення з'єднання;
– SMTP read timeout - обмеження на відповідь після команди RCPT TO;
– Максимальна тривалість усієї сесії - обмеження загального часу перевірки однієї адреси.

Якщо одне із часових обмежень перевищено - перевірка припиняється, адреса отримує статус "невизначено" з відповідною поміткою ("timeout", "no response", "temporary error").

Для врахування нестабільних SMTP-серверів передбачена повторна перевірка через затримку, за умови обмеженої кількості спроб.

Окремі алгоритми оптимізовано під обробку груп електронних адрес:

– об'єднання запитів до одного SMTP-сервера для кількох адрес на одному домені;
– повторне використання відкритого з'єднання (keep-alive);
– пріоритезація перевірок для доменів із найменшим latency.

Це суттєво покращує швидкість при перевірці великих списків та знижує навантаження на систему.

Алгоритми перевірки в модулі валідації розроблено з урахуванням:

– відповідності міжнародним стандартам;
– поведінки реальних SMTP-серверів;
– потреб у масштабованості та fault-tolerance.

Вони забезпечують надійне визначення валідності адрес, а в поєднанні з ML-аналізом (див. розділ 3.1) — дозволяють ефективно

фільтрувати як помилкові, так і підозрілі адреси, зберігаючи баланс між точністю, швидкістю та економічністю.

### *3.3 Інтеграція з чергами повідомлень (Kafka, RabbitMQ)*

У платформі валідації електронних адрес важливо забезпечити асинхронну, надійну та масштабовану обробку запитів, яка не блокує основні компоненти системи (зокрема, API та інтерфейс користувача). Для цього використано два незалежні, але взаємодіючі механізми черг повідомлень - RabbitMQ та Apache Kafka. Вони виконують різні ролі в архітектурі, доповнюючи одне одного.

RabbitMQ використовується як брокер завдань для валідаційного модуля. Його основне призначення - приймати завдання від серверної частини й передавати їх валідаційним виконувачам у фоні. Це дає змогу:

- уникнути блокування HTTP-запитів користувача;
- паралелізувати обробку великої кількості електронних адрес;
- балансувати навантаження між кількома інстансами validator;
- гарантувати повторну обробку у випадку збою виконувача.

Типова послідовність обробки в RabbitMQ:

1. Користувач надсилає список електронних адрес через API.
2. Backend формує повідомлення про задачу перевірки (у форматі JSON) та публікує його в чергу RabbitMQ.
3. Один із виконувачів validator зчитує повідомлення з черги, виконує перевірку та надсилає результат далі через Kafka.
4. У випадку збоїв виконувач не підтверджує обробку (ack), і повідомлення повертається до черги.

Завдяки використанню acknowledgment-механізму RabbitMQ гарантує стійкість до втрати задач у разі збоїв.

Apache Kafka використовується як система подій (event bus), яка передає результати перевірки від validator до notifications. Її призначення - розповсюджувати повідомлення про завершення обробки між кількома підписниками з мінімальною затримкою.

Переваги Kafka у цій архітектурі:

- Висока пропускна здатність для потокової передачі подій;
- Надійність доставки завдяки журналюванню та збереженню подій;
- Масштабованість за рахунок тем та груп споживачів;
- Можливість незалежного додавання нових компонентів, які слухають ті самі події (наприклад, аналітика, логування, ML-підсистеми).

Типова схема взаємодії з Kafka:

1. Сервіс validator після завершення перевірки публікує повідомлення в Kafka-топік validation.results.
2. Сервіс notifications підписаний на цей топік і миттєво отримує повідомлення.
3. notifications передає інформацію користувачу через відкрите SSE-з’єднання.

Це дозволяє отримувати результати в реальному часі, з мінімальним навантаженням на головний API-сервер.

Використання одночасно RabbitMQ і Kafka зумовлене розподілом функціональності:

- RabbitMQ - підходить для чітко визначених задач з відповідальністю, коли важлива гарантія обробки;
- Kafka - ідеально підходить для широкомовної подієвої комунікації, де результат потрібен кільком системам.

Таким чином, платформа використовує RabbitMQ для внутрішньої оркестрації обробки, а Kafka - для зовнішньої дистрибуції результатів.

***Висновки до розділу 3***

У третьому розділі детально розглянуто архітектуру та функціонування модуля валідації електронних адрес як ключового елемента платформи. Було здійснено класифікацію основних типів перевірки, які реалізовано у системі:

- Синтаксична валідація - перевірка відповідності електронної адреси базовим правилам RFC;
- DNS- та MX-перевірка - перевірка існування домену та поштового сервера;
- SMTP-емуляція - встановлення з’єднання з реальним поштовим сервером для визначення наявності поштової скриньки;
- Catch-all виявлення - перевірка того, чи домен приймає всі листи незалежно від username;
- ML-аналіз - визначення, чи є username випадковим (keysmash), з використанням нейромережевої моделі.

У підрозділі 3.2 представлено алгоритмічну реалізацію перевірок, з урахуванням обмежень реальних SMTP-серверів, таймаутів, необхідності коректного завершення сесій та обробки нестандартних відповідей поштових провайдерів.

Розділ також висвітлює інтеграцію модуля валідації з системою міжсервісної комунікації. За допомогою RabbitMQ задачі на перевірку розподіляються асинхронно між виконувачами-валідаторами, що дозволяє забезпечити масштабованість і стійкість до навантажень. Результати перевірок надсилаються через Kafka, а подальші дії з ініціації сповіщень або оновлень на стороні клієнта відбуваються у режимі реального часу.

# РОЗДІЛ 4. РЕАЛІЗАЦІЯ ПЛАТФОРМИ

## *4.1 Реалізація клієнтської частини з використанням React.js*

Клієнтська частина SaaS-платформи для валідації електронних адрес розроблена із застосуванням сучасного стеку технологій, з фокусом на швидкодію, інтерактивність і зручність користувача. Основою є бібліотека React.js, яка забезпечує гнучкий компонентний підхід до побудови інтерфейсу. Її використання дозволяє легко масштабувати систему, організовувати повторне використання логіки та підтримувати розділення відповідальностей між елементами інтерфейсу. Мова програмування - TypeScript, яка надає статичну типізацію, знижує ризик помилок на етапі розробки та покращує зручність роботи над кодом. Як інструмент збирання обрано Vite, що забезпечує швидку компіляцію, гаряче оновлення модулів під час розробки і загалом пришвидшує цикл зворотного зв’язку.

Для стилізації використовуються Tailwind CSS та бібліотека компонентів shadcn/ui, що дозволяє створювати сучасний, адаптивний інтерфейс із чітко структурованими візуальними компонентами. Завдяки цим засобам інтерфейс платформи виглядає професійно, лишається легким і водночас функціональним (рисунки 4.1-4.3).

Користувацький інтерфейс виконує низку ключових функцій. По-перше, він забезпечує зручне введення електронних адрес: як вручну, так і через імпорт файлів у форматі CSV. По-друге, користувач отримує результати перевірки у режимі реального часу - реалізовано інтеграцію з сервером через технологію Server-Sent Events (SSE), що дозволяє передавати оновлення миттєво, без потреби в циклічному опитуванні API. Результати перевірки подаються у вигляді таблиці з інформативними

статусами, кольоровим маркуванням, поясненнями причин недійсності адреси та можливістю експорту очищеного списку.

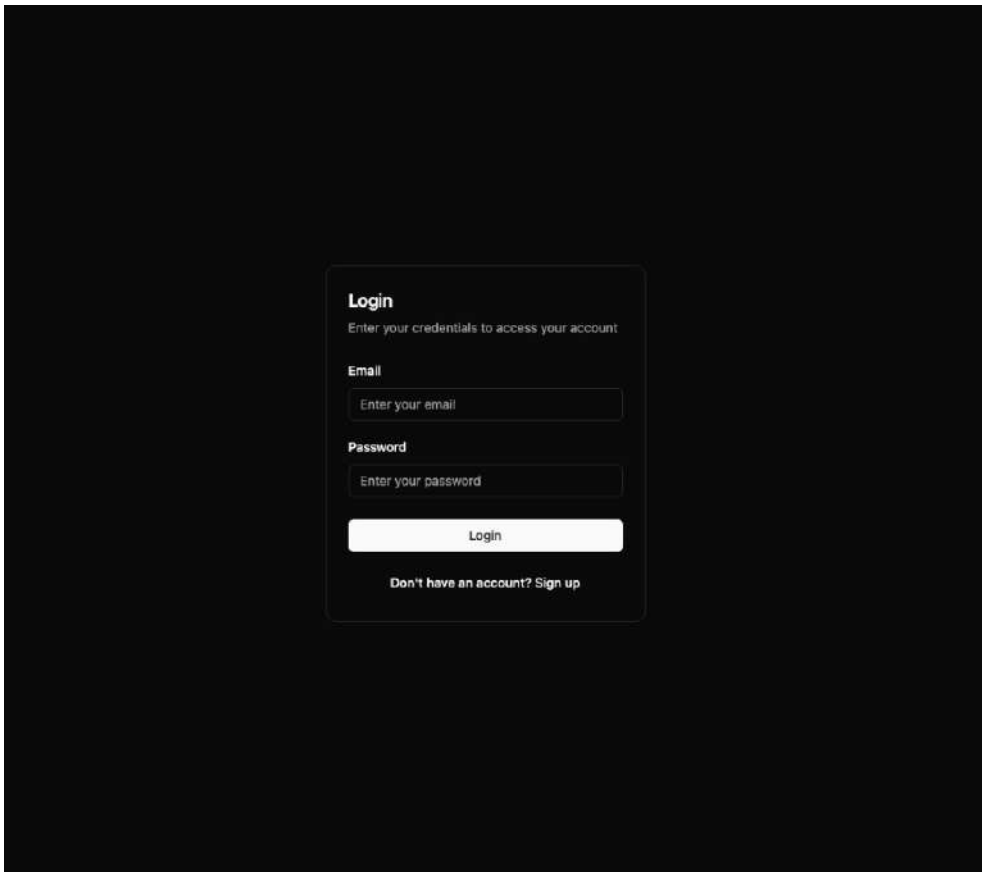


Рисунок 4.1 - Екран форми входу в кабінет

Окрему увагу приділено управлінню обліковим записом. Користувач може переглядати інформацію про свій тарифний план, залишок ліміту на перевірки, історію транзакцій, а також отримувати повідомлення про помилки або події системи через toast-сповіщення. Застосунок побудований так, щоб гарантувати безперебійну роботу навіть за великого навантаження - важкі компоненти підвантажуються динамічно (lazy loading), що дозволяє зберегти продуктивність і швидкий рендер сторінок.

З технічної точки зору, React-застосунок взаємодіє з бекендом через REST API - для надсилання запитів на перевірку, отримання історії, керування акаунтом - та SSE - для підписки на події зміни статусу перевірки. Усі HTTP-запити та підключення до подій інкапсульовані в

окремі обгортки, що забезпечують обробку помилок, автоматичні повторні спроби, контроль тайм-аутів та повторне встановлення з’єднань при втраті доступу до мережі.

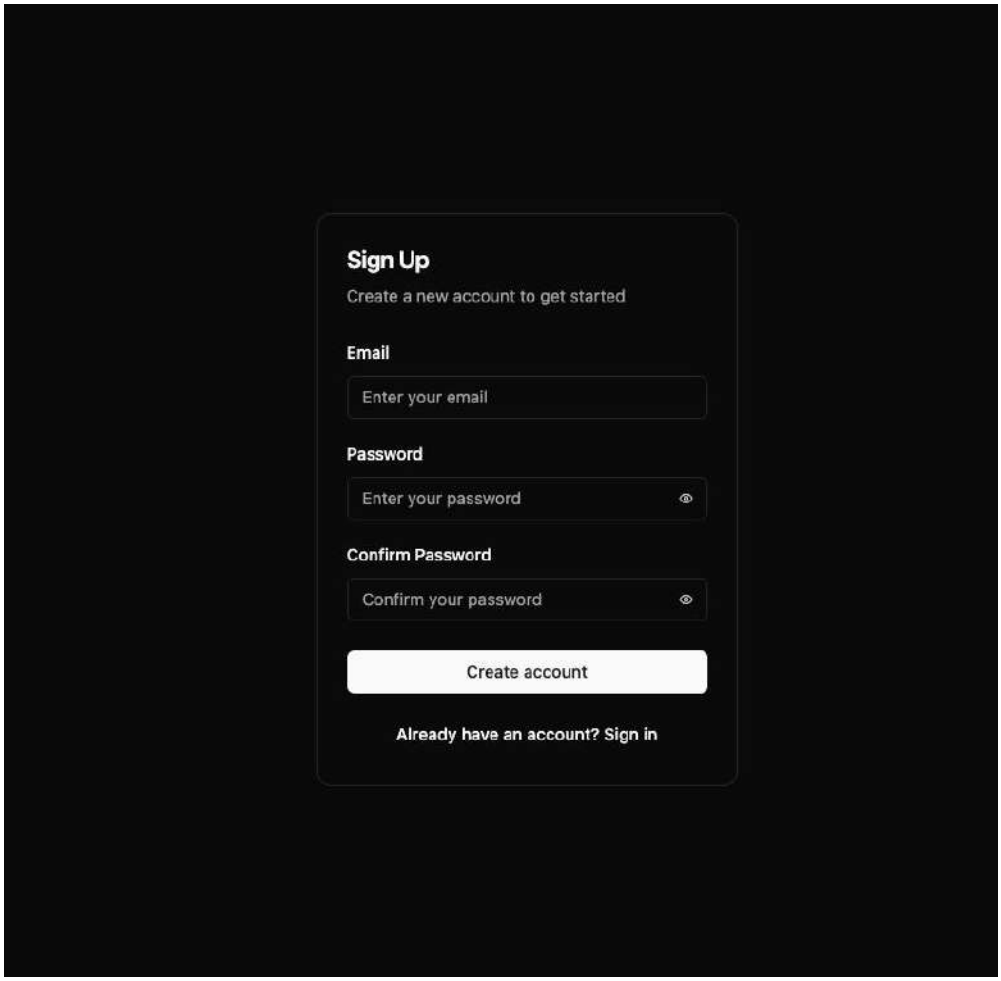


Рисунок 4.2 - Екран форми реєстрації

Інтерфейс користувача є адаптивним і добре працює як на десктопах, так і на мобільних пристроях. Важливим є також забезпечення клієнтської валідації введених даних: перевіряється коректність електронної адреси, відповідність CSV-файлів структурі, попереджається про повтори чи порожні поля.

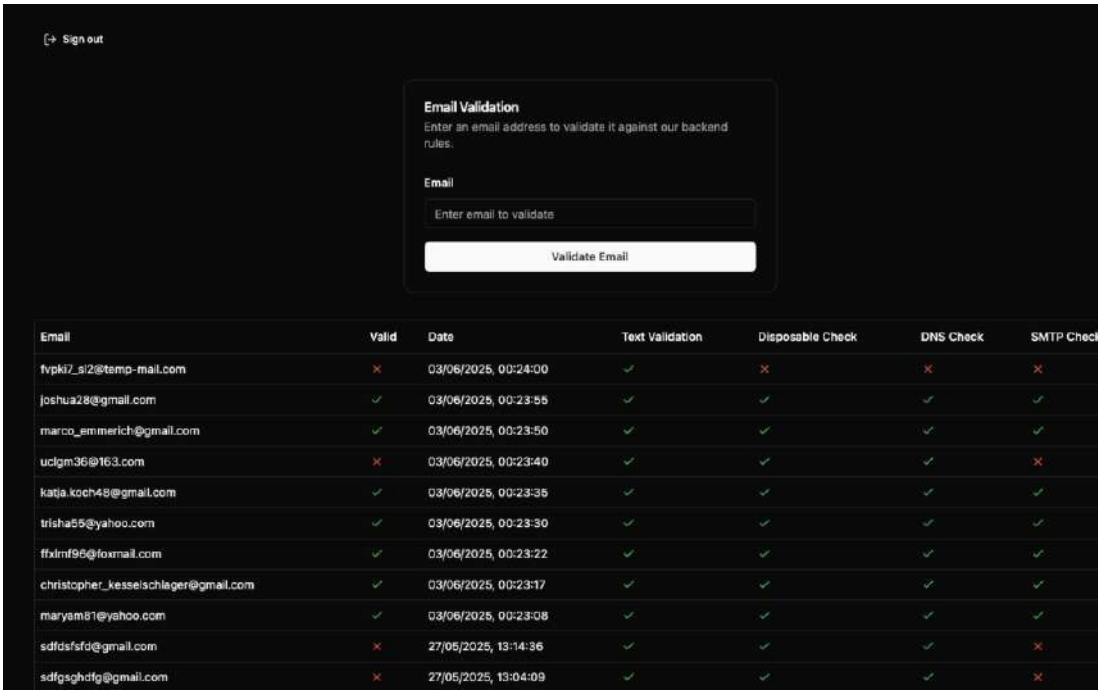


Рисунок 4.3 - Приклад екрану зі списком проведених валідацій

Загалом клієнтська частина платформи є повноцінним інтерактивним засобом доступу до основного функціоналу сервісу.

## *4.2 Реалізація серверної частини на NestJS*

Бекенд SaaS-платформи реалізовано з використанням фреймворку NestJS, що побудований поверх Node.js і TypeScript і поєднує гнучкість, масштабованість та суворе дотримання принципів модульної архітектури. Такий вибір обґрунтований необхідністю створення надійної серверної частини, яка підтримує обробку великої кількості запитів у реальному часі, є легко підтримуваною, а також забезпечує інтеграцію з системами обміну повідомленнями та зовнішніми сервісами.

Основними завданнями бекенда є: обробка вхідних запитів на перевірку електронних адрес (як з боку API, так і через веб-інтерфейс), постановка задач у чергу на валідацію, автентифікація користувачів,

контроль тарифних обмежень, зберігання історії перевірок і підтримка потокового зв’язку з клієнтом для оновлення статусів. Усі ці функції реалізовані у вигляді модулів, кожен із яких відповідає за конкретну частину бізнес-логіки.

Архітектурно бекенд розроблений згідно з підходом “чистої архітектури”, де контролери відповідають за прийом HTTP-запитів і делегують обробку сервісам, а сервіси реалізують прикладну логіку, взаємодіючи з чергами RabbitMQ, базою даних або зовнішніми API. Логічна структура проекту поділена на окремі модулі (наприклад, ValidationModule, AuthModule, UserModule, BillingModule), що забезпечує гнучкість і масштабованість системи. Додатково реалізовано фільтри, інтерсептори та пайпи, які централізовано обробляють помилки, виконують валідацію даних і форматують відповіді.

Важливою особливістю є взаємодія бекенда з зовнішніми сервісами та мікросервісами. Завдання на валідацію передаються через RabbitMQ, а результати повертаються через Kafka. Потокові події доставляються користувачеві через SSE (Server-Sent Events). Для автентифікації та зберігання користувацьких даних використовується Supabase, який виконує роль провайдера авторизації, а також як доступ до бази даних (через PostgreSQL-клієнт або REST API).

Інтерфейс програмування застосунків (REST API) включає в себе набір маршрутів, які реалізують основну функціональність системи:

- GET /api/auth/session — отримання інформації про сесію користувача;
- POST /api/auth/signup — реєстрація нового користувача;
- POST /api/auth/signin — автентифікація користувача;
- POST /api/email/validate — надсилання списку електронних адрес на перевірку;

- GET /api/email/validations — отримання історії перевірок користувача;
- GET /api2/notifications/stream — ініціалізація SSE-з’єднання для push-оновлень;
- GET /billing/info — перегляд тарифного плану та залишку перевірок.

API побудовано з урахуванням принципів безпеки: реалізовано авторизацію за допомогою JWT-токенів, логування всіх запитів, обмеження швидкості доступу (rate limiting), захист від перевантаження та несанкціонованого доступу. Кожен запит перевіряється на наявність дійсного токена, а задачі — на відповідність тарифному плану користувача. Для захисту інфраструктури передбачено механізми throttling та обмеження кількості одночасних SSE-з’єднань.

Система протестована як на рівні окремих сервісів (unit-тести), так і в форматі наскрізного тестування основних сценаріїв взаємодії користувача з API (end-to-end тести). Логування подій реалізовано за допомогою бібліотеки Winston, що дозволяє зручно інтегрувати бекенд у системи моніторингу, такі як Grafana + Loki.

Таким чином, бекенд виступає ядром всієї платформи, забезпечуючи зв’язок між клієнтським інтерфейсом, сервісами валідації та базою даних, і виконує всі критично важливі функції обробки, маршрутизації та обліку.

### *4.3 CI/CD-пайплайни для автоматизації розгортання*

Для забезпечення стабільного, повторюваного та контрольованого процесу розгортання всіх компонентів платформи реалізовано повноцінний CI/CD-процес (Continuous Integration / Continuous Deployment). Автоматизація розгортання дозволяє мінімізувати людський фактор, прискорити доставку нових змін до продакшн-середовища та гарантувати,

що кожна версія коду проходить через валідацію, тестування і збірку перед публікацією (рисунок 4.4).

У реалізації CI/CD-процесу використано такі інструменти:

– GitHub Actions - як основна платформа CI/CD;
– Docker - для контейнеризації сервісів;
– Docker Hub - як реєстр образів;
– AWS EC2 - для хостингу продакшн-сервісів;
– Supabase - як кероване середовище для бази даних і автентифікації;
– Nginx - як реверс-проксі та TLS-термінатор на продакшн-сервері.

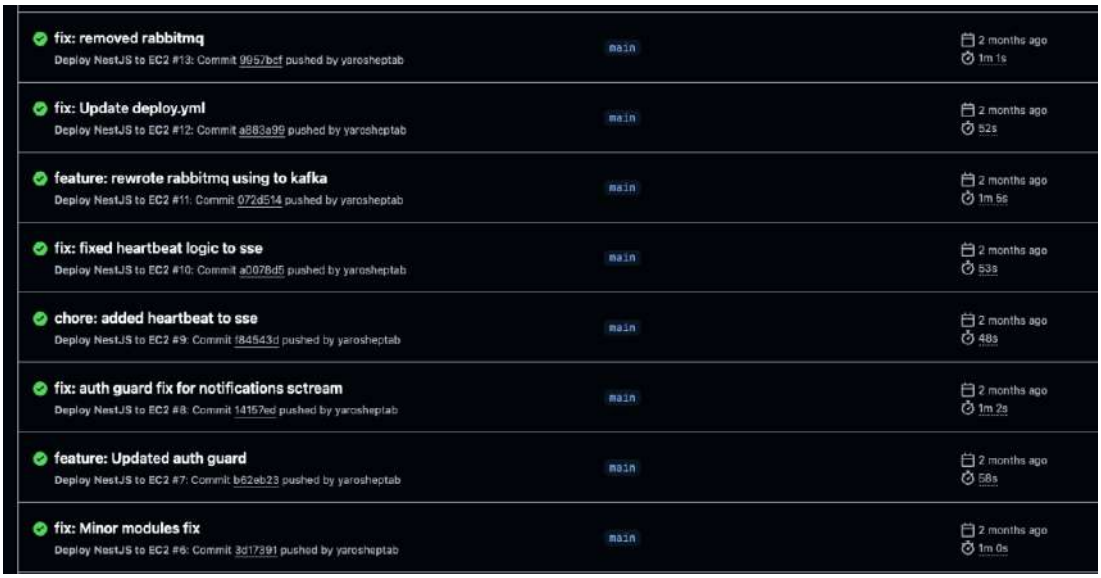


Рисунок 4.4 - Історія проведених CI/CD пайплайнів

CI-процес запускається автоматично при кожному push або pull request у основну гілку репозиторію. Він включає:

1. Перевірку синтаксису та типів за допомогою eslint і tsc;
2. Запуск unit-тестів (Jest для серверної частини та React Testing Library для клієнтської частини);
3. Перевірку форматування коду (prettier --check);
4. Збірку Docker-образів для backend, frontend, validator, notifications;

5. Відправку образів у Docker Hub з тегами latest або vX.Y.Z.

Після успішного завершення CI-етапу автоматично виконується розгортання:

1. SSH-підключення до AWS EC2 через GitHub Actions;
2. Витягування актуальних Docker-образів з Docker Hub;
3. Перезапуск сервісів через docker-compose down && docker-compose up -d;
4. Очистка старих образів і логів для економії дискового простору;
5. Оновлення конфігурацій через .env файли та секрети з GitHub Secrets / AWS SSM.

Використовуючи обраний підхід, забезпечується швидка доставка оновлень без ручного втручання, можливість швидкого rollback у разі збоїв, прозорий процес з історією всіх змін, єдина точка входу для автоматизованого тестування, складання та релізу та безпечна передача секретів через GitHub Secrets та AWS IAM-ролі.

Таким чином, CI/CD-пайплайн є критично важливим елементом платформи, що забезпечує її стабільне функціонування, оперативність оновлень та надійність у продакшн-середовищі.

### *4.4 Моніторинг та логування (PM2+, Grafana, Loki, Prometheus)*

Для забезпечення стабільної роботи платформи в реальному часі, швидкого виявлення проблем та аналізу навантаження реалізовано комплексну систему моніторингу та логування. Вона охоплює усі ключові компоненти - від серверної частини й виконувачів до брокерів повідомлень і клієнтської частини - та забезпечує візуалізацію метрик, збір журналів подій і автоматичні сповіщення.

Для запуску та моніторингу сервісів, реалізованих на Node.js (backend, notifications), використовується PM2 у поєднанні з PM2 Plus (рисунок 4.5):

- забезпечує автоматичний перезапуск процесів у разі збоїв;
- дозволяє відстежувати CPU, RAM, затримки, кількість оброблених задач;
- надає веб-інтерфейс з історією інцидентів, сповіщеннями та логами;
- підтримує кластерний режим запуску для горизонтального масштабування [22].

PM2 також інтегрується з системою логування та надсилає події у централізовані журнали.

Для збору кількісних показників продуктивності використовується Prometheus:

- кожен сервіс експортує метрики на /metrics у форматі Prometheus exposition format;
- метрики включають кількість оброблених електронних адрес, час відповіді, навантаження на черги, статуси SMTP-з’єднань;
- Prometheus регулярно опитує ці шляхи та зберігає історичні дані.

Це дозволяє будувати аналітичні графіки, виявляти аномалії та аналізувати продуктивність системи за різні періоди часу.

Для візуального представлення метрик використовуються дашборди Grafana, які підключені до Prometheus і Loki:

- побудовано окремі панелі для кожного сервісу (backend, validator, notifications);
- на графіках можна бачити активність у реальному часі: кількість перевірок, помилки, час відповіді SMTP-серверів;
- реалізовано сповіщення для ситуацій, коли метрики перевищують критичні пороги (наприклад, висока затримка або велика кількість невдалих підключень).

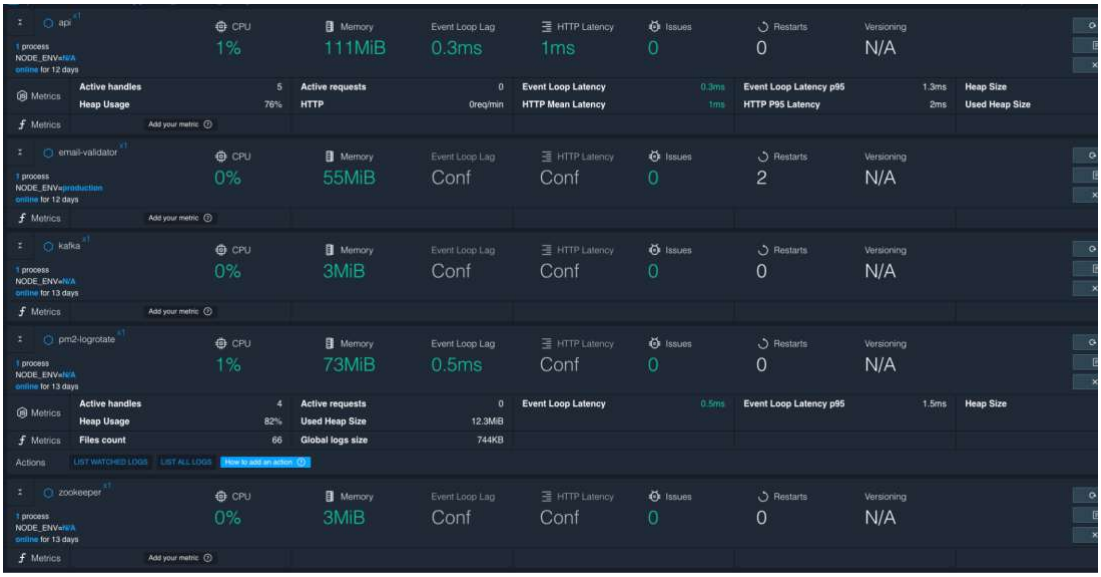


Рисунок 4.5 - Сторінка моніторингу сервісів PM2+

Grafana також дозволяє інтегрувати сповіщення у Slack, Telegram або електронну пошту у разі інцидентів.

Для збору, зберігання та пошуку логів у реальному часі використовується Grafana Loki (рисунок 4.6):

– кожен сервіс передає свої логи у форматі JSON або plaintext через драйвер promtail або stdout інтеграцію;
– у логах фіксується інформація про статуси перевірок, з’єднання з SMTP, помилки DNS, спроби входу користувачів;
– логи фільтруються за сервісами, рівнем важливості (info, warn, error) та часовими мітками.

Завдяки Loki вся подієва історія платформи зберігається централізовано, що дає змогу швидко аналізувати інциденти або відстежувати проблеми користувачів.

Рисунок 4.6 - Приклад логів мікросервісів

### *Висновки до розділу 4*

У четвертому розділі було представлено практичну реалізацію ключових компонентів платформи валідації електронних адрес у реальному часі.

Клієнтську частину реалізовано за допомогою React.js, що забезпечує динамічний інтерфейс користувача, інтеграцію з API у режимі реального часу, відображення статусів перевірок та роботу з потоковими оновленнями за допомогою SSE (Server-Sent Events). Було реалізовано функціонал подачі списку електронних адрес на перевірку, перегляду історії запитів і візуалізації результатів.

Бекенд побудовано на основі NestJS - модульного Node.js-фреймворку, який забезпечив чітку структуру проєкту, обробку запитів

REST, авторизацію через Supabase, генерацію задач у чергу RabbitMQ та взаємодію з Kafka для обробки результатів. Застосовано сучасні принципи проектування: розділення відповідальностей, використання DTO (Data Transfer Object) та DI (Dependency Injection).

Особливу увагу приділено CI/CD-процесам, що автоматизовують збірку, тестування та деплой платформи. Налаштовано пайплайни, які забезпечують безперервну доставку оновлень на staging та production середовища за допомогою GitHub Actions і Docker.

Система моніторингу та логування реалізована через стек Grafana, Loki, Prometheus та PM2+. Це дозволяє контролювати стан мікросервісів, збирати метрики продуктивності, переглядати лог-файли та оперативно реагувати на збої або аномалії в роботі платформи.

Таким чином, у межах цього розділу було описано повний цикл створення, розгортання та підтримки SaaS-платформи, яка відповідає сучасним вимогам до продуктивності, безперервної інтеграції та надійного моніторингу.

# РОЗДІЛ 5. ОЦІНКА ЕФЕКТИВНОСТІ ТА МОЖЛИВОСТІ РОЗВИТКУ

## *5.1 Навантажувальне тестування та вимірювання продуктивності*

Для перевірки стабільності, швидкодії та масштабованості платформи було проведено комплексне навантажувальне тестування. Основною метою цього етапу було визначити граничні можливості обробки великої кількості запитів у реальному часі, виявити вузькі місця в архітектурі та оцінити час реакції системи за різних умов (рисунок 5.1).

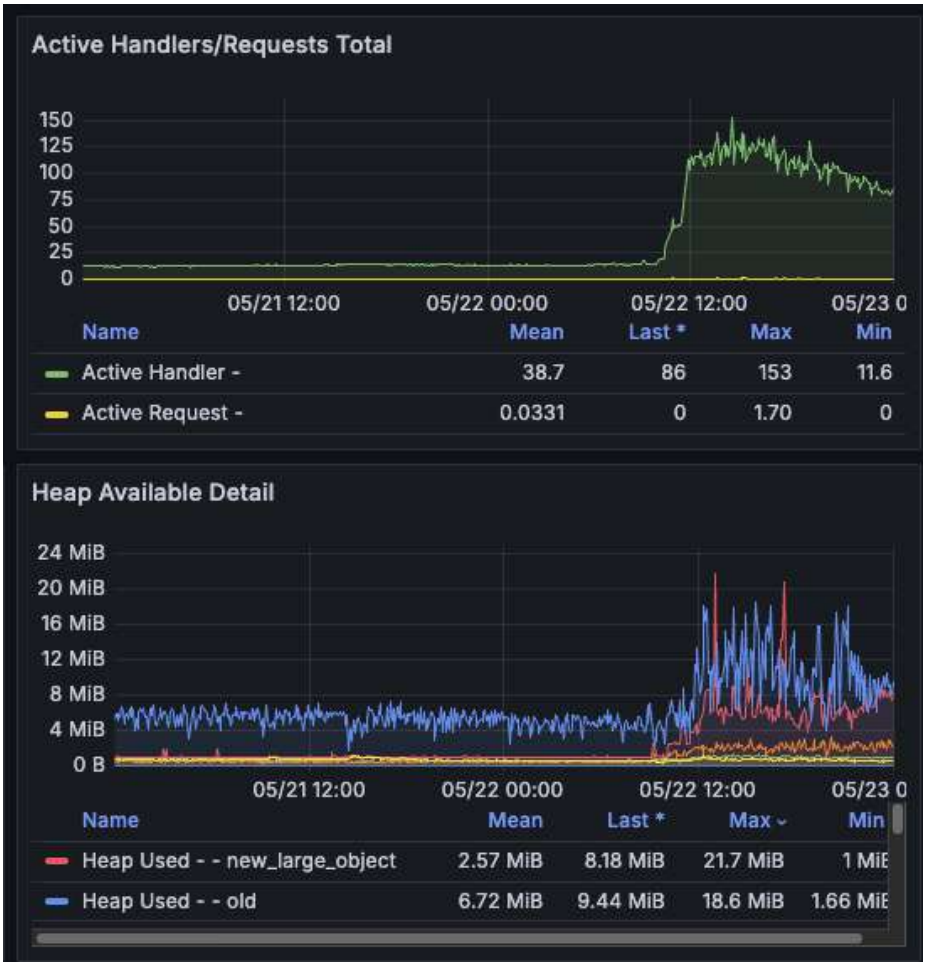


Рисунок 5.1 - Різке підняття використання ресурсів з початком тестування

Метою тестування є:

- Визначити максимальну кількість одночасно оброблюваних електронних адрес без зниження продуктивності.
- Оцінити середній час повного циклу перевірки однієї адреси електронної пошти (від надсилання до отримання результату).
- Перевірити стійкість сервісів до пікових навантажень.
- Змоделювати сценарії зростання черг повідомлень у RabbitMQ та обсягу подій у Kafka.

При тестуванні було використано наступні інструменти:

- k6 - для імітації навантаження на REST API;
- Apache JMeter - для моделювання масових запитів до валідатора;
- Prometheus + Grafana - для збору метрик у реальному часі;
- Loki + PM2 - для спостереження за логами та станом процесів під час тестування.

Задля оптимального результату було обрано кілька варіантів імітації навантаження:

1. Паралельне надсилання 10,000 електронних адрес з рівномірним розподілом протягом 10 хвилин.
2. Пікове навантаження - 1000 електронних адрес/сек протягом короткого періоду (burst-тест).
3. Довготривале навантаження - стабільний потік у 100 запитів/сек протягом 1 години.
4. Тест стабільності SMTP-симулятора - оцінка обробки великої кількості одночасних SMTP-сесій.
5. SSE-тест - перевірка поведінки системи при 1000 одночасно відкритих з’єднаннях на події Kafka.

Провівши тестування за описаними вище сценаріями отримані такі результати:

– Середній час повної валідації однієї адреси електронної пошти (включаючи SMTP): 420–650 мс, залежно від швидкодії зовнішніх поштових серверів.
– Максимальна стабільна пропускна здатність: понад 7,000 електронних адрес/хв без помітної деградації продуктивності.
– SSE впевнено підтримує >1000 одночасних з’єднань на звичайному EC2 t3.medium.
– Kafka успішно обробляє потік у 20,000 повідомлень/хв без втрат подій.
– RabbitMQ демонструє лінійне масштабування черг і споживачів без переповнення при зростанні навантаження.

Протягом процесу тестування використання ресурсів було підвищеним, проте не критичним та без помітних всплесків (рисунок 5.2).


Рисунок 5.2 - Метрики Grafana під час тестування

### *5.2 Виявлені вузькі місця та шляхи оптимізації*

- SMTP-діалоги є найбільш повільним етапом перевірки, бо залежать від зовнішніх серверів;
- при обробці понад 10 тис. одночасних перевірок помітне навантаження на базу даних (Supabase);
- швидкість оновлення статусів у Kafka може знижуватись, якщо повідомлення не батчуються.

Після проведеного тестування вирішено покращити підходи задля більш стабільної роботи при високому навантаження:

- введення обмеження на час SMTP-з’єднання (timeout 5 с);
- додавання кешування DNS/MX-запитів для пришвидшення повторної перевірки доменів;
- розподілення виконувачів валідатора на кілька інстансів із балансуванням через queue group;
- додаткове введення rate-limiting на API-рівні для уникнення зловживань.

### *5.3 Перспективи розвитку: API для інтеграцій, machine learning*

Розроблена SaaS-платформа вже на початковому етапі забезпечує широкий функціонал для високоточних перевірок електронних адрес, однак має значний потенціал для подальшого розвитку. Нижче наведено ключові напрями, які дозволять підвищити практичну цінність системи, адаптувати її під ширші бізнес-завдання та розширити коло користувачів.

Одним із пріоритетних напрямів розвитку є вдосконалення та стандартизація API з метою:

- інтеграції з CRM, ESP та маркетинговими платформами (наприклад, HubSpot, Mailchimp, ActiveCampaign);
- підтримки вебхуків та callback-механізмів для асинхронної обробки результатів;
- реалізації SDK для різних мов програмування (Node.js, Python, PHP), що полегшить підключення до платформи сторонніми розробниками;
- запровадження OAuth2 та granular-permission API-токенів для B2B-партнерств.

Це зробить платформу більш відкритою до корпоративного використання та дозволить розробникам легко вбудовувати валідацію електронних адрес у свої сервіси.

Хоча класичні методи валідації (синтаксис, SMTP, MX) є ефективними, вони не завжди здатні розпізнати адреси електронної пошти, пов’язані з ботами, масовими реєстраціями або спам-діяльністю. Тому наступним логічним кроком є впровадження нейромережевого класифікатора, що визначає “підозрілі” електронні адреси за рядом ознак:

- структура username (наявність keysmash-послідовностей, великої кількості цифр тощо);
- повторюваність шаблонів серед великої кількості електронних адрес;
- поєднання доменів і username, які часто зустрічаються в спам-базах;
- поведінкові сигнали (наприклад, одні й ті самі електронні адреси на різних IP).

Цей підхід дозволить формувати рівень спаму - числову оцінку ймовірності того, що адреса електронної пошти використовується не для легітимної реєстрації, а для шахрайських або автоматизованих дій.

У перспективі також можна інтегрувати:

- базу злитих електронних адрес (наприклад, “Have I Been Pwned”);
- сервіси репутаційної оцінки доменів (Spamhaus, Talos, Cisco Umbrella);

– локальні чорні списки, створені користувачами платформи.

Це дозволить адаптувати механізми фільтрації до актуальних загроз і захистити кінцевих користувачів ще до моменту першого запиту.

У майбутніх оновленнях планується додати:

– аналітичну панель із трендами типових помилок (наприклад, доменів, що часто повертають “catch-all”);
– рекомендації щодо фільтрації бази контактів;
– оцінку якісного складу бази електронних адрес для маркетингових рішень.

***Висновки до розділу 5***

У п’ятому розділі було здійснено оцінку ефективності розробленої платформи та визначено її потенціал до подальшого розвитку.

Під час навантажувального тестування проведено імітацію реального трафіку з високою інтенсивністю запитів. Тестування показало, що платформа здатна обробляти тисячі валідацій паралельно з мінімальними затримками завдяки асинхронній обробці. Було зафіксовано стабільну роботу системи при пікових навантаженнях.

У ході аналізу вузьких місць виявлено залежність загального часу перевірки від швидкості відповіді SMTP-серверів та необхідність гнучкого таймаут-контролю. Для вирішення цієї проблеми впроваджено механізм адаптивного обмеження таймаутів та повторних спроб. Також рекомендовано винести ML-аналіз у окремий виконувач з можливістю масштабування під окреме навантаження.

У підрозділі щодо перспектив розвитку було окреслено напрямки майбутнього розширення платформи:

- Розробка повноцінного публічного API для інтеграцій з іншими системами (CRM, реєстраційні форми тощо).
- Застосування моделей машинного навчання для класифікації електронних адрес за ризиком (вірогідність спаму, шкідливі дії, боти).
- Розширення функціоналу сповіщень, аналітики та аудиту для бізнес-користувачів.

Таким чином, оцінка продуктивності підтвердила стабільність і масштабованість системи, а визначені напрямки розвитку відкривають можливості для розширення функціоналу платформи та її подальшої комерціалізації.

# ВИСНОВКИ

У межах виконання магістерської роботи було спроєктовано та реалізовано масштабовану програмну платформу типу Software-as-a-Service для валідації електронних адрес у режимі реального часу. Основна увага приділялася побудові багаторівневого підходу до перевірки електронних адрес, який поєднує традиційні механізми валідації (синтаксичний аналіз, перевірка наявності домену та поштових серверів, емуляція SMTP-діалогу) з інтелектуальними методами, зокрема аналізом імені користувача за допомогою нейронної мережі для виявлення випадково згенерованих або ненадійних адрес. Архітектура системи побудована на основі мікросервісного підходу, що забезпечило гнучкість, масштабованість і можливість паралельної обробки великої кількості запитів.

Було реалізовано повний цикл обробки електронних адрес, що включає асинхронну постановку задач у чергу за допомогою RabbitMQ, обробку кожного запиту окремим сервісом валідації, передавання результатів через систему Kafka та відправлення сповіщень користувачу в реальному часі за допомогою технології Server-Sent Events. Значну увагу було приділено аспектам автоматизації розгортання (CI/CD), безпеки передачі даних, моніторингу стану сервісів та централізованого логування. Проведене навантажувальне тестування підтвердило ефективність обраної архітектури, виявило критичні точки навантаження й надало змогу оцінити продуктивність системи в умовах реального використання.

Результати роботи повністю відповідають цілям і завданням, поставленим у вступі. Зокрема, було створено функціональну архітектуру, адаптовану до розподіленої обробки запитів, реалізовано всі ключові етапи перевірки електронних адрес, включаючи мережеву перевірку та машинне навчання, побудовано інтерфейс користувача із відображенням статусів у режимі реального часу, розгорнуто засоби контролю працездатності

системи та автоматизовано процеси розгортання усіх компонентів платформи.

З метою подальшого розвитку платформи доцільно зосередити зусилля на таких напрямках: розширення функціональності API шляхом підтримки вебхуків, створення SDK для різних мов програмування та впровадження гнучкіших механізмів авторизації; удосконалення нейронної моделі для точнішого виявлення спамових або автоматично створених адрес, а також інтеграція з зовнішніми репутаційними системами; реалізація панелі аналітики для користувачів з рекомендаціями щодо покращення якості баз даних; підвищення відмовостійкості платформи шляхом розподілу черг за пріоритетами та масштабування брокерів повідомлень; адаптація рішення для корпоративного сегменту шляхом впровадження “white-label” моделей або інтеграційних модулів для бізнес-клієнтів.

Урахування зазначених напрямків дозволить перетворити платформу з сервісу перевірки електронних адрес у комплексну систему для управління якістю контактних даних з можливістю гнучкої адаптації до потреб різних категорій користувачів.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. NestJS – A progressive Node.js framework : [електронний ресурс] – Режим доступу: https://docs.nestjs.com – Назва з екрана.
2. Apache Kafka Documentation : [електронний ресурс] – Режим доступу: https://kafka.apache.org/documentation/ – Назва з екрана.
3. RabbitMQ Documentation : [електронний ресурс] – Режим доступу: https://www.rabbitmq.com/documentation.html – Назва з екрана.
4. Hickson I. Server-Sent Events // WHATWG Living Standard [Електронний ресурс]. – Режим доступу: https://html.spec.whatwg.org/multipage/server-sent-events.html – Назва з екрана.
5. Cloudflare. What is an MX record? // Cloudflare Learning Center [Електронний ресурс]. – Режим доступу: https://www.cloudflare.com/learning/dns/dns-records/dns-mx-record/ – Назва з екрана.
6. Простий поштовий транспортний протокол (SMTP) : [електронний ресурс] // IETF – Режим доступу: https://datatracker.ietf.org/doc/html/rfc5321 – Назва з екрана.
7. Формат повідомлень електронної пошти (Internet Message Format) : [електронний ресурс] // IETF – Режим доступу: https://datatracker.ietf.org/doc/html/rfc5322 – Назва з екрана.
8. TempMail – Temporary Disposable Email Address // TempMail [Електронний ресурс]. – Режим доступу: https://temp-mail.org – Назва з екрана.
9. GuerrillaMail – Disposable Temporary E-Mail Address // GuerrillaMail [Електронний ресурс]. – Режим доступу: https://www.guerrillamail.com – Назва з екрана.

10. What is an SNMP Trap? All About SNMP Traps : [електронний ресурс] // Solarwinds – Режим доступу: https://www.solarwinds.com/resources/it-glossary/snmp-traps – Назва з екрана.
11. ZeroBounce Documentation : [електронний ресурс] – Режим доступу: https://www.zerobounce.net/docs/ – Назва з екрана.
12. NeverBounce API Documentation : [електронний ресурс] – Режим доступу: https://developers.neverbounce.com – Назва з екрана.
13. EmailListVerify – Full-Featured Email Verification. // EmailListVerify [Електронний ресурс]. – Режим доступу: https://www.emaillistverify.com – Назва з екрана.
14. MailboxValidator – Email Verification Service // MailboxValidator [Електронний ресурс]. – Режим доступу: https://mailboxvalidator.com – Назва з екрана.
15. Fielding R. T. Architectural Styles and the Design of Network-based Software Architectures: Doctoral Dissertation. – University of California, Irvine, 2000. – [Електронний ресурс]. – Режим доступу: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm – Назва з екрана.
16. Prometheus Documentation : [електронний ресурс] – Режим доступу: https://prometheus.io/docs/ – Назва з екрана.
17. Grafana Documentation : [електронний ресурс] – Режим доступу: https://grafana.com/docs/ – Назва з екрана.
18. Loki Documentation : [електронний ресурс] – Режим доступу: https://grafana.com/oss/loki/ – Назва з екрана.
19. Hunter.io Email Verifier API : [електронний ресурс] – Режим доступу: https://hunter.io/api-docs#email-verifier – Назва з екрана.
20. Docker Documentation : [електронний ресурс] – Режим доступу: https://docs.docker.com – Назва з екрана.

21. Amazon Web Services Documentation : [електронний ресурс] – Режим доступу: https://docs.aws.amazon.com – Назва з екрана.
22. PM2 Documentation : [електронний ресурс] – Режим доступу: https://pm2.keymetrics.io/docs/ – Назва з екрана.
23. Email Validation Guide: Syntax, MX, SMTP, and more : [електронний ресурс] / Mailtrap – Режим доступу: https://mailtrap.io/blog/email-validation/ – Назва з екрана.
24. Supabase Documentation : [електронний ресурс] – Режим доступу: https://supabase.com/docs – Назва з екрана.
25. React Documentation : [електронний ресурс] – Режим доступу: https://react.dev/learn – Назва з екрана.
26. Google. Machine Learning Crash Course : [електронний ресурс] – Режим доступу: https://developers.google.com/machine-learning/crash-course – Назва з екрана.
27. Microsoft Azure Architecture Center. Best Practices for SaaS Applications : [електронний ресурс] – Режим доступу: https://learn.microsoft.com/en-us/azure/architecture/ – Назва з екрана.
28. Saini D. Email Validation Using Deep Learning : [електронний ресурс] // Towards Data Science, 2022 – Режим доступу: https://towardsdatascience.com/email-validation-using-deep-learning – Назва з екрана.