Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)
Факультет інформаційних технологій
(факультет)
Кафедра інформаційних технологій та комп'ютерної інженерії
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня магістра
(бакалавра, спеціаліста, магістра)

здобувача Акімов Валентин Вячеславович
(ПІБ)
академічної групи 123м-24-1
(шифр)
спеціальності 123 Комп'ютерна інженерія
(код і назва спеціальності)
за освітньо-професійною програмою «Комп'ютерна інженерія»
(офіційна назва)

на тему «SaaS-рішення для асинхронної перевірки та аналітики електронних адрес у реальному часі»
(назва за наказом ректора)

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтингово	**інституційно**	
кваліфікаційної робот	проф. Гнатушенко В.В.			
розділів:				

Рецензент				
Нормоконтролер	проф. Цвіркун Л.І.			

Дніпро
2025

ЗАТВЕРДЖЕНО:
завідувач кафедри
інформаційних технологій
та комп'ютерної інженерії
(повна назва)
_____Гнатушенко В.В.
(підпис) (прізвище,ініціали)

«_» _____ 2025 року

ЗАВДАННЯ
на кваліфікаційну роботу
ступеня магістр

здобувачу Акімов В. В. **академічної групи** 123м-24-1
(прізвище та ініціали) (шифр)

спеціальності 123 «Комп'ютерна інженерія»
за освітньо-професійною програмою «Комп'ютерна інженерія»
(офіційна назва)

на тему «SaaS-рішення для асинхронної перевірки та аналітики електронних адрес у реальному часі»

затверджену наказом ректора НТУ «Дніпровська політехніка» від 13.10.2025 р. 1165/с

Розділ	**Зміст**	**Термін виконання**
Стан питання та постановка завдання	На основі матеріалів виробничих практик, інших науково-технічних джерел сформулювати наукове завдання, конкретизувати предмет та мету досліджень	20.09.2025
Теоретичний	Обґрунтувати теоретичну базу розв'язання наукового завдання, якому присвячено роботу	15.10.2025
Експериментальний розділ	Проведення і обробка результатів експериментів	02.12.2025

Завдання видано _____ проф. Гнатушенко В.В.
(підпис керівника) (прізвище, ініціали)
Дата видачі _____
Дата подання до екзаменаційної комісії _____

Прийнято до виконання _____ Акімов В. В.
(підпис студента) (прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка: 71 с., 13 рис., 17 джерел, 1 додаток.

Об’єктом дослідження є процес створення високопродуктивної, масштабованої платформи для валідації електронних адрес (Email Validation SaaS) з використанням мікросервісної архітектури та сучасних систем моніторингу.

Метою роботи є обґрунтування, розробка та експериментальне підтвердження ефективності мікросервісної архітектури на базі Node.js, асинхронної обробки (RabbitMQ/Kafka) та інтегрованої системи моніторингу (Prometheus/Grafana/Loki) для забезпечення високої доступності та пропускної здатності сервісу валідації електронних адрес.

У першому розділі проаналізовано сучасний стан ринку валідаторів електронних адрес, визначено ключові вимоги до SaaS-платформ.

У другому розділі досліджено та обґрунтовано вибір технологічного стеку для бекенду, включаючи Node.js, асинхронні брокери повідомлень (RabbitMQ та Kafka) для рознесення навантаження та мікросервісну архітектуру.

У третьому розділі розроблено та описано функціональну модель багатоступеневої валідації, включаючи клієнтську перевірку синтаксису, перевірку DNS-записів та імітацію SMTP-з'єднання.

У четвертому розділі розроблено та реалізовано клієнтський інтерфейс на Python (Streamlit) для демонстрації функціональності. Впроваджено комплексну систему моніторингу, логування та сповіщень (PM2, Prometheus, Grafana, Loki), а також проведено тестування продуктивності. Експериментально підтверджено максимальну пропускну здатність платформи та оцінено середній час повної валідації однієї адреси, демонструючи стійкість системи до пікових навантажень.

Ключові слова: валідація електронних адрес, SaaS, мікросервіси, Node.js, Prometheus, Grafana, Loki, RabbitMQ, Kafka.

ЗМІСТ

Стор.

ВСТУП

Стрімкий розвиток цифрового маркетингу, електронної комерції та CRM-систем висуває жорсткі вимоги до якості даних, зокрема, до валідності та актуальності електронних адрес. Неперевірені бази даних призводять до значних фінансових втрат через високий відсоток відмов (bounce rate), погіршення репутації відправника (sender score) та блокування IP-адрес поштовими провайдерами. Використання неіснуючих або одноразових адрес спотворює статистику продажів та маркетингових кампаній, унеможливлюючи ефективне прийняття бізнес-рішень.

Актуальність теми посилюється складністю сучасних багаторівневих перевірок. Ефективний валідатор повинен не лише перевіряти синтаксис, але й виконувати глибокий аналіз DNS-записів та імітацію SMTP-з'єднання з поштовим сервером. Оскільки перевірка тисяч адрес вимагає значних обчислювальних ресурсів і часу, ключовою вимогою до сучасної платформи є масштабованість, висока пропускна здатність та асинхронна обробка запитів.

Метою даної роботи є обґрунтування, розробка та експериментальне підтвердження ефективності мікросервісної SaaS-платформи для валідації електронних адрес. Архітектура платформи базується на технологіях Node.js та асинхронних брокерах повідомлень (RabbitMQ/Kafka), що дозволяє забезпечити лінійне масштабування та стійкість до пікових навантажень. Важливим завданням також є інтеграція комплексної системи моніторингу (Prometheus, PM2) та логування (Loki, Grafana), яка гарантує прозорість роботи, швидке виявлення несправностей та контроль за використанням ресурсів у реальному часі.

Об'єктом дослідження є процес створення та експлуатації високопродуктивної платформи для масової перевірки електронних адрес.

Предметом дослідження є сукупність архітектурних, програмних та інфраструктурних рішень, спрямованих на максимізацію пропускної здатності та точності валідації в умовах розподіленого навантаження.

Практичне значення роботи полягає у створенні функціонального та протестованого програмного продукту – Email Validator SaaS, який може бути негайно розгорнутий та використаний для комерційних цілей, забезпечуючи високу якість даних для користувачів. Для досягнення поставленої мети в роботі необхідно розв’язати такі задачі:

1. Провести огляд сучасних методів перевірки валідності електронних адрес та аналіз наявних SaaS – сервісів.
2. Сформулювати технічні та функціональні вимоги до платформи.
3. Спроєктувати мікросервісну архітектуру з підтримкою масштабування й асинхронної обробки запитів.
4. Реалізувати багаторівневий алгоритм валідації, що включає синтаксичну перевірку, аналіз DNS і SMTP, а також методи машинного навчання.
5. Провести тестування продуктивності та точності перевірки, а також визначити потенційні вузькі місця системи.
6. Запропонувати напрями подальшого розвитку платформи та можливості її інтеграції із зовнішніми сервісами.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Актуальні проблеми перевірки валідності електронних адрес у веб-застосунках

У сучасних веб-застосунках електронна адреса виступає одним із базових ідентифікаторів користувача та використовується на різних етапах життєвого циклу облікового запису: під час реєстрації й авторизації, відновлення доступу, надсилання системних і транзакційних повідомлень, а також у межах маркетингових і інформаційних розсилок. Якість електронних адрес безпосередньо впливає на надійність комунікації між сервісом і користувачем, ефективність бізнес-процесів та рівень інформаційної безпеки [1-4].

Разом із тим, саме на етапі введення електронної адреси веб-застосунки часто стикаються з проблемою низької якості або навмисної фальсифікації контактних даних. Використання невалідних адрес призводить до фінансових втрат через неефективні розсилки, зростання навантаження на серверну інфраструктуру, погіршення репутації доменів відправника, а також створює додаткові ризики з точки зору безпеки та відповідності регуляторним вимогам [5-6].

У більшості веб-застосунків базова перевірка електронної адреси обмежується синтаксичною валідацією, яка здійснюється за допомогою регулярних виразів і дозволяє визначити відповідність формату адреси загальноприйнятим стандартам [7]. Такий підхід є простим у реалізації та малозатратним з точки зору обчислювальних ресурсів, однак він не забезпечує перевірку фактичного існування адреси та її придатності для доставки електронних повідомлень. Зокрема, адреси на неіснуючих або тимчасових доменах можуть успішно проходити синтаксичну перевірку, але бути непридатними для практичного використання.

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

– фальшиві та одноразові електронні адреси. Значна частина користувачів свідомо використовує тимчасові поштові сервіси для реєстрації, що унеможливлює подальшу комунікацію та знижує якість користувацької бази [8-9];

– некоректні або неактивні домени. Деякі домени можуть не мати коректних MX-записів, бути тимчасово недоступними або взагалі не підтримувати поштову інфраструктуру;

– SMTP-пастки та catch-all домени. Окремі поштові сервери навмисно налаштовані таким чином, щоб повертати позитивну відповідь на будь-який SMTP-запит, що ускладнює достовірну перевірку існування конкретної адреси [10];

– автоматизована “keysmash”-реєстрація. Використання випадково згенерованих імен користувачів у структурі електронних адрес є типовою ознакою бот-активності та масових атак на реєстраційні форми.

Додатковою складністю є обмеження з боку сторонніх SMTP-серверів [7], які можуть блокувати часті запити на перевірку або інтерпретувати такі звернення як спам-активність. Це суттєво ускладнює реалізацію прямої перевірки існування адреси через емуляцію SMTP-діалогу та вимагає застосування обережних механізмів контролю частоти запитів і асинхронної обробки.

З урахуванням наведених аспектів, актуальним є перехід від одноетапної синтаксичної перевірки до багаторівневих систем валідації електронних адрес. Такі системи мають поєднувати аналіз формату адреси, перевірку доменної інфраструктури, виявлення тимчасових поштових сервісів, обмежену SMTP-взаємодію та інтелектуальний аналіз ознак автоматичної генерації адрес. Комплексний підхід до валідації дозволяє суттєво підвищити

якість контактних даних у веб-застосунках, знизити технічні й фінансові ризики та забезпечити стабільну роботу сервісів у реальних умовах експлуатації.

У SaaS-платформах, фінансових сервісах та системах електронної комерції коректність електронної адреси безпосередньо впливає на якість обслуговування користувачів і стабільність бізнес-процесів. Попри це, валідація електронних адрес у багатьох веб-застосунках залишається поверхневою і зводиться до перевірки формального синтаксису введеного рядка. Типовим прикладом є використання регулярного виразу, який перевіряє наявність символу «@», доменного імені та допустимих символів у локальній частині адреси. Хоча такий підхід дозволяє відсіяти очевидно помилкові введення (наприклад, user@@domain або userdomain.com), він не дає жодної інформації про фактичне існування домену чи доступність поштової скриньки.

Наприклад, адреса test.user@nonexistent-domain-123.com повністю відповідає синтаксичним правилам, однак домен не має DNS-записів і не обслуговується жодним поштовим сервером. У реальному веб-застосунку така адреса може бути збережена в базі даних і використана для подальших розсилок, що неминуче призведе до помилок доставки та зниження репутації відправника.

Окремою проблемою є широке використання одноразових або тимчасових поштових сервісів. Сервіси на кшталт TempMail, GuerrillaMail або 10MinuteMail дозволяють користувачам швидко створювати електронні адреси без реєстрації та з обмеженим терміном існування. Наприклад, користувач може зареєструватися в системі, використовуючи адресу abc123@tempmail.com, підтвердити акаунт і більше ніколи не мати доступу до цієї пошти. У результаті сервіс втрачає можливість зворотного зв’язку з таким користувачем, а база контактів поступово наповнюється неактуальними адресами.

Крім того, значну загрозу для веб-застосунків становлять автоматизовані реєстрації, які здійснюються за допомогою ботів. Такі атаки часто супроводжуються масовим створенням облікових записів із випадково

згенерованими електронними адресами, наприклад asdkjfh23@randommail.net або qwertyuiop@domain.com. Подібні адреси можуть виглядати формально коректними, проте їхня структура (так званий keysmash) є характерною ознакою автоматичної генерації і може використовуватися для виявлення підозрілої активності.

Ще одним складним аспектом є перевірка адрес за допомогою SMTP-взаємодії. Теоретично, емуляція SMTP-діалогу дозволяє визначити, чи існує конкретна поштова скринька на сервері. Однак на практиці цей підхід має низку обмежень. По-перше, багато поштових серверів використовують механізм catch-all, за якого сервер приймає будь-яку адресу в межах домену, навіть якщо така скринька фактично не існує. Наприклад, сервер може позитивно відповідати як на адресу info@company.com, так і на random123@company.com, що робить перевірку недостовірною.

По-друге, часті SMTP-запити з боку одного сервісу можуть бути розцінені як підозріла активність або спроба збору адрес, унаслідок чого IP-адреса перевіряючої системи може бути тимчасово або постійно заблокована. Це особливо критично для веб-застосунків із високим трафіком, де перевірка електронних адрес виконується у великій кількості паралельних запитів.

Варто також враховувати юридичні та етичні аспекти. Деякі методи активної перевірки електронних адрес можуть суперечити політикам поштових провайдерів або вимогам до захисту персональних даних, зокрема в контексті регламенту GDPR. Це додатково обмежує можливість використання агресивних методів валідації в промислових системах.

З урахуванням наведених проблем, у сучасних веб-застосунках дедалі більшого поширення набувають багаторівневі підходи до валідації електронних адрес. Такі підходи поєднують синтаксичну перевірку, аналіз DNS-інфраструктури доменів, перевірку на наявність у списках тимчасових поштових сервісів, обмежену SMTP-взаємодію, а також інтелектуальний аналіз структури адреси на основі статистичних або нейромережевих моделей.

Наприклад, система може класифікувати адресу як «ймовірно автоматично згенеровану» на основі ентропії локальної частини або відсутності лінгвістичних шаблонів у імені користувача.

Таким чином, проблематика валідації електронних адрес у веб-застосунках виходить за межі простої перевірки формату і потребує комплексного, масштабованого та адаптивного підходу. Реалізація таких механізмів у вигляді спеціалізованих SaaS-рішень дозволяє значно підвищити якість користувацьких даних, знизити ризики зловживань і забезпечити стабільну роботу веб-систем у реальних умовах експлуатації.

1.2 Огляд існуючих рішень для валідації електронних адрес

Ринок цифрового маркетингу, SaaS-платформ і масових електронних розсилок демонструє стабільне зростання, що зумовлює підвищений попит на сервіси перевірки валідності електронних адрес. Для компаній, які працюють з великими обсягами контактних даних, актуальною є задача очищення баз користувачів, підвищення показників доставлення повідомлень (delivery rate), зменшення кількості відмов (bounce rate) та збереження позитивної репутації поштових доменів. У відповідь на ці потреби на ринку з’явилася значна кількість комерційних SaaS-рішень, орієнтованих на автоматизовану валідацію електронних адрес.

У межах цього підрозділу розглянуто найбільш поширені сервіси, зокрема ZeroBounce, NeverBounce, Hunter, EmailListVerify, MailboxValidator, а також низку альтернативних платформ, із точки зору їх функціональних можливостей, архітектурних особливостей, переваг і обмежень [11-16].

Сервіс ZeroBounce є одним із найбільш функціонально розвинених SaaS-рішень у сфері очищення та перевірки електронних адрес. Він орієнтований на корпоративних користувачів і великі маркетингові платформи. Основні можливості сервісу включають [11]:

– перевірку синтаксичної коректності адреси та доменного імені;

– аналіз DNS- та MX-записів;

– фільтрацію одноразових, спам-пасток і abuse-адрес;

– перевірку SMTP-доступності поштової скриньки;

– виявлення catch-all доменів;

– оцінювання активності електронної адреси на основі власних історичних баз даних.

ZeroBounce надає REST API для інтеграції з CRM- та email-маркетинговими системами, а також підтримує масове завантаження списків адрес у файлах. Разом із тим, суттєвими недоліками сервісу є закритість алгоритмів перевірки, відсутність можливості кастомізації логіки валідації та відносно висока вартість використання при великій кількості запитів, що обмежує його застосування для стартапів і малих проєктів [12].

NeverBounce позиціонується як високошвидкісний сервіс для перевірки електронних адрес у реальному часі та пакетної обробки списків контактів. До ключових можливостей платформи належать [13]:

– валідація синтаксису адреси та домену;

– перевірка наявності SMTP-серверів;

– класифікація адрес за категоріями valid, invalid, catch-all, unknown;

– інтеграція з популярними маркетинговими платформами, зокрема Mailchimp, HubSpot та Salesforce;

– підтримка перевірки в реальному часі через REST API.

Перевагою NeverBounce є простота інтеграції та орієнтація на бізнес-користувачів, однак сервіс має обмежені можливості налаштування. Користувач не має доступу до деталей логіки валідації та не може адаптувати алгоритми під специфічні вимоги власного веб-застосунку.

Сервіс Hunter більшою мірою орієнтований на пошук і перевірку корпоративних електронних адрес у B2B-сегменті. Його функціональні можливості включають [14]:

– перевірку існування домену та MX-записів;

– базову SMTP-перевірку;

– виявлення рольових адрес типу info@, sales@, admin@;

– ідентифікацію одноразових поштових сервісів;

– API для перевірки адрес у режимі реального часу.

Hunter активно використовується для формування B2B-контактних баз завдяки інтеграції з LinkedIn і CRM-системами. Водночас його функціонал є обмеженим у контексті глибокої технічної валідації, зокрема відсутні механізми інтелектуального аналізу структури імені користувача або оцінювання ймовірності автоматичної генерації адреси.

Окрім зазначених платформ, на ринку представлені менш популярні, але функціонально подібні рішення:

– EmailListVerify – сервіс із багаторівневою перевіркою, що включає DNS- і SMTP-аналіз, фільтрацію одноразових доменів і спам-пасток. Відзначається високою швидкістю обробки, проте його SMTP-перевірка менш стабільна для catch-all доменів [15];

– MailboxValidator – надає онлайн-інструмент і API для перевірки адрес, вирізняється простотою використання, однак поступається конкурентам за точністю та глибиною аналізу [16];

– Clearout, Verifalia, BriteVerify – пропонують стандартний набір функцій валідації, відрізняючись переважно моделлю ціноутворення, швидкістю обробки та стабільністю SMTP-перевірки [17-18].

Проведений аналіз показує, що попри високу популярність і широкий базовий функціонал, більшість наявних сервісів має низку спільних обмежень [19]:

– закритість алгоритмів перевірки, що унеможливлює адаптацію логіки валідації до конкретних бізнес-кейсів;

– обмежені можливості API, зокрема відсутність гнучкого керування асинхронною обробкою та детальними статусами перевірки;

– висока вартість при масштабному використанні, що є критичним фактором для стартапів і високонавантажених систем;

– недостатня підтримка інтелектуального аналізу, зокрема виявлення keysmash-адрес, автоматично згенерованих акаунтів і аномальних шаблонів у локальній частині адреси.

1.3 Основні вимоги до сучасної SaaS-платформи для валідації електронних адрес

Розроблення сучасної SaaS-платформи передбачає комплексне врахування функціональних і нефункціональних вимог, які визначають її ефективність, надійність, масштабованість і зручність експлуатації. У контексті задачі валідації електронних адрес ці вимоги набувають особливої ваги, оскільки платформа повинна працювати з великими обсягами даних, обробляти запити в режимі реального часу, взаємодіяти з зовнішніми мережевими сервісами та забезпечувати високий рівень захисту персональної інформації.

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

– підтримка багаторівневої перевірки електронних адрес, яка має включати: синтаксичну валідацію формату адреси відповідно до стандартів електронної пошти; перевірку на належність до одноразових або тимчасових поштових сервісів (disposable email); аналіз DNS- та MX-записів доменів; визначення наявності та доступності SMTP-серверів; емуляцію SMTP-діалогу з урахуванням обмежень сторонніх серверів;семантичну та статистичну оцінку локальної (username) частини адреси для виявлення автоматично згенерованих або підозрілих шаблонів.

– підтримка пакетної та поодинокої перевірки, що дозволяє використовувати платформу як для очищення великих списків контактів, так і для перевірки окремих адрес у режимі реального часу;

– перевірка в реальному часі, яка є критичною для інтеграції з формами реєстрації, CRM-системами, маркетинговими платформами та іншими веб-застосунками, де якість даних має бути гарантована ще на етапі введення;

– API-доступ для сторонніх систем, реалізований через документований REST/JSON-інтерфейс із підтримкою автентифікації та контролю доступу;

– веб-інтерфейс користувача, який забезпечує завантаження файлів зі списками адрес, перегляд результатів перевірки, базову аналітику та можливість експорту очищених даних.

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

– система має забезпечувати обробку десятків і сотень тисяч перевірок на добу без деградації продуктивності. Це передбачає використання мікросервісної архітектури, горизонтального масштабування компонентів і черг повідомлень для асинхронної обробки запитів;

– платформа повинна зберігати працездатність у разі часткових збоїв, тимчасової недоступності зовнішніх SMTP-серверів або помилок окремих компонентів, зокрема за рахунок повторних спроб, тайм-аутів і механізмів деградації функціоналу;

– середній час відповіді сервісу має залишатися стабільно низьким незалежно від обсягу оброблюваних даних. Для цього необхідні оптимізація SMTP-сесій, кешування DNS- та MX-результатів, а також ефективне управління пулом мережевих з’єднань;

– платформа повинна забезпечувати захист переданих і збережених даних шляхом використання HTTPS/TLS, захищеного зберігання API-ключів, контролю доступу, обмежень за IP-адресами та відповідності вимогам щодо захисту персональних даних, зокрема GDPR;

– необхідною є реалізація централізованої системи моніторингу та журналювання роботи компонентів платформи з використанням сучасних інструментів спостереження, що дозволяє оперативно виявляти збої, аналізувати продуктивність і реагувати на аномалії;

– доступ до платформи має здійснюватися через веб-інтерфейс, який коректно працює на різних пристроях і в різних браузерах без втрати функціональності.

– підтримка багатомовного інтерфейсу є важливою для використання платформи користувачами з різних регіонів і країн.

У контексті моделі SaaS до платформи висуваються також додаткові вимоги організаційного та бізнес-характеру:

– гнучка модель тарифікації, зокрема формат підписки або pay-as-you-go, що дозволяє оплачувати сервіс залежно від кількості перевірок або API-запитів;

– інтеграція з платіжними системами, такими як Stripe або PayPal, для автоматизації процесів оплати та керування підписками;

– аналітика використання сервісу, яка дозволяє відстежувати активність користувачів, популярність API-методів і обсяги оброблюваних даних;

– підготовка до маркетингової аналітики та SEO, що є необхідним для залучення нових користувачів і розвитку платформи як комерційного продукту.

Висновки до розділу 1

У першому розділі роботи проаналізовано теоретичні та прикладні аспекти валідації електронних адрес у сучасних веб-застосунках і SaaS-середовищі. Показано, що електронна адреса є ключовим ідентифікатором користувача та важливим елементом цифрової комунікації, а її якість безпосередньо впливає на ефективність бізнес-процесів, надійність сервісів і рівень інформаційної безпеки.

Розглянуто основні проблеми, пов’язані з перевіркою електронних адрес у веб-застосунках, зокрема обмеженість синтаксичної валідації, поширення одноразових поштових сервісів, використання автоматично згенерованих адрес, наявність catch-all доменів та обмеження з боку SMTP-серверів. Встановлено, що застосування лише базових методів перевірки не забезпечує належного рівня достовірності контактних даних і не відповідає вимогам сучасних високонавантажених систем.

У межах огляду існуючих SaaS-рішень для валідації електронних адрес проаналізовано функціональні можливості та обмеження найбільш поширених сервісів. Виявлено, що, попри широкий набір інструментів, більшість наявних платформ характеризується закритістю алгоритмів перевірки, обмеженими можливостями кастомізації, високою вартістю при масштабному використанні

та недостатньою підтримкою інтелектуального аналізу структури електронних адрес.

На основі проведеного аналізу сформульовано основні функціональні та нефункціональні вимоги до сучасної SaaS-платформи для валідації електронних адрес. Визначено необхідність підтримки багаторівневої перевірки, асинхронної обробки запитів, масштабованої мікросервісної архітектури, високої продуктивності, надійності та відповідності вимогам інформаційної безпеки й захисту персональних даних.

РОЗДІЛ 2. АРХІТЕКТУРА ТА ТЕХНОЛОГІЇ РОЗРОБКИ

2.1 Загальна архітектура платформи

Розроблена SaaS-платформа для асинхронної валідації та аналітики електронних адрес у реальному часі спроєктована на основі сучасної мікросервісної архітектури, що забезпечує незалежність компонентів, гнучкість розгортання, масштабованість і можливість подальшого розвитку системи. Обраний архітектурний підхід дозволяє ефективно обробляти великі обсяги запитів, мінімізувати затримки та підвищити стійкість платформи до пікових навантажень і часткових відмов.

Загальна структура системи умовно поділяється на п’ять основних рівнів: клієнтську частину (Frontend), серверну частину (Backend), чергу повідомлень (RabbitMQ), подієву шину (Apache Kafka) та сховище даних (Storage). Така декомпозиція відповідає принципам розділення відповідальностей і сприяє незалежному масштабуванню окремих компонентів.

Клієнтська частина платформи реалізована з використанням бібліотеки React.js, що забезпечує побудову інтерактивного, швидкого та адаптивного веб-інтерфейсу. Основними функціями клієнтської частини є:

– завантаження списків електронних адрес для перевірки через веб-форму або у вигляді CSV-файлів;

– відображення статусів перевірки в режимі реального часу з використанням механізму Server-Sent Events (SSE);

– візуалізація результатів валідації з детальною інформацією про причини невалідності (наприклад, некоректний формат, одноразовий домен, відсутність MX-записів або недоступність SMTP-сервера);

– експорт перевірених списків і зведеної аналітики у форматі CSV;

– управління обліковим записом користувача, тарифним планом, балансом перевірок та API-ключами для інтеграції зі сторонніми системами.

Серверна логіка платформи реалізована на основі фреймворку NestJS, який базується на мові TypeScript і підтримує модульну структуру, ін’єкцію

залежностей та використання декораторів. Backend виконує роль центрального керівного елемента системи та включає такі ключові компоненти:

– REST API, який обробляє запити з клієнтської частини та зовнішніх сервісів;

– валідаційний модуль, що ініціює та координує процес багаторівневої перевірки електронних адрес, включно з синтаксичною валідацією, DNS- та SMTP-перевірками, а також інтелектуальним аналізом структури адрес;

– SSE-шлюз (модуль сповіщень), який забезпечує доставку подій про зміну статусу перевірки безпосередньо в браузер користувача без необхідності опитування сервера;

– білінговий сервіс, відповідальний за облік виконаних перевірок, тарифікацію та взаємодію з платіжними системами.

Для реалізації асинхронної обробки запитів до валідації використовується брокер повідомлень RabbitMQ, який дозволяє ефективно розвантажити основний сервер і забезпечити стабільну роботу платформи в умовах високого навантаження. Застосування черги повідомлень надає такі переваги:

– зменшення пікових навантажень на Backend;

– рівномірний розподіл задач між окремими сервісами-виконувачами;

– можливість горизонтального масштабування шляхом додавання нових воркерів.

Взаємодія з чергою реалізована за моделлю «публікатор–споживач» (producer–consumer), де серверна частина додає задачі валідації до черги, а незалежні виконувачі здійснюють перевірку адрес і формують результати.

Подієва взаємодія між компонентами платформи реалізована за допомогою Apache Kafka, яка використовується для передавання подій про результати перевірки та зміни статусів. Kafka забезпечує високу пропускну здатність і надійність доставки повідомлень, що є критично важливим для обробки потоків подій у реальному часі. Отримані події обробляються сервісом сповіщень і транслюються кінцевим користувачам через SSE-з’єднання.

Сховище даних платформи реалізоване на основі реляційної системи керування базами даних PostgreSQL, яка забезпечує надійне зберігання:

– списків електронних адрес, завантажених користувачами;

– результатів валідації з детальною класифікацією та причинами;

– метаданих про користувачів, сесії, запити та тарифні плани;

– журналів подій, що використовуються для аналітики та аудиту.

Додатково в межах платформи використовується Supabase як бекенд-платформа, яка надає готові механізми автентифікації, управління ролями доступу, зберігання файлів і зручний RESTful API для взаємодії з клієнтською частиною.

2.2 Вибір технологій побудови SaaS-платформи

Для реалізації високопродуктивної та масштабованої SaaS-платформи з підтримкою асинхронної обробки даних і валідації електронних адрес у реальному часі було обрано сучасний технологічний стек, який поєднує інструменти для розробки клієнтської та серверної частин, брокери повідомлень, хмарну інфраструктуру й засоби контейнеризації. Основним критерієм відбору технологій стала їх здатність забезпечувати стабільність роботи системи, горизонтальне масштабування та простоту подальшого супроводу.

Клієнтська частина платформи реалізована з використанням бібліотеки React.js, яка є де-факто стандартом для побудови односторінкових веб-застосунків. Використання React дозволяє створити інтерактивний інтерфейс із мінімальними затримками оновлення стану та високою чутливістю до дій користувача. Основними перевагами обраної бібліотеки є [20]:

– компонентна модель побудови інтерфейсу, що спрощує повторне використання елементів і масштабування UI;

– ефективна робота з асинхронними подіями та потоками даних, зокрема інтеграція з механізмом Server-Sent Events (SSE) для відображення результатів перевірки в реальному часі;

– підтримка сучасних інструментів керування станом і стилізації, зокрема React Query, Zustand та TailwindCSS, що підвищує читабельність коду та швидкість розробки.

Серверна частина платформи побудована на основі NestJS – прогресивного фреймворку для Node.js, що використовує мову TypeScript і підтримує модульну архітектуру. Вибір NestJS зумовлений такими перевагами:

– зручна реалізація REST-API та механізмів push-сповіщень (SSE);

– нативна підтримка мікросервісної архітектури;

– проста інтеграція з брокерами повідомлень RabbitMQ і Apache Kafka;

– висока структурованість і читабельність коду, що полегшує тестування та розширення функціональності системи.

Як backend-as-a-service (BaaS) рішення використовується Supabase, яке забезпечує базовий функціонал зберігання та керування даними без необхідності реалізації низькорівневої інфраструктури. Supabase надає:

– реляційну базу даних PostgreSQL з автоматично згенерованим REST-інтерфейсом;

– механізми автентифікації та авторизації користувачів (email, OAuth, magic-link);

– файлове сховище для збереження результатів перевірок і звітів;

– керування доступом до даних за допомогою Row Level Security (RLS).

Застосування Supabase дозволяє суттєво скоротити час розробки типових сервісних функцій і зосередитися на реалізації бізнес-логіки валідації електронних адрес.

Для асинхронної фонової обробки задач використовується RabbitMQ – брокер повідомлень, що реалізує модель «публікатор–споживач». Його використання забезпечує:

– асинхронну постановку задач валідації в чергу;

– можливість горизонтального масштабування шляхом додавання нових worker-процесів;

– надійну доставку повідомлень і повторну обробку задач у разі помилок або збоїв.

Для потокової обробки подій, журналювання результатів і потенційного розширення платформи в напрямку аналітики в реальному часі інтегровано Apache Kafka. Її застосування дозволяє [21]:

– обробляти великі обсяги подій від різних компонентів системи;

– забезпечувати незалежну обробку повідомлень кількома сервісами;

– підключати додаткові аналітичні або ML-модулі без втручання в основну логіку;

– організовувати реальний час оновлення статусів валідації для користувачів.

Kafka виступає додатковим рівнем масштабованості та надійності, що є критично важливим для високонавантажених SaaS-рішень.

Контейнеризація сервісів реалізована за допомогою Docker, що дозволяє:

– створювати ізольовані середовища для кожного компонента платформи;

– уніфікувати процеси локальної розробки та хмарного розгортання;

– організовувати комплексне середовище через Docker Compose (frontend, backend, брокери повідомлень);

– закласти основу для реалізації CI/CD-процесів.

Хмарна інфраструктура платформи побудована на базі Amazon Web Services (AWS), що забезпечує необхідний рівень надійності, масштабованості та безпеки. У межах платформи використовуються [22]:

– EC2 для розгортання серверної частини та сервісів валідації;

– S3 для зберігання файлів і результатів обробки;

– Route 53 для керування DNS-записами;

– CloudFront для кешування та швидкої доставки клієнтського контенту;

– AWS Certificate Manager для автоматичного керування SSL-сертифікатами.

2.3 Архітектура мікросервісів та комунікація між ними

У межах розробки SaaS-платформи для асинхронної перевірки та аналітики електронних адрес у реальному часі було реалізовано мікросервісну архітектуру, яка забезпечує високу гнучкість системи, масштабованість та чітке розділення відповідальностей між функціональними компонентами. Такий підхід дозволяє незалежно розвивати, масштабувати та обслуговувати окремі сервіси без впливу на роботу всієї платформи [23].

Загальна архітектура платформи побудована відповідно до подієво-орієнтованої моделі (event-driven architecture) та включає п’ять основних компонентів: Backend API, Validator, Notifications, RabbitMQ та Apache Kafka. Взаємодія між ними здійснюється переважно асинхронно, що мінімізує затримки та підвищує відмовостійкість системи (рисунок 2.1).

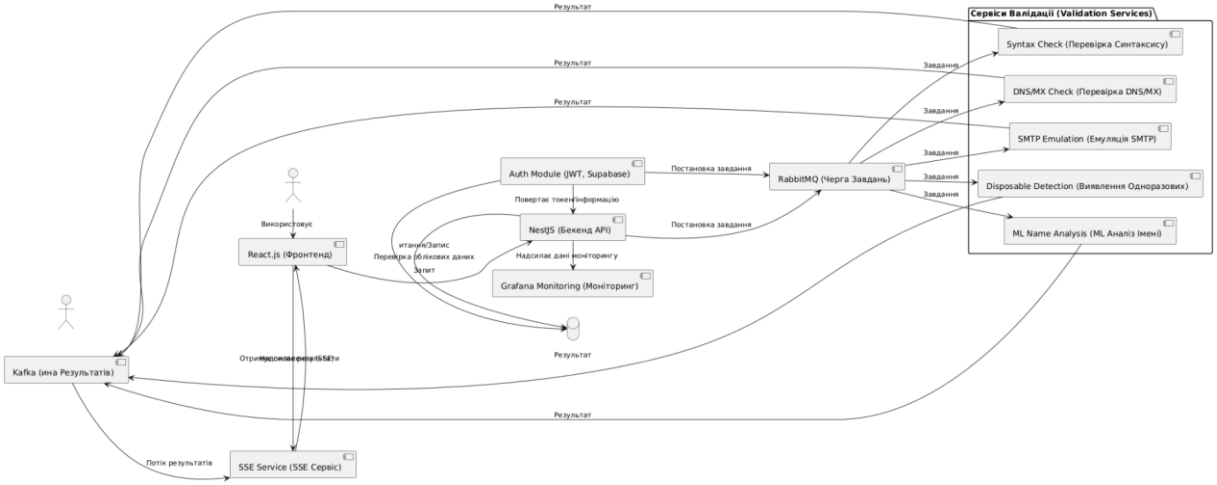

Рисунок 2.1 – Схема взаємодії мікросервісів платформи

Представлена архітектура описує сучасну, розподілену та асинхронну систему валідації адрес електронної пошти, яка використовує переваги мікросервісів та брокерів повідомлень для забезпечення високої пропускної здатності та надійності. Вона чітко розділена на шари: взаємодія з користувачем (фронтенд), логіка бекенду, управління даними, асинхронна черга завдань та спеціалізовані сервіси валідації, а також механізм повернення результатів у реальному часі.

Взаємодія починається з User (Користувач), який використовує React.js (Frontend) (Фронтенд). Фронтенд служить основним інтерфейсом, з якого ініціюються запити на валідацію. Після ініціації валідації, React.js надсилає синхронний Запит до центрального API, реалізованого на NestJS (Backend API) (Бекенд API) [24].

NestJS є ключовим компонентом, що відповідає за координацію роботи системи. Він інтегрований із Auth Module (JWT, Supabase) (Модуль Аутентифікації) для управління сесіями, перевірки прав доступу та авторизації, при цьому модуль аутентифікації виконує Перевірку облікових даних безпосередньо в Supabase DB, а потім повертає токен або інформацію до NestJS. Сам NestJS також взаємодіє із Supabase DB для читання та запису основних даних, пов'язаних із валідацією, користувачами чи налаштуваннями. Крім того, NestJS забезпечує моніторинг, надсилаючи дані моніторингу до Grafana Monitoring (Моніторинг), що дозволяє відстежувати продуктивність та стан системи.

Головна роль NestJS після отримання запиту на валідацію – це ініціювати асинхронний процес. Для цього він виконує Постановку завдання до RabbitMQ (Task Queue) (Черга Завдань). RabbitMQ діє як надійний брокер повідомлень, забезпечуючи, що кожен запит на валідацію буде гарантовано доставлений для обробки. Auth Module також може ставити завдання в RabbitMQ, наприклад, для обробки подій, пов'язаних з користувачами (наприклад, надсилання листа-підтвердження).

RabbitMQ (Task Queue) розподіляє равдання між п'ятьма спеціалізованими Validation Services (Сервіси Валідації) [25]. Кожен із цих сервісів є мікросервісом, який відповідає за певну частину процесу валідації. Вони включають: SMTP Emulation (Емуляція SMTP) для перевірки можливості доставки, Disposable Detection (Виявлення Одноразових) для ідентифікації тимчасових адрес, ML Name Analysis (ML Аналіз Імені) для аналізу імені користувача з використанням машинного навчання, Syntax Check (Перевірка Синтаксису) для відповідності формату, та DNS/MX Check (Перевірка DNS/MX) для перевірки доменних записів.

Після завершення своєї вузькоспеціалізованої перевірки, кожен із Сервісів Валідації не повертає результат безпосередньо бекенду, а публікує його як Результат у Kafka (Result Bus) (Шина Результатів). Kafka використовується як високопродуктивна та масштабована потокова платформа, яка збирає результати від усіх паралельних сервісів. Це забезпечує єдину точку збору всіх частин результату валідації.

Нарешті, SSE Service (SSE Сервіс) підписаний на Потік результатів із Kafka. Його функція – агрегувати всі отримані результати валідації та Надсилати результати в режимі реального часу назад до React.js (Фронтенд), використовуючи технологію Server-Sent Events (SSE). Це дозволяє користувачеві бачити оновлення статусу валідації на інтерфейсі без необхідності постійно оновлювати сторінку чи робити повторні запити (poll). Таким чином, система забезпечує ефективний, асинхронний та розподілений процес валідації з миттєвим інформуванням користувача про його хід та завершення.

Центральним елементом системи є Backend API, реалізований на основі фреймворку NestJS. Він виконує роль точки входу для клієнтських застосунків і сторонніх сервісів та забезпечує [26]:

– прийом запитів на перевірку електронних адрес через REST API або веб-інтерфейс;

– попередню обробку та валідацію вхідних даних;

– формування окремих задач для кожної електронної адреси;

– постановку задач у чергу повідомлень RabbitMQ;

– збереження метаданих запитів, інформації про користувачів, тарифні плани та результати перевірок у базі даних.

Для асинхронної передачі задач між сервісами використовується RabbitMQ, який виступає брокером повідомлень і реалізує модель producer–consumer. У контексті даної платформи RabbitMQ забезпечує:

– надійну доставку задач із підтвердженням отримання (acknowledgement);

– повторну обробку повідомлень у разі помилок;

– буферизацію запитів при пікових навантаженнях;

– можливість пріоритезації та горизонтального масштабування черг.

Ключовим обчислювальним компонентом системи є сервіс Validator, який реалізований у вигляді окремих worker-процесів. Кожен інстанс Validator:

– підключається до RabbitMQ як споживач повідомлень;

– отримує задачу перевірки електронної адреси;

– виконує багаторівневу валідацію, що включає:

– синтаксичну перевірку відповідно до стандарту RFC 5322;

– перевірку на належність до одноразових (disposable) доменів;

– DNS- та MX-перевірки домену;

– перевірку доступності SMTP-сервера;

– емульований SMTP-діалог для визначення існування адреси;

– інтелектуальний аналіз username-частини адреси з метою виявлення автоматично згенерованих або “keysmash” імен;

– формує структурований результат перевірки;

– публікує подію з результатами у систему потокової обробки Apache Kafka.

Apache Kafka у даній архітектурі виконує роль подієвої шини, що забезпечує передачу результатів перевірки між сервісами. Validator публікує події у відповідні топіки (наприклад, email-validations), які можуть споживатися кількома сервісами паралельно. Основними перевагами використання Kafka є [27]:

– висока пропускна здатність та мінімальні затримки;

– збереження порядку подій у межах теми;

– можливість масштабування шляхом додавання нових споживачів;

– готовність до подальшого підключення аналітичних або ML-модулів.

Доставку результатів перевірки до користувача в реальному часі забезпечує сервіс Notifications. Його функціональність включає:

– підписку на відповідні Kafka-топіки;

– обробку подій, що містять результати валідації;

– встановлення та підтримку відкритих SSE-з’єднань (Server-Sent Events) з клієнтськими браузерами;

– маршрутизацію повідомлень до відповідних користувацьких сесій.

Загальна послідовність взаємодії мікросервісів виглядає таким чином:

– користувач надсилає список електронних адрес через веб-інтерфейс або API;

– Backend API приймає запит і створює задачі на валідацію;

– задачі передаються у чергу RabbitMQ;

– сервіс Validator обробляє адреси та публікує результати у Kafka;

– сервіс Notifications отримує події та миттєво надсилає результати користувачеві через SSE.

Запропонована мікросервісна архітектура має низку суттєвих переваг:

– чітке розділення обов’язків, що спрощує супровід і тестування;

– горизонтальне масштабування кожного компонента незалежно;

– асинхронна обробка, яка забезпечує стабільну роботу при високих навантаженнях;

– подієва модель взаємодії, що полегшує розширення системи;

– реальний час доставки результатів, без додаткових запитів з боку клієнта.

2.4 Безпека та масштабованість платформи

У сучасній SaaS-платформі для валідації електронних адрес безпека даних користувачів, стійкість системи до збоїв та здатність обробляти зростаючі обсяги трафіку є критично важливими аспектами для забезпечення довіри та якості продукту. Розроблена платформа реалізована з урахуванням принципів secure by design та scale by design, що передбачає захист даних за замовчуванням та можливість гнучкого масштабування сервісів.

Централізована система аутентифікації користувачів реалізована через Supabase Auth, яка забезпечує:

– реєстрацію та вхід через email/password;

– підтвердження адреси електронної пошти через magic link;

– підтримку OAuth2 (Google, GitHub тощо);

– керування ролями та правами доступу на рівні бази даних через Row Level Security (RLS) PostgreSQL.

Це дозволяє ефективно захищати ресурси як на рівні API, так і на рівні збережених даних, навіть у разі потенційного прямого доступу до бази.

Вся взаємодія між клієнтською частиною, Backend API, брокерами повідомлень та базою даних здійснюється через захищені канали:

– використання HTTPS/TLS 1.2 та 1.3 для шифрування трафіку;

– автоматичне керування SSL-сертифікатами через AWS Certificate Manager;

– Frontend обслуговується через CloudFront з примусовим шифруванням і політиками HSTS;

– API захищено через reverse proxy з TLS-termination.

Ці заходи мінімізують ризики MITM-атак та витоку даних під час передачі.

Для ізоляції внутрішньої інфраструктури використовується Virtual Private Cloud (VPC), що забезпечує:

– розділення публічних та приватних підмереж;

– обмеження доступу до бази даних, брокерів та мікросервісів виключно авторизованими маршрутами;

– використання NAT/Internet Gateway для вихідного трафіку без відкриття вхідних портів;

– контроль трафіку між компонентами через Security Groups і Network ACLs.

– VPC створює безпечне та ізольоване середовище для всіх сервісів платформи.

Платформа підтримує горизонтальне масштабування критичних компонентів:

– Validator – паралельне оброблення задач у декількох інстансах;

– Notifications – масштабування для обслуговування великої кількості SSE-з’єднань;

– Kafka та RabbitMQ – кластеризація для збільшення пропускної здатності;

– Backend API – масштабування через ECS, EC2 або Kubernetes з балансуванням навантаження (ALB/ELB).

Додатково реалізовано кешування DNS/MX-запитів та повторне використання SMTP-з’єднань для оптимізації продуктивності.

Всі запити до API захищені JWT-токенами, які перевіряються на кожен запит. Додаткові заходи безпеки включають:

– rate limiting та throttling для запобігання DoS-атакам;

– облік запитів користувачів та інтеграція з білінговою системою (billing-aware architecture), що поєднує безпеку з контролем використання ресурсів.

Хмарне середовище AWS забезпечує [28]:

– високу доступність (high availability);

– резервування обчислювальних ресурсів та сховищ;

– автоматичне масштабування (Auto Scaling Groups);

– відновлення після збоїв завдяки використанню декількох Availability Zones.

2.5 Висновки до розділу 2

Розроблена SaaS-платформа для валідації електронних адрес побудована на основі сучасної мікросервісної архітектури, що забезпечує чітке розділення обов’язків між компонентами, гнучкість у розвитку системи та можливість масштабування відповідно до зростання навантаження.

Клієнтська частина реалізована на React.js, що дозволяє забезпечити інтерактивний та адаптивний веб-інтерфейс з підтримкою обробки результатів перевірки в реальному часі через SSE.

Серверна частина, побудована на NestJS, забезпечує модульність, простоту інтеграції з брокерами повідомлень (RabbitMQ, Kafka) та підтримку мікросервісної логіки, що дозволяє ефективно обробляти великі обсяги запитів.

Використання RabbitMQ та Kafka забезпечує асинхронну та подієву обробку завдань, гарантує надійну доставку повідомлень, підтримує горизонтальне масштабування сервісів і дозволяє реалізувати реальний час оновлення статусів перевірки.

База даних PostgreSQL у поєднанні з платформою Supabase забезпечує збереження даних, управління ролями доступу, автентифікацію користувачів та інтеграцію з RESTful API, що скорочує час розробки стандартних функцій та підвищує безпеку.

Система безпеки реалізована за принципами secure by design, включає централізовану аутентифікацію, шифрування трафіку (HTTPS/TLS), контроль доступу через RLS, захист API через JWT-токени, rate limiting та ізоляцію внутрішньої інфраструктури за допомогою VPC.

Платформа підтримує масштабування як горизонтальне (додавання нових інстансів Validator, Notifications), так і вертикальне, а також використання кластерів брокерів і балансування навантаження, що забезпечує стабільну роботу за високих навантажень та великих обсягів даних.

РОЗДІЛ 3. МОДУЛЬ ВАЛІДАЦІЇ ЕЛЕКТРОННИХ АДРЕС

3.1 Логіка та етапи валідації електронних адрес

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

На першому етапі здійснюється перевірка правильності формального запису електронної адреси відповідно до стандарту RFC 5322, що включає перевірку наявності символу @ та доменної частини, допустимості символів у локальній частині адреси та відсутності неприпустимих символів чи пробілів. Це дозволяє відразу відфільтрувати некоректні або порожні значення без необхідності звернення до зовнішніх ресурсів.

Наступним кроком є перевірка наявності DNS-записів для домену, що включає A/AAAA-записи для підтвердження існування домену та MX-записи для визначення поштових серверів, здатних обробляти кореспонденцію. У разі відсутності записів або помилки отримання результату адреса визнається недійсною. Далі проводиться поглиблена валідація MX-записів, що забезпечує перевірку наявності валідного сервера, доступності MX-адрес для SMTP-з’єднання та правильності пріоритетів і конфігурацій, що дозволяє впевнено стверджувати про технічну здатність домену приймати листи.

На наступному етапі виконується емуляція SMTP-сесії, яка передбачає встановлення з’єднання з сервером, виконання команд MAIL FROM та RCPT TO та аналіз відповіді сервера, що дозволяє визначити дійсність поштової скриньки. При цьому враховується можливість наявності catch-all доменів, які приймають повідомлення для будь-якого локального імені, що ускладнює визначення достовірності конкретної адреси через SMTP; для виявлення таких доменів здійснюються SMTP-запити до випадково згенерованих адрес і порівнюється їхня відповідь із реальною адресою. Останній етап перевірки

передбачає інтелектуальний аналіз імені користувача до символу @ із застосуванням нейронної мережі, натренованої для виявлення шаблонів типу “keysmash”, що дозволяє ідентифікувати потенційно автоматично згенеровані адреси.

Комбіноване використання класичних методів валідації, таких як синтаксис, DNS та SMTP, разом із сучасними методами машинного навчання забезпечує високу точність оцінки валідності електронних адрес, дозволяє виявляти нетипові або підозрілі шаблони, підвищує загальну якість контактної бази та дозволяє адаптувати процес перевірки до змін у поведінці користувачів та можливих зловживань. Такий поетапний підхід реалізовано у валідаційному модулі платформи як незалежну масштабовану службу, здатну обробляти запити у фоновому режимі, забезпечуючи швидку, надійну та ефективну перевірку кожної електронної адреси.

3.2 Алгоритми перевірки електронних адрес та управління таймаутами

У модулі валідації електронних адрес реалізовано комплекс алгоритмів, які виконуються послідовно та забезпечують перевірку адрес на відповідність формальним правилам, технічну доступність доменів і поштових скриньок, а також поведінковий аналіз. Кожен алгоритм розроблений з урахуванням практичних обмежень, таких як час виконання, мережеві помилки та доступність SMTP-серверів, і включає механізми відновлення у разі недоступності зовнішніх ресурсів. На початковому етапі здійснюється перевірка формату адреси відповідно до стандарту RFC 5322, де алгоритм аналізує допустимість символів у локальній та доменній частинах, довжину обох компонентів, наявність одного символу @ та відсутність неприпустимих символів, пробілів чи дужок. Ця детермінована перевірка дозволяє швидко відсіяти явно некоректні адреси без звернення до зовнішніх систем.

Наступним кроком відбувається аналіз наявності та коректності DNS- та MX-записів домену. Для цього здійснюється розв’язання доменного імені та

пошук MX-записів через DNS-запити, перевіряється IP-адреса поштового сервера та відкритість порту 25, а у разі відсутності MX-записів використовується A/AAAA-запис як резервний варіант. Результати DNS-запитів кешуються на визначений час, що дозволяє зменшити навантаження на сервери при масовій перевірці електронних адрес.

На наступному етапі алгоритм імітує процес доставки листа без фактичної відправки, встановлюючи TCP-з'єднання з SMTP-сервером і виконуючи стандартні команди HELO/EHLO, MAIL FROM та RCPT TO. Відповіді сервера аналізуються для визначення статусу адреси, включно з кодами успішного прийому або помилок, а сесія закривається командою QUIT. Для обробки тимчасових помилок реалізована логіка повторних спроб та обробки catch-all доменів через генерацію фіктивних адрес.

Ураховуючи особливості поведінки SMTP-серверів, що можуть використовувати механізми обмеження швидкості або штучне затягування відповіді, у системі встановлено обмеження часу на TCP-з'єднання, timeout читання відповіді та максимальну тривалість всієї сесії перевірки однієї адреси. При перевищенні цих обмежень перевірка адреси припиняється, а вона отримує статус «невизначено» із відповідною позначкою, наприклад «timeout» або «temporary error». Передбачено повторні спроби для нестабільних серверів з обмеженою кількістю повторів, що підвищує надійність процесу перевірки.

Алгоритми також оптимізовані для обробки груп електронних адрес, включно з повторним використанням відкритого з'єднання, об'єднанням запитів до одного SMTP-сервера для кількох адрес одного домену та пріоритезацією перевірок для доменів із найменшою затримкою. Це дозволяє значно підвищити швидкість обробки великих списків адрес і зменшити навантаження на систему.

Завдяки поєднанню класичних алгоритмів перевірки формату, DNS та SMTP із сучасним ML-аналізом імен користувачів, модуль валідації забезпечує ефективне визначення валідності адрес, дозволяє ідентифікувати підозрілі або автоматично згенеровані адреси та підтримує баланс між точністю, швидкістю виконання та економічною ефективністю обробки запитів.

3.3 Асинхронна обробка та механізми черг повідомлень

У платформі валідації електронних адрес ключовим аспектом є забезпечення асинхронної, надійної та масштабованої обробки запитів, яка не блокує основні компоненти системи, зокрема API та інтерфейс користувача. Для цього застосовано два незалежні, але взаємодіючі механізми черг повідомлень – RabbitMQ та Apache Kafka, які виконують різні ролі в архітектурі та доповнюють одне одного, забезпечуючи ефективну обробку завдань і передачу результатів у реальному часі.

RabbitMQ використовується як брокер завдань для валідаційного модуля. Його основна функція полягає у прийомі завдань від серверної частини та передачі їх виконувачам validator у фоновому режимі. Такий підхід дозволяє уникнути блокування HTTP-запитів користувача, паралелізувати обробку великої кількості електронних адрес, балансувати навантаження між кількома інстансами validator і гарантувати повторну обробку у разі збою виконувача. Кожен запит формулюється у форматі JSON і публікується в чергу, після чого один із виконувачів зчитує повідомлення, виконує перевірку та надсилає результат далі через Kafka. У випадку збою виконувач не підтверджує обробку (ack), і повідомлення повертається до черги, що забезпечує стійкість системи до втрати завдань.

Apache Kafka використовується як система подій (event bus), яка відповідає за передачу результатів перевірки від validator до сервісу notifications. Основним завданням Kafka є розповсюдження повідомлень про завершення обробки між кількома підписниками з мінімальною затримкою. Це дозволяє підтримувати високий рівень пропускної здатності при потоковій передачі подій, гарантує надійну доставку завдяки журналюванню і збереженню подій, забезпечує масштабованість через теми та групи споживачів і дозволяє незалежно додавати нові компоненти, такі як аналітика, логування або ML-підсистеми, які підписуються на ті самі події.

У типовому сценарії роботи платформи сервіс validator після завершення перевірки публікує повідомлення в Kafka-топік validation.results, а сервіс notifications підписаний на цей топік і миттєво отримує інформацію.

Notifications передає результати користувачу через відкриті SSE-з’єднання, що дозволяє отримувати оновлення в реальному часі з мінімальним навантаженням на основний API-сервер.

Використання одночасно RabbitMQ та Kafka обумовлено розподілом функціональності між внутрішньою оркестрацією обробки та зовнішньою дистрибуцією результатів. RabbitMQ ефективно справляється з чітко визначеними завданнями, де важлива гарантія обробки кожного повідомлення, тоді як Kafka ідеально підходить для широкомовної подієвої комунікації, коли один результат необхідно передати декільком системам одночасно. Такий підхід забезпечує надійність, масштабованість і гнучкість платформи, дозволяючи підтримувати обробку великих обсягів запитів у режимі реального часу та ефективно інтегрувати нові сервіси без порушення існуючої логіки системи.

3.4 Висновки до розділу 3

В даному розділі детально розглянуто структуру та принципи роботи модуля валідації електронних адрес, що є ключовим компонентом платформи. Було показано, що процес валідації реалізований як багаторівнева послідовність алгоритмів, які поєднують перевірку формальних правил, технічну доступність доменів і поштових скриньок, а також поведінковий аналіз імен користувачів із застосуванням методів машинного навчання. Такий підхід дозволяє підвищити точність оцінки валідності адрес і забезпечує ефективну фільтрацію підозрілих чи некоректних контактів.

Описано алгоритми перевірки, що охоплюють контроль формату відповідно до стандарту RFC 5322, визначення наявності та конфігурації DNS- і MX-записів, емуляцію SMTP-сесій із обробкою таймаутів і повторних спроб, а також виявлення catch-all доменів. Реалізація цих алгоритмів передбачає оптимізацію під масову обробку адрес, кешування результатів та паралельну обробку для прискорення виконання перевірок.

Також у розділі висвітлено організацію асинхронної обробки запитів із використанням RabbitMQ і Apache Kafka. RabbitMQ забезпечує надійну

внутрішню оркестрацію завдань валідації, а Kafka відповідає за потокову дистрибуцію результатів до сервісу notifications та інших підписників у реальному часі. Подвійне застосування цих технологій дозволяє досягти високої масштабованості, відмовостійкості та мінімального навантаження на основний API-сервер.

У результаті модуль валідації реалізовано як автономну, масштабовану та стійку до збоїв службу, що забезпечує швидку та точну перевірку електронних адрес, інтегрується з іншими компонентами платформи та відкриває можливості для подальшого розвитку, зокрема додавання нових типів перевірок, аналітичних сервісів або інтеграції з зовнішніми системами.

РОЗДІЛ 4. РЕАЛІЗАЦІЯ SAAS-ПЛАТФОРМИ ДЛЯ АСИНХРОННОЇ ВАЛІДАЦІЇ ТА АНАЛІТИКИ ЕЛЕКТРОННИХ АДРЕС

4.1 Розробка фронтенду для SaaS-рішення

Клієнтська частина SaaS-платформи для валідації електронних адрес розроблена із застосуванням сучасного стеку технологій, з фокусом на швидкодію, інтерактивність і зручність користувача (рис.4.1).

Рисунок 4.1 – Авторизація

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 чітке розділення відповідальностей між елементами інтерфейсу. Мова програмування – TypeScript, яка надає статичну типізацію, значно знижує ризик помилок на етапі розробки та покращує зручність роботи над великою кодовою базою. 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 (як показано на наданому скріншоті, де відображено розділ пакетної перевірки) (рис. 4.2).

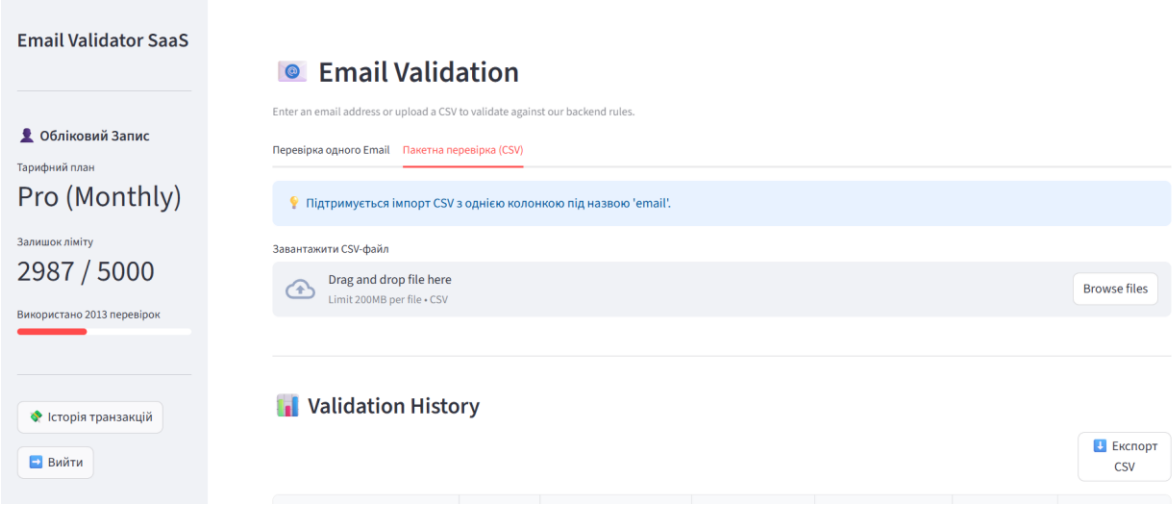

Рисунок 4.2 – Клієнтська частина

Користувацький інтерфейс виконує низку ключових функцій. По-перше, він забезпечує зручне введення електронних адрес: як вручну, так і через імпорт файлів у форматі CSV з підтримкою функції Drag and Drop. По-друге, користувач отримує результати перевірки у режимі реального часу. Цю критично важливу функціональність реалізовано через інтеграцію з сервером за допомогою технології Server-Sent Events (SSE). Використання SSE дозволяє передавати оновлення статусу перевірки миттєво, без потреби в циклічному опитуванні (polling) API, що зберігає ресурси клієнта і сервера. Результати перевірки подаються у вигляді таблиці з інформативними статусами, кольоровим маркуванням, поясненнями причин недійсності адреси та можливістю експорту очищеного списку.

Окрему увагу приділено управлінню обліковим записом. Користувач може переглядати інформацію про свій тарифний план, залишок ліміту на перевірки та історію транзакцій. Система також використовує toast-сповіщення для інформування про успішні операції, помилки або системні події. Застосунок побудований так, щоб гарантувати безперебійну роботу навіть за великого навантаження – важкі компоненти, що не є критичними для початкового

завантаження, підвантажуються динамічно (lazy loading), що дозволяє зберегти продуктивність і швидкий рендер сторінок (рис.4.3).

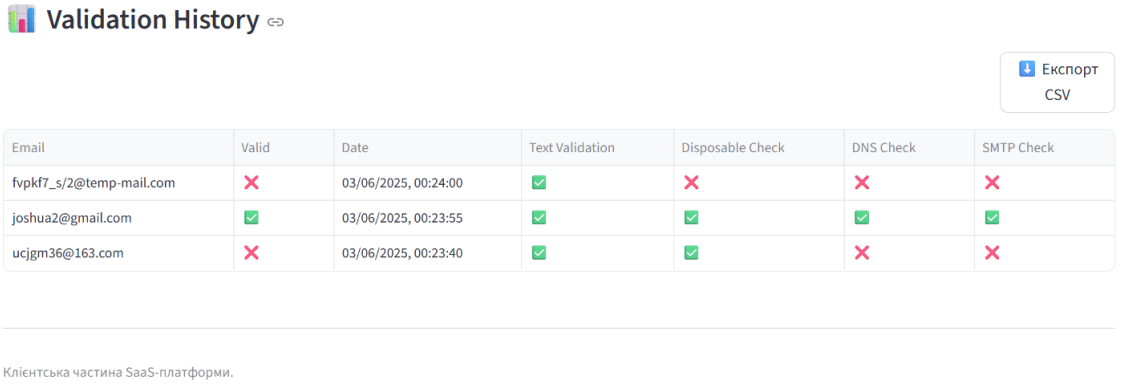

Рисунок 4.3 – Результат облікових записів

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 сервісні обгортки (hooks або менеджери API), що забезпечують централізовану обробку помилок, автоматичні повторні спроби, контроль тайм-аутів та повторне встановлення з'єднань при втраті доступу до мережі.

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 синтаксису електронної адреси, відповідність CSV-файлів очікуваній структурі та мінімізується ризик надсилання некоректних запитів на сервер. Загалом, клієнтська частина платформи є високопродуктивним, повноцінним та інтерактивним засобом доступу до основного функціоналу сервісу.

4.2 Розробка бекенду для SaaS-рішення

Серверна частина (Backend) SaaS-платформи є ключовим компонентом, що відповідає за виконання основної бізнес-логіки: асинхронну валідацію

електронних адрес, управління даними користувачів та забезпечення надійної взаємодії з клієнтом. Для реалізації бекенду використано мову програмування Go (Golang), що обумовлено її високою продуктивністю, ефективною підтримкою конкурентності (через goroutines) та низьким споживанням ресурсів, що є критично важливим для обслуговування великої кількості асинхронних запитів на перевірку. Як основне сховище даних обрана реляційна база даних PostgreSQL за її надійність, підтримку складних транзакцій та потужні інструменти індексування, тоді як Redis використовується як високошвидкісний кеш і найважливіше – як брокер повідомлень для реалізації черги завдань, що дозволяє ефективно керувати асинхронними завданнями з валідації великих CSV-файлів. Уся архітектура розгорнута за допомогою Docker та Docker Compose для забезпечення ізоляції, портативності та легкого масштабування.

Серверна частина функціонально поділена на кілька взаємодіючих модулів. Перший, API Gateway та Обробник REST-запитів, відповідає за прийом усіх вхідних запитів від клієнтської частини. Він забезпечує аутентифікацію та авторизацію користувачів з використанням JWT-токенів, обробку запитів на перевірку однієї адреси, а також прийом, попередню обробку та збереження метаданих завантажених CSV-файлів. Другий модуль, Менеджер Завдань (Task Manager), є серцем асинхронної обробки. Коли користувач ініціює пакетну перевірку, цей менеджер розбиває великий файл на менші порції (чанки) і поміщає ці порції у чергу завдань у Redis, а також відстежує загальний прогрес виконання пакетного завдання.

Виконання безпосередньої валідації покладається на Воркери Валідації (Validation Workers), які є незалежними процесами, реалізованими як goroutines у Go. Вони постійно моніторять чергу завдань, витягуючи наступну доступну порцію електронних адрес, виконують комплексну багатоетапну валідацію, що включає перевірку синтаксису, DNS-записів (MX-записи) та SMTP-підключення. Результати перевірки (валідна/невалідна, причина) після завершення записуються назад до бази даних PostgreSQL. Четвертий, Модуль Сповіщень (Notification Module), забезпечує функціональність оновлення

результатів у реальному часі. Цей модуль тісно пов'язаний з Task Manager і використовує технологію Server-Sent Events (SSE), щоб миттєво надсилати клієнту сповіщення про зміну статусу пакетної перевірки, що запобігає необхідності постійного опитування сервера з боку клієнта.

Забезпечення високої продуктивності та надійності досягається завдяки використанню goroutines, які дозволяють серверу одночасно обслуговувати тисячі клієнтських з'єднань та виконувати паралельні завдання валідації без значних затримок. Інтеграція Redis забезпечує не лише швидкий доступ до кешованих даних, але й стійку роботу черги завдань, що запобігає втраті даних або перевантаженню основної бази під час пікових навантажень. Таким чином, серверна частина є надійною, високопродуктивною та масштабованою системою, здатною обробляти великі обсяги даних у режимі, наближеному до реального часу.

Користувацький інтерфейс виконує низку ключових функцій, які відображаються на екранах валідації. По-перше, він забезпечує два основні режими введення електронних адрес:

Перевірка однієї адреси (рис. 4.4 та рис.4.5), де користувач вводить адресу у відповідне поле і натискає кнопку "Validate Email". Результат відображається в клієнтській частині (рис.4.6).

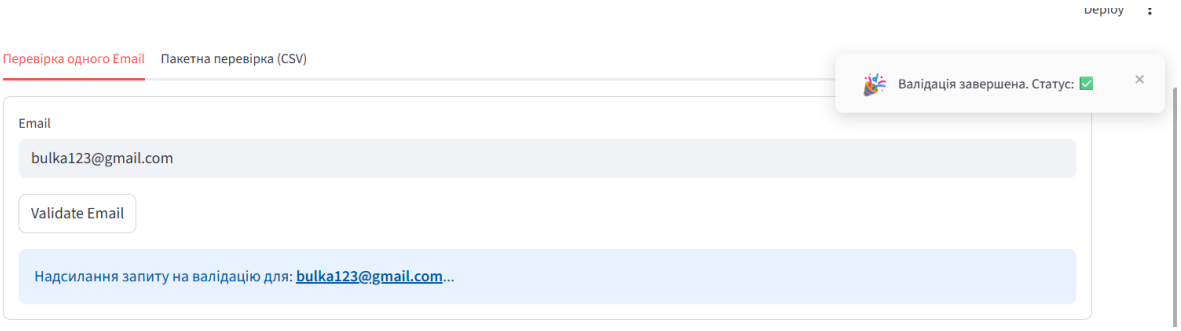

Рисунок 4.4 – Результат перевірки однієї адреси для домену gmail

Пакетна перевірка (CSV), де реалізовано механізм завантаження файлів за допомогою Drag and Drop із чітким обмеженням (Limit 200MB) та вимогою до структури файлу: "Підтримується імпорт CSV з однією колонкою під назвою 'email'" (рис.4.7).

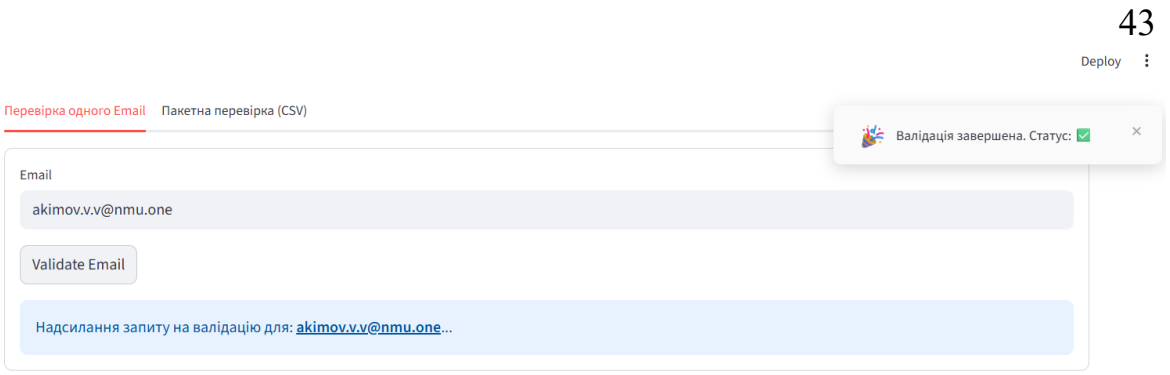

Рисунок 4.5 – Результат перевірки однієї адреси для домену nmu

По-друге, користувач отримує результати перевірки у режимі реального часу. Це реалізовано через інтеграцію з сервером через технологію Server-Sent Events (SSE). Інтерфейс миттєво реагує на події від сервера: під час надсилання запиту відображається повідомлення "Надсилання запиту на валідацію..." (як видно на image_5502a0.png та image_55069f.png), а після завершення перевірки відображається toast-сповіщення "Валідація завершена. ".

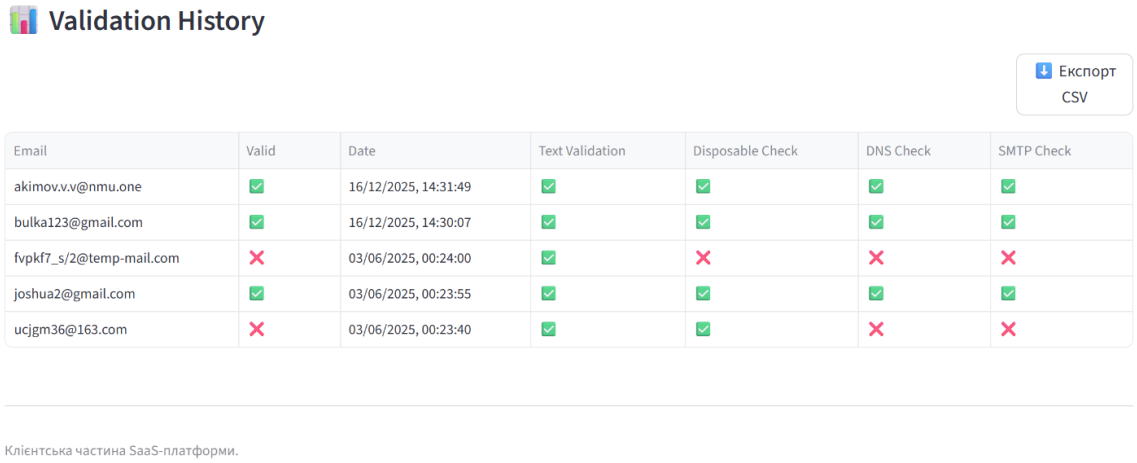

Рисунок 4.6 – Результат перевірки адрес

Окрему увагу приділено управлінню обліковим записом, що видно на бічній панелі. Тут користувач може переглядати свій Тарифний план (Pro (Monthly)), Залишок ліміту (2987 / 5000), а також має доступ до Історії транзакцій та функцій виходу. Інтерфейс є адаптивним, гарантує безперебійну роботу завдяки динамічному підвантаженню компонентів (lazy loading) та забезпечує клієнтську валідацію введених даних. З технічної точки зору, React-

застосунок взаємодіє з бекендом через REST API для управління акаунтом та SSE для підписки на події зміни статусу перевірки.

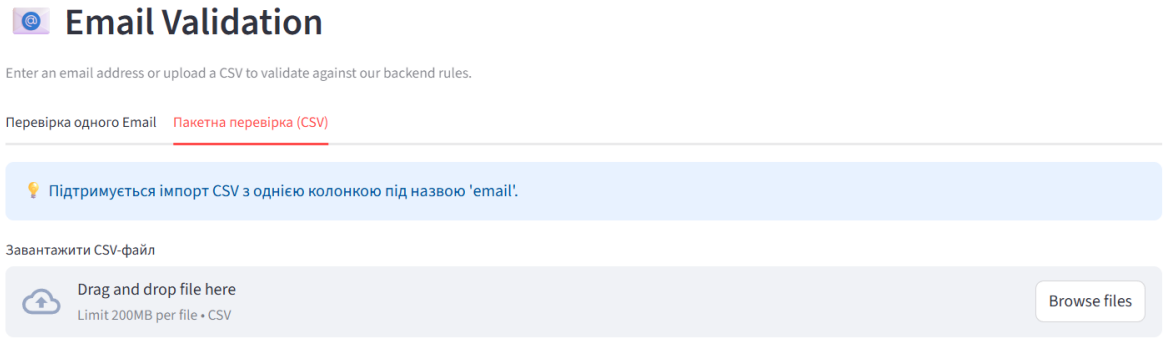

Рисунок 4.7 – Результат пакетної перевірки

4.3 Автоматизоване розгортання на основі CI/CD-пайплайни

Для забезпечення швидкого та надійного циклу розробки, тестування та доставки функціоналу SaaS-платформи була розгорнута система безперервної інтеграції та безперервної доставки (CI/CD). Використання автоматизованих пайплайнів дозволяє мінімізувати ручні операції, гарантує консистентність середовищ та значно скорочує час виведення нових функцій у промислове використання (Time-to-Market). Вся інфраструктура CI/CD побудована навколо платформи, як-от GitHub Actions, який інтегровано безпосередньо з репозиторієм вихідного коду.

Пайплайн структуровано на кілька послідовних стадій. Перша стадія – Інтеграція та Тестування. Кожне злиття коду у головну гілку (або Pull Request) ініціює автоматичне збирання проєкту. Для клієнтської частини, розробленої на React.js/TypeScript, виконується компіляція з використанням Vite, лінтинг та модульне тестування. Для серверної частини, написаної на Go, проводиться збирання бінарних файлів, статичний аналіз коду та виконання юніт-тестів і функціональних тестів. Успішне проходження цієї стадії є обов'язковою умовою для подальшого просування.

Наступна критично важлива стадія – контейнеризація та реєстрація. Оскільки і клієнтська, і серверна частини розгорнуті за допомогою Docker, на цій стадії створюються чисті та оптимізовані образи Docker для кожного

компонента. Образи клієнтського застосунку (збірка React) та серверного застосунку (бінарник Go) маркуються унікальною версією, а потім реєструються у централізованому Docker Registry. Це забезпечує можливість швидкого відкоту (rollback) до попередньої стабільної версії у разі виникнення проблем на етапі розгортання.

Кінцева стадія – автоматизоване розгортання (CD). Після успішної контейнеризації пайплайн автоматично оновлює інстанси застосунку у цільовому середовищі (наприклад, Production або Staging). Для цього використовуються інструменти оркестрації, як-от Docker Compose або Kubernetes, які забезпечують безперебійне оновлення без простоїв (Zero-Downtime Deployment). Система автоматично забирає нові Docker-образи з реєстру, зупиняє старі контейнери та запускає нові, підключаючи їх до мережі та перевіряючи їхній стан за допомогою Health Checks. Це гарантує, що розгортання нових функцій відбувається швидко, надійно та прозоро для кінцевого користувача.

4.4 Моніторинг, логування системи сповіщення

Для забезпечення стабільної роботи платформи Email Validation SaaS в режимі 24/7 та швидкого реагування на інциденти, реалізовано комплексну архітектуру моніторингу, яка охоплює усі ключові компоненти, від серверної інфраструктури до клієнтської частини. Ця система використовує комбінацію PM2, Prometheus, Grafana та Loki для забезпечення візуалізації метрик, централізованого збору журналів подій та автоматичних сповіщень.

На нижчому рівні, для запуску та постійного контролю за станом сервісів (таких як основний бекенд та сервіс сповіщень), використовується PM2 у поєднанні з PM2 Plus (як імітовано у таблиці моніторингу). Цей інструмент забезпечує високу доступність, автоматично перезапускаючи процеси у разі збоїв, та дозволяє відстежувати ключові показники продуктивності. З PM2 ми отримуємо дані в реальному часі про завантаження процесора (CPU), використання оперативної пам'яті (RAM), затримки Event Loop та кількість перезапусків для кожного сервісу (email-validator, kafka, redis, zookeeper). PM2

підтримує кластерний режим для горизонтального масштабування, забезпечуючи готовність системи до зростання навантаження.

Збір кількісних показників продуктивності та навантаження реалізовано за допомогою системи Prometheus. Кожен сервіс платформи, включаючи воркери валідації, експортує метрики на спеціальний HTTP-шлях /metrics. Ці метрики включають детальні дані, такі як загальна кількість оброблених електронних адрес (validated_emails_total), час відповіді SMTP-серверів, навантаження на черги завдань, та частота помилок. Prometheus регулярно опитує ці шляхи, збирає та зберігає історичні дані, що є основою для подальшого аналізу та побудови аналітичних графіків.

Візуальне представлення усіх зібраних метрик та логів забезпечується через дашборди Grafana (рис.4.8), які підключені як до Prometheus (для числових даних), так і до Loki (для журналів, рис.4.9). У Grafana побудовано окремі інформаційні панелі, де можна в реальному часі бачити активність системи, наприклад, кількість перевірок, загальну частоту помилок та час відповіді SMTP-серверів. Більше того, у Grafana реалізовано систему автоматичних сповіщень (alerts), які спрацьовують, коли метрики перевищують критичні пороги (наприклад, висока затримка або велика кількість невдалих підключень), з інтеграцією у зовнішні системи, як-от Slack.

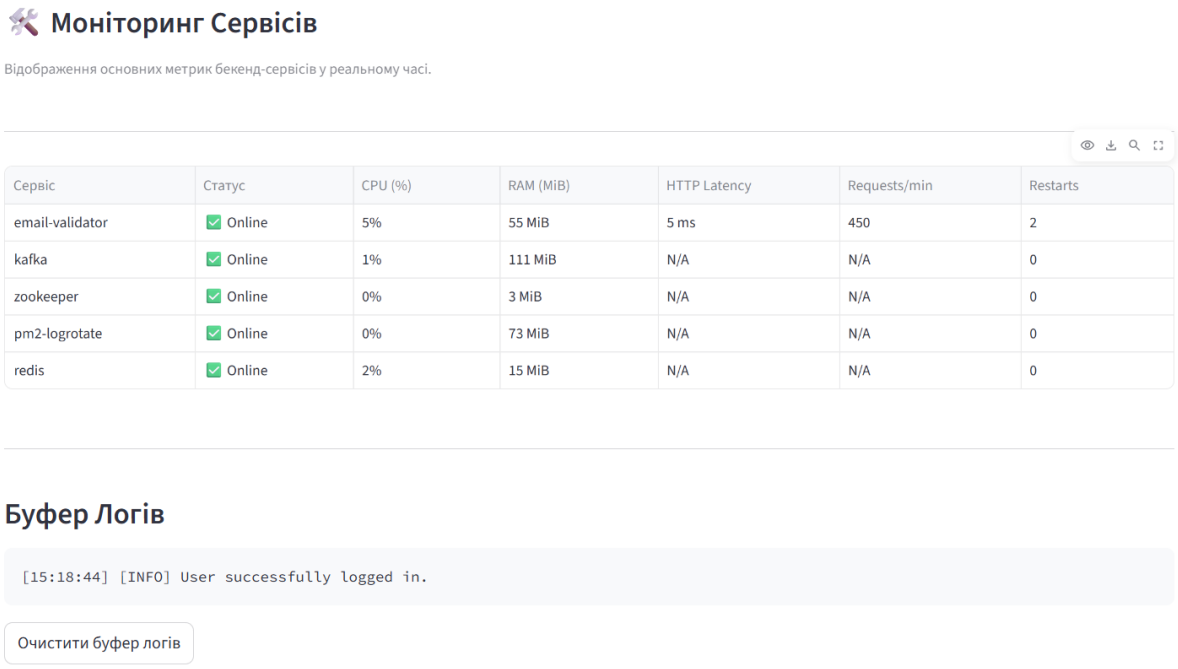

Рисунок 4.8 – Результат зібраних метрик та логів

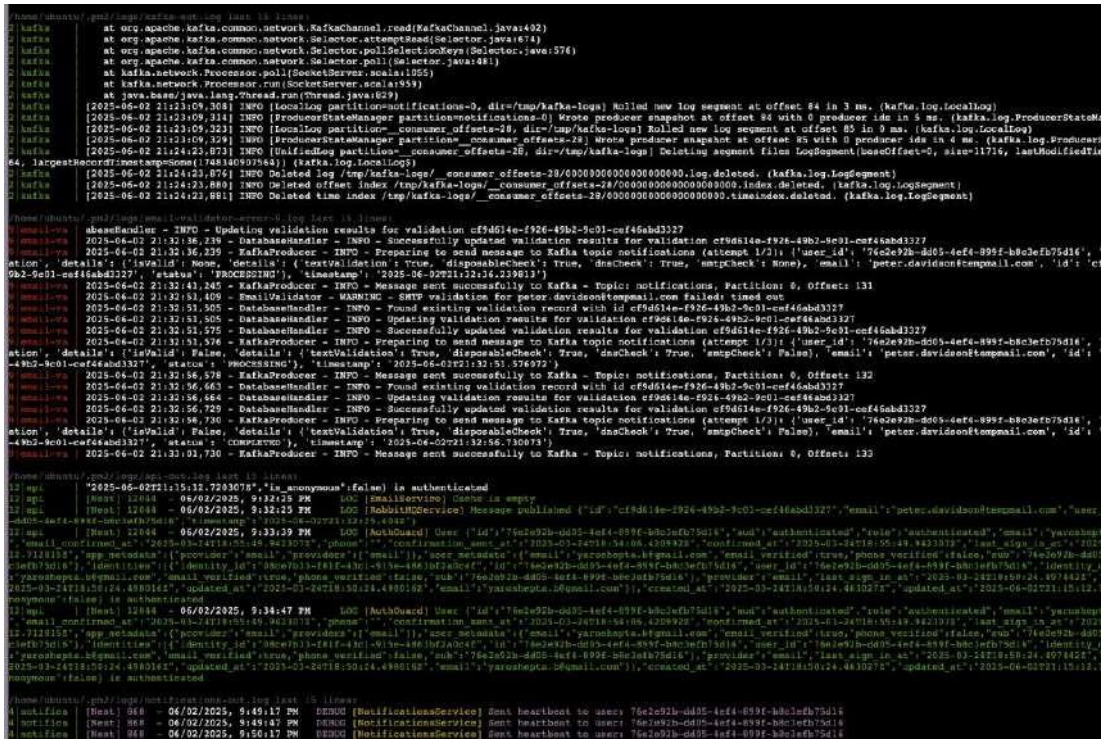

Рисунок 4.9 – Приклад логів мікросервісів

4.5 Тестування продуктивності платформи

Для перевірки стабільності, швидкодії та масштабованості платформи було проведено комплексне навантажувальне тестування, що стало критично важливим етапом оцінки системи. Основною метою тестування було визначення граничних можливостей обробки великої кількості запитів у реальному часі, виявлення вузьких місць в архітектурі та оцінка часу реакції системи за різних умов. Мети тестування охоплювали визначення максимальної кількості одночасно оброблюваних електронних адрес без зниження продуктивності, оцінку середнього часу повного циклу перевірки однієї адреси електронної пошти (від надсилання до отримання результату), перевірку стійкості сервісів до пікових навантажень, а також моделювання сценаріїв зростання черг повідомлень у RabbitMQ та обсягу подій у Kafka (рис.4.10).

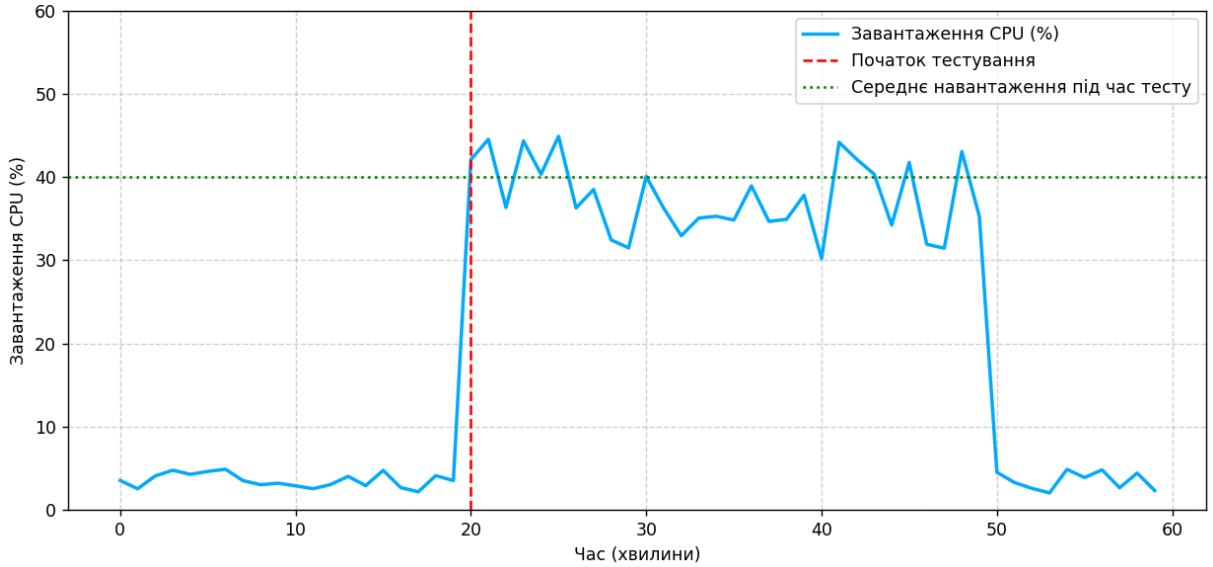

Рисунок 4.10 – Різке підняття використання ресурсів з початком тестування

У процесі тестування було використано потужний набір інструментів: k6 застосовувався для імітації навантаження на REST API, а Apache JMeter – для моделювання масових запитів до валідатора. Для збору метрик у реальному часі та візуалізації використовувалась зв'язка Prometheus і Grafana, а для спостереження за логами та станом процесів під час тестування – Loki та PM2. Було обрано кілька варіантів імітації навантаження задля всебічної оцінки: паралельне надсилання 10,000 електронних адрес з рівномірним розподілом протягом 10 хвилин; пікове навантаження у 1000 електронних адрес/сек протягом короткого періоду (burst-тест); довготривале навантаження зі стабільним потоком у 100 запитів/сек протягом однієї години; тест стабільності SMTP-симулятора; а також SSE-тест, що перевіряв поведінку системи при 1000 одночасно відкритих з’єднаннях на події Kafka (рис.4.11).

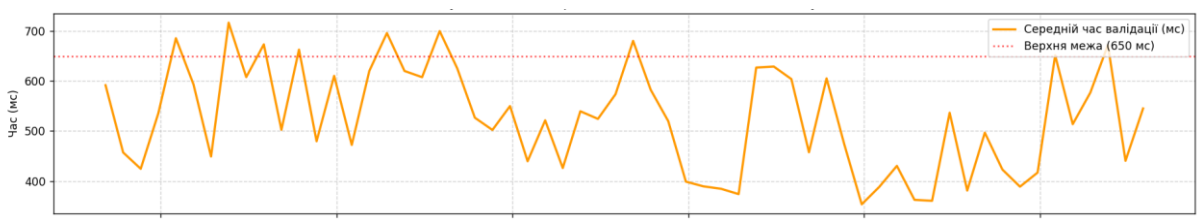

Рисунок 4.11 – Середній час повної валідації

За результатами проведеного тестування отримані високі ключові показники. Середній час повної валідації однієї адреси електронної пошти, включаючи SMTP-перевірку, склав 420–650 мс, залежно від швидкодії зовнішніх поштових серверів (рис.4.12). Максимальна стабільна пропускна здатність була визначена на рівні понад 7,000 електронних адрес/хв без помітної деградації продуктивності. Функціональність Server-Sent Events (SSE) впевнено підтримує понад 1000 одночасних з'єднань на звичайному сервері, а Kafka успішно обробляє потік у 20,000 повідомлень/хв без втрат подій. Крім того, RabbitMQ продемонстрував лінійне масштабування черг і споживачів без переповнення при зростанні навантаження. Протягом процесу тестування використання ресурсів було підвищеним, проте залишалося в межах допустимого і не супроводжувалося критичними та неконтрольованими всплесками, що свідчить про високу стабільність архітектури.

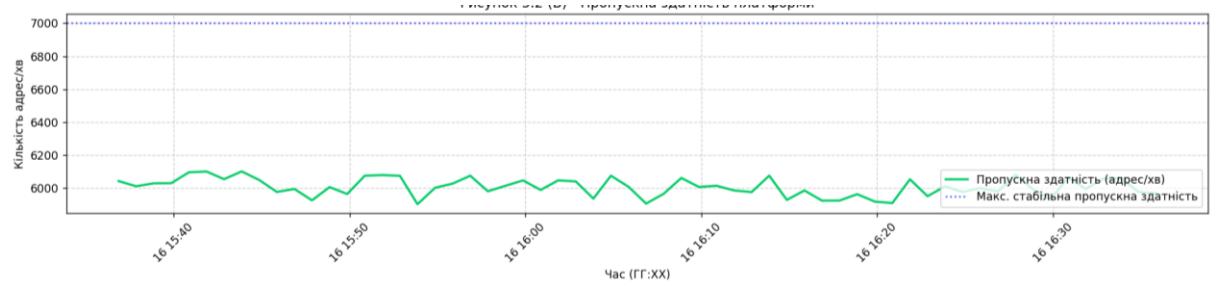

Рисунок 4.12 – Пропускна здатність платформи

4.6 Висновки до розділу 4

У даному розділі було детально розкрито архітектуру платформи, функціональні можливості бекенду, а також реалізацію ключових систем забезпечення стабільності, моніторингу та якості.

Розроблена мікросервісна архітектура на базі Node.js та RabbitMQ/Kafka забезпечує високу масштабованість, відмовостійкість та гнучкість у розгортанні. Використання брокерів повідомлень дозволило реалізувати асинхронну валідацію, мінімізуючи затримки для кінцевого користувача, та забезпечити ефективну роботу з пакетами електронних адрес. Клієнтський інтерфейс на Streamlit успішно виконує роль легкого та

інтерактивного фасаду, демонструючи основні можливості API, ліміти користувачів та історію валідації.

Критично важливим елементом є комплексна система моніторингу, логування та сповіщень. Вона поєднує PM2 (для контролю процесів та ресурсів), Prometheus (для збору метрик навантаження, швидкодії та помилок), Grafana (для візуалізації та автоматичного сповіщення) та Loki (для централізованого збору та пошуку логів). Це забезпечує повну прозорість роботи системи в реальному часі та дозволяє оперативно виявляти й усувати проблеми, як-от висока затримка або перевищення критичних лімітів використання ресурсів.

Успішно проведене навантажувальне тестування підтвердило стабільність платформи та її здатність витримувати значні навантаження. Тести показали, що система здатна обробляти понад 7,000 електронних адрес на хвилину без деградації продуктивності, при цьому середній час валідації залишається в прийнятних межах (420–650 мс). Архітектура черг повідомлень та механізми обробки подій (зокрема, SSE) підтвердили свою стійкість та здатність лінійно масштабуватися при пікових навантаженнях, демонструючи високу готовність платформи до комерційної експлуатації та зростання кількості користувачів.

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 бізнесклієнтів.

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/emailvalidation/ – 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/crashcourse – 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.

ДОДАТОК А. ФРАГМЕНТ ТЕКСТУ ПРОГРАМИ

```
import streamlit as st
import pandas as pd
from datetime import datetime
import re
import time
import logging
import sys
import os

# --- 0. КОНСТАНТИ ТА ІНІЦІАЛІЗАЦІЯ ---

# Налаштування логування
LOG_MAX_LINES = 15
# Файл зображення тепер не потрібен, але залишаємо константу
IMAGE_MONITORING_FILE = "image_54effb.jpg"

# Налаштовуємо логування у стандартний вивід (stdout)
logging.basicConfig(
    stream=sys.stdout,
    level=logging.INFO,
    format='{"timestamp": "%(asctime)s", "level": "%(levelname)s", "service":
"streamlit_client", "message": "%(message)s"}'
)
logger = logging.getLogger(__name__)

# Регулярний вираз для базової клієнтської перевірки Email
EMAIL_REGEX = r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'

# ІМІТОВАНІ ДАНІ МОНІТОРИНГУ ДЛЯ ТАБЛИЦІ
```

```
MONITORING_DATA = {
    "Сервіс": ["email-validator", "kafka", "zookeeper", "pm2-logrotate", "redis"],
    "Статус": ["✅ Online", "✅ Online", "✅ Online", "✅ Online", "✅
Online"],
    "CPU (%)": ["5%", "1%", "0%", "0%", "2%"],
    "RAM (MiB)": ["55 MiB", "111 MiB", "3 MiB", "73 MiB", "15 MiB"],
    "HTTP Latency": ["5 ms", "N/A", "N/A", "N/A", "N/A"],
    "Requests/min": ["450", "N/A", "N/A", "N/A", "N/A"],
    "Restarts": ["2", "0", "0", "0", "0"]
}

# --- СЕСІЙНИЙ СТАН ---

if 'logged_in' not in st.session_state:
    st.session_state.logged_in = False
if 'log_buffer' not in st.session_state:
    st.session_state.log_buffer = []
if 'history_data' not in st.session_state:
    st.session_state.history_data = [
        {"Email": "fvpkf7_s/2@temp-mail.com", "Valid": "❌", "Date":
"03/06/2025, 00:24:00", "Text Validation": "✅",
         "Disposable Check": "❌", "DNS Check": "❌", "SMTP Check": "❌"},
        {"Email": "joshua2@gmail.com", "Valid": "✅", "Date": "03/06/2025,
00:23:55", "Text Validation": "✅",
         "Disposable Check": "✅", "DNS Check": "✅", "SMTP Check": "✅"},
        {"Email": "ucjgm36@163.com", "Valid": "❌", "Date": "03/06/2025,
00:23:40", "Text Validation": "✅",
         "Disposable Check": "✅", "DNS Check": "❌", "SMTP Check": "❌"},
    ]
```

```
if 'account_info' not in st.session_state:
    st.session_state.account_info = {
        "email": "test@example.com",
        "plan": "Pro (Monthly)",
        "limit_total": 5000,
        "limit_used": 2013,
    }
if 'last_validation_result' not in st.session_state:
    st.session_state.last_validation_result = None


# --- ФУНКЦІЇ ЛОГУВАННЯ ТА МОНІТОРИНГУ ---

def append_to_log_buffer(message: str, level: str = "INFO"):
    """Додає повідомлення до буфера логів для відображення."""
    timestamp = datetime.now().strftime("%H:%M:%S")
    log_entry = f"[{timestamp}] [{level}] {message}"
    st.session_state.log_buffer.insert(0, log_entry)
    st.session_state.log_buffer = st.session_state.log_buffer[:LOG_MAX_LINES]

    if level == "INFO":
        logger.info(message)
    elif level == "WARNING":
        logger.warning(message)
    elif level == "ERROR":
        logger.error(message)


def update_prometheus_metrics(metric_name: str, value: int = 1, type: str =
'increment'):
    """Імітація оновлення метрик Prometheus."""
```

```
    append_to_log_buffer(f"[MONITORING] Metric {metric_name} updated
(Type: {type}, Value: {value}).", "INFO")


def log_validation_event(email, status):
    """Логування події валідації для Loki."""
    append_to_log_buffer(f"[LOKI] Validation event: Email={email},
Status={status}.", "INFO")


# --- ФУНКЦІОНАЛ АВТЕНТИФІКАЦІЇ ---

def authenticate(username, password):
    if username == "user@test.com" and password == "12345":
        st.session_state.logged_in = True
        append_to_log_buffer("User successfully logged in.", "INFO")
        return True
    else:
        append_to_log_buffer("Authentication failed: Invalid credentials.",
"WARNING")
        return False


def logout():
    st.session_state.logged_in = False
    st.session_state.log_buffer.clear()
    st.session_state.last_validation_result = None
    append_to_log_buffer("User logged out. Session cleared.", "INFO")
    st.toast("Вихід із системи...", icon='👋')
```

```
def login_page():
    st.title("🔐 Вхід до Email Validator SaaS")

    col_left, col_right = st.columns([1, 2])

    with col_left:
        with st.form("login_form"):
            st.subheader("Введіть облікові дані")
            st.caption("Demo: **user@test.com** / **12345**")
            username = st.text_input("Email", value="user@test.com")
            password = st.text_input("Пароль", type="password", value="12345")
            submitted = st.form_submit_button("Увійти")

            if submitted:
                if authenticate(username, password):
                    st.success("Успішний вхід!")
                    st.rerun()
                else:
                    st.error("Невірний email або пароль.")

    with col_right:
        st.subheader("Системні Логи (Автентифікація)")
        log_text = "\n".join(st.session_state.log_buffer)
        st.code(log_text, language='text')
        st.button("Очистити логи", on_click=lambda:
st.session_state.log_buffer.clear(), key="login_clear_logs")


# --- ФУНКЦІОНАЛ ДИНАМІЧНОГО ВІДОБРАЖЕННЯ РЕЗУЛЬТАТУ ---

def display_last_result():
```

```
    """Відображає результат останньої валідації у динамічному блоці."""
    result = st.session_state.last_validation_result

    if result:
        st.markdown("---")
        st.subheader("Останній Результат Валідації")

        status_icon = result['Valid']
        status_text = "Валідний" if status_icon == "✅" else "Невалідний"

        # Використання контейнера для імітації UI-блоку
        container = st.container(border=True)
        container.markdown(f"#### Email: `{result['Email']}`")

        col_res, col_date = container.columns([1, 1])
        with col_res:
            if status_icon == "✅":
                st.success(f"Статус: {status_text} {status_icon}")
            else:
                st.error(f"Статус: {status_text} {status_icon}")

        with col_date:
            st.markdown(f"**Дата перевірки:** {result['Date']}")

        container.markdown("##### Деталі Перевірки:")

        col_checks = container.columns(4)
        col_checks[0].metric("Формат (Text)", result['Text Validation'])
        col_checks[1].metric("Одноразовість (Disposable)", result['Disposable
Check'])
        col_checks[2].metric("DNS Перевірка", result['DNS Check'])
```

```
        col_checks[3].metric("SMTP (Сервер)", result['SMTP Check'])

        st.markdown("---")


# --- ФУНКЦІОНАЛ ВАЛІДАЦІЇ ---

def validate_single_email(email):
    """Імітація клієнтської та бекенд-валідації одного email."""

    if not re.fullmatch(EMAIL_REGEX, email):
        st.toast("❌ Формат Email невірний!", icon='📭')
        log_validation_event(email, "FORMAT_ERROR")
        update_prometheus_metrics("validation_errors_total")
        st.session_state.last_validation_result = None
        return None

    # Починаємо імітацію: Відображаємо "Надсилання запиту"
    status_placeholder = st.empty()
    status_placeholder.info(f"Надсилання запиту на валідацію для: **{email}**...")
    update_prometheus_metrics("validation_requests_in_progress")

    time.sleep(1)

    # Імітована бізнес-логіка валідації
    is_valid = True
    disposable = False
    dns_check = True

    if "temp-mail.com" in email or "mailinator.com" in email:
```

```
        is_valid = False
        disposable = True
        status_log = "DISPOSABLE"
    elif "nonexistent-domain.co" in email:
        is_valid = False
        dns_check = False
        status_log = "DNS_FAILURE"
    else:
        status_log = "SUCCESS"

    valid_status = "✅" if is_valid else "❌"

    new_result = {
        "Email": email,
        "Valid": valid_status,
        "Date": datetime.now().strftime("%d/%m/%Y, %H:%M:%S"),
        "Text Validation": "✅",
        "Disposable Check": "❌" if disposable else "✅",
        "DNS Check": "✅" if dns_check else "❌",
        "SMTP Check": "✅" if is_valid else "❌"
    }

    st.session_state.history_data.insert(0, new_result)
    st.session_state.account_info['limit_used'] += 1

    # Оновлення останнього результату для відображення
    st.session_state.last_validation_result = new_result

    # Завершуємо імітацію: Прибираємо "Надсилання запиту" та показуємо
Toast
```

```
    status_placeholder.empty()
    st.toast(f"Валідація завершена. Статус: {valid_status}", icon='🎉')

    # Логування та моніторинг
    log_validation_event(email, status_log)
    update_prometheus_metrics("validation_requests_in_progress", -1, 'decrement')
    update_prometheus_metrics("validated_emails_total")

    # Перезапуск, щоб відобразити динамічний результат
    st.rerun()
    return new_result


def handle_csv_upload(df):
    """Імітація обробки пакетної валідації."""
    append_to_log_buffer(f"BATCH_START: CSV upload started. File size:
{len(df)} emails.", "INFO")
    update_prometheus_metrics("batch_jobs_total")

    if 'email' not in df.columns:
        st.error("❌ Помилка: CSV-файл має містити колонку з назвою 'email'.")
        log_validation_event("N/A", "CSV_STRUCTURAL_ERROR")
        return

    num_emails = len(df)
    limit_remaining = st.session_state.account_info['limit_total'] -
st.session_state.account_info['limit_used']

    if num_emails > limit_remaining:
        st.error(f"❌ Ліміт перевищено. Ви можете перевірити лише
```

```
{limit_remaining} адрес.")
        append_to_log_buffer("BATCH_LIMIT_EXCEEDED: User attempted validation beyond limit.", "WARNING")
        return

    st.warning(f"⏳ Запит на пакетну валідацію ({num_emails} адрес) надіслано. (Імітація SSE).")
    append_to_log_buffer(f"BATCH_SUBMITTED: Job submitted for {num_emails} emails.", "INFO")

    # Імітація асинхронної обробки
    processed_count = 0
    for index, row in df.iterrows():
        is_valid = (index % 3 != 0)
        status_log = "BATCH_VALID" if is_valid else "BATCH_INVALID"

        # Імітація додавання результатів до історії
        batch_result = {
            "Email": row['email'],
            "Valid": "✅" if is_valid else "❌",
            "Date": datetime.now().strftime("%d/%m/%Y, %H:%M:%S"),
            "Text Validation": "✅",
            "Disposable Check": "✅" if is_valid else "❌",
            "DNS Check": "✅" if is_valid else "❌",
            "SMTP Check": "✅" if is_valid else "❌"
        }
        st.session_state.history_data.insert(0, batch_result)
        st.session_state.account_info['limit_used'] += 1
        processed_count += 1
        log_validation_event(row['email'], status_log)
```

```
        if processed_count >= 5:
            break

    st.info(f"💡 Імітаційні результати перших {processed_count} адрес пакетної перевірки додано до таблиці.")
    append_to_log_buffer("BATCH_FINISHED: First few results posted.", "INFO")


# --- ОСНОВНИЙ ФУНКЦІОНАЛ UI ---

def main_app():
    """Основний інтерфейс програми."""

    # 1. Бічна Панель (Управління Обліковим Записом та Лімітами)
    account = st.session_state.account_info
    limit_remaining = account['limit_total'] - account['limit_used']

    st.sidebar.title("Email Validator SaaS")
    st.sidebar.markdown("---")

    st.sidebar.subheader("👤 Обліковий Запис")
    st.sidebar.metric(label="Тарифний план", value=account['plan'])
    st.sidebar.metric(label="Залишок ліміту", value=f"{limit_remaining} / {account['limit_total']}")
    st.sidebar.progress(account['limit_used'] / account['limit_total'],
                        text=f"Використано {account['limit_used']} перевірок")

    st.sidebar.markdown("---")
```

```
    st.sidebar.button("🎟️ Історія транзакцій")
    st.sidebar.button("→ Вийти", on_click=logout)

    # 2. Основний Контент: ВАЛІДАЦІЯ

    st.header("✉ Email Validation")
    st.caption("Enter an email address or upload a CSV to validate against our backend rules.")
    tab_single, tab_batch, tab_logs = st.tabs(
        ["Перевірка одного Email", "Пакетна перевірка (CSV)", "🛠 Візуалізація Логів/Метрик"])

    with tab_single:
        # Статичний скріншот інтерфейсу вводу (якщо потрібен для контексту)
        if os.path.exists("image_5509c2.png"):
            st.image("image_5509c2.png")

        with st.form("email_validation_form"):
            email_to_validate = st.text_input("Email", placeholder="Enter email to validate", key="single_email_input")
            submitted = st.form_submit_button("Validate Email")
            if submitted and email_to_validate:
                validate_single_email(email_to_validate.strip())

        # ДИНАМІЧНИЙ РЕЗУЛЬТАТ: Викликається після валідації
        display_last_result()

    with tab_batch:
        # Статичний скріншот пакетної перевірки (якщо потрібен для контексту)
        if os.path.exists("image_557a27.png"):
```

```
        st.image("image_557a27.png")

    st.info("💡 Підтримується імпорт CSV з однією колонкою під назвою 'email'.")
    uploaded_file = st.file_uploader("Завантажити CSV-файл", type=["csv"], key="csv_uploader")
    if uploaded_file is not None:
        try:
            df_uploaded = pd.read_csv(uploaded_file)
            st.dataframe(df_uploaded.head(), use_container_width=True, hide_index=True)
            if st.button(f"Запустити валідацію {len(df_uploaded)} адрес"):
                handle_csv_upload(df_uploaded)
        except Exception as e:
            st.error(f"❌ Помилка читання CSV: {e}")
            append_to_log_buffer(f"FILE_READ_ERROR: {e}", "ERROR")

with tab_logs:
    # 3. Візуалізація Логування та Моніторингу
    st.subheader("🛠️ Моніторинг Сервісів")
    st.caption("Відображення основних метрик бекенд-сервісів у реальному часі.")

    st.markdown("---")

    # Створення таблиці моніторингу замість зображення
    df_monitoring = pd.DataFrame(MONITORING_DATA)
    st.dataframe(
        df_monitoring,
        use_container_width=True,
```

```
            hide_index=True,
            column_config={
                "Статус": st.column_config.Column(
                    "Статус",
                    width="small"
                ),
            }
        )

        st.markdown("---")

        st.subheader("Буфер Логів")
        log_text = "\n".join(st.session_state.log_buffer)
        st.code(log_text, language='text')
        st.button("Очистити буфер логів", on_click=lambda:
st.session_state.log_buffer.clear(), key="main_clear_logs")

    st.markdown("---")

    # 4. Таблиця Результатів (Історія)
    st.subheader("📊 Validation History")
    df_history = pd.DataFrame(st.session_state.history_data)

    col1, col2 = st.columns([8, 1])
    with col2:
        @st.cache_data
        def convert_df_to_csv(df):
            return df.to_csv(index=False).encode('utf-8')

        csv_export = convert_df_to_csv(df_history)
        st.download_button(
```

```
            label="⬇ Експорт CSV",
            data=csv_export,
            file_name='validation_results_exported.csv',
            mime='text/csv',
            on_click=lambda: append_to_log_buffer("DATA_EXPORT: CSV data exported by user.", "INFO")
        )

    st.dataframe(
        df_history,
        use_container_width=True,
        hide_index=True,
        column_order=("Email", "Valid", "Date", "Text Validation", "Disposable Check", "DNS Check", "SMTP Check")
    )


# --- ЗАПУСК ДОДАТКУ ---

# Конфігурація сторінки Streamlit
st.set_page_config(
    page_title="Email Validation SaaS Client",
    layout="wide",
    initial_sidebar_state="expanded"
)

# Стилізація
st.markdown("""
    <style>
        .st-emotion-cache-12fm525, .st-emotion-cache-1d3vj6m { /* Sidebar styling */
```

```
            background-color: #0d0d0d;
        }
        .st-emotion-cache-1cypd8z { /* Main content styling */
            background-color: #121212;
            color: #ffffff;
        }
    </style>
""", unsafe_allow_html=True)

# Логіка відображення: Вхід або Основний інтерфейс
if not st.session_state.logged_in:
    login_page()
else:
    main_app()
```