Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Автоматики і комп'ютеризованих технологій
(повна назва)

Кафедра Комп'ютерно-інтегрованих технологій, автоматизації та робототехніки
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

другий (магістерський)
(рівень вищої освіти)

РОЗРОБЛЕННЯ КОМП'ЮТЕРИЗОВАНОЇ СИСТЕМИ ДЛЯ АВТОМАТИЗАЦІЇ ПРОЦЕСУ РОЗСИЛКИ EMAIL-ПОВІДОМЛЕНЬ ЗА РЕЗУЛЬТАТАМИ АНАЛІЗУ ПОДІЙ
(тема)

Виконав здобувач ІІ року навчання,
групи КТРСм-23-1
Лебідь Георгій Тарасович
(прізвище, ім'я, по батькові)

Спеціальності 174 Автоматизація, комп'ютерно-інтегровані технології та робототехніка
(код і повна назва напряму)

Тип програми Освітньо-професійна
Освітня програма Комп'ютеризовані та робототехнічні системи
(назва)

Керівник доц. Іванов Л.С.
(посада, прізвище, ініціали)

Допускається до захисту
Зав. кафедри КІТАР _____ Невлюдов І. Ш.
(підпис) (прізвище, ініціали)

2025

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

Факультет АКТ
Кафедра КІТАР
Рівень вищої освіти другий (магістерський)
Спеціальність 174 Автоматизація, комп'ютерно-інтегровані технології та робототехніка
Тип програми Освітньо-професійна
Освітня програма Комп'ютеризовані та робототехнічні системи
(шифр і назва)

ЗАТВЕРДЖУЮ:
Зав.кафедри _____
(підпис)
«__»_____2025р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Лебедю Георгію Тарасовичу
(шифр і назва)

1. Тема роботи: Розроблення комп'ютеризованої системи для автоматизації процесу розсилки email-повідомлень за результатами аналізу подій
Затверджена наказом університету від 25.11.2024 №1239Ст
2. Термін подання здобувачем роботи до екзаменаційної комісії 16.01.2025р.
3. Вихідні дані до роботи: 3.1 Автоматизація розсилки; 3.2 Аналіз подій

4. Перелік питань, що потрібно опрацювати в роботі: 4.1 Вступ; 4.2 Аналіз існуючих рішень для автоматизації розсилки повідомлень; 4.3 Опис архітектури мікросервісу; 4.4 Вимоги до системи для підтримки email-розсилки та інтеграції з месенджерами; 4.5 Визначення функціональних вимог до програмного забезпечення; 4.6 Вибір мови програмування та середовища розробки БД; 4.7 Визначення структур даних для зберігання подій та налаштувань користувачів; 4.8 Розробка ER-діаграми бази даних та опис зв'язків між таблицями; 4.9 Проектування фізичної моделі БД; 4.10 Опис бізнес-логіки роботи ПЗ; 4.11 Розробка алгоритмів автоматизації розсилки email-повідомлень та повідомлень у месенджери; 4.12 Розробка функцій обробки подій і тригерів для розсилки повідомлень, логування та протоколювання роботи сервісу; 4.13 Розробка серверної частини мікросервісу; 4.15 Розробка інтерфейсу користувача; 4.16 Оцінка продуктивності та надійності програмного забезпечення; 4.17 Висновки

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій Графічний демонстраційний матеріал в форматі PowerPoint(*.ppt) формату А4 –15 сторінок.

6. Консультанти розділів роботи

Найменування розділу	Консультант (посада, прізвище, ім'я, по-батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз сучасних технологій автоматизації розсилки повідомлень	22.09.2024-08.10.2024	виконано
2	Визначення функціональних вимог до програмного забезпечення	09.10.2024-21.10.2024	виконано
3	Вибір мови програмування та середовища розробки БД	22.10.2024-01.11.2024	виконано
4	Визначення структур даних для зберігання подій та налаштувань користувачів	02.11.2024-5.11.2024	виконано
5	Розробка ER-діаграми бази даних та опис зв'язків між таблицями	6.11.2024-10.11.2024	виконано
6	Проектування фізичної моделі БД	11.11.2024-16.11.2024	виконано
7	Розробка бізнес-логіки програмного забезпечення	17.11.2024-01.12.2024	виконано
8	Реалізація програмного забезпечення	02.12.2024-14.12.2024	виконано
9	Оцінка продуктивності та надійності програмного забезпечення	15.12.2024-27.12.2024	виконано
10	Захист кваліфікаційної роботи	16.01.2025	

Дата видачі завдання 2 вересня 2024р.

Здобувач _____ Лебідь Г. Т.
(підпис) (прізвище, ініціали)

Керівник роботи _____ Іванов Л.С.
(підпис) (прізвище, ініціали)

Я, як студент ХНУРЕ, розумію та підтримую політику закладу з академічної доброчесності. Я не надавав та не одержував недозволену допомогу під час підготовки кваліфікаційної роботи. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

06.01.2025  Лебідь Г.Т.

РЕФЕРАТ

Пояснювальна записка: 85 с., 14 табл., 23 рис., 3 дод., 21 джерело.

АВТОМАТИЗАЦІЯ РОЗСИЛКИ, МІКРОСЕРВІСНА АРХІТЕКТУРА, EMAIL-ПОВІДОМЛЕННЯ, ІНТЕГРАЦІЯ З МЕСЕНДЖЕРАМИ, БАЗА ДАНИХ MONGODB, АНАЛІЗ ПОДІЙ, ОБРОБКА ТРИГЕРІВ.

Мета дослідження – удосконалення процесу автоматизації розсилки за рахунок впровадження іноваційних технічних рішень.

Об’єкт дослідження – процес автоматизації розсилки повідомлень.

Предмет дослідження – мікросервісні архітектури та технічні рішення для реалізації автоматизованих розсилок.

В кваліфікаційній роботі було проведено аналіз існуючих рішень для автоматизації розсилки повідомлень, описано архітектуру мікросервісу та вимоги до системи для підтримки email-розсилки та інтеграції з месенджерами. Визначено функціональні вимоги до програмного забезпечення та структуру даних для зберігання подій та налаштувань користувачів. Розроблено ER-діаграми бази даних та опис зв’язків між таблицями, спроектовано фізичну модель БД та описано бізнес-логіку роботи ПЗ. Розроблено алгоритмів автоматизації розсилки email-повідомлень та повідомлень у месенджери та функції обробки подій і тригерів для розсилки повідомлень. Розроблено функції логування та протоколювання роботи сервісу, серверної частини мікросервісу, базу даних на MongoDB з використанням мови SQL та інтерфейс користувача. Проведено оцінку продуктивності та надійності програмного забезпечення.

Результати кваліфікаційної роботи можна віднести до Цілі сталого розвитку 9 «Промисловість, інновації та інфраструктура», а саме 9.4.

ABSTRACT

Explanatory note: 85 pages, 14 tables, 23 figures, 3 app, 21 sources.

AUTOMATION OF MAILING, MICROSERV ARCHITECTURE, EMAIL MESSAGES, INTEGRATION WITH MESSENGERS, MONGODB DATABASE, EVENT ANALYSIS, TRIGGER PROCESSING.

The purpose of the research is to improve the process of mailing automation through the implementation of innovative technical solutions.

The object of the study is the process of automating the sending of messages.

The subject of the study is microservice architectures and technical solutions for implementing automated mailings.

The qualification work analyzed existing solutions for automating the sending of messages, described the microservice architecture and system requirements for supporting email mailings and integration with messengers. Functional requirements for the software and the data structure for storing events and user settings were determined. ER-diagrams of the database and a description of the relationships between tables were developed, a physical model of the database was designed and the business logic of the software was described. Algorithms for automating the sending of email messages and messages to messengers and event processing functions and triggers for sending messages were developed. Logging and reporting functions of the service, the server part of the microservice, a database on MongoDB using the SQL language and a user interface were developed. The performance and reliability of the software were assessed. The results of the qualification work can be attributed to Sustainable Development Goal 9 "Industry, Innovation and Infrastructure", namely 9.4.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

AWS – amazon web services;

E2EE – end-to-end encryption;

JWT – JSON web token;

REST – representational state transfer;

SaaS – software as a service;

SES – simple email service.

ВСТУП

У сучасних умовах, у бізнесі, освіті та інших сферах діяльності важливо забезпечити своєчасне інформування користувачів про події, зміни та інші важливі аспекти. Традиційні методи комунікації вимагають значних ресурсів і часу, що знижує їхню ефективність. Використання автоматизованих систем розсилки повідомлень дозволяє оптимізувати цей процес, підвищуючи швидкість і точність взаємодії.

Розвиток мікросервісних архітектур, інструментів аналізу подій та інтеграції з популярними месенджерами відкриває нові можливості для створення ефективних рішень. У цьому контексті дослідження та розробка програмного забезпечення для автоматизації розсилок є важливим завданням, яке дозволяє вдосконалити процеси комунікації, скоротити витрати та підвищити задоволеність користувачів.

Мета дослідження – удосконалення процесу автоматизації розсилки за рахунок впровадження іноваційних технічних рішень.

Об’єкт дослідження – процес автоматизації розсилки повідомлень.

Предмет дослідження – мікросервісні архітектури та технічні рішення для реалізації автоматизованих розсилок.

Для досягнення поставленої мети необхідно вирішити такі завдання:

– провести аналіз існуючих рішень для автоматизації розсилки повідомлень;

– описати архітектуру мікросервісу;

– описати вимоги до системи для підтримки email-розсилки та інтеграції з месенджерами;

– визначити функціональні вимоги до програмного забезпечення;

– провести вибір мови програмування та середовища розробки БД;

– визначити структуру даних для зберігання подій та налаштувань

користувачів;

– розробити ER-діаграму бази даних та опис зв’язків між таблицями;

– спроектувати фізичну модель БД;

– описати бізнес-логіку роботи ПЗ;

– розробити алгоритми автоматизації розсилки email-повідомлень та повідомлень у месенджери;

– розробити функції обробки подій і тригерів для розсилки повідомлень;

– провести налаштування конфігурацій користувачів для автоматизованих розсилок;

– розробити функції логування та протоколювання роботи сервісу;

– розробити серверну частину мікросервісу;

– реалізувати базу даних на MongoDB з використанням мови SQL;

– розробити інтерфейс користувача;

– провести оцінку продуктивності та надійності програмного забезпечення.

Кваліфікаційна робота виконана згідно ДСТУ 3008 – 15 [1], керуючись навчальним посібником з дипломного проекту [2] та методичними вказівками [3], результати кваліфікаційної роботи отримали апробацію в науковій статті [4].

1 АНАЛІЗ СУЧАСНИХ ТЕХНОЛОГІЙ АВТОМАТИЗАЦІЇ РОЗСИЛКИ ПОВІДОМЛЕНЬ

1.1 Аналіз існуючих рішень для автоматизації розсилки повідомлень

Сучасний ринок пропонує широкий вибір інструментів для автоматизації розсилки повідомлень, які умовно можна розділити на три основні категорії: SaaS-платформи, open-source рішення та кастомні розробки. Кожна з цих категорій має свої особливості, переваги та недоліки, що дозволяє компаніям вибирати варіант, який найкраще відповідає їхнім потребам і бюджету [5].

SaaS-платформи (Software as a Service) – це сервіси, які надають функціональність автоматизованої розсилки в рамках підписки. Вони працюють у хмарі, що дозволяє користувачам легко масштабувати їх відповідно до поточних потреб. Приклади SaaS-платформ включають Mailchimp, SendGrid, Twilio, Amazon SES, які надають потужні функції для email-маркетингу, транзакційної розсилки та інтеграції з месенджерами. Однією з ключових переваг SaaS-платформ є те, що користувачі не потребують встановлення чи підтримки програмного забезпечення на власних серверах. Це дозволяє бізнесам швидко запустити автоматизацію розсилки без великих початкових інвестицій у інфраструктуру та персонал. Крім того, SaaS-платформи зазвичай пропонують інтеграцію з популярними інструментами CRM, що дозволяє легко збирати дані про клієнтів та налаштовувати персоналізовані повідомлення.

SaaS-рішення часто включають готові шаблони та функції для сегментації аудиторії, що дозволяє маркетологам зосередитися на створенні персоналізованих кампаній без необхідності програмування. Окрім того, такі платформи, як правило, пропонують аналітичні інструменти, які дозволяють

відстежувати показники ефективності розсилок, наприклад, кількість відкриттів, кліків та конверсій. Це спрощує аналіз ефективності маркетингових кампаній та дозволяє коригувати стратегію в реальному часі. Однак, SaaS-платформи мають і свої обмеження. Наприклад, вони зазвичай пропонують стандартний набір функцій, який не завжди можна адаптувати під конкретні бізнес-потреби. Користувачі SaaS також стикаються з обмеженнями за обсягами розсилок та необхідністю сплачувати підписку, що може бути фінансово обтяжливим для малого бізнесу. Компанії часто використовують Mailchimp для email-маркетингу завдяки простоті налаштування та швидкому старту. Наприклад, стартап з обмеженим бюджетом може запускати маркетингові кампанії без інвестицій у технічну інфраструктуру та персонал [6].

Open-source рішення – це програмне забезпечення з відкритим кодом, яке компанії можуть завантажити, встановити та налаштувати під свої потреби. Прикладами таких інструментів є Mautic, Postal, Mailtrain. Open-source рішення часто використовуються компаніями, які прагнуть знизити витрати на програмне забезпечення або забезпечити максимальний контроль над функціональністю системи. Основною перевагою open-source рішень є їхня гнучкість та можливість повного контролю над розсилками. Компанії можуть вносити зміни до коду для адаптації рішення під свої потреби, додаючи додаткові функції або інтегруючи з внутрішніми системами. Це дозволяє уникнути обмежень SaaS-платформ та розробити індивідуальні рішення, які максимально відповідають специфіці бізнесу.

Незважаючи на те, що open-source рішення не потребують щомісячної підписки, їх впровадження може бути дорогим через необхідність технічної підтримки та інфраструктури. Наприклад, компанії, які використовують open-source, мають забезпечити власні сервери для обробки розсилок та підтримувати їх у робочому стані. Крім того, open-source рішення часто вимагають технічних навичок для налаштування та обслуговування, що може

бути складним для компаній, які не мають досвідчених розробників. Mautic – популярне open-source рішення для автоматизації маркетингу, яке компанії можуть адаптувати під свої потреби. Наприклад, маркетингова агенція може використовувати Mautic для сегментації аудиторії та тригерних розсилок, інтегруючи його з власною CRM [7].

Кастомні розробки передбачають створення власного програмного забезпечення з нуля або на базі вже існуючого open-source рішення. Це дозволяє компаніям повністю контролювати процес автоматизації розсилок і отримати рішення, що враховує всі специфічні потреби. Кастомні рішення можуть включати особливі функції, інтеграцію з внутрішніми системами і повну відповідність технічним та безпековим вимогам компанії. Основна перевага кастомних розробок полягає у високій гнучкості та можливості створення унікальних функцій. Наприклад, компанії можуть розробити специфічні алгоритми для персоналізації повідомлень або налаштувати автоматизацію процесів відповідно до вимог галузі. Кастомні рішення також дозволяють уникнути залежності від сторонніх постачальників і краще захищати дані клієнтів. Великі корпорації, такі як Amazon чи Google, інвестують у кастомні рішення для обробки величезних обсягів повідомлень та інтеграції з іншими сервісами. Наприклад, для персоналізації та безперебійної роботи платформи Amazon використовує власну систему повідомлень [8].

Кастомні розробки є одними з найдорожчих рішень для автоматизації розсилок, оскільки вимагають значних інвестицій в час, ресурси та технічні компетенції. Створення програмного забезпечення з нуля може зайняти місяці або навіть роки, залежно від складності функціоналу. Крім того, компанії мають забезпечити підтримку та оновлення свого рішення, щоб воно залишалося конкурентоспроможним та безпечним. Після проведеного аналізу можна виділити такі переваги та недоліки трьох категорій рішень автоматизації розсилки повідомлень, представлених у таблиці 1.1.

Таблиця 1.1 – Порівняльна таблиця функціональностей рішень

Функціонал	SaaS-платформи	Open-source рішення	Кастомні розробки
Швидкість впровадження	Висока. Можливий швидкий запуск із мінімальними налаштуваннями	Середня. Потребує налаштування та інсталяції	Низька. Потребує часу для розробки з нуля
Гнучкість налаштування	Обмежена. Налаштування обмежені функціоналом платформи	Висока. Можливі значні зміни в коді	Максимальна. Можна розробити функціонал під конкретні потреби
Витрати на підтримку	Низькі. Підтримка забезпечується провайдером	Середні. Потрібні власні ресурси для підтримки	Високі. Потребує постійної підтримки та оновлень
Масштаб	Висока. Масштабування автоматичне	Висока, але залежить від налаштувань інфраструктури	Висока, але потребує додаткових ресурсів
Безпека даних	Забезпечується провайдером, відповідає стандартам	Залежить від налаштувань компанії	Максимальний контроль, можливі власні засоби захисту
Інтеграція з іншими системами	Стандартні інтеграції з CRM, API для популярних сервісів	Можлива інтеграція через відкритий код	Можлива будь-яка інтеграція з внутрішніми системами
Аналітика та звітність	Вбудовані інструменти для аналітики	Базова аналітика або інтеграція з іншими системами	Можливість створення індивідуальної аналітики
Технічні вимоги до користувача	Мінімальні. Не потребує технічних знань	Середні. Потрібні базові технічні навички	Високі. Потребує кваліфікованих розробників
Вартість	Залежить від підписки	Безкоштовна, витрати на інфраструктуру	Висока стартова вартість через розробку з нуля

Вибір між SaaS, open-source та кастомними рішеннями залежить від бюджету, технічних ресурсів і потреб бізнесу: SaaS забезпечує швидкий запуск та базову безпеку за передплатою, що підходить стартапам; open-

source дозволяє знизити початкові витрати, але вимагає технічної підтримки та власних налаштувань безпеки; кастомні рішення, хоча й найдорожчі, забезпечують максимальну гнучкість і контроль над функціоналом та захистом. Часто використовують комбіновані підходи, наприклад, поєднання SaaS з open-source або кастомними елементами для специфічних потреб.

Наступним етапом проведемо аналіз популярних інструментів для автоматизації розсилки повідомлень:

– SendGrid – одна з найпопулярніших платформ для автоматизації email-розсилок, яка надає потужний API для масової відправки транзакційних і маркетингових повідомлень. SendGrid відрізняється високою доставлюваністю, що особливо важливо для компаній з великим обсягом розсилок. Платформа пропонує розширені функції аналітики, шаблони листів і простий інтерфейс для управління розсилками. Водночас, SendGrid добре підходить для інтеграції з різними системами та програмами через API, що робить її універсальним рішенням для автоматизації комунікацій [9];

– Mailchimp – інструмент, відомий своєю орієнтованістю на маркетингові команди та бізнеси, які фокусуються на email-маркетингу. Крім відправки повідомлень, Mailchimp надає інструменти для побудови сегментацій аудиторії, налаштування кампаній та детального аналізу ефективності. Він також включає в себе функціонал автоматизації на основі поведінки користувачів, наприклад, тригерні розсилки. Це робить Mailchimp зручним вибором для компаній, що хочуть створювати персоналізовані та адаптивні комунікаційні кампанії [10];

– Amazon SES (Simple Email Service) – це рішення для автоматизації розсилок від Amazon Web Services (AWS), яке відрізняється низькими витратами та широкими можливостями налаштування. Amazon SES підходить для компаній, які вже використовують інфраструктуру AWS і шукають гнучке рішення для транзакційних та маркетингових повідомлень. Однак, на відміну від інших сервісів, SES орієнтований більше на

розробників і вимагає базових технічних знань для інтеграції та налаштування. Незважаючи на це, SES пропонує високу надійність, масштабованість і низькі витрати, що робить його популярним серед великих корпорацій [11];

– Twilio – платформа, яка спеціалізується на мультиканальній комунікації, включаючи SMS, email та голосові повідомлення. Основна перевага Twilio полягає в можливості легко створювати та налаштовувати повідомлення через різні канали, що робить його ефективним інструментом для автоматизації не тільки email, але і SMS-розсилок та інших видів комунікацій. Twilio API підтримує інтеграцію з багатьма CRM та іншими платформами, що дозволяє автоматизувати спілкування з клієнтами на різних етапах взаємодії [12];

– Slack API – популярний інструмент для автоматизації внутрішньої комунікації в компаніях. З його допомогою команди можуть відправляти автоматичні повідомлення в Slack-канали, що особливо корисно для швидкого сповіщення про події, що відбуваються в бізнес-процесах. Slack API дозволяє інтегрувати Slack з різними системами та сервісами, наприклад, для отримання сповіщень з CRM або аналітичних систем. Це робить Slack API корисним для компаній, які прагнуть підвищити ефективність внутрішньої комунікації та знизити час реакції на критичні події [13].

Графік, який відображає популярність різних рішень для автоматизації розсилок. Як можна бачити, SaaS рішенню надається найбільша популярність, за ним слідують open-source рішення та кастомні розробки, представлено на рисунку 1.1, а на рисунку 1.2 представлено структурні схеми інтеграції з додатками.

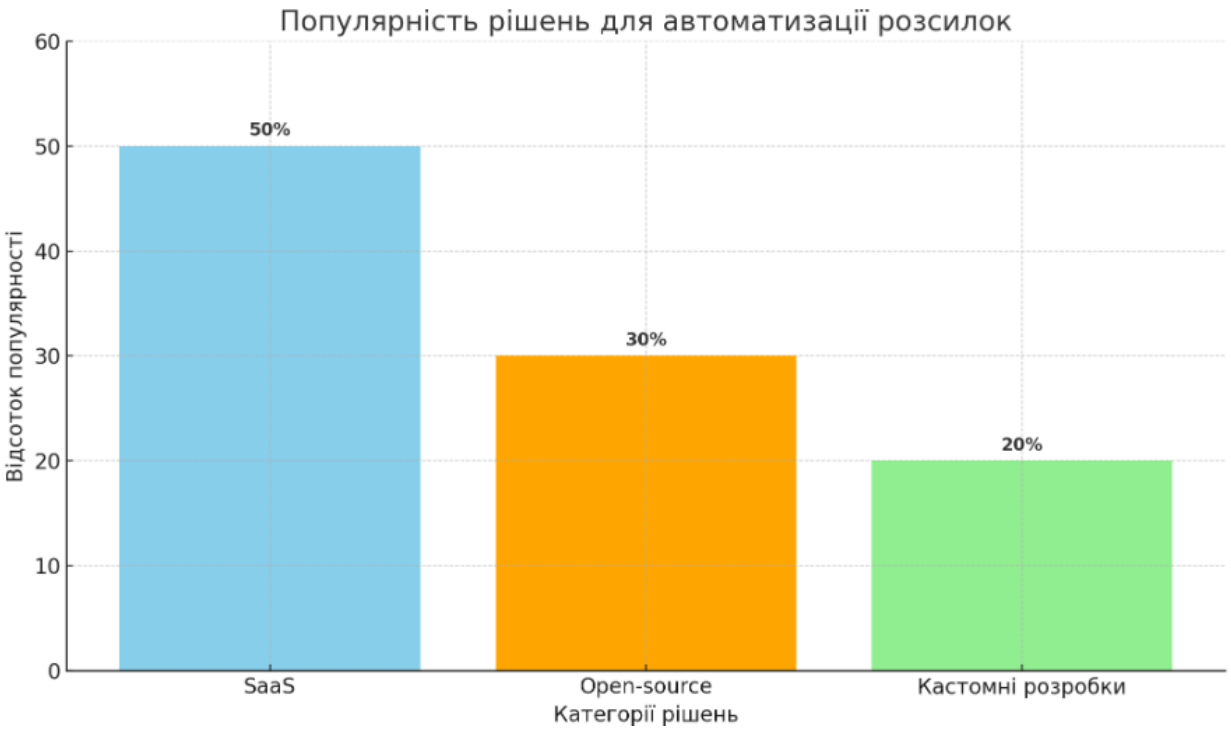


Рисунок 1.1 – Популярність різних рішень для автоматизації розсилок

а)

б)

а) структурна схема інтеграції SaaS-рішення (Mailchimp);

б) структурна схема інтеграції кастомного рішення

Рисунок 1.2 – Структурні схеми інтеграції з додатками

Структурна схема інтеграції SaaS-рішення (Mailchimp) включає користувачів, платформу Mailchimp, CRM-систему та систему аналітики, а зв'язки показують, як користувачі взаємодіють з Mailchimp, а також

інтеграцію з CRM і аналітичними системами. Структурна схема інтеграції кастомного рішення включає користувачів, кастомне рішення, базу даних та API сторонніх сервісів, а зв'язки демонструють, як користувачі взаємодіють із кастомним рішенням і як воно інтегрується з базою даних і сторонніми сервісами через API.

Далі, проведемо порівняльний аналіз підтримки месенджерів у таких платформах як WhatsApp, Telegram та Viber API. WhatsApp API є популярним інструментом для інтеграції месенджерів, орієнтованим на бізнес-комунікацію. WhatsApp API дозволяє компаніям відправляти повідомлення клієнтам, включаючи підтвердження замовлень, сповіщення про доставку або технічну підтримку. Проте API має обмеження: для використання необхідно пройти верифікацію через офіційного партнера (Business Solution Provider) і дотримуватися політики WhatsApp. Це робить WhatsApp API гарним вибором для великих компаній, які готові вкладати ресурси у відповідність правилам платформи. Telegram API забезпечує більшу гнучкість, ніж WhatsApp, завдяки відкритому доступу та менш обмеженій політиці використання. Telegram дозволяє компаніям створювати ботів для автоматизації комунікацій, відправляти повідомлення без необхідності верифікації та використовувати API для широкого спектра функцій, від маркетингових кампаній до технічної підтримки. Ця платформа підходить для стартапів та середнього бізнесу завдяки простоті інтеграції та можливості налаштувати ботів на власних серверах без обмежень з боку платформи. Viber API дозволяє компаніям взаємодіяти з клієнтами через чат-боти або публічні облікові записи, що є корисним для сегментованих рекламних кампаній та масових повідомлень. Хоча Viber має обмежену популярність у деяких країнах, у ряді регіонів, особливо у Східній Європі, Viber користується широкою підтримкою. API Viber підтримує інтеграцію з різноманітними маркетинговими платформами, надаючи інструменти для

створення кастомних ботів і інтеграцій на рівні бізнесу, що робить його зручним вибором для компаній, які хочуть охопити специфічні регіони [14].

Аналіз вартості та доступності показує, що Telegram є найменш витратним з трьох, оскільки не вимагає оплати за використання API або обмежень на повідомлення. WhatsApp, хоч і вимагає щомісячної підписки для бізнес-акаунтів, забезпечує високу конверсію та довіру користувачів. Viber займає проміжну позицію: він підтримує платні функції для корпоративних клієнтів, зокрема спонсоровані повідомлення, але забезпечує безкоштовний базовий функціонал для малих підприємств. Telegram надає найбільшу гнучкість і дозволяє відправляти будь-який тип повідомлень, тоді як WhatsApp API більш орієнтований на транзакційні повідомлення, а Viber API обмежений в налаштуваннях та функціях інтерфейсу. Для максимальної гнучкості підходить Telegram, для високої надійності та конверсії – WhatsApp, а для цільових регіонів і маркетингових повідомлень – Viber. Представимо у таблиці 1.2 порівняння можливостей інтеграції для платформ WhatsApp, Telegram та Viber.

Таблиця 1.2 – Порівняння можливостей інтеграції для платформ WhatsApp, Telegram та Viber

Функціональні можливості	WhatsApp API	Telegram API	Viber API
1	2	3	4
Відкритість API	Обмежена (потрібна верифікація)	Відкрита (без верифікації)	Відкрита, з базовими обмеженнями
Підтримка чат-ботів	Обмежена	Повна підтримка	Повна підтримка
Сегментація аудиторії	Обмежена	Відсутня	Доступна для публічних акаунтів
Можливості кастомізації	Мінімальна	Максимальна	Обмежена
Вартість	Платна (в залежності від постачальника)	Безкоштовна	Безкоштовна для базового функціоналу

Продовження таблиці 1.2

1	2	3	4
Безпека і шифрування	Висока (end-to-end)	Висока (end-to-end)	Висока (end-to-end)
Географічна популярність	Широка	Широка	Висока у Східній Європі
Інтеграція з CRM та аналітичними системами	Обмежена	Повна підтримка	Часткова підтримка
Маркетингові можливості	Обмежені	Обмежені	Розширені (спонсоровані повідомлення)

Архітектура систем із інтеграцією в месенджери часто складається з мікросервісів, кожен із яких відповідає за конкретну функцію, таку як обробка повідомлень, аналітика або робота з базою даних. У типовій системі існує центральний сервіс, який приймає події або повідомлення з вебдодатка або CRM-системи та направляє їх до відповідного месенджера через API. Наприклад, при оновленні статусу замовлення в CRM система генерує подію, що обробляється центральним сервісом, а потім надсилається через Telegram або Viber API до клієнта. Впровадження кешування для частих запитів та налаштування черг повідомлень (наприклад, RabbitMQ) дозволяє підвищити продуктивність і надійність доставки.

Для інтеграції системи з кількома месенджерами використовуються адаптерні або шлюзові сервіси, які працюють із різними API та спрощують управління повідомленнями. Наприклад, мікросервісний підхід із застосуванням Node.js або Python дозволяє створювати окремі модулі для кожного месенджера (WhatsApp, Telegram, Viber), які адаптують повідомлення до специфіки платформи. Така архітектура полегшує масштабування та оновлення окремих модулів, зберігаючи загальну стійкість системи [15].

1.2 Опис архітектури мікросервісу

Мікросервісна архітектура відрізняється від монолітної своєю структурою і підходом до розробки. У мікросервісній архітектурі додаток складається з окремих сервісів, кожен з яких відповідає за конкретну функцію, наприклад, авторизацію, обробку даних чи роботу з месенджерами. Ці сервіси взаємодіють через API, що дозволяє їм працювати незалежно один від одного та бути розробленими різними командами. Таку систему легко масштабувати горизонтально: кожен сервіс можна налаштувати так, щоб обробляти більше запитів, не впливаючи на інші частини системи. Це також полегшує процес оновлення – кожен сервіс може змінюватися чи оновлюватися незалежно від інших.

У монолітній архітектурі всі функції та модулі об’єднані в один єдиний блок, який запускається як єдине ціле. Така архітектура є простою для реалізації на початкових етапах, особливо для невеликих проєктів, оскільки немає необхідності налаштовувати окремі канали зв'язку між компонентами. Проте зі зростанням системи моноліт стає важчим в управлінні: будь-які зміни чи оновлення в одному компоненті можуть вплинути на всю систему, що ускладнює тестування та розгортання нових функцій. Масштабування такої архітектури потребує масштабування всієї системи, навіть якщо потрібно збільшити продуктивність тільки одного її модуля.

З точки зору надійності, мікросервісна архітектура має перевагу, оскільки помилки в одному сервісі не обов'язково порушують роботу інших компонентів. У монолітній системі збої або проблеми в одному модулі можуть призвести до збою всього додатка, оскільки всі компоненти взаємопов’язані. Однак мікросервісна архітектура вимагає складнішого управління з точки зору інфраструктури, адже вона потребує системи для оркестрації, моніторингу та забезпечення надійності комунікації між сервісами.

У плані швидкості розробки, монолітні системи часто розробляти швидше, оскільки вони простіші і не вимагають налаштування зв’язків між різними сервісами. Для мікросервісної архітектури потрібне більше початкове налаштування, зокрема для взаємодії сервісів, управління мережею та деплойменту, але вона забезпечує гнучкість та швидший розвиток у довгостроковій перспективі. Кожна команда розробників може працювати над своїм сервісом незалежно, що дозволяє паралельно реалізовувати нові функції [16].

Далі розглянемо окремі модулі, інтеграція яких дозволяє створювати систему автоматизованої розсилки, яка працює стабільно навіть при великих навантаженнях. Кожен компонент має свою функцію, тому система залишається масштабованою та надійною. Оркестрація сервісів, використання черг повідомлень і незалежна робота API для кожного месенджера дозволяють системі адаптуватися під потреби бізнесу та змінюватися без ризику збоїв в інших частинах.

Сервіс для обробки подій – це ключовий модуль, який приймає події з різних зовнішніх систем, таких як CRM, ERP або вебдодатки. Його завдання – зареєструвати подію, проаналізувати її та передати відповідні дані далі для обробки, наприклад, для надсилання повідомлення клієнту. Для оптимізації роботи сервіс для обробки подій зазвичай використовує черги повідомлень, такі як RabbitMQ або Kafka, що дозволяє організувати асинхронний обмін даними між компонентами. Це підвищує продуктивність, оскільки зменшує навантаження на систему, забезпечуючи обробку подій незалежно від решти сервісів.

Сервіс для формування повідомлень – це модуль, який приймає дані з сервісу обробки подій та генерує повідомлення на основі отриманої інформації. Він гнучкий у налаштуваннях, дозволяючи формувати різні типи повідомлень: текстові, мультимедійні (зображення, відео) або інтерактивні (кнопки, посилання). Важливим аспектом є персоналізація повідомлень

відповідно до даних клієнтів, що зберігаються в базі користувачів. Це дозволяє налаштувати контент залежно від уподобань і поведінки клієнта, збільшуючи ефективність взаємодії та підвищуючи рівень залучення.

API для месенджерів – кожен месенджер, такий як WhatsApp, Telegram чи Viber, має окремий модуль для інтеграції, що адаптує повідомлення відповідно до вимог платформи. Ці API надають можливість надсилати повідомлення, відстежувати статус доставки, а також обробляти відповіді клієнтів. Модулі API розроблені так, щоб забезпечити надійну передачу даних у відповідному форматі кожного месенджера, а також автоматично обробляти винятки, якщо платформа недоступна або виникає помилка при відправленні. Вони дозволяють інтеграції з різними платформами, що забезпечує охоплення ширшої аудиторії.

Приведемо схематичне представлення роботи компонентів мікросервісної архітектури для автоматизованої розсилки повідомлень, представлене на рисунку 1.3. Опишемо детально кожен елемент схеми:

– зовнішні системи (CRM, ERP) генерують події (наприклад, зміни статусу замовлення);

– сервіс обробки подій приймає події, реєструє їх та передає в чергу подій;

– черга подій забезпечує асинхронне оброблення, розсилаючи події іншим компонентам для оптимізації навантаження;

– сервіс для формування повідомлень створює відповідне повідомлення, яке потім спрямовується до API відповідного месенджера (WhatsApp, Telegram, тощо);

– API месенджера адаптує повідомлення під вимоги конкретної платформи та відправляє його клієнту;

– сервіс аналітики збирає інформацію про статуси повідомлень (доставлено, прочитано) для аналізу ефективності;

– система моніторингу та логування забезпечує контроль за системою в реальному часі, що дозволяє оперативно реагувати на можливі збої.

Рисунок 1.3 – Схематичне представлення роботи компонентів мікросервісної архітектури для автоматизованої розсилки повідомлень

Наступним етапом опишемо стандарти взаємодії між сервісами: REST, WebSockets. REST (Representational State Transfer) і WebSockets є двома популярними стандартами для взаємодії між сервісами в мікросервісній архітектурі. REST заснований на протоколі HTTP і є більш традиційним підходом, який дозволяє сервісам обмінюватися даними за допомогою простих запитів (GET, POST, PUT, DELETE). У RESTful-архітектурі кожен

сервіс надає певний набір ресурсів, які можна викликати через унікальні URL. REST добре підходить для запитів, які не потребують миттєвих відповідей або постійного з’єднання, що робить його надійним для передачі статичних даних, таких як отримання інформації про користувача чи запит на створення нового об'єкта в системі.

WebSockets, з іншого боку, підтримують двостороннє з’єднання між клієнтом і сервером, що робить його ідеальним для випадків, коли необхідно постійно оновлювати дані або забезпечити низьку затримку в реальному часі. WebSocket-підключення дозволяє серверам відправляти дані клієнту без необхідності запиту від клієнтської сторони. Це дозволяє уникати постійних запитів, що покращує продуктивність і знижує навантаження на сервер. WebSockets часто використовуються для таких застосувань, як чати, сповіщення, а також інтеграції з IoT, де потрібні часті оновлення.

Вибір між REST і WebSockets залежить від специфіки застосування. REST є більш популярним та простішим у реалізації та підтримці, тоді як WebSockets підходять для інтерактивних або динамічних додатків, де важливі миттєві оновлення. У великих системах часто комбінують обидва стандарти: REST використовується для основних CRUD-операцій, а WebSockets – для забезпечення реального часу в критично важливих частинах додатка.

Для масштабування мікросервісів зазвичай використовуються контейнери, зокрема за допомогою інструментів Docker і Kubernetes. Docker дозволяє «запаковувати» кожен мікросервіс у контейнер разом з усіма необхідними бібліотеками та залежностями. Це забезпечує сумісність середовищ розробки та продуктивного середовища, дозволяючи легко розгортати та масштабувати кожен сервіс незалежно. Docker робить процеси масштабування гнучкими та дозволяє легко управляти окремими сервісами, що особливо важливо в динамічних системах.

Kubernetes є платформою для оркестрації контейнерів, яка дозволяє автоматизувати управління контейнерами Docker у великій інфраструктурі. Він надає механізми автоматичного масштабування, балансування навантаження, розподілення трафіку та моніторингу. Kubernetes управляє кластерами контейнерів, які можуть динамічно створювати нові екземпляри сервісів при зростанні навантаження, а також автоматично «вимикати» їх при зниженні активності. Це значно спрощує підтримку великих додатків та знижує витрати на ресурси.

Поєднання Docker і Kubernetes дозволяє створювати масштабовані, розподілені додатки, які можуть стабільно працювати при великих навантаженнях. Docker забезпечує легку контейнеризацію, а Kubernetes додає додатковий рівень автоматизації, управління та оркестрації, що дозволяє сервісам залишатися продуктивними навіть у складних інфраструктурах. Це ідеальне рішення для великих організацій, які потребують гнучкості та надійності в умовах зростаючих вимог.

Для підтримки високих навантажень мікросервісні архітектури використовують схему горизонтального масштабування, що дозволяє збільшувати кількість інстансів кожного сервісу відповідно до поточних вимог. Це досягається за допомогою балансувальника навантаження, який рівномірно розподіляє вхідні запити між наявними екземплярами. Кожен мікросервіс можна запустити в декількох контейнерах, що дозволяє обробляти більше запитів паралельно. При збільшенні трафіку або навантаження система автоматично створює додаткові контейнери, щоб зберігати стабільну продуктивність. Окрім балансування навантаження, система моніторингу є важливим елементом схеми масштабування. Вона відстежує ключові метрики, такі як затримка відповідей, споживання ресурсів та активність користувачів. У разі виявлення зростання навантаження, моніторингова система взаємодіє з оркестратором (наприклад, Kubernetes), щоб масштабувати окремі сервіси відповідно до поточних

потреб. Це автоматичне масштабування допомагає уникнути перевантажень і збоїв у системі. Схема також передбачає автоматичне відключення зайвих ресурсів у періоди низької активності, що знижує витрати на інфраструктуру. Таким чином, мікросервісна архітектура з підтримкою навантаження забезпечує стабільну роботу додатка при зростанні трафіку і гарантує ефективне використання ресурсів [17].

1.3 Вимоги до системи для підтримки email-розсилки та інтеграції з месенджерами

Продуктивність мікросервісної системи для автоматизованої розсилки повідомлень залежить від кількості подій, які система обробляє, і кількості активних користувачів. При високому обсязі подій (наприклад, одночасному надходженні тисяч запитів) можуть виникати проблеми з продуктивністю, якщо інфраструктура недостатньо оптимізована. Система повинна бути готова обробляти як значні пікові навантаження, так і стабільні обсяги подій при зростанні кількості користувачів.

Ключові показники для моніторингу продуктивності включають середній час відповіді, відсоток успішно доставлених повідомлень, пропускну здатність черг повідомлень і здатність до масштабування. Наприклад, при збільшенні кількості користувачів затримка в обробці подій може збільшуватися, якщо система не встигає масштабуватися відповідно до вимог. Використання автоматичного масштабування, балансувальників навантаження і розподілу ресурсів допомагає підтримувати продуктивність на належному рівні [18].

Планування інфраструктури має враховувати максимальні навантаження, передбачаючи можливості розширення черг повідомлень, обробки API-запитів і потужностей бази даних. За допомогою тестів навантаження можна перевірити, як система поводиться за різних рівнів

активності користувачів та кількості подій, щоб встановити необхідні технічні вимоги. Представимо технічні вимоги для підтримки великої кількості повідомлень у таблиці 1.3.

Таблиця 1.3 – Технічні вимоги для підтримки великої кількості повідомлень

Компонент	Технічні вимоги
Сервіс обробки подій	Висока пропускна здатність черг подій (мін. 1000 подій/сек); підтримка RabbitMQ/Kafka; можливість горизонтального масштабування
Черга повідомлень	Можливість розподілення подій між сервісами; обробка від 10,000 подій/сек у пікових навантаженнях; резервне копіювання
Сервіс для формування повідомлень	Швидкодія при обробці текстових і мультимедійних повідомлень; підтримка персоналізації; інтеграція з аналітикою
API інтеграції з месенджерами	Стабільна робота при високих навантаженнях; підтримка 10-15 підключень/сек на кожен канал; адаптація під вимоги платформ
Система моніторингу	Підтримка реального часу (Prometheus, Grafana); відстеження затримки відповіді, відмов у обробці подій
База даних користувачів	Масштабованість; зберігання великих обсягів даних (до 1 млн користувачів); підтримка швидкого пошуку та запитів
Автоматичне масштабування	Динамічне розгортання нових екземплярів сервісів при навантаженні; інтеграція з оркестратором Kubernetes
Безпека	Шифрування даних; автентифікація та авторизація користувачів; захист API від атак (наприклад, DDoS)

Забезпечення безпеки даних в системах автоматизованої розсилки повідомлень є важливим аспектом, особливо при обробці конфіденційної інформації та даних користувачів. Основні вимоги включають шифрування даних як при зберіганні (data-at-rest), так і під час передачі (data-in-transit), а також автентифікацію та авторизацію для доступу до системи. Шифрування надає додатковий рівень захисту, що знижує ризики несанкціонованого доступу та перехоплення даних. У таких системах також необхідні заходи

щодо захисту від атак, таких як DDoS, та запобігання ін’єкційним атакам, які можуть призвести до витоку даних.

Захист даних під час передачі забезпечується за допомогою стандартів SSL/TLS, які дозволяють шифрувати весь трафік між сервером і клієнтом. Це захищає дані від перехоплення, що є особливо важливим при інтеграції з месенджерами та іншими зовнішніми сервісами. Додатково, при використанні мікросервісної архітектури бажано застосовувати VPN для захисту внутрішнього трафіку між сервісами, а також інструменти шифрування, такі як AES або RSA, для зашифрованого зберігання даних у базах даних [19].

Наведемо приклади безпекових рішень для розсилки повідомлень:

– JWT (JSON Web Token) використовується для безпечної передачі даних між клієнтами та серверами. JWT забезпечує надійну автентифікацію користувачів, а також дозволяє шифрувати певну частину токена для захисту конфіденційних даних. JWT часто використовується для автентифікації API-запитів, дозволяючи уникнути передачі паролів або інших чутливих даних;

– OAuth 2.0, поширений протокол для авторизації, який надає доступ до системи без необхідності розкривати паролі користувача. У месенджерах, таких як Slack та Telegram, OAuth 2.0 часто використовується для забезпечення безпеки інтеграцій, надаючи контроль над правами доступу та зберігаючи дані користувачів у безпеці;

– End-to-End Encryption (E2EE), для систем, що надсилають чутливі дані через месенджери, важливо підтримувати наскрізне шифрування. Це забезпечує, що тільки відправник і отримувач можуть прочитати повідомлення. Приклади E2EE є стандартами в WhatsApp та Signal, де повідомлення шифруються на пристрої відправника і розшифровуються тільки на пристрої отримувача;

– Rate Limiting та захист від DDoS-атак, для систем з високим обсягом запитів важливо обмежувати швидкість запитів (rate limiting) та мати захист

від DDoS-атак, щоб уникнути перевантаження серверів. Інструменти, як-от Cloudflare, можуть забезпечити рівень захисту від таких атак і забезпечити додатковий бар'єр перед API-сервісами;

– моніторинг безпеки, постійний моніторинг є важливим для своєчасного виявлення потенційних загроз. Інструменти, такі як Prometheus або Datadog, дозволяють відстежувати незвичайну активність у системі, зміну патернів трафіку або підозрілі спроби доступу, що дозволяє оперативно реагувати на інциденти безпеки.

Інтеграція з месенджерами, такими як WhatsApp, Telegram, та Viber, вимагає врахування технічних, безпекових і регуляторних вимог, які гарантують надійний обмін повідомленнями між системою і платформами. Вибір відповідного API для інтеграції, встановлення протоколів обміну даними, забезпечення безперервного з’єднання та адаптація під специфічні вимоги кожного месенджера є ключовими аспектами, які потрібно врахувати. Крім того, необхідно підтримувати захист даних, оскільки месенджери можуть передавати конфіденційну інформацію користувачів [20].

1.4 Постановка задач дослідження

Розроблення комп’ютеризованої системи для автоматизації процесу розсилки email-повідомлень за результатами обумовлене необхідністю підвищення ефективності комунікації в організаціях, які обробляють великий обсяг подій і потребують оперативного інформування учасників про їх результати. Ручне виконання таких задач часто призводить до затримок, помилок та значних витрат часу, що особливо критично в умовах сучасних динамічних процесів. Автоматизація цього процесу дозволяє забезпечити своєчасну і точну передачу інформації, зменшити людський фактор у помилках, а також оптимізувати робочі ресурси. Крім того, така система забезпечує інтеграцію з іншими інформаційними платформами та гнучке

налаштування під специфіку конкретної організації, що робить її універсальним рішенням для покращення управління процесами.

В ході проведеного у першому розділі аналізу в кваліфікаційній роботі, було виявлено, що тема дослідження є актуальною. Метою дослідження є удосконалення процесу автоматизації розсилки за рахунок впровадження іноваційних технічних рішень. Об’єктом дослідження є процес автоматизації розсилки повідомлень. Предметом дослідження є мікросервісні архітектури та технічні рішення для реалізації автоматизованих розсилок. Для досягнення поставленої мети необхідно вирішити наступні завдання:

– визначити функціональні вимоги до програмного забезпечення;

– провести вибір мови програмування та середовища розробки БД;

– визначити структуру даних для зберігання подій та налаштувань користувачів;

– розробити ER-діаграму бази даних та опис зв’язків між таблицями;

– спроектувати фізичну модель БД;

– описати бізнес-логіку роботи ПЗ;

– розробити алгоритми автоматизації розсилки email-повідомлень та повідомлень у месенджери;

– розробити функції обробки подій і тригерів для розсилки повідомлень;

– провести налаштування конфігурацій користувачів для автоматизованих розсилок;

– розробити функції логування та протоколювання роботи сервісу;

– розробити серверну частину мікросервісу;

– реалізувати базу даних на MongoDB з використанням мови SQL;

– розробити інтерфейс користувача;

– провести оцінку продуктивності та надійності програмного забезпечення.

2 РОЗРОБКА ER-ДІАГРАМИ БАЗИ ДАНИХ ТА ПРОЕКТУВАННЯ ЛОГІЧНОЇ ТА ФІЗИЧНОЇ МОДЕЛІ БД

2.1 Визначення функціональних вимог до програмного забезпечення

Процес автоматизації розсилки e-mail повідомлень є важливим інструментом для оперативного інформування, реагування на події та підтримання зв’язку з клієнтами чи користувачами. Вимоги до розробки такого ПЗ формуються з урахуванням його мети – забезпечення точності, своєчасності, безпеки та зручності у використанні.

Аналіз вимог дозволяє вирішити наступні завдання:

– забезпечити відповідність системи потребам користувачів;

– мінімізувати ризики, пов’язані з помилками в розсилці (наприклад, надсилання зайвих повідомлень або помилки в контенті);

– оптимізувати витрати на розробку завдяки чіткому визначенню функціоналу.

Виходячи з проведеного аналіз у першому розділі, для розроблюваного ПЗ обрано наступний перелік функціональних вимог, які представлено в таблиці 2.1.

Таблиця 2.1 – Перелік функціональних вимог, які пред’являються розроблюваному ПЗ

Вимога до ПЗ	Опис	Обґрунтування
1	2	3
Збір та аналіз подій	ПЗ повинно забезпечувати збір інформації про події з визначених джерел (лог-файли, API, бази даних тощо) та її аналіз для прийняття рішення про необхідність розсилки	Забезпечує релевантність розсилки, знижуючи ризик надсилання непотрібних повідомлень

Продовження таблиці 2.1

1	2	3
Фільтрація подій за критеріями	ПЗ повинна мати можливість налаштовувати критерії для фільтрації подій (наприклад, за типом, рівнем пріоритету, джерелом)	Дозволяє зосередити увагу на важливих подіях, що потребують дій
Генерація шаблонів повідомлень	Повідомлення повинні автоматично формуватися на основі шаблонів, які містять змінні, що заповнюються динамічними даними про подію	Забезпечує персоналізацію повідомлень та економію часу на їх створення
Масова розсилка	ПЗ повинно підтримувати відправку великої кількості повідомлень одночасно з використанням черги для уникнення перевантажень	Знижує ризик блокування серверів розсилки та забезпечує стабільність системи
Розклад та відкладена відправка	Реалізація можливості планування розсилки на визначений час	Дозволяє врахувати часові зони отримувачів або оптимізувати час відправки для більшої ефективності
Моніторинг статусів розсилки	ПЗ повинна відстежувати статуси повідомлень (відправлено, доставлено, помилка) і зберігати їх у базі даних	Дозволяє оцінити ефективність розсилки та виявити технічні проблеми
Журнал подій та аудиту	Фіксування всіх дій системи, пов’язаних із розсилкою (запуск, зміна налаштувань, помилки)	Забезпечує прозорість і можливість аналізу роботи системи для виправлення помилок або оптимізації
Управління списками отримувачів	ПЗ повинна підтримувати управління базою даних отримувачів (додавання, видалення, сегментація)	Дозволяє більш точно орієнтувати розсилку на відповідні групи користувачів
Забезпечення безпеки даних	Захист особистих даних отримувачів через шифрування, автентифікацію доступу та дотримання норм GDPR	Важливо для уникнення витоку даних та юридичних проблем

Продовження таблиці 2.1

1	2	3
Сумісність із платформами	ПЗ повинна бути доступною через веб-інтерфейс і підтримувати інтеграцію з іншими платформами через API	Забезпечує універсальність та легкість у використанні
Налаштування та масштабованість	Можливість зміни конфігурацій ПЗ (кількість одночасних розсилок, шаблони, інтеграції)	Сприяє адаптації системи під змінні потреби користувачів.
Звітування	Генерація звітів про розсилку (кількість відправлених листів, відкриття, переходи за посиланнями)	Дає змогу оцінити ефективність розсилки та покращити її результати

Виконання всіх перелічених вимог, представлених у таблиці 2.1 дозволить створити програмне забезпечення, що відповідає потребам автоматизації розсилки e-mail повідомлень. Аналіз і обґрунтування кожної вимоги забезпечує розробку стабільної, безпечної та ефективної системи, яка задовольняє вимоги сучасного бізнесу та кінцевих користувачів [4].

2.2 Вибір мови програмування та середовища розробки БД

Розробка програмного забезпечення для автоматизації розсилки e-mail повідомлень вимагає врахування кількох ключових факторів: продуктивності, зручності у розробці, можливості масштабування, роботи з API та інтеграції з базами даних. У цьому контексті доцільно порівняти сучасні мови програмування та середовища розробки БД. Порівняльний аналіз мов програмування представлено в таблиці 2.2.

Таблиця 2.2 – Порівняльний аналіз сучасних мов програмування для розробки ПЗ

Мова програмування	Переваги	Недоліки
Python	простий синтаксис, що прискорює розробку; велика кількість бібліотек для роботи з e-mail (наприклад, smtplib, email) та API (наприклад, requests); підтримка багатьох фреймворків для веб-розробки (Flask, Django)	відносно низька продуктивність для великих систем; не завжди підходить для обробки високонавантажених систем
JavaScript (Node.js)	асинхронна модель роботи забезпечує високу продуктивність для систем із великою кількістю запитів; широкий вибір модулів для роботи з e-mail (наприклад, nodemailer); один стек технологій для серверної та клієнтської частини (JS на сервері та в браузері)	потребує обережного підходу до обробки винятків через асинхронність; більше уваги до оптимізації коду, особливо в складних проєктах
Java	надійність і масштабованість; велика кількість бібліотек для роботи з e-mail та інтеграції з базами даних (наприклад, JavaMail API)	висока складність та час на розробку; більший обсяг коду для реалізації стандартних функцій у порівнянні з JavaScript чи Python
C# (.NET)	інтеграція з Microsoft середовищами; потужний інструментарій для роботи з API, базами даних та багатопотоковістю	залежність від платформи Windows (хоча .NET Core працює і на інших ОС); відносно висока вартість інфраструктури

Порівняльний аналіз мов програмування, приведений в таблиці 2.2 показав, що JavaScript (Node.js) є оптимальним вибором для створення програмного забезпечення для автоматизації розсилки e-mail повідомлень. Python забезпечує простоту розробки та має багатий набір бібліотек, але

поступається Node.js у продуктивності для високонавантажених систем. Java демонструє високу надійність і масштабованість, але потребує більше часу на розробку та більших ресурсів. C# є потужним інструментом, проте його залежність від екосистеми Microsoft та складніша інтеграція робить його менш привабливим у цьому контексті. Node.js відрізняється асинхронною архітектурою, яка забезпечує обробку великої кількості запитів із високою швидкістю, а також має численні бібліотеки, такі як nodemailer, які спрощують роботу з e-mail сервісами. Завдяки єдиній мові програмування для серверної і клієнтської частини, JavaScript скорочує час розробки та забезпечує легкість масштабування. Таким чином, Node.js забезпечує найкращий баланс між продуктивністю, зручністю розробки, інтеграцією та масштабованістю, що робить його оптимальним вибором для розробки такого типу програмного забезпечення.

Базуючись на вибраній мові програмування JavaScript (Node.js), необхідно наступним етапом обрати БД та СУБД для реалізації функцій зберігання даних, для розроблюваного ПЗ для автоматизації розсилки e-mail повідомлень. Порівняльний аналіз сучасних БД та СУБД представлено в таблиці 2.3.

Таблиця 2.3 – Порівняльний аналіз сучасних БД та СУБД

БД	Переваги	Недоліки
1	2	3
MySQL	надійність і популярність; підтримка складних запитів, транзакцій	менш ефективний для роботи з неструктурованими даними; складніший у масштабуванні порівняно з NoSQL базами
PostgreSQL	потужна підтримка складних запитів і реляційної логіки; підтримка JSON-формату для зберігання неструктурованих даних	вища складність налаштування та обслуговування, порівняно з MySQL

Продовження таблиці 2.3

1	2	3
MongoDB	гнучкість у роботі з неструктурованими даними (JSON-подібний формат); легкість у масштабуванні (sharding, реплікація); проста інтеграція з Node.js (за допомогою драйвера MongoDB)	обмеженість складних транзакцій; залежність від оперативної пам'яті для швидкодії
SQLite	легкість у використанні, немає необхідності в сервері; підходить для невеликих проєктів	не підходить для високонавантажених систем
Firebase Realtime Database	інтеграція в реальному часі; хмарне зберігання даних	висока вартість для великих обсягів даних; обмеження у складних запитах

Аналіз середовищ розробки баз даних виявив, що MongoDB є найкращим вибором для автоматизації процесу розсилки e-mail повідомлень завдяки своїм перевагам у роботі з неструктурованими даними. Реляційні бази, такі як MySQL і PostgreSQL, добре підходять для складних транзакцій і реляційних моделей, але їх гнучкість обмежується при роботі з динамічними або змінними структурами даних, які часто потрібні для зберігання подій чи історії повідомлень. MongoDB, працюючи з JSON-подібними документами, спрощує організацію даних, дозволяючи легко адаптувати структуру до змін без складної міграції. Firebase добре інтегрується з реальним часом, але його обмеження у складних запитах та висока вартість роблять його менш ефективним для масштабних проєктів. MongoDB також забезпечує легкість масштабування завдяки вбудованій підтримці реплікації та розподілу даних (sharding), що критично важливо для зростаючої кількості записів. Інтеграція з Node.js через офіційний драйвер MongoDB є простою, швидкою та ефективною. Завдяки цьому MongoDB забезпечує високу гнучкість, масштабованість і продуктивність, що робить його оптимальним вибором для такого типу програмного забезпечення.

2.3 Визначення структур даних для зберігання подій та налаштувань користувачів

Для розроблюваного програмного забезпечення автоматизації розсилки e-mail повідомлень обґрунтовано використання документної моделі даних MongoDB. Цей підхід забезпечує гнучкість, оскільки дозволяє легко адаптувати структуру документів до змін бізнес-логіки. Структури даних спроектовані так, щоб забезпечити зберігання інформації про події та налаштування користувачів у вигляді JSON-подібних документів. Дані подій включають ідентифікатор події, тип події, час створення, статус обробки та деталі, які можуть бути специфічними для кожного типу подій. Налаштування користувачів включають унікальний ідентифікатор користувача, його електронну адресу, налаштування частоти розсилки, список підписок на типи подій та час останнього оновлення налаштувань. Така структура дозволяє зберігати всі необхідні дані в одній колекції або використовувати зв’язки між окремими колекціями для більшого контролю над даними. Завдяки MongoDB можна використовувати вкладені документи для детальних налаштувань або великих масивів даних, що значно спрощує зберігання динамічної інформації. Для Node.js структура даних легко інтегрується через Mongoose або інші ODM-бібліотеки, забезпечуючи високопродуктивну обробку даних у реальному часі.

Обґрунтування структури даних для зберігання подій та налаштувань користувачів базується на принципах ефективної організації даних, підтримки гнучкості та масштабованості системи для автоматизації розсилки e-mail повідомлень. Розподіл бази на чотири таблиці забезпечує чітке розмежування функціональних завдань і спрощує обробку запитів. Таблиця Users дозволяє зберігати основні дані про користувачів, включаючи ім'я та електронну адресу, що є необхідним мінімумом для ідентифікації та взаємодії із системою. Events відображає доступні аналітичні функції чи

події, які користувачі можуть обирати для отримання розсилок. Це гарантує централізоване зберігання даних про події, спрощуючи їх оновлення та управління. Посередницька таблиця UserSettings створює зв’язок між користувачами та подіями, що дозволяє підтримувати гнучкі підписки. Використання унікального ключа для поєднання користувача та події забезпечує відсутність дублювання даних і точність налаштувань. Таблиця WorkProtocol дає змогу вести історію взаємодії з користувачами, що є важливим для моніторингу та аналізу ефективності системи. Зберігання дат та ідентифікаторів у WorkProtocol забезпечує можливість ретроспективного аналізу, виявлення проблемних точок і формування звітності. Така структура є оптимальною, оскільки забезпечує простоту взаємодії з даними, можливість масштабування бази за рахунок додавання нових атрибутів або зв’язків та гнучкість у роботі з користувацькими підписками. Це рішення відповідає принципам нормалізації даних, знижуючи ризик дублювання і підвищуючи швидкість роботи системи, особливо для обробки великої кількості підписок і подій. Обґрунтування вибору наступної структури приведена в таблиці 2.4.

Таблиця 2.4 – Причина вибору та призначення таблиць БД

Таблиця	Призначення	Причина вибору
Users	Зберігає інформацію про користувачів	Простота ідентифікації та персоналізації
Events	Описує доступні аналітичні функції та події	Централізоване управління подіями
UserSettings	Встановлює зв’язок між користувачами та подіями	Гнучкість у підтримці підписок
WorkProtocol	Фіксує історію відправок	Підтримка аудиту та аналізу ефективності

Ця структура гарантує, що дані залишаються послідовними, легко керованими й масштабованими, що особливо важливо для великих систем автоматизації e-mail розсилок.

2.4 Розробка ER-діаграми бази даних та опис зв’язків між таблицями

Розробка ER-діаграми бази даних є важливим етапом у створенні програмного забезпечення для автоматизації процесу розсилки e-mail повідомлень за результатами аналізу подій. Вона дозволяє забезпечити чітке уявлення про структуру даних та взаємозв’язки між основними сутностями, що становлять основу системи. Завдяки цьому стає можливим ефективне моделювання та реалізація бази даних, яка відповідає вимогам проекту, зокрема щодо зберігання, обробки та управління інформацією про користувачів, події, налаштування підписок і журнали роботи.

На рисунку 2.1 представлено ER-діаграму, яка відображає ключові сутності бази даних, такі як «Користувачі», «Події», «Налаштування Користувачів» та «Журнал Роботи», а також зв’язки між ними, що дозволяють реалізувати функціональність системи. Відображення взаємозв’язків між таблицями дозволяє гарантувати цілісність даних, уникаючи дублювання інформації, та забезпечує гнучкість у розширенні системи. Наприклад, посередницька таблиця «Налаштування Користувачів» ефективно моделює зв’язок «багато-до-багатьох» між користувачами та подіями, що критично важливо для динамічного налаштування підписок. Також таблиця «Журнал Роботи» дозволяє вести історію відправки повідомлень для кожного користувача, зберігаючи часові мітки, що забезпечує можливість аудиту та аналізу. Таким чином, ER-діаграма виступає основою для правильного проєктування бази даних і слугує інструментом, який сприяє ефективному розробленню програмного забезпечення.

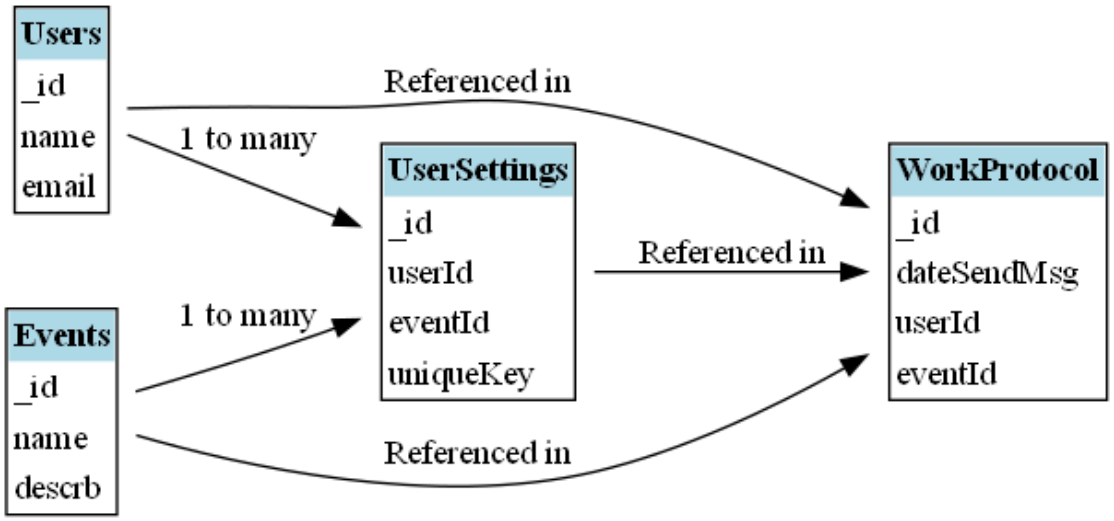


Рисунок 2.1 – ER-діаграма бази даних та опис зв’язків між таблицями

2.5 Проектування фізичної моделі БД

Проектування фізичної моделі бази даних передбачає створення структури таблиць із визначенням полів, типів даних, розмірності та встановленням зв’язків між таблицями відповідно до логічної моделі. У програмному забезпеченні для автоматизації процесу розсилки e-mail повідомлень структура бази даних побудована на чотирьох основних таблицях: Users, Events, UserSettings та WorkProtocol. Розглянемо фізичну реалізацію кожної таблиці в контексті використання MongoDB. Опис таблиць БД з об означенням типу даних опису приведено в таблицях 2.5-2.8.

Таблиця Users зберігає інформацію про користувачів системи, необхідну для ідентифікації та подальшого спілкування через e-mail. Фізична модель приведена в таблиці 2.5.

Таблиця 2.5 – Фізична модель таблиці Users

Поле	Тип даних	Опис
_id	ObjectId	Унікальний ідентифікатор користувача (генерується автоматично MongoDB)
name	String	Ім'я користувача, до 100 символів
email	String	Електронна адреса користувача, до 150 символів

Таблиця Events містить інформацію про події, доступні для користувачів, які вони можуть обрати для аналізу. Фізична модель приведена в таблиці 2.6.

Таблиця 2.6 – Фізична модель таблиці Events

Поле	Тип даних	Опис
_id	ObjectId	Унікальний ідентифікатор події
name	String	Назва події, до 200 символів
descrb	String	Опис події, необмежена кількість символів (тип Text)

Таблиця UserSettings реалізує зв’язок «багато-до-багатьох» між користувачами і подіями, що дозволяє одному користувачеві підписуватись на декілька подій, і навпаки. Фізична модель приведена в таблиці 2.7.

Таблиця 2.7 – Фізична модель таблиці UserSettings

Поле	Тип даних	Опис
_id	ObjectId	Унікальний ідентифікатор запису
userId	ObjectId	Ідентифікатор користувача з таблиці Users
eventId	ObjectId	Ідентифікатор події з таблиці Events
uniqueKey	String	Унікальний ключ, створений на основі userId та eventId, до 250 символів

Таблиця WorkProtocol зберігає інформацію про кожне відправлене повідомлення, включаючи дату і час, користувача та подію. Фізична модель приведена в таблиці 2.8.

Таблиця 2.8 – Фізична модель таблиці WorkProtocol

Поле	Тип даних	Опис
_id	ObjectId	Унікальний ідентифікатор запису
dateSendMsg	Date	Дата та час відправлення повідомлення
userId	ObjectId	Ідентифікатор користувача з таблиці Users
eventId	ObjectId	Ідентифікатор події з таблиці Events

Вибір типів полів і їх розмірності для бази даних, що використовується в програмному забезпеченні для автоматизації процесу розсилки e-mail повідомлень, обґрунтований вимогами до продуктивності, масштабованості та забезпечення коректності даних. Поля типу ObjectId обрано для унікальних ідентифікаторів, оскільки цей тип є стандартним для MongoDB, гарантує унікальність кожного запису в колекції та займає фіксований обсяг пам'яті, що забезпечує швидкий доступ до даних. Поля типу String використовуються для текстових даних, таких як імена, електронні адреси та описи, оскільки вони забезпечують гнучкість у зберіганні інформації, яка може відрізнятися за довжиною. Обмеження розмірності таких полів, як ім'я (100 символів) або електронна адреса (150 символів), враховує стандарти для таких даних, уникаючи перевантаження системи та забезпечуючи ефективне використання пам’яті. Поля з необмеженою кількістю символів, такі як описи подій, обрані з метою збереження великої кількості текстової інформації, наприклад, детального пояснення аналітичних функцій. Тип Date для збереження часу і дати розсилки повідомлень дозволяє виконувати точний аналіз журналу відправлень, а також сортування й фільтрацію даних. Поле uniqueKey, створене з комбінації ідентифікаторів користувача та події, гарантує унікальність підписок і мінімізує ризик дублювання, що критично важливо для коректного виконання аналітичних функцій. Такий підхід дозволяє забезпечити оптимальний баланс між функціональністю бази даних, її продуктивністю та здатністю масштабуватися в умовах зростання обсягів даних.

2.6 Висновки до 2 розділу

У другому розділі було виконано проектування бази даних, яка є основою для програмного забезпечення автоматизації процесу розсилки e-mail повідомлень. Першим етапом стало визначення функціональних

вимог, що дозволило сформулювати ключові завдання системи, включаючи зберігання інформації про користувачів, події, їх налаштування та журнал дій. На основі цих вимог було обґрунтовано вибір мови програмування JavaScript (Node.js) і бази даних MongoDB, які забезпечують високу продуктивність і гнучкість роботи з динамічними даними. Визначення структури даних дозволило сформувати чітку модель для зберігання інформації про події та налаштування користувачів, що забезпечує ефективну інтеграцію з іншими компонентами системи. Розробка ER-діаграми, представлена на рисунку 2.1, візуалізувала зв’язки між сутностями, такими як «Користувачі», «Події», «Налаштування Користувачів» та «Журнал Роботи», забезпечуючи цілісність даних і простоту управління ними. Проектування фізичної моделі бази даних надало детальне визначення типів даних і розмірностей полів, враховуючи вимоги до продуктивності системи та оптимізації зберігання інформації. Усі підрозділи цього розділу взаємопов’язані та спрямовані на створення надійної основи для розробки програмного забезпечення, що забезпечує масштабованість, гнучкість і високу ефективність роботи з даними.

3 РОЗРОБКА БІЗНЕС-ЛОГІКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Опис бізнес-логіки роботи ПЗ

Бізнес-логіка роботи програмного забезпечення (ПЗ) – це сукупність правил, алгоритмів і процесів, що визначають, як програмне забезпечення обробляє дані, взаємодіє з користувачами, виконує операції та забезпечує досягнення поставлених цілей. Вона відображає специфічні потреби бізнесу чи організації, які вирішує ПЗ, і є центральним елементом архітектури будь-якої програми. Бізнес-логіка визначає порядок виконання функцій, умови прийняття рішень і способи обробки інформації відповідно до потреб кінцевих користувачів. Її розробка дозволяє зробити програму гнучкою, масштабованою та пристосованою до змін у процесах або вимогах.

Бізнес-логіка є необхідною складовою розробки програмного забезпечення, оскільки вона встановлює зв'язок між технічними можливостями системи та її функціональними цілями. Вона дозволяє розробникам створювати рішення, які відповідають конкретним задачам, оптимізуючи внутрішні процеси й забезпечуючи ефективну взаємодію між користувачами та системою. У контексті автоматизації розсилки e-mail повідомлень бізнес-логіка визначає, які події вимагають надсилання повідомлень, як ці повідомлення генеруються, персоналізуються та доставляються до користувачів. Її розробка гарантує, що система реагуватиме на події своєчасно, релевантно та відповідно до встановлених бізнес-вимог. Це знижує ризик помилок, підвищує ефективність роботи системи та сприяє задоволенню потреб кінцевих користувачів.

Для програмного забезпечення автоматизації процесу розсилки e-mail повідомлень обрана подієво-орієнтована модель бізнес-логіки. Ця модель є оптимальною, оскільки забезпечує реагування системи на конкретні події, що

зберігаються в базі даних або надходять через API. Подієво-орієнтована модель дозволяє збудувати чіткий і структурований процес обробки подій, які запускають відповідні дії, включаючи генерацію повідомлень, персоналізацію контенту та їх розсилку.

Подієво-орієнтована модель забезпечує гнучкість і масштабованість системи, дозволяючи легко додавати нові типи подій чи функціональні можливості без значних змін у базовій архітектурі. Крім того, така модель добре інтегрується із сучасними сервісами аналітики та API для збору даних, дозволяючи реагувати на результати аналізу подій у реальному часі. Це важливо для досягнення основних цілей розроблюваного ПЗ, включаючи персоналізацію, швидкість реагування та ефективність комунікації.

Подієво-орієнтована модель бізнес-логіки може бути описана у вигляді математичної системи, яка базується на множинах, функціях і правилах. Для розробки моделі бізнес-логіки на базі подієво-орієнтована моделі введемо наступні параметри, які представлені в таблиці 3.1.

Таблиця 3.1 – Параметри та їх опис

Параметр	Опис
u_i	унікальний ідентифікатор користувача
e_j	дентифікатор події.
m_k	унікальне повідомлення, пов’язане з подією
t_p	час активації події або відправки повідомлення
f_U	визначає логіку зв’язків між користувачами та подіями
f_S	слідкує за статусом доставки повідомлень

У контексті автоматизації розсилки e-mail повідомлень, модель складається з таких основних компонентів:

– множини та параметри розроблювального ПЗ:

$$U = \{u_1, u_2, \dots, u_n\}, \tag{3.1}$$

$$E = \{e_1, e_2, \dots, e_m\}, \tag{3.2}$$

$$M = \{m_1, m_2, \dots, m_k\}, \tag{3.3}$$

$$T = \{t_1, t_2, \dots, t_p\}, \tag{3.4}$$

де U – множина користувачів;

u_1 – користувач системи;

E – множина подій;

e_j – подія, що може викликати розсилку;

M – множина повідомлень;

m_k – повідомлення, що генерується системою;

T – множина часу;

t_p – часовий параметр, який визначає момент, коли подія відбулася або повідомлення було відправлено.

– функції та залежності подієво-орієнтована модель бізнес-логіки для розроблювального ПЗ.

Функція обробки подій:

$$f_E: E \to M, \tag{3.5}$$

де $f_E(e_j)$ – визначає повідомлення m_k, яке повинно бути згенероване для події e_j.

Функція підписок користувачів:

$$f_U: U \to 2^E, \tag{3.6}$$

де $f_U(u_i)$ – визначає множину подій, на які підписаний користувач u_i;

2^E – множина всіх підмножин E.

Функція генерації повідомлень:

$$f_M: M \times U \to Content, \tag{3.7}$$

де $f_M(m_k, u_i)$ – формує контент повідомлення m_k для користувача u_i з урахуванням персоналізації.

Функція часу:

$$f_T: T \to \{0,1\}, \tag{3.8}$$

де $f_T(t_p) = 1$, якщо повідомлення m_k успішно доставлене, і 0 в іншому випадку.

Бізнес-логіка роботи системи описується як послідовність виконання функцій:

– вхідна подія e_j перевіряється функцією часу f_T, щоб визначити її актуальність;

– для кожного користувача u_i, що має підписку на подію e_j, визначену через f_U та викликається функція обробки подій f_E, яка створює повідомлення m_k;

– згенероване повідомлення передається у функцію генерації контенту f_M, що формує персоналізований текст;

– повідомлення m_k відправляється користувачу u_i і перевіряється функцією статусу доставки f_S.

Розроблена математична модель забезпечує формальне представлення подієво-орієнтованої бізнес-логіки, на основі якої будується розробка програмного забезпечення. Модель візуалізує ключові компоненти бізнес-логіки, такі як множини користувачів, подій, повідомлень, часу та функції, які описують їх взаємодію. Модель допомагає зрозуміти, як обробляються події, генеруються повідомлення, перевіряється їх актуальність та

забезпечується доставка. Графічне представлення математичної моделі подієво-орієнтованої бізнес-логіки для програмного забезпечення для автоматизації процесу розсилки e-mail повідомлень по результатам аналізу подій представлено на рисунку 3.1.

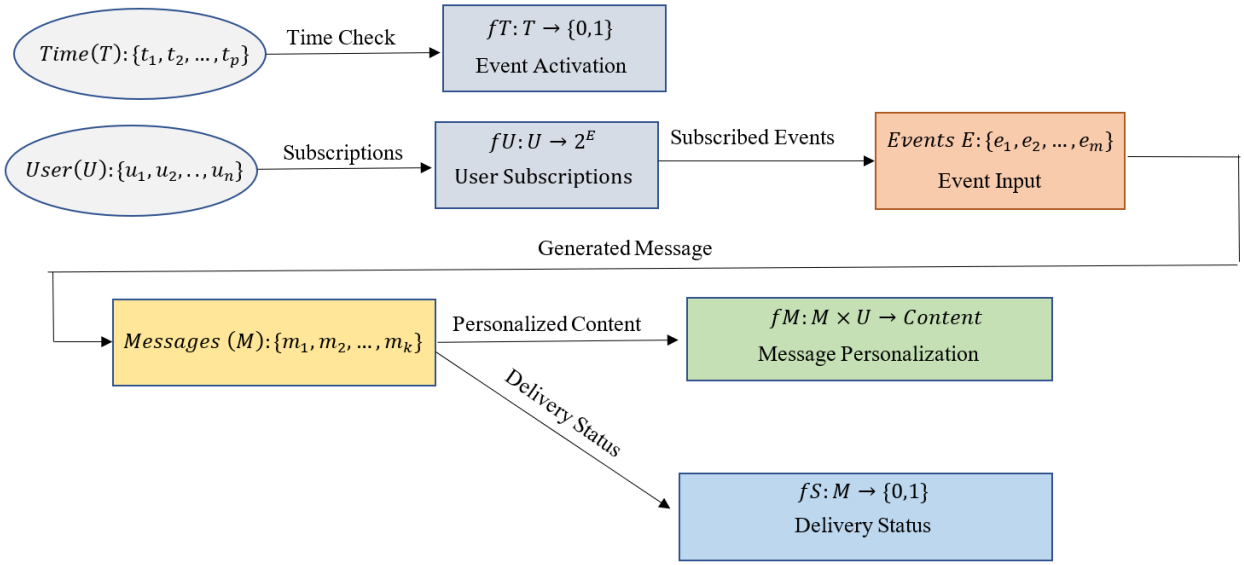


Рисунок 3.1 – Графічне представлення математичної моделі подієво-орієнтованої бізнес-логіки для програмного забезпечення для автоматизації процесу розсилки e-mail повідомлень по результатам аналізу подій

3.2 Розробка алгоритмів автоматизації розсилки email-повідомлень та повідомлень у месенджери

На основі розробленої структури бази даних пропонується наступне рішення, що програмне забезпечення побудовано за принципом подієво-орієнтованої моделі, де ключовими компонентами є події (Events), користувачі (Users), підписки користувачів (UserSettings) та журнал роботи (WorkProtocol). Це забезпечує гнучкість у створенні нових аналітичних івентів, управлінні користувачами та їхніми підписками, а також веденні журналу відправок. Логіка роботи передбачає інтеграцію з API для збору даних, їх аналіз у вигляді подій і автоматичну відправку результатів на

електронну пошту або месенджери. Загальний вид алгоритму представлено на рисунку 3.2.

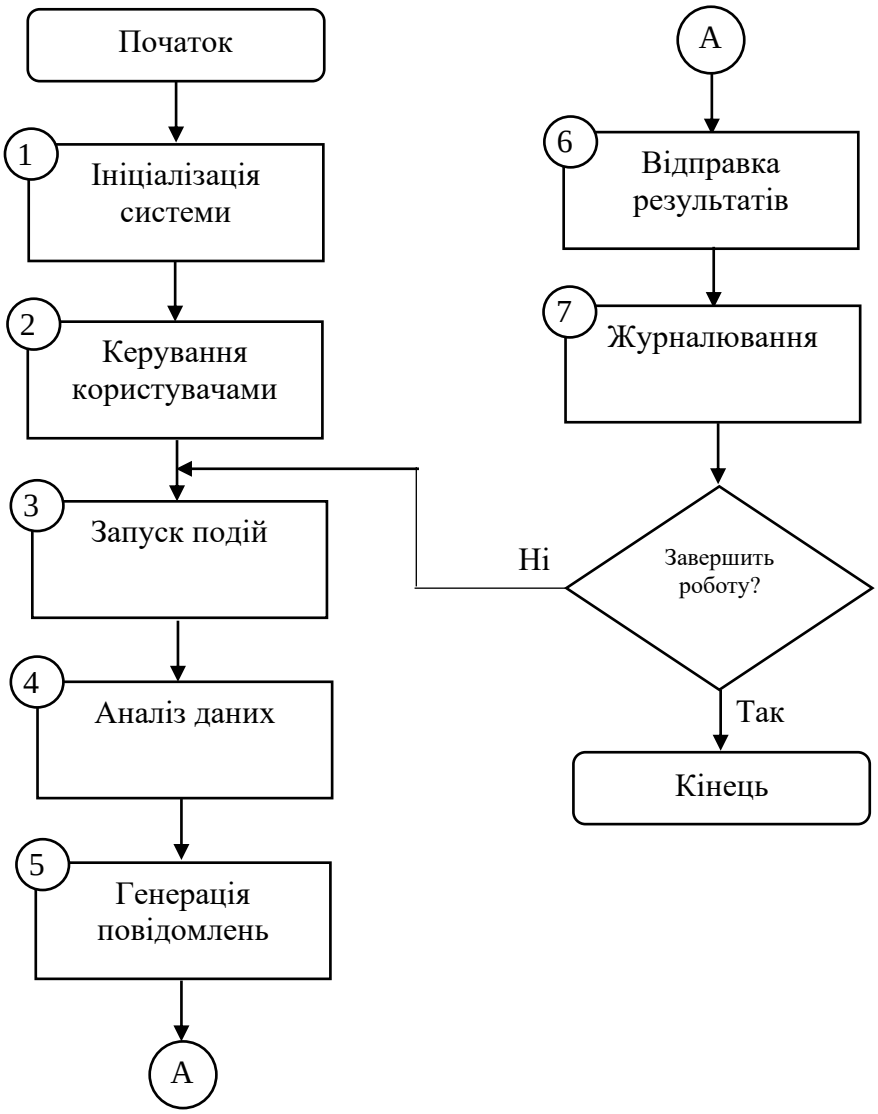


Рисунок 3.2 – Загальний алгоритм роботи ПЗ автоматизації процесу розсилки e-mail повідомлень

Опишемо призначення кожного блоку загального алгоритму роботи ПЗ автоматизації процесу розсилки e-mail повідомлень, яка представлена на рисунку 3.2.

Ініціалізація системи:

– завантаження базових даних подій із таблиці Events;

– підключення до API для збору інформації.

Керування користувачами:

– додавання/редагування користувачів через інтерфейс управління;

– налаштування підписок на події з таблиці UserSettings.

Запуск подій:

– події можуть бути запущені вручну користувачем або автоматично, якщо передбачено регулярний запуск;

– збір даних із зовнішніх джерел (наприклад, зовнішніх API).

Аналіз даних – виконання обчислень і формування результатів у форматі, готовому для відправлення (текстовий звіт або таблиця).

Генерація повідомлень – персоналізація контенту для кожного користувача на основі його підписок і специфічних налаштувань.

Відправка результатів:

– інтеграція з сервісами електронної пошти або месенджерів;

– відправка повідомлень користувачам, які підписані на відповідні події.

Журналювання:

– запис даних про відправлення повідомлень у таблицю WorkProtocol;

– збереження інформації про дату, користувача, подію та статус відправки.

Алгоритм починається із запуску події, яка ініціює процес збору даних з API. Система аналізує отримані дані, порівнюючи їх із заданими параметрами події, та формує аналітичний звіт. На основі підписок користувачів визначаються ті, кому необхідно надіслати результати. Далі відбувається персоналізація контенту, враховуючи ім’я користувача, його підписки та формат повідомлення. Згенеровані повідомлення додаються в чергу для відправлення через інтеграцію з e-mail сервісами або API месенджерів. Усі операції логуються в журналі для можливості подальшого аналізу.

Розроблений алгоритм для автоматизації розсилки e-mail повідомлень і повідомлень у месенджери має низку переваг, які забезпечують ефективність, гнучкість і надійність роботи системи. Він дозволяє обробляти події як у ручному, так і автоматичному режимі, що підвищує зручність використання для різних сценаріїв. Завдяки інтеграції з API для збору даних, алгоритм може динамічно аналізувати інформацію та швидко реагувати на зміни. Персоналізація повідомлень забезпечує точну відповідність контенту потребам користувачів, що сприяє кращій взаємодії з клієнтами. Наявність журналювання дозволяє відстежувати історію відправлень, аналізувати результати і забезпечувати прозорість роботи. Використання черг для відправки повідомлень мінімізує перевантаження системи та гарантує стабільну роботу навіть при великій кількості запитів. Така структура алгоритму підтримує масштабованість і легко адаптується до нових вимог або додаткових функцій.

3.3 Розробка функцій обробка подій і тригерів для розсилки повідомлень

Обробка подій і тригери для розсилки повідомлень є ключовими компонентами системи автоматизації, які забезпечують своєчасне реагування на зміну стану даних або виконання певних умов. Реалізація цих функцій дозволяє автоматизувати процес надсилання e-mail повідомлень і повідомлень у месенджери, зменшуючи втручання користувачів і підвищуючи ефективність роботи системи. Тригери виступають як механізми запуску розсилки, активуючи алгоритми обробки подій за певних умов, наприклад, оновлення даних чи виконання аналітичних розрахунків. У цьому підрозділі розглянуто методи створення функцій обробки подій і тригерів, які забезпечують точність і надійність автоматизованої системи. Приведемо

приклад програмної реалізації функцій обробки подій і тригерів для розсилки повідомлень, яка представлена нижче:

```
exports.getEvents = async (req, res) => {
  try {
    const events = await Event.find();
    const users = await User.find();
    const userSettings = await UserSettings.find().lean(); // Подписки пользователей

    const eventsWithSubscribers = events.map(event => {
      const subscribers = userSettings
        .filter(sub => sub.eventId.toString() === event._id.toString())
        .map(sub => users.find(user => user._id.toString() === sub.userId.toString()));

      return { ...event.toObject(), subscribers };
    });

    res.render('events', { events: eventsWithSubscribers, users });
  } catch (err) {
    res.status(500).send('Error fetching events');
  }
};
```

Цей фрагмент коду відповідає за отримання списку подій (івентів) із бази даних, визначення користувачів, підписаних на ці події, та відображення результатів у вигляді сторінки з доступними подіями. Код збирає дані про події, користувачів і їхні підписки, об’єднує ці дані, додаючи до кожної події список підписників, і передає їх до шаблону для рендерингу сторінки. У разі

виникнення помилки повертається відповідь із кодом статусу 500 та повідомленням про проблему:

```
let analysisResult;
switch (event.name) {
  case 'Daily Stock Change Analysis':
    analysisResult = await analyzeDailyStockChange(symbol);
    break;
  case 'Weekly Average Price Analysis':
    analysisResult = await analyzeWeeklyAverage(symbol);
    break;
  case 'Monthly High and Low Price Analysis':
    analysisResult = await analyzeMonthlyHighLow(symbol);
    break;
  case 'Rising Trend Check (3 days)':
    analysisResult = await checkRisingTrend(symbol);
    break;
  case 'Monthly Stock Change Analysis':
    analysisResult = await analyzeMonthlyStockChange(symbol);
    break;
  case '10-Day Volatility Analysis':
    analysisResult = await analyzeVolatility(symbol);
    break;
  case '2-Week Price Swing Analysis':
    analysisResult = await analyzeBiggestSwings(symbol);
    break;
  case 'Weekly Growth Percentage':
    analysisResult = await analyzeWeeklyGrowth(symbol);
    break;
  case '5-Day Volume Spike Check':
```

```
    analysisResult = await analyzeVolumeSpike(symbol);
    break;
  case '30-Day Daily Volatility Analysis':
    analysisResult = await analyzeDailyVolatility(symbol);
    break;
  default:
    return res.status(400).send('Unknown event');
}
```

Цей фрагмент коду виконує аналіз на основі обраного типу події, використовуючи механізм умовного вибору (switch-case). У залежності від назви події (event.name) викликається відповідна функція аналізу, яка обробляє дані для конкретного символу акції (symbol) та повертає результат. Якщо назва події не відповідає жодному із заданих випадків, повертається помилка з кодом статусу 400 і повідомленням про невідому подію:

```
let result;
switch (event.name) {
  case 'Daily Stock Change Analysis':
    result = await analyzeDailyStockChange(symbol);
    break;
  // Додати інші кейси для аналізу, якщо є
}

if (!result || result.includes('Error')) {
  return res.status(400).send(`Error analyzing stock data for ${symbol}.`);
}

// Генеруємо графік
const historicalData = await getHistoricalData(symbol);
```

```
const predictedData = predictNextWeekPrices(historicalData, 52); // Прогноз на 52 тижні (1 рік)
const chartImageBuffer = await generateChart(historicalData, predictedData);
```

Цей фрагмент коду призначений для виконання аналізу даних на основі назви події та генерації результату. У залежності від типу події (event.name) викликається відповідна функція аналізу для обраного символу акції (symbol). Якщо аналіз не вдається або повертає помилку, відправляється відповідь із кодом 400. Далі на основі історичних даних про акції створюється прогноз на наступний рік, і генерується графік, який зберігається у вигляді буферного зображення.

3.4 Налаштування конфігурацій користувачів для автоматизованих розсилок

Наступним кроком необхідно розробити механізми створення, редагування та збереження налаштувань, які дозволяють користувачам керувати своїми підписками на події. Конфігурації користувачів забезпечують гнучкість системи, дозволяючи персоналізувати отримання повідомлень відповідно до індивідуальних потреб і вподобань. Використання бази даних для зберігання підписок гарантує, що дані залишаються цілісними й доступними для обробки. Крім того, налаштування включають можливість вибору способу доставки результатів, частоти відправлень і типів подій, що робить систему зручною й ефективною для автоматизації розсилок. Приклад програмної реалізації приведено нижче:

```
exports.getUsers = async (req, res) => {
  try {
    const users = await User.find();
    const events = await Event.find(); // Получаем список всех событий
```

```
    res.render('users', { users, events });
  } catch (err) {
    res.status(500).send('Error fetching users and events');
  }
};
```

Фрагмент коду призначений для отримання списку всіх користувачів і подій із бази даних. Після успішного отримання даних вони передаються до шаблону для рендерингу сторінки, яка відображає користувачів і доступні події. У разі виникнення помилки під час виконання запиту повертається відповідь із кодом статусу 500 та повідомленням про проблему:

```
exports.addUser = async (req, res) => {
  const { name, email, eventIds } = req.body;
  try {
    const newUser = new User({ name, email });
    await newUser.save();

    // Якщо є події для передплати, додаємо їх у userSettings
    if (Array.isArray(eventIds)) {
      for (const eventId of eventIds) {
        await UserSettings.create({
          userId: newUser._id,
          eventId,
          uniqueKey: `${newUser._id}_${eventId}`
        });
      }
    }

    res.redirect('/users');
  } catch (err) {
```

```
    res.status(500).send('Error adding user and subscriptions');
  }
};
```

Цей фрагмент коду призначений для додавання нового користувача до системи та налаштування його підписок на події. На основі даних із запиту створюється новий користувач із зазначеними іменем та електронною поштою, а також зберігається у базі даних. Якщо вказані події для підписки, для кожного обраного ідентифікатора події створюється запис у таблиці UserSettings, що зв’язує користувача з подією. Після успішного виконання запиту користувача перенаправляють на сторінку списку користувачів, а у разі помилки повертається статус 500:

```
exports.getResultsBySubscriptions = async (req, res) => {
  try {
    const userId = req.params.id;
    const { symbol } = req.body;
    const user = await User.findById(userId);
    const userSubscriptions = await UserSettings.find({ userId }).lean();

    if (!symbol || symbol.trim() === '') {
      return res.status(400).send('Stock symbol is required');
    }

    if (!userSubscriptions.length) {
      return res.status(400).send('User has no subscriptions.');
    }

    const subscribedEvents = await Event.find({ _id: { $in: userSubscriptions.map(sub => sub.eventId) } });
    let results = [];
```

Цей фрагмент коду призначений для отримання результатів аналізу на основі підписок конкретного користувача. Система знаходить користувача за його ID, перевіряє наявність підписок у таблиці UserSettings та отримує пов’язану інформацію про події. Якщо користувач не має підписок або не вказано символ акцій для аналізу, повертається помилка з відповідним статусом. Після цього дані про підписані події використовуються для формування результатів, які будуть представлені користувачу.

3.5 Розробка функцій логування та протоколювання роботи сервісу

Наступним необхідним рішенням потрібно програмно реалізувати методи забезпечення прозорості та контролю за виконанням основних процесів у системі автоматизації розсилок. Логування дозволяє фіксувати ключові події, помилки та успішні операції, що важливо для моніторингу стану сервісу та швидкого реагування на можливі проблеми. Протоколювання забезпечує збереження історії виконаних дій, включаючи відправку повідомлень і взаємодію з користувачами, що сприяє аналітиці та аудитам. Використання структурованих логів і протоколів дозволяє інтегрувати сервіс із системами моніторингу, забезпечуючи масштабованість і стабільність роботи. Приведемо фрагменти програмної реалізації:

```
const WorkProtocol = require('../models/WorkProtocol');
const User = require('../models/User');
const Event = require('../models/Event');

exports.getLogs = async (req, res) => {
  try {
    const logs = await WorkProtocol.find()
      .populate('userId', 'name email')
      .populate('eventId', 'name')
```

```
        .lean();

      res.render('logs', { logs });
    } catch (error) {
      console.error('Error fetching logs:', error);
      res.status(500).send('Error fetching logs');
    }
  };
```

Фрагмент коду отримання журналу дій (логів) сервісу із таблиці WorkProtocol. Він завантажує записи журналу, додаючи до них інформацію про користувачів (ім’я, електронна пошта) та події (назва) за допомогою функції populate. Отримані дані передаються в шаблон для рендерингу сторінки з логами. У разі виникнення помилки виводиться повідомлення про проблему та повертається статус 500:

```
        await WorkProtocol.create({
          userId: user._id,
          eventId: event._id,
          dateSendMsg: new Date()
        });

        console.log(`Email sent to ${user.email}`);
      } catch (error) {
        console.error(`Error sending email to ${user.email}:`, error);
      }
    }

    res.send('Результати успішно надіслані вибраним користувачам та залоговані');
  } catch (err) {
```

```
    console.error('Error sending result:', err);
    res.status(500).send('Error sending result');
  }
};
```

Цей фрагмент коду відповідає за створення запису в журналі роботи (WorkProtocol) після успішної відправки повідомлення користувачеві. У записі зберігається ідентифікатор користувача, події та дата відправки. У разі успіху в консоль виводиться підтвердження про відправлення e-mail, а в разі помилки реєструється відповідне повідомлення про помилку. Після завершення всіх операцій система повідомляє про успішну розсилку результатів користувачам.

3.6 Висновки до 3 розділу

У третьому розділі було розроблено бізнес-логіку програмного забезпечення для автоматизації процесу розсилки e-mail повідомлень та повідомлень у месенджери. Детально описано подієво-орієнтовану модель, яка забезпечує ефективне реагування на події, автоматизацію процесів генерації повідомлень та їх розсилки. Також були створені алгоритми автоматизації, що включають всі етапи обробки, від збору даних до логування результатів. Реалізовано функції налаштувань користувачів, які дозволяють персоналізувати підписки та спосіб доставки повідомлень, а також створено механізми логування для забезпечення прозорості роботи сервісу.

Розроблений код базується на сучасних технологіях, таких як Node.js та MongoDB, що забезпечує високу продуктивність, гнучкість і масштабованість системи. Перевагами коду є чітка структура, легкість інтеграції з зовнішніми сервісами та висока надійність завдяки розробленим тригерам і механізмам обробки помилок. Алгоритми та функції дозволяють

ефективно керувати підписками, генерувати контент для повідомлень та здійснювати автоматизовану розсилку з мінімальним втручанням користувача. Таким чином, виконана робота створює надійну основу для реалізації функціонального та адаптивного програмного забезпечення.

4 РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Розробка серверної частини мікросервісу

Серверна частина програмного забезпечення для автоматизації процесу розсилки e-mail повідомлень розроблена з використанням мови програмування JavaScript на платформі Node.js. Ця платформа забезпечує ефективне асинхронне виконання завдань, що є критично важливим для обробки великої кількості запитів одночасно. Основним завданням серверної частини є прийом і обробка даних, взаємодія з базою даних MongoDB, а також забезпечення коректної роботи бізнес-логіки розсилки повідомлень.

Архітектура серверної частини базується на принципах REST API, що дозволяє легко інтегрувати програму з іншими системами. Ключовими компонентами є модулі для обробки HTTP-запитів, управління сесіями користувачів, авторизації та аутентифікації, а також логування. Завдяки використанню бібліотек Express.js сервер має компактний і масштабований код, який забезпечує високу продуктивність і легкість у підтримці.

База даних MongoDB була обрана через її гнучкість у зберіганні даних. Документоорієнтована модель MongoDB дозволяє зберігати дані у форматі JSON, що добре інтегрується з JavaScript і Node.js. Серверна частина реалізує модулі для CRUD-операцій (створення, читання, оновлення та видалення даних), які використовуються для роботи з колекціями бази даних. Зокрема, до бази даних зберігається інформація про користувачів, події, які аналізуються, і шаблони e-mail повідомлень. Для оптимізації запитів застосовані індекси, що значно скорочують час пошуку даних.

Особливу увагу приділено механізмам аутентифікації користувачів. Для цього використовується токенізація на основі JSON Web Token (JWT). Це забезпечує високий рівень безпеки та спрощує авторизацію, дозволяючи

користувачам безпечно отримувати доступ до функціоналу системи. Також сервер реалізує захист від атак типу SQL-ін'єкцій (хоча база даних не SQL, вжиті заходи загалом убезпечують від небезпечних запитів) та XSS через використання бібліотек для валідації та очищення даних.

Одним із ключових аспектів розробки є забезпечення бізнес-логіки автоматизації розсилки повідомлень. Для цього реалізовано механізм планування завдань на основі бібліотеки node-schedule. Ця бібліотека дозволяє запускати функції у визначений час або періодично. Серверна частина отримує дані про події, які аналізуються, визначає відповідність умовам для відправки повідомлень і формує шаблони e-mail. Шаблони повідомлень містять як статичну, так і динамічну інформацію, яка генерується на основі даних із бази.

Взаємодія з поштовими сервісами здійснюється за допомогою бібліотеки Nodemailer. Цей інструмент підтримує різні поштові протоколи, такі як SMTP, що дозволяє надсилати повідомлення надійно та швидко. Nodemailer інтегрується із зовнішніми поштовими сервісами, такими як Gmail чи Outlook, що забезпечує додаткову гнучкість у виборі платформи для розсилки. Сервер забезпечує контроль статусу доставки повідомлень і зберігає цю інформацію у базі даних для подальшого аналізу.

Для моніторингу та відстеження стану роботи серверної частини впроваджено систему логування. Логи записуються у файли або у зовнішні сервіси на кшталт Logstash або Kibana для зручного аналізу. Також впроваджено механізми автоматичного сповіщення розробників про критичні помилки через інтеграцію зі службами, такими як Slack або Telegram.

Серверна частина також інтегрується з модулем аналітики подій. Цей модуль дозволяє отримувати інформацію про події з різних джерел, аналізувати її та приймати рішення щодо необхідності відправки повідомлень. Сервер обробляє отримані дані, виконує їхню валідацію та

передає результати до інших компонентів системи. Завдяки цьому забезпечується висока точність і релевантність розсилок.

Для підвищення надійності роботи сервера реалізовано механізми горизонтального масштабування через кластеризацію Node.js. Це дозволяє використовувати всі ядра процесора і обробляти більше запитів. Також впроваджено резервне копіювання даних із бази, що забезпечує їхню збереженість у разі збою.

Таким чином, серверна частина є критично важливим компонентом системи, що забезпечує її стабільну роботу, високу продуктивність і безпеку. Її архітектура та реалізація враховують сучасні вимоги до розробки програмного забезпечення, що робить систему ефективною для вирішення поставлених завдань.

4.2 Реалізація бази даних на MongoDB з використанням мови SQL

MongoDB це документоорієнтована база даних, яка ідеально підходить для сучасних програмних рішень завдяки своїй гнучкості та продуктивності. У контексті розробки програмного забезпечення для автоматизації розсилки e-mail повідомлень MongoDB забезпечує ефективне зберігання й обробку великих обсягів даних, таких як інформація про користувачів, події, шаблони повідомлень та історію розсилок. На відміну від традиційних реляційних баз даних, MongoDB використовує документи у форматі JSON, які легко інтегруються з мовою програмування JavaScript і дозволяють більш природно працювати з даними. Побудова структури бази даних починається з моделювання ключових колекцій, які відповідають основним сутностям системи:

– Users (Користувачі), Events (Події);

– UserSettings (Налаштування Користувачів);

– WorkProtocol (Журнал Роботи).

Кожна таблиця має свої специфічні атрибути та зв’язки. Опис таблиць та їх зв'язків:

– Users (Користувачі) Таблиця Users зберігає інформацію про користувачів системи. Основні атрибути включають: о _id: Унікальний ідентифікатор користувача. о name: Ім'я користувача. о email: Електронна адреса користувача. Користувачі можуть підписуватися на різні події, що зберігаються у таблиці Events, за допомогою таблиці UserSettings;

– Events (Події) Таблиця Events зберігає інформацію про події або аналітичні функції, доступні для користувачів. Основні атрибути включають: о _id: Унікальний ідентифікатор події. о name: Назва події. о descrb: Опис події. Ці події відображають можливі аналітичні функції для акцій, які користувачі можуть обрати;

– UserSettings (Налаштування Користувачів) Таблиця UserSettings є посередником між таблицями Users та Events і зберігає підписки користувачів на події. Основні атрибути включають: о _id: Унікальний ідентифікатор запису. о userId: Ідентифікатор користувача, який пов'язує запис з таблицею Users. о eventId: Ідентифікатор події, який пов'язує запис з таблицею Events. о uniqueKey: Унікальний ключ, створений на основі userId та eventId, що гарантує унікальність кожної підписки користувача на подію. Завдяки цій таблиці кожен користувач може мати кілька підписок на події, а кожна подія може бути пов’язана з кількома користувачами;

– WorkProtocol (Журнал Роботи) Таблиця WorkProtocol використовується для ведення журналу відправлених результатів аналізу. Основні атрибути включають: о _id: Унікальний ідентифікатор запису. о dateSendMsg: Дата та час відправлення повідомлення. о userId: Ідентифікатор користувача, якому було надіслано результат. о eventId: Ідентифікатор події, для якої було виконано аналіз. Ця таблиця дозволяє зберігати історію відправок результатів аналізу для кожного користувача та подій. Приклад

програмного коду ініціалізації таблиці Events з типовими подіями, приведено на рисунку 4.1.

```
const mongoose = require('mongoose');
const Event = require('../models/Event'); // шлях до моделі Event

async function addDefaultEvents() {
  const events = [
    { name: 'Daily Stock Change Analysis', description: 'Analyze daily stock
price changes.' },
    { name: 'Weekly Average Price Analysis', description: 'Analyze average
closing price over 7 days.' },
  ];

  try {
    for (const event of events) {
      const newEvent = new Event(event);
      await newEvent.save();
    }
    console.log('Default events added successfully.');
  } catch (error) {
    console.error('Error adding default events:', error);
  }
}

mongoose.connect('mongodb://localhost:27017/yourdbname', { useNewUrlParser:
true, useUnifiedTopology: true })
  .then(() => {
    addDefaultEvents().then(() => mongoose.disconnect());
  })
  .catch(err => console.error('Database connection error:', err));
```

Рисунок 4.1 – Фрагмент програмного коду ініціалізації таблиці Events

Аналогічно приведемо приклад моделі для таблиці UserSetting, яка представлена на рисунку 4.2.

```
const mongoose = require('mongoose');

const userSettingsSchema = new mongoose.Schema({
  userId: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },
  eventId: { type: mongoose.Schema.Types.ObjectId, ref: 'Event', required: true
},
  uniqueKey: { type: String, unique: true, required: true } // Унікальний ключ
для унікальності кожної підписки
});

const UserSettings = mongoose.model('UserSettings', userSettingsSchema);

module.exports = UserSettings;
```

Рисунок 4.2 – Фрагмент програмного коду моделі таблиці UserSetting

Ця модель представляє таблицю UserSettings, яка зв'язує користувачів з їх підписками на події.

У проєкті використовується MongoDB, яка є NoSQL базою даних, що організована як колекції документів у форматі JSON. Це дозволяє легко зберігати та управляти даними, але відсутність суворої схеми й нормалізації (як у SQL) дає більше гнучкості.

Якщо б ми використовували SQL-базу даних, ми б мали окремі таблиці для кожної сутності з чіткими зв'язками через первинні та зовнішні ключі.

Users (користувачі): таблиця users зберігає інформацію про користувачів, таких як ім'я та email. У SQL це була б таблиця з полями id (первинний ключ), name, та email. Створення таблиці у термінах SQL мови приведена на рисунку 4.3.

```
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    name VARCHAR(255),
    email VARCHAR(255) UNIQUE
);
```

Рисунок 4.3 – Створення таблиці user у термінах SQL мови

Events (події): таблиця events зберігає інформацію про події, такі як назва і опис. У SQL це була б таблиця з полями id (первинний ключ) та name. Створення таблиці у термінах SQL мови приведена на рисунку 4.4.

```
CREATE TABLE events (
    id SERIAL PRIMARY KEY,
    name VARCHAR(255),
    description TEXT
);
```

Рисунок 4.4 – Створення таблиці events у термінах SQL мови

userSettings (налаштування користувачів): таблиця, що зв'язує users та events, представляє підписки користувачів на певні події. У SQL ця таблиця

називається таблицею зв'язків (join table) і включає зовнішні ключі до users та events. Створення таблиці у термінах SQL мови приведена на рисунку 4.5.

```
CREATE TABLE userSettings (
    id SERIAL PRIMARY KEY,
    userId INT REFERENCES users(id),
    eventId INT REFERENCES events(id),
    UNIQUE(userId, eventId)
);
```

Рисунок 4.5 – Створення таблиці userSetting у термінах SQL мови

workProtocol (протокол роботи): таблиця, що зберігає лог відправки повідомлень користувачам щодо подій. У SQL вона б включала userId, eventId, та dateSendMsg (дату відправки повідомлення) і мала б зовнішні ключі для зв'язку з таблицями users та events. Створення таблиці у термінах SQL мови приведена на рисунку 4.6.

```
CREATE TABLE workProtocol (
    id SERIAL PRIMARY KEY,
    userId INT REFERENCES users(id),
    eventId INT REFERENCES events(id),
    dateSendMsg TIMESTAMP
);
```

Рисунок 4.6 – Створення таблиці workProtocol у термінах SQL мови

Приведемо приклади реалізацій роботи з даними у СУБД MongoDB та їх SQL-аналогії. Створення запису (INSERT), у MongoDB для створення нового користувача або події ми використовуємо метод save() або insertOne(). У SQL аналогом є оператор INSERT INTO. Приклад реалізацій команди INSERT INTO приведено на рисунку 4.7.

```
INSERT INTO users (name, email) VALUES ('Ім'я
користувача', 'email@example.com');
```

Рисунок 4.7 – Приклад реалізацій команди INSERT INTO

Об’єднання таблиць (JOIN), у MongoDB, коли ми хочемо отримати дані, що потребують зв’язку між таблицями, наприклад, які події підписані користувачем, ми використовуємо populate() або проводимо додаткові запити. У SQL для таких запитів використовують JOIN. Приклад реалізацій приведено на рисунку 4.8.

```
UserSettings.find({ userId: userId
}).populate('eventId');

SQL:

SELECT users.name, events.name
FROM userSettings

JOIN users ON userSettings.userId = users.id
JOIN events ON userSettings.eventId = events.id
WHERE users.id = 1;
```

Рисунок 4.8 – Приклад реалізацій команди JOIN

4.3 Розробка інтерфейсу користувача

Розробка інтерфейсу користувача для програмного забезпечення автоматизації процесу розсилки e-mail повідомлень за результатами аналізу подій є ключовим етапом, що забезпечує ефективну взаємодію між користувачем і системою. У цьому підрозділі розглядається створення інтерфейсу на основі клієнт-серверної архітектури із застосуванням HTTPS протоколу та мови програмування JavaScript. Такий підхід обрано через його здатність забезпечити високу гнучкість, масштабованість та безпеку системи. Клієнт-серверна модель дозволяє розділити функціональність між клієнтською частиною, яка відповідає за взаємодію з користувачем, та серверною, що виконує основну логіку і обробку даних. Це забезпечує оптимізацію ресурсів і можливість одночасного обслуговування великої кількості користувачів. Використання HTTPS протоколу гарантує захист переданої інформації від несанкціонованого доступу шляхом шифрування даних, що є критично важливим при обробці чутливих даних, таких як

результати аналізу подій та електронні адреси користувачів. JavaScript обрано як основну мову для розробки клієнтської частини завдяки її популярності, широким можливостям для створення динамічного контенту, а також здатності працювати безпосередньо у веб-браузері, що значно спрощує доступ до системи. Така архітектура забезпечує простоту розгортання, високу швидкість виконання запитів, а також інтеграцію з іншими компонентами програмного забезпечення. У результаті реалізований інтерфейс дозволить користувачам ефективно налаштовувати параметри розсилки, переглядати результати аналізу подій і взаємодіяти з системою у захищеному середовищі. Головна сторінка для програмного забезпечення автоматизації процесу розсилки e-mail повідомлень за результатами аналізу подій представлена на рисунку 4.9.

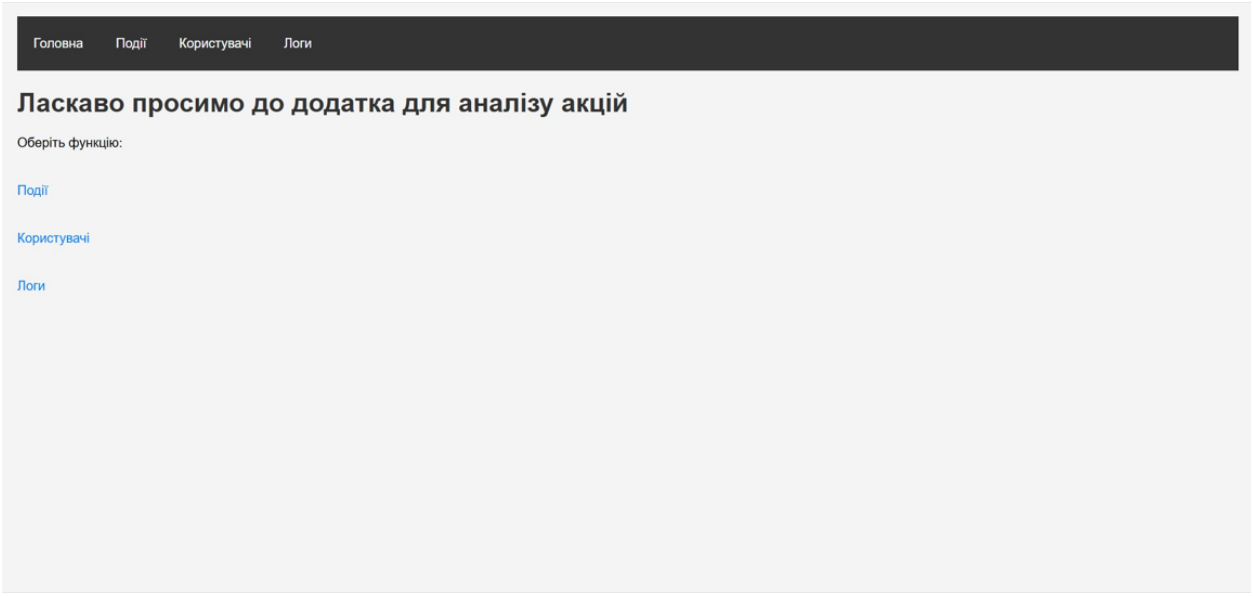


Рисунок 4.9 – Головна сторінка для програмного забезпечення автоматизації процесу розсилки e-mail повідомлень за результатами аналізу подій

Користувач може обрати наступні функцій: «Подій», «Користувачі» та «Логи». Функція «Подій», яка представлена на рисунку 4.10, дає можливість створить подію.

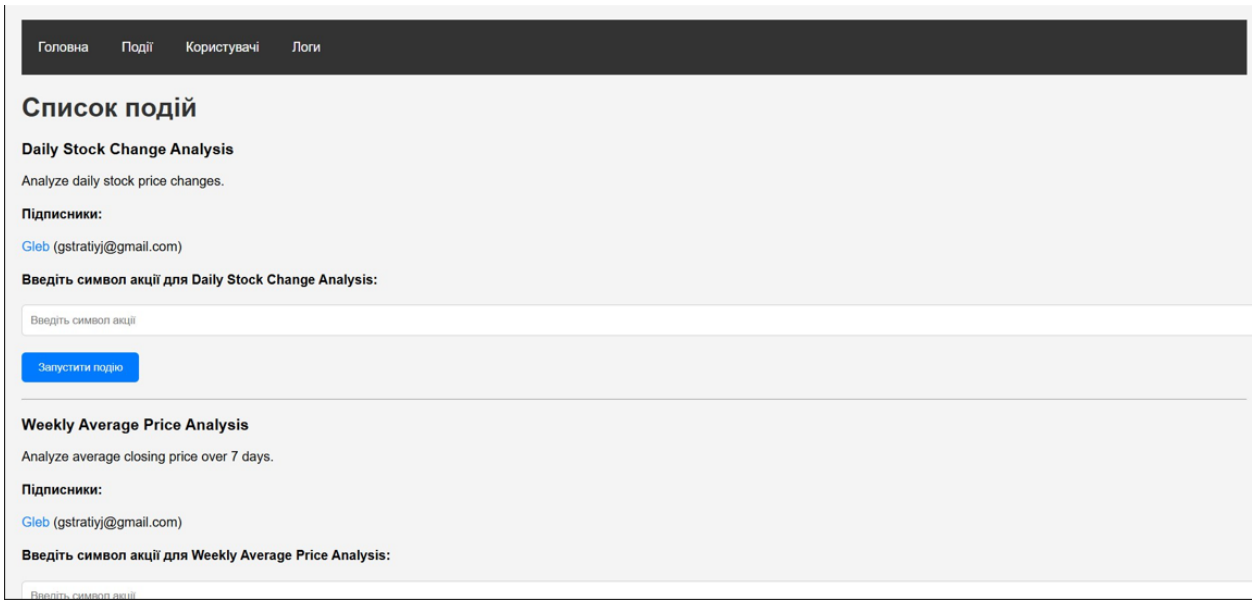


Рисунок 4.10 – Вікно функцій «Подій»

Для керування користувачами створеної подій, та редукування підписки на подій, був розроблено наступній інтерфейс «Користувачі», якій представлено на рисунку 4.11.

Рисунок 4.11 – Вікно функцій «Користувачі»

Для реалізації функцій адміністрування користувачами, реалізовано вікно, яке представлено на рисунку 4.12, що дає можливість додати нового користувача, видалити або редагувати його дані в програми.

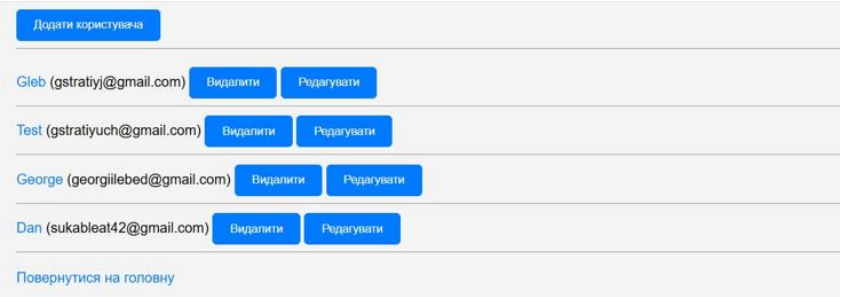
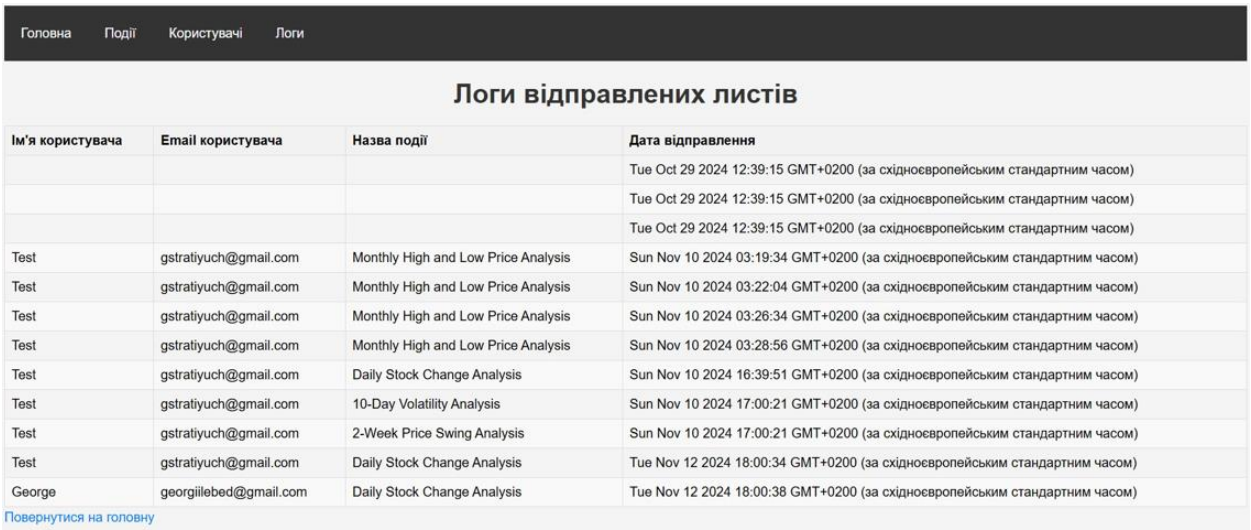


Рисунок 4.12 – Вікно адміністрування користувачів

Для загального аналізу відправлених подій, розроблено вікно «Логи відправлених подій», яке представлено на рисунку 4.13.



Ім'я користувача	Email користувача	Назва події	Дата відправлення
			Tue Oct 29 2024 12:39:15 GMT+0200 (за східноєвропейським стандартним часом)
			Tue Oct 29 2024 12:39:15 GMT+0200 (за східноєвропейським стандартним часом)
			Tue Oct 29 2024 12:39:15 GMT+0200 (за східноєвропейським стандартним часом)
Test	gstratiyuch@gmail.com	Monthly High and Low Price Analysis	Sun Nov 10 2024 03:19:34 GMT+0200 (за східноєвропейським стандартним часом)
Test	gstratiyuch@gmail.com	Monthly High and Low Price Analysis	Sun Nov 10 2024 03:22:04 GMT+0200 (за східноєвропейським стандартним часом)
Test	gstratiyuch@gmail.com	Monthly High and Low Price Analysis	Sun Nov 10 2024 03:26:34 GMT+0200 (за східноєвропейським стандартним часом)
Test	gstratiyuch@gmail.com	Monthly High and Low Price Analysis	Sun Nov 10 2024 03:28:56 GMT+0200 (за східноєвропейським стандартним часом)
Test	gstratiyuch@gmail.com	Daily Stock Change Analysis	Sun Nov 10 2024 16:39:51 GMT+0200 (за східноєвропейським стандартним часом)
Test	gstratiyuch@gmail.com	10-Day Volatility Analysis	Sun Nov 10 2024 17:00:21 GMT+0200 (за східноєвропейським стандартним часом)
Test	gstratiyuch@gmail.com	2-Week Price Swing Analysis	Sun Nov 10 2024 17:00:21 GMT+0200 (за східноєвропейським стандартним часом)
Test	gstratiyuch@gmail.com	Daily Stock Change Analysis	Tue Nov 12 2024 18:00:34 GMT+0200 (за східноєвропейським стандартним часом)
George	georgiilebed@gmail.com	Daily Stock Change Analysis	Tue Nov 12 2024 18:00:38 GMT+0200 (за східноєвропейським стандартним часом)

Рисунок 4.13 – Вікно «Логи відправлених подій»

Для зручності візуалізації та аналізу актуальної інформацій, пропонується представить данні у вигляді графіку з прогнозом, як показано на рисунку 4.14.

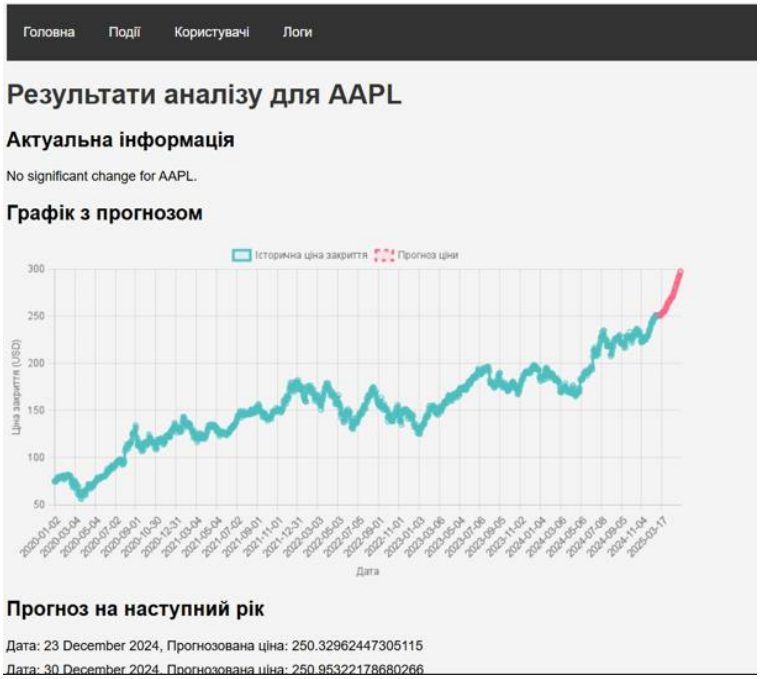


Рисунок 4.14 – Вікно «Результати аналізу»

Для створення автоматизованого процесу розсилки e-mail повідомлень за результатами аналізу подій, адміністратор обирає користувачів які отримають результат на пошту, як показано на рисунку 4.15

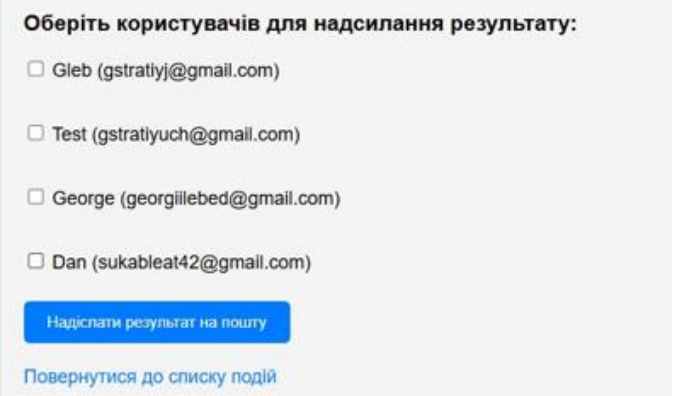


Рисунок 4.15 – Вікно «Вибору користувачів для надсилання результату на e-mail повідомлень»

Розроблений інтерфейс користувача має низку важливих переваг, що роблять його ефективним, зручним і надійним для виконання завдань автоматизації розсилки e-mail повідомлень. По-перше, використання клієнт-

серверної архітектури забезпечує високий рівень масштабованості, що дозволяє легко адаптувати систему до зростаючих обсягів обробки даних та кількості користувачів. Інтеграція з HTTPS протоколом додає значний рівень безпеки, оскільки всі передані дані шифруються, захищаючи їх від несанкціонованого доступу або модифікації. Крім того, завдяки використанню мови JavaScript у клієнтській частині забезпечується висока динамічність і інтерактивність, що робить роботу з інтерфейсом інтуїтивно зрозумілою навіть для некваліфікованих користувачів. Гнучкість JavaScript дозволяє реалізовувати складні функції, такі як динамічне оновлення контенту без перезавантаження сторінки, що значно підвищує зручність використання. Серверна частина, завдяки централізованій обробці запитів, забезпечує стабільну та ефективну роботу системи незалежно від технічних параметрів клієнтських пристроїв. Це рішення дозволяє створити сучасний інструмент для автоматизації, який поєднує швидкість, безпеку та простоту використання, що відповідає високим вимогам сучасних IT-рішень.

4.4 Оцінка продуктивності та надійності програмного забезпечення

Перший експеримент – перевірка продуктивності при високому навантаженні, великої кількості одночасних запитів для тестування серверної частини, бази даних і черги розсилки. Це дозволить оцінити, як система справляється з піковими навантаженнями. Для цього буде використовуватися наступні інстументи для тестування продуктивності, такими як Apache JMeter, Postman з колекціями запитів, або спеціальними бібліотеками на Node.js, такими як Artillery. Отримані результати першого експерименту представлені в таблиці 4.1, а також представлені на рисунку 4.16 у вигляді комбінованого графіку.

Таблиця 4.1 – Отримані результати при проведені першого експерименту

Кількість одночасних запитів	Середній час відповіді (мс)	Максимальний час відповіді (мс)	Відсоток успішних запитів (%)	CPU (%)	RAM (МБ)
100	150	250	100	25	300
200	180	400	99,5	40	350
400	300	750	98,7	60	420
600	500	1200	96,5	75	480
800	850	1800	93,8	85	550
1000	1200	2500	90,2	95	600

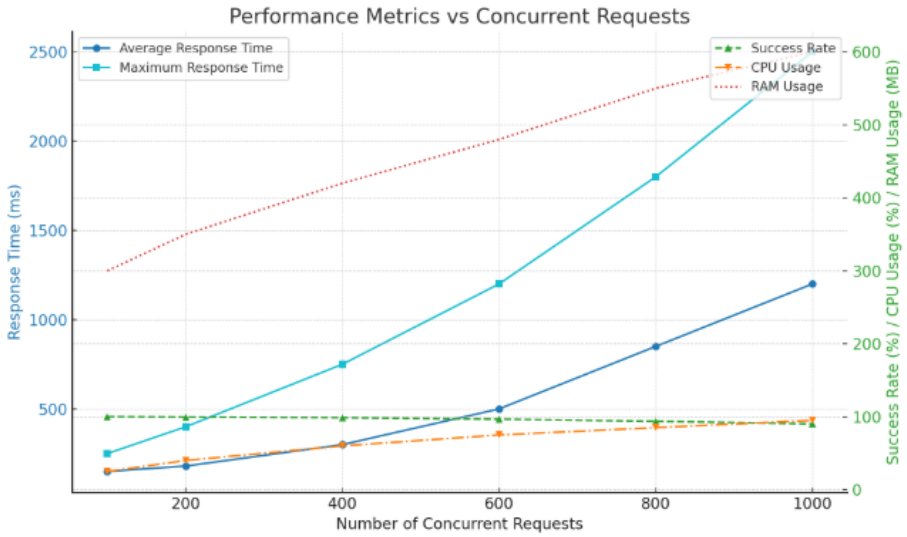


Рисунок 4.16 – Графік метрик продуктивності системи залежно від кількості одночасних запитів

На базі отриманих результатів першого експеримент у можна зробить наступні висновки, що середній час відповіді системи збільшується зі зростанням навантаження, досягаючи 1200 мс при максимальному рівні запитів, що свідчить про поступове зниження її швидкодії. Максимальний час відповіді демонструє, як система справляється з найскладнішими запитами, вказуючи на можливі "вузькі місця" у роботі. Відсоток успішних запитів починає знижуватися після перевищення порогу у 600 запитів, що вказує на перевантаження та необхідність оптимізації. Використання процесора зростає пропорційно навантаженню, досягаючи пікових значень

при 1000 запитах, що свідчить про повне завантаження серверних ресурсів. Обсяг використання оперативної пам'яті також поступово збільшується, але залишається в межах, що не викликають збоїв у роботі системи, демонструючи її стабільність навіть під значним навантаженням.

Другий експеримент направлено на аналіз затримок розсилки, тобто вимірювання часу між активацією події і фактичною відправкою повідомлення. Результати допоможуть виявити можливі "вузькі місця" в процесі автоматизації. Отримані результати другого експерименту приведені в таблиці 4.2, а також представлені у вигляді комбінованого графіку на рисунку 4.17.

Таблиця 4.2 – Отримані результати при проведенні другого експерименту

Подія	Час активації (с)	Затримка в черзі (с)	Час відправки повідомлення (с)	Загальна затримка (с)
Event A	0,5	1,1	0,8	2,3
Event B	1,0	1,3	1,0	3,1
Event C	1,3	1,8	1,1	4,4
Event B	1,8	2,4	1,3	5,8
Event E	2,1	3,1	1,5	7,1

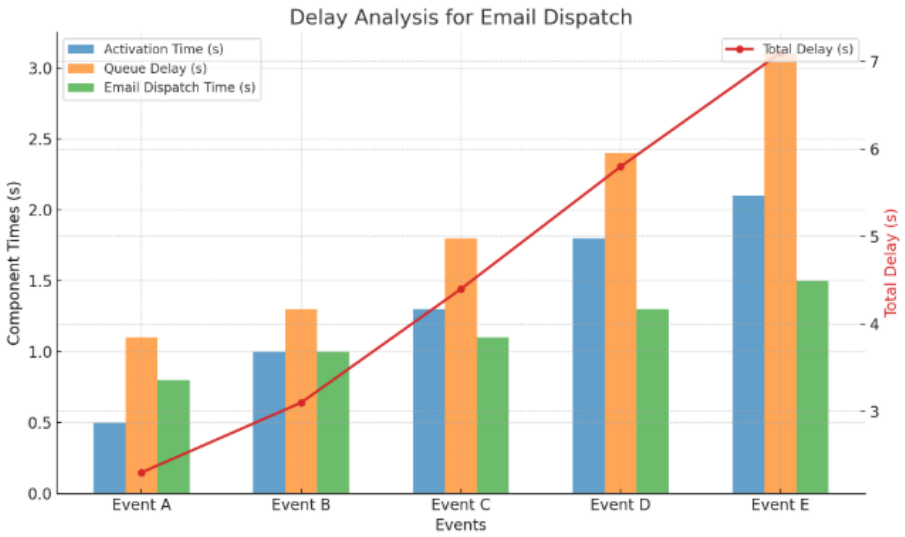


Рисунок 4.17 – Комбінований графік для аналізу затримок розсилки повідомлень

Аналіз отриманих результатів демонструє, що загальна затримка розсилки повідомлень поступово зростає зі збільшенням складності обробки подій. Час активації подій є найменшою складовою загальної затримки, але зростає у міру збільшення кількості оброблюваних даних або складності самої події. Основний внесок у загальну затримку припадає на час перебування повідомлень у черзі, який значно збільшується зі зростанням навантаження на систему. Це вказує на те, що черга може стати потенційним "вузьким місцем" при підвищенні обсягів оброблюваних запитів. Час відправки повідомлення залишається відносно стабільним і зростає лінійно, що свідчить про ефективність обраного поштового сервісу. Загальна тенденція до збільшення затримки із ростом складності або кількості подій вказує на необхідність оптимізації процесу управління чергою для зменшення впливу цього фактора на продуктивність системи. Загальний час розсилки залишається в межах допустимих для середнього навантаження, але при значному підвищенні інтенсивності роботи системи може знадобитися впровадження додаткових механізмів оптимізації.

4.5 Охорона праці

При розробці програмного забезпечення для автоматизації розсилки повідомлень значна увага приділяється забезпеченню комфортних і безпечних умов праці для програмістів та інших фахівців, які працюють у приміщенні. Основними аспектами охорони праці є організація робочого місця відповідно до ергономічних вимог, дотримання норм мікроклімату, вентиляції та рівня освітлення, що безпосередньо впливають на продуктивність праці й здоров'я співробітників.

Розрахунок освітлення для робочого приміщення проведено на основі нормативного рівня освітленості, який для офісних приміщень становить 300 лк – 500 лк. Для приміщення площею 20 м² із висотою стелі 2,7 м було

обрано світлодіодні лампи потужністю 18 Вт із світловим потоком 1800 лм кожна. За формулою розрахунку кількості ламп:

$$N = E \cdot S / F \cdot K, \quad (4.1)$$

де E = 400 лк (середній рівень освітленості);

S = 20 м² (площа приміщення);

F = 1800 лм (світловий потік лампи);

K = 0,8 (коефіцієнт запасу), було визначено, що для забезпечення нормативного рівня освітлення необхідно встановити 6 світлодіодних ламп.

Окрім освітлення, передбачено встановлення сучасної системи вентиляції для підтримання комфортної температури та рівня вологості повітря. Робочі місця оснащено ергономічними меблями, що дозволяють знизити навантаження на опорно-руховий апарат. Також рекомендовано використовувати монітори з фільтрами синього світла та забезпечувати регулярні перерви для уникнення зорової втоми. Комплексний підхід до питань охорони праці сприяє збереженню здоров'я співробітників і підвищенню ефективності їхньої роботи [20].

4.6 Висновки до 4 розділу

У четвертому розділі було реалізовано програмне забезпечення для автоматизації процесу розсилки email-повідомлень на основі аналізу подій. Розроблено серверну частину мікросервісу, що забезпечує стабільну роботу системи, інтеграцію з базою даних, а також виконання бізнес-логіки автоматизації розсилок. Реалізована база даних на основі MongoDB дозволила ефективно зберігати дані про події, налаштування користувачів і логи роботи системи, забезпечуючи високу продуктивність та масштабованість.

Також було створено інтерфейс користувача, що забезпечує зручний доступ до налаштувань системи та управління розсилками. Проведена оцінка продуктивності програмного забезпечення підтвердила його відповідність заданим вимогам щодо надійності, швидкості роботи та масштабованості. Таким чином, реалізована система демонструє ефективність у вирішенні задач автоматизації комунікацій, що підтверджує доцільність обраного підходу та використаних технологій.

ВИСНОВКИ

У кваліфікаційній роботі було розроблено комп’ютеризовану систему для автоматизації розсилки email-повідомлень та повідомлень у месенджери на основі аналізу подій. Провели аналіз існуючих рішень для автоматизації розсилки повідомлень. Описали архітектуру мікросервісу та вимоги до системи для підтримки email-розсилки та інтеграції з месенджерами. Визначили функціональні вимоги до програмного забезпечення та структуру даних для зберігання подій та налаштувань користувачів. Провели вибір мови програмування та середовища розробки БД. Розробили ER-діаграму бази даних та описали зв’язки між таблицями. Спроектували фізичну модель БД та описали бізнес-логіку роботи ПЗ. Розробили алгоритми автоматизації розсилки email-повідомлень та повідомлень у месенджери та функції обробки подій і тригерів для розсилки повідомлень. Провели налаштування конфігурацій користувачів для автоматизованих розсилок. Розробили функції логування та протоколювання роботи сервісу та серверну частину мікросервісу. Реалізували базу даних на MongoDB з використанням мови SQL та розробили інтерфейс користувача. Провели оцінку продуктивності та надійності програмного забезпечення. Отримані результати роботи можна віднести до Цілі сталого розвитку 9 «Промисловість, інновації та інфраструктура», а саме 9.4 «Сприяти прискореному розвитку високо- та середньовисокотехнологічних секторів переробної промисловості, які формуються на основі використання ланцюгів «освіта – наука – виробництво» та кластерного підходу за напрямами: розвиток інноваційної екосистеми; розвиток інформаційнотелекомунікаційних технологій (ІКТ); застосування ІКТ в АПК, енергетиці, транспорті та промисловості; високотехнологічне машинобудування; створення нових матеріалів; розвиток фармацевтичної та біоінженерної галузей» [21].

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. ДСТУ 3008-15. Документація. Звіти у сфері науки та техніки. структура та правила оформлення. – Введ. 2015-06-22. – К. Держстандарт України, 2017 – 29 с.

2. Невлюдов, І.Ш. Дипломне проектування для студентів усіх форм навчання спеціальностей 151 «Автоматизація та комп’ютерно-інтегровані технології» [Текст]: навч. посіб. / І.Ш. Невлюдов, А.О. Андрусевич, О.В. Токарєва, Г.В. Пономарьова. – Київ-58, пр. Космонавта Комарова, 1, 2016. – 320с.

3. Методичні вказівки з підготовки та захисту кваліфікаційної роботи здобувачами другого (магістерського) рівня вищої освіти спеціальності 174 Автоматизація та комп’ютерно-інтегровані технології, освітньо-професійних програм: «Автоматизоване управління технологічними процесами»; «Комп’ютерно-інтегровані технологічні процеси і виробництва»; «Комп’ютеризовані та робототехнічні системи» / Упоряд.: І. Ш. Невлюдов Р. В. Артюх В. В. Безкоровайний Н. П. Демська В. В. Євсєєв О. І. Филипенко О. М. Цимбал. Харків: ХНУРЕ, 2021. 55 с.

4. Лебідь Г. Автоматизація розсилки email-повідомлень та повідомлень в месенджери на основі аналізу подій. мікросервісний підхід / Лебідь Г., Іванов Л. // Інформаційні технології і автоматизація – 2024 : матеріали XVII міжнародної науково-практичної конференції, 31 жовтня-1 листопада 2024 р. – Одеса : Видавництво ОНТУ, 2024 р. – С. 286-289.

5. Мороз М. В. Розробка системи автоматизації для розсилки email повідомлень по результатам аналізу подій з автоматизованою розсилкою в месенджери : пояснювальна записка до атестаційної роботи здобувача вищої освіти на першому (бакалаврському) рівні, спеціальність 151 – Автоматизація

та комп`ютерно-інтегровані технології / М. В. Мороз ; М-во освіти і науки України, Харків. нац. ун-т радіоелектроніки. – Харків, 2024. – 51 с.

6. Трофімов О. Розробка чат-орієнтованої SaaS-платформи для удосконалення процесу прийняття бізнес-рішень. Магістерська робота (073 "Менеджмент"). Український католицький університет. Кафедра управління та організаційного розвитку. Львів: УКУ 2024, 59 с.

7. Фатко В.В. Модель системи виявлення і запобігання мережевих вторгнень на основі open-source стеку Suricata + ELKE : кваліф. робота магістра зі спеціальності 123 «Комп'ютерна інженерія» / В.В. Фатко. – Полтава : Національний університет імені Юрія Кондратюка, 2021. – 68 с.

8. Козуб Ю. Особливості розробки мультиплатформних застосунків на Kotlin / Ю. Козуб, Г. Козуб // Вісник Хмельницького національного університету. – 2023. – Том 1, №1 (317). – С. 224-229.

9. Satapathi, A., Mishra, A., Satapathi, A., & Mishra, A. (2021). OTP Mailer with Queue Storage Trigger and SendGrid Binding. Hands-on Azure Functions with C# Build Function as a Service (FaaS) Solutions, 63-90.

10. Surana, R., & Tiwari, P. (2024, March). Comprehensive Analysis of Email Marketing Tools for Enhancing Customer-Business Relationship. In International Conference On Emerging Trends In Expert Applications & Security (pp. 29-42). Singapore: Springer Nature Singapore.

11. Saralaya, S., Hehar, J., Pereira, A. S., Saldanha, A., & Fernandes, M. (2023, April). RetroMailer-An Email Marketing Campaign using Amazon SES. In 2023 2nd International Conference on Smart Technologies and Systems for Next Generation Computing (ICSTSN) (pp. 1-6). IEEE.

12. Imogen, P. V., Rakshana, R., Kaviya, M., & Sharmila, S. (2024). Real-Time Fire Surveillance with Machine Learning Twilio Integration. IRO Journal on Sustainable Wireless Systems, 6(2), 110-122.

13. Bobhate, R., & Malhotra, J. (2021). Slack Feedback Analyzer (SFBA). In Computational Methods and Data Engineering: Proceedings of ICMDE 2020, Volume 1 (pp. 397-405). Springer Singapore.

14. Плужник, О., & Дрок, П. (2023). Електронний документообіг в системі роботи сучасних бібліотек: ефективність, безпека та інновації. Society. Document. Communication. Соціум. Документ. Комунікація, (19), 198-212.

15. Тулашвілі, Ю. Й., Кошелюк, В. А., & Лапайчук, А. В. (2023, November). Моніторинг безпеки архітектури мікросервісів з використанням aws cloudwatch. In The 3 rd International scientific and practical conference “Current challenges of science and education”(November 13-15, 2023) MDPC Publishing, Berlin, Germany. 2023. 592 p. (p. 200).

16. Кіяшко Д. Г. Вивчення засобів та архітектури для взаємодії мікросервісів у вебзастосунках : пояснювальна записка до кваліфікаційної роботи здобувача вищої освіти на другому (магістерському) рівні, спеціальність 122 Комп’ютерні науки / Д. Г. Кіяшко ; М-во освіти і науки України, Харків. нац. ун-т радіоелектроніки. – Харків, 2024. – 60 с.

17. Киселевич, В. В., & Усата, О. Ю. (2024). Переваги та недоліки застосування мікросервісної архітектури на платформі. Net Core. Інформаційні технології: теорія і практика. I (VII) міжнародна науково-практична конференція здобувачів вищої освіти і молодих учених «Інформаційні технології: теорія і практика», 275-279.

18. Кібкало А. Є. Впровадження чат-ботів в маркетингову діяльність торговельних підприємств: кваліфікаційна робота на здобуття ступеня вищої освіти «Бакалавр»: спец. 075 - маркетинг; наук. кер. О. М. Прядко. Харків: ДБТУ, 2024. 70 с.

19. Масуд Б.Н. Вдосконалення маркетингової діяльності компанії "Genesis" на міжнародному ринку з використанням комплексного Інтернет-маркетингу. - Кваліфікаційна (дипломна) робота випускника освітнього

ступеня "магістр" за спеціальністю 075 "Маркетинг". - Національний авіаційний університет. - Київ, 2023. - 117 с.

20. Охорона праці на виробництві // Сайт GCC, 2024. URL: https://gc.ua/uk/oxorona-pracivofisivimogidorobochogomiscyaofisnogopracivnika/ (дата звернення: 10.12.2024).

21. Ціль 9. Промисловість, інновації та інфраструктура // Diia business, 2024. URL: https://business.diia.gov.ua/entrepreneur-handbook/item/cil_9_promislovist_innovaciyi_ta_infrastruktura (дата звернення: 15.12.2024).