Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти перший(бакалаврський)

Інтелектуальний Email-сервіс з AI-помічником

(тема)

Виконав:

здобувач 4 року навчання,

групи КІУКІ-21-2

Валерій ДМИТРЕНКО
(власне ім’я, прізвище)

Спеціальність

123 «Комп’ютерна інженерія»
(код і повна назва спеціальності)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма

Комп’ютерна інженерія
(повна назва освітньої програми)

Керівник: доц. Володимир ФЕДОРЧЕНКО
(посада, власне ім’я, прізвище)

Допускається до захисту

Зав. кафедри ЕОМ _____ Андрій КОВАЛЕНКО
(підпис) (власне ім’я, прізвище)

2025 р.

Харківський національний університет радіоелектроніки

Факультет комп'ютерної інженерії та управління

Кафедра електронних обчислювальних машин

Рівень вищої освіти перший(бакалаврський)

Спеціальність 123 «Комп'ютерна інженерія»
(код і повна назва)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Комп'ютерна інженерія
(повна назва)

ЗАТВЕРДЖУЮ:
Зав.
кафедри _____
(підпис)
"____" _____ 20__ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Дмитренку Валерію
(прізвище, ім'я, по батькові)

1. Тема роботи Інтелектуальний Email-сервіс з AI-помічником

затверджена наказом по університету від " 26 " травня 2024 р. № 424 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 17 червня 2025 р.

3. Вхідні дані до роботи 1) протоколи обміну повідомленнями (SMTP, IMAP/POP3);
2) інтерфейси та бібліотеки: JavaFX, Spring Boot, API хмарних сервісів;
3) архітектурні рішення щодо використання Docker, PostgreSQL.

4. Перелік питань, що потрібно опрацювати у роботі
1) аналіз існуючих рішень для електронної пошти, визначення їхніх переваг і недоліків;
2) формулювання функціональних і нефункціональних вимог до системи;
3) розробка архітектури програмного комплексу (інтерфейс, сервер, база даних);
4) реалізація захищеної аутентифікації, авторизації та управління користувачами;
5) інтеграція модуля штучного інтелекту для автоматизації роботи з листами;
6) створення зручного користувацького інтерфейсу на JavaFX;
7) організація процесу обробки вкладень та роботи з файлами;
8) налаштування розгортання додатку за допомогою Docker.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів)

Слайд-презентація – 15 слайдів

6. Консультанти розділів роботи (заповнюється за наявності консультантів згідно з наказом, зазначеним у п.1)

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

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Розгляд інтернет-тенденцій з приводу	27.05.25-28.05.25	+
	розроблення застосунків		
2	Розгляд теми пошти сервісу та вибір	29.05.25-30.05.25	+
	бібліотек		
3	Робота з кодом і доповнення функціоналу	31.05.25-05.06.25	+
4	Поліпшення функціональності та продуктивно-	06.06.25-08.06.25	+
	сті		
5	Додавання анімацій у додаток	09.06.25-10.06.25	+
6	Завершення розробки застосунку та робота з	11.06.25-16.06.25	+
	звітом		

Дата видачі завдання " 26 " травня 2025 р.

Здобувач _____
(підпис)

Керівник роботи _____ доц. Володимир ФЕДОРЧЕНКО
(підпис) (посада, власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 88 с., 17 рис., 11 табл., 12 дод., 21 джерел.

EMAIL, ПОШТОВИЙ СЕРВІС, КОРИСТУВАЧ, ВКЛАДЕННЯ, ШТУЧНИЙ ІНТЕЛЕКТ, API, КОНТЕЙНЕР, DOCKER, SPRING BOOT, JAVAFX, POSTGRESQL, АУТЕНТИФІКАЦІЯ, JWT, ХМАРНІ СЕРВІСИ

Мета кваліфікаційної роботи це розробка сучасного та безпечного сервісу електронної пошти «SeeYaa» з зрозумілим інтерфейсом, можливістю працювати з вкладеннями, інтеграцією штучного інтелекту для автоматизації звичних дій і захистом особистих даних користувачів.

Під час виконання роботи було спроєктовано і реалізовано багатошарову структуру для сервісу електронної пошти на основі JavaFX (для інтерфейсу користувача) та Spring Boot (для логіки бекенду і безпеки). Для збереження даних використано реляційну базу PostgreSQL. Додаток забезпечує безпечну реєстрацію і авторизацію користувачів із застосуванням сучасних підходів до захисту даних, а також інтегрує функції AI-асистента на базі хмарних моделей для аналізу текстів й створення підказок. Архітектура проєкту передбачає можливість легкого розгортання через контейнерну платформу Docker и подальшого масштабування з використанням хмарних серсісів У процесі розробки проведено комплексне тестування функціоналу, враховано продуктивність системи, а також усунено основні вузькі місця у використанні ресурсів і роботі з базою даних. Дослідження та отримані результати довели, що нові технології і способи працюють для створення зручних, безпечних, простих IT-рішень в електронній зв'язку.

ABSTRACT

Bachelor’s thesis 88 pages, 17 figures, 11 tables, 12 appendices, 21 sources.

EMAIL, EMAIL SERVICE, USER, ATTACHMENTS, ARTIFICIAL INTELLIGENCE, API, CONTAINER, DOCKER, SPRING BOOT, JAVAFX, POSTGRESQL, AUTHENTICATION, JWT, CLOUD SERVICES

The major goal of this thesis is to develop a modern and secure «SeeYaa» email service with a user-friendly interface, the ability to work with attachments, integration of artificial intelligence to automate common actions, and protection of users' personal data.

In order to the work, a multi-layered structure for the email service was designed and implemented based on Java FX (for the user interface) and Spring Boot (for backend logic and security). The PostgreSQL relational database is used for data storage. The application provides secure user registration and authorization using modern data protection approaches, as well as integrates the functions of an AI assistant based on cloud models for analyzing texts and creating subquestions. The project architecture provides for easy deployment via the Docker container platform and further scaling using cloud services During the development process, the team conducted comprehensive testing of the functionality, took into account system performance, and eliminated the main bottlenecks in resource utilization and database operations. The research and the results obtained have proven that new technologies and methods work to create convenient, secure, simple IT solutions in electronic communication.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

2FA – Two-Factor Authentication, двофакторна автентифікація — підвищення рівня безпеки при вході

AI, AI – Artificial Intelligence, штучний інтелект, галузь комп'ютерних наук, що розробляє системи для імітації людського мислення

API – Application Programming Interface, набір визначень для взаємодії програмних компонентів

API key – унікальний ідентифікатор для контролю доступу до API сторонніх сервісів

CI/CD – Continuous Integration / Continuous Delivery, безперервна інтеграція та доставка змін у коді

CRUD – Create, Read, Update, Delete; базові операції над даними

DB – Database, база даних

Docker – технологія контейнеризації для створення ізольованих середовищ запуску додатків

DTO – Data Transfer Object, об'єкт для передачі даних між частинами системи/рівнями архітектури

gRPC – Google Remote Procedure Call, фреймворк для високопродуктивної міжпроцесної взаємодії через мережу на основі HTTP/2 і Protocol Buffers

IDE – Integrated Development Environment, інструмент для розробки (наприклад, IntelliJ IDEA, Eclipse)

JavaFX – фреймворк для розробки графічних інтерфейсів користувача на Java

JWT – JSON Web Token, відкритий стандарт для безпечної передачі інформації у формі JSON-об'єктів

ORM – Object-Relational Mapping, технологія для зручної роботи з базами даних через об'єктно-орієнтовані моделі

REST – Representational State Transfer, архітектурний стиль для розробки веб-сервісів

Spring Boot – фреймворк для спрощеної розробки й запуску Java-додатків, з широкою підтримкою мікросервісної архітектури

TLS – Transport Layer Security, криптографічний протокол для захищеної передачі даних у мережі

UI – User інтерфейс, користувацький інтерфейс

ВСТУП

Сучасний розвиток інформаційних технологій зумовив появу численних інноваційних рішень у сфері комунікацій. Одним із важливих напрямків є автоматизація процесів обміну електронною поштою, що дозволяє підвищити ефективність комунікації як у бізнес-середовищі, так і у приватному секторі.

У цій праці розглядається створення системи поштової служби, яка поєднує нові технології для надання стабільного та безпечного обміну повідомленнями. Система зроблена з використанням JavaFX для створення простого в користуванні інтерфейсу, Spring Boot для управління серверною логікою та Docker для легкого розгортання програми у різних середах.

Актуальність обраної теми обумовлена зростаючою потребою в інтегрованих рішеннях, здатних забезпечити високий рівень автоматизації робочих процесів, а також необхідністю гарантувати безпеку обміну даними. Використання модульної архітектури дозволяє легко масштабувати систему та інтегрувати додаткові сервіси, що є важливим фактором у сучасних IT-рішеннях.

У роботі вирішено низку завдань, що включають аналіз існуючих аналогів, розробку технічних вимог до системи, створення прототипу та проведення комплексного тестування розробленого продукту. Запропонований підхід дозволяє не лише підвищити ефективність обробки електронної пошти, а й розширити функціональні можливості системи шляхом інтеграції додаткових сервісів, зокрема AI-асистента.

1 ЗАГАЛЬНА ХАРАКТЕРИСТИКА ПРОЕКТУ

1.1 Мета та завдання проекту

Електронна пошта залишається одним із основних каналів комунікації навіть у століття месенджерів: щодня у світі надсилається понад 347 мільярдів email-повідомлень (Radicati Group, 2023), але більшість із них проходить майже незахищено. Зростає обсяг спаму, а шкідливі вкладення завдають компаніям багатомільярдних збитків.

Проте навіть сучасні поштові сервіси часто або зосереджені лише на бізнесі, або не надають належної приватності й інтелектуальної допомоги користувачу. Саме тут виникає потреба в поштовому клієнті нового покоління – безпечному, швидкому, розумному і насправді зручному.

Мета проєкту – розробка настільного клієнта електронної пошти «SeeYaa», який:

a) гарантує високий рівень безпеки листування та захист даних за допомогою вбудованого шифрування й сучасної автентифікації (Spring Security);

b) дає користувачу інтуїтивний та легкий інтерфейс (JavaFX) для щоденного листування;

c) інтегрує AI-помічник, що допомагає формувати відповіді, аналізує вміст листів, підказує важливі дії;

d) використовує gRPC для пришвидшеної та ефективної передачі файлів і синхронізації даних між клієнтом і сервером, що дозволяє без затримок обробляти великі вкладення;

e) легко масштабується, працює на Windows та Linux, і може бути розгорнутий у хмарному середовищі.

Основні завдання:

1) створення адаптивного графічного інтерфейсу на JavaFX із

сучасними дизайнерськими підходами (темна/світла тема, drag-and-drop, фільтрація/сортування повідомлень);

2) розробка бекенд-архітектури на Spring Boot з використанням технологій Spring Security, gRPC та підключення до PostgreSQL;

3) впровадження швидкої багаторівневої системи авторизації і захисту персональних даних, підтримка двофакторної автентифікації;

4) реалізація функцій надсилання, отримання, групування листів, роботи з вкладеннями, а також інтелектуального (AI-асистента) аналізу листування;

5) інтеграція gRPC для оптимізованої відправки великих файлів та швидкої синхронізації пошти/вкладень;

6) забезпечення надійного зберігання, історії листування та гнучкого відновлення/видалення листів;

7) підтримка багатокористувацької адресної книги, шаблонів листів, розширеного пошуку та офлайн-режиму;

8) подальша можливість інтеграції з зовнішніми сервісами та масштабування в хмарному середовищі.

1.2 Аналіз існуючих рішень та обґрунтування вибору технологій

Попри велику кількість email-клієнтів, справжня конкуренція точиться лише в кількох нішах. Нижче приведені приклади email сервісів, які використовуються щоденно:

- Gmail – наймасовіший сервіс (понад 1,8 млрд користувачів), пропонує базові AI-фішки (Smart Reply, фільтрація спаму), але не має наскрізного шифрування й дуже залежить від хмарної екосистеми Google (рис. 1.1).

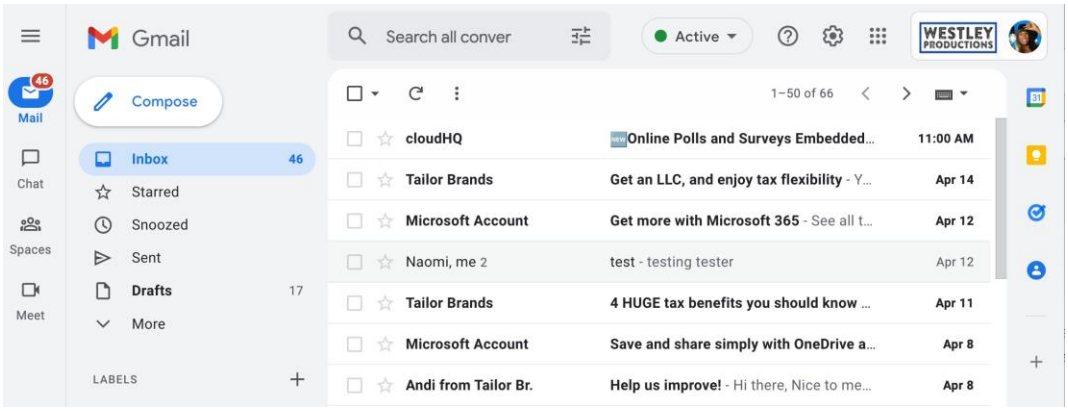

Рисунок 1.1 – Інерфейс використання Gmail

- Outlook – гігант корпоративного сегменту, де фокус зроблено на інтеграції з календарем і офісними застосунками, AI-функції з’явилися лише останні два роки, інтерфейс залишається перевантаженим (рис. 1.2).

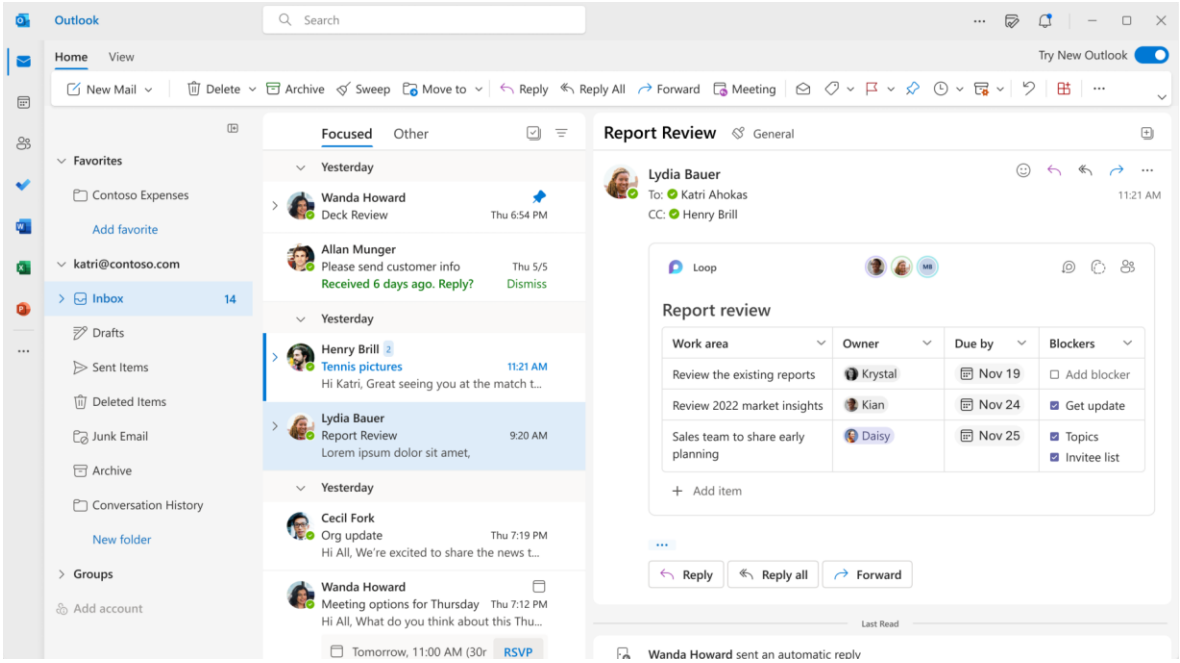

Рисунок 1.2 – Інерфейс використання Outlook

- Thunderbird – один із небагатьох open source-клієнтів, активно розвивається, легко масштабується функціями завдяки плагінам, однак інтеграція AI відсутня, а UI часто критикують за «застарілість» (рис. 1.3).

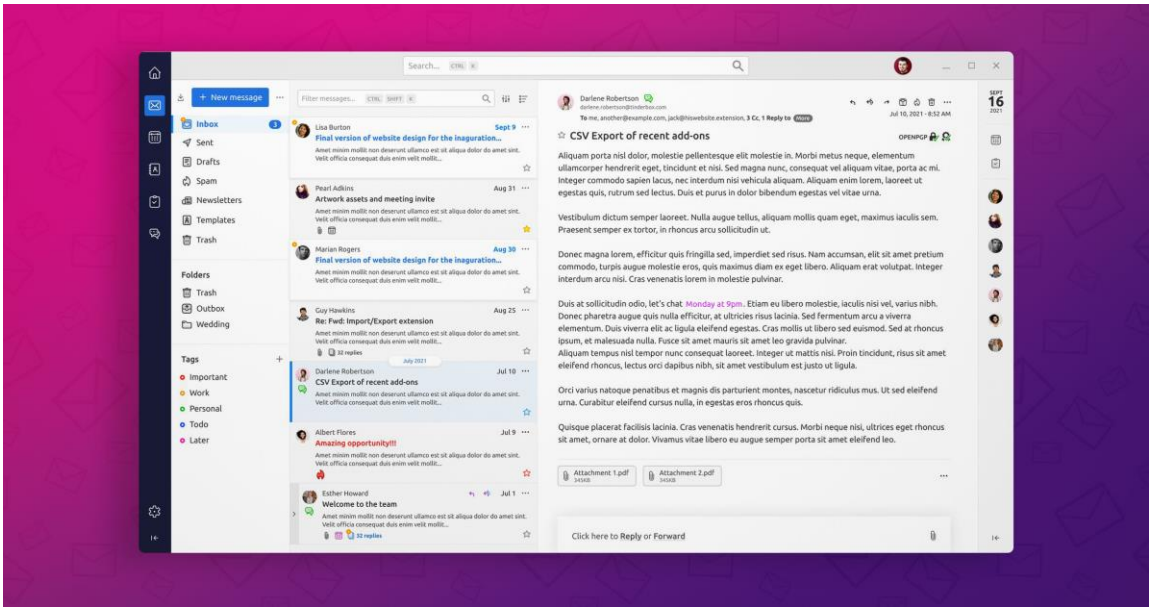

Рисунок 1.3 – Інерфейс використання Thunderbird

- ProtonMail – один із лідерів з безпеки та конфіденційності (E2E-шифрування default), проте клієнти для десктопа перебувають у beta (рис. 1.4), AI-можливостей (пошук, автозаповнення) нема.

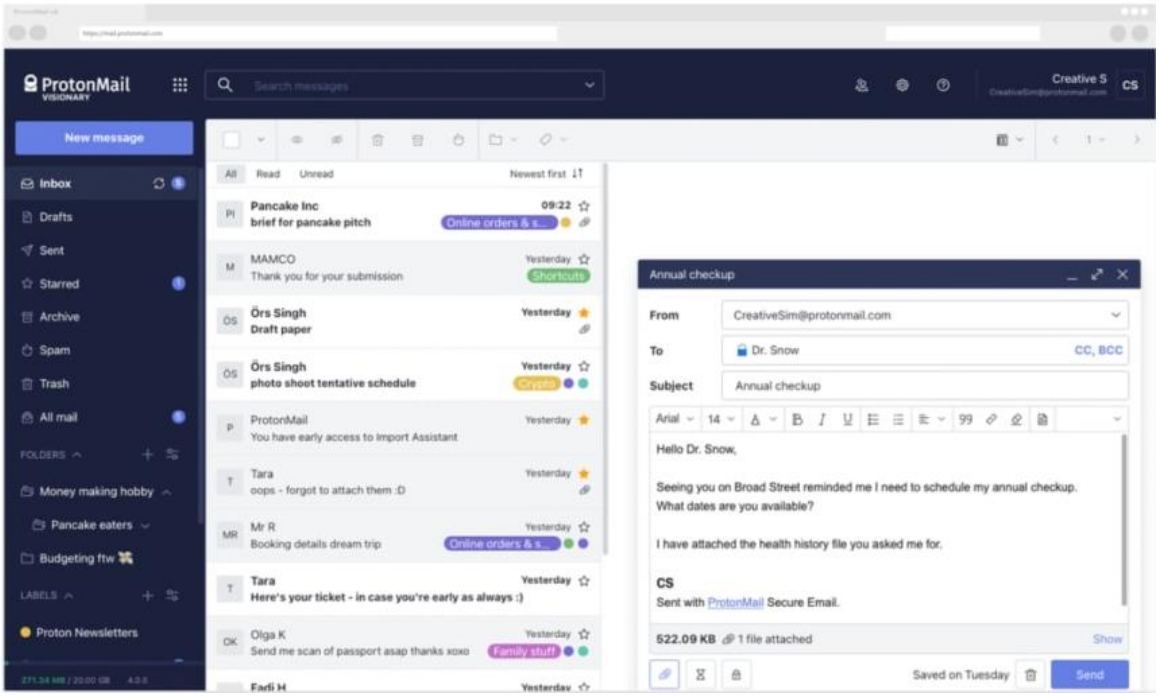

Рисунок 1.4 – Інерфейс використання ProtonMail

Отже, більшість рішень на ринку:

- не підтримують високопродуктивний обмін файлами в реальному часі (gRPC майже не використовується);
- не поєднують потужний AI-інструментарій та приватність;
- мають або складну архітектуру, або не піддаються кастомізації під конкретного користувача.

Обґрунтування вибору технологій:

1. основу даного проєкту становить стек Java 21 + JavaFX 21 для UI — це гарантує кросплатформеність, надійність та хорошу продуктивність;
2. бекенд реалізовано на Spring Boot 3 та Spring Security для забезпечення гнучкості, модульності, потужного захисту, легкого масштабування та швидкого розгортання;
3. зберігання даних виконується у PostgreSQL 17, яка забезпечує транзакційність, підтримку повнотекстового пошуку та гнучку роботу з вкладеннями;
4. gRPC (Google Remote Procedure Calls) використовується для швидкої передачі пошти, файлів-вкладень та синхронізації – завдяки цьому досягається низька затримка та висока пропускна здатність навіть на повільних мережах (на відміну від класичних REST API);
5. інтеграція AI реалізована завдяки використанню сучасних моделей (локальних чи через API), що робить роботу з листуванням ефективнішою і кориснішою.

1.3 Основні вимоги до системи

Під час розробки системи, було знайдено кілька функцій та нефункціональних потреб, які мають дати стабільну безпечну і дієву роботу додатку згідно його цілі.

а) Функціональні вимоги

1) реєстрація та авторизація користувачів із захистом облікових записів (використання Spring Security, шифрування паролів);

2) пошук листів: Можливість швидкого та ефективного пошуку листів за різними критеріями (відправник, отримувач, тема, ключові слова у тілі листа, дата, наявність вкладень);

3) фільтрація та сортування листів: Можливість фільтрувати листи (наприклад, непрочитані, з прапорцем, з вкладеннями) та сортувати їх за різними атрибутами (дата, відправник, розмір);

4) сповіщення: Система сповіщень про нові листи (можливо, з налаштуванням їх інтенсивності та типу);

5) можливість створення, надсилання та отримання електронних листів, включно з підтримкою вкладених файлів;

6) інтеграція AI-помічника для формування відповідей на листи та аналізу вмісту повідомлень;

7) функція переміщення листів між різними категоріями (Вхідні, Відправлені, Спам, Кошик) та шанс відновлення або останнього видалення;

8) збереження історії листування та відповіді в базі даних із урахуванням цілісності та взаємозв’язків;

9) інтерфейс користувача, реалізований з використанням JavaFX, що забезпечує доступність, інтуїтивність і зручність взаємодії з додатком.

б) Нефункціональні вимоги

1) безпека: застосування механізмів шифрування, автентифікації, обмеження доступу до ресурсів та захист від несанкціонованого втручання;

2) масштабованість: можливість подальшого розширення функціоналу та перенесення додатку в хмарне середовище;

3) надійність: збереження стабільної роботи додатку за різних умов, обробка виняткових ситуацій без втрати даних;

2) продуктивність: оптимізація запитів до бази даних, ефективне управління пам’яттю та асинхронна обробка завдань;

3) пошук листів: Можливість швидкого та ефективного пошуку листів за різними критеріями (відправник, отримувач, тема, ключові слова у тілі листа, дата, наявність вкладень);

4) фільтрація та сортування листів: Можливість фільтрувати листи (наприклад, непрочитані, з прапорцем, з вкладеннями) та сортувати їх за різними атрибутами (дата, відправник, розмір);

5) сповіщення: Система сповіщень про нові листи;

6) кросплатформеність: підтримка запуску на основних операційних системах (Windows, Linux).

Поставлені вимоги стали основою для створення архітектурного проєктування та технічної реалізації системи (рисунок 1.5), допомагаючи вийти до надійного, зручного а також новаторського сервісу електронна пошта.

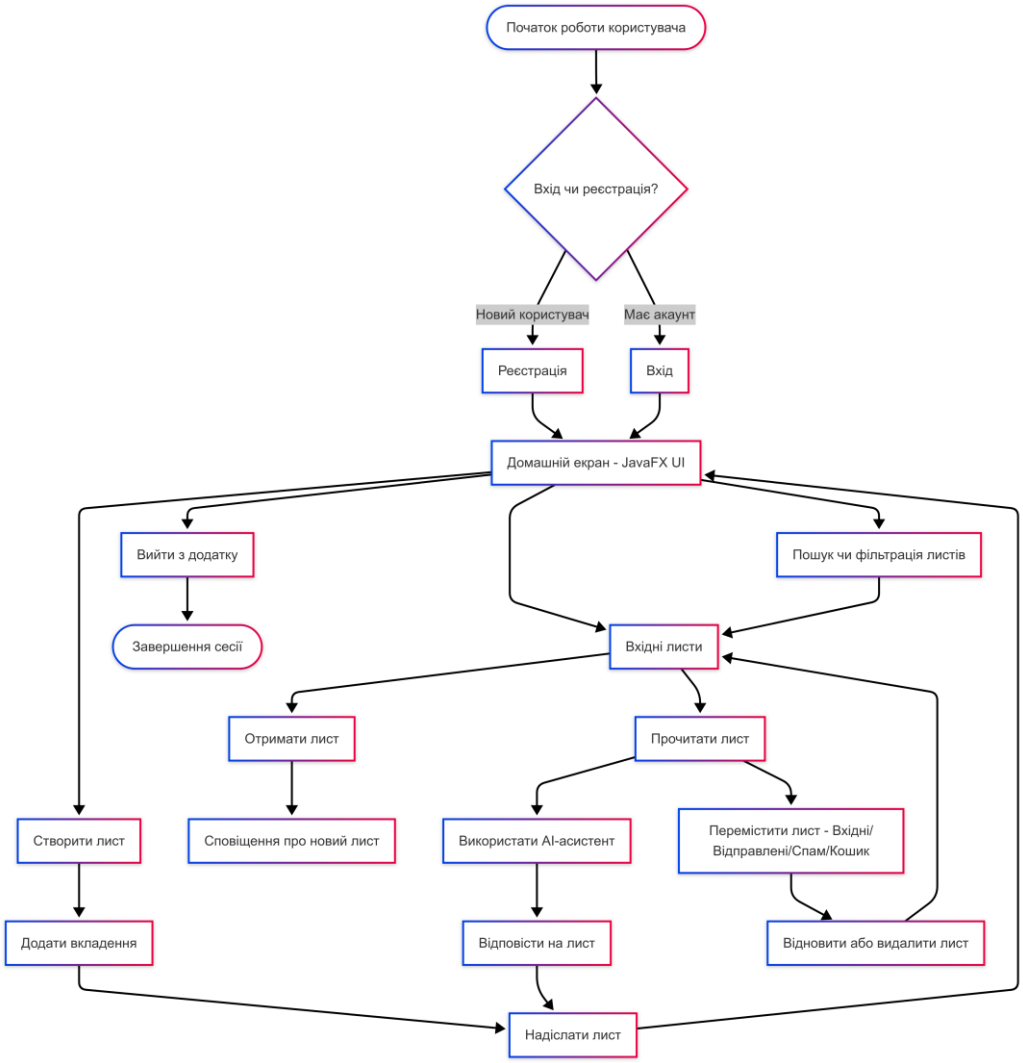

Рисунок 1.5 – Типовий сценарій використання користувачем

1.4 Системні вимоги

Мінімальні:

- Необхідний 64-бітний процесор та операційна система.
- Операційна система: Windows 10 (64-bit), Ubuntu 20.04 або macOS 11.
- Процесор: Intel Core i3-6100 / AMD Ryzen 3 1200.
- Оперативна пам’ять: 2 ГБ RAM.
- Графіка: інтегрована (Intel HD Graphics 530) або аналогічна дискретна відеокарта.
- Мережа: доступ до Інтернету.
- Дисковий простір: 350 МБ доступного місця.
- Додатково: встановлений Java JDK 24.

Рекомендовані:

- Операційна система: Windows 10/11 (64-bit), Ubuntu 22.04, macOS 12+.
- Процесор: Intel Core i5-8250U / AMD Ryzen 5 2500U або потужніший.
- Оперативна пам’ять: 8 ГБ RAM.
- Графіка: NVIDIA GeForce MX150 / AMD Radeon Vega 8 або краще.
- Дисковий простір: 1 ГБ вільного місця.

Пояснення: Завдяки використанню сучасних технологій – JavaFX для інтерфейсу, Spring Boot для серверної логіки, оптимізованого протоколу gRPC для швидкої обробки даних та асинхронного виконання завдань – додаток забезпечує стабільну і швидку роботу навіть на ноутбуках та комп’ютерах базового рівня. Ефективність архітектури дозволяє залишати низьке споживання ресурсів, швидке завантаження листів/файлів і плавність роботи інтерфейсу, що робить «SeeYaa» доступним для широкого кола користувачів.

2 АРХІТЕКТУРА ТА ТЕХНОЛОГІЇ РОЗРОБКИ

2.1 Загальна архітектура додатка

2.1.1 Архітектура бази даних

База даних системи «SeeYaa» реалізована за допомогою реляційної СУБД PostgreSQL. Вона забезпечує зберігання всіх основних сутностей додатку, включаючи користувачів, електронні листи, відповіді, вкладення, а також інформацію про переміщення листів. Архітектура побудована таким чином, щоб забезпечити цілісність даних, логічну зв’язність сутностей та ефективність пошуку й обробки інформації [4] (рисунок 2.1).

1. Користувачі (users)

Призначення: зберігання облікових даних та персональної інформації користувачів системи (табл 2.1).

Таблиця 2.1 – Архітектура сутності User

Поле	Тип	Призначення
id	`varchar(255)`	Унікальний ідентифікатор
email	`varchar(255)`	Email користувача
firstname	`varchar(255)`	Ім’я користувача
lastname	`varchar(255)`	Прізвище користувача
password	`varchar(255)`	Хешований пароль
username	`varchar(255)`	Унікальне ім’я користувача

2. Листи (letter)

Призначення: основна таблиця для зберігання листів, які надсилаються між користувачами (табл. 2.2).

Таблиця 2.2 – Архітектура сутності User

Поле	Тип	Призначення
id	`varchar(255)`	Унікальний ідентифікатор листа
active_letter	`boolean`	Прапорець активності листа
created_at	`timestamp(6)`	Дата створення
delete_time	`timestamp(6)`	Запланована дата видалення
text	`varchar(5000)`	Текст повідомлення
topic	`varchar(62)`	Тема листа
by_user_id	`varchar(255)`	ID користувача, який надіслав лист
to_user_id	`varchar(255)`	ID користувача, який отримує лист

3. Відповіді на листи (answer)

Призначення: зберігання відповідей на існуючі листи (табл. 2.3).

Таблиця 2.3 – Архітектура сутності Answer

Поле	Тип	Призначення
id	`varchar(255)`	Унікальний ідентифікатор відповіді
answer_text	`varchar(255)`	Текст відповіді
created_at	`timestamp(6)`	Дата створення відповіді
current_letter_id	`varchar(255)`	ID листа, до якого належить відповідь
user_by_id	`varchar(255)`	ID користувача, що відповів

4. Переміщення листів (moved_letter)

Призначення: реалізація логіки переміщення листів по "теках" (вхідні, спам тощо, табл. 2.4).

Таблиця 2.4 – Архітектура сутності MovedLetter

Поле	Тип	Призначення
id	varchar(255)	Унікальний ідентифікатор запису
type_of_letter	varchar(255)	Тип теки (INBOXES, SPAM, SENT, GARBAGE...)
will_delete_at	timestamp(6)	Дата запланованого автоматичного видалення
letter_id	varchar(255)	ID листа, який переміщується
moved_by_id	varchar(255)	ID користувача, який перемістив лист

5. Вкладення (files)

Призначення: зберігання метаданих файлів, прикріплених до листів (табл. 2.5).

Таблиця 2.5 – Архітектура сутності Files

Поле	Тип	Призначення
id	integer	Унікальний ідентифікатор файлу
data	oid	Посилання на двійкові дані файлу
name	varchar(255)	Назва файлу
size	bigint	Розмір файлу
type	varchar(255)	Тип файлу (VIDEO, IMAGE, DOCUMENT тощо)
file_id	varchar(255)	ID листа, до якого прикріплено файл

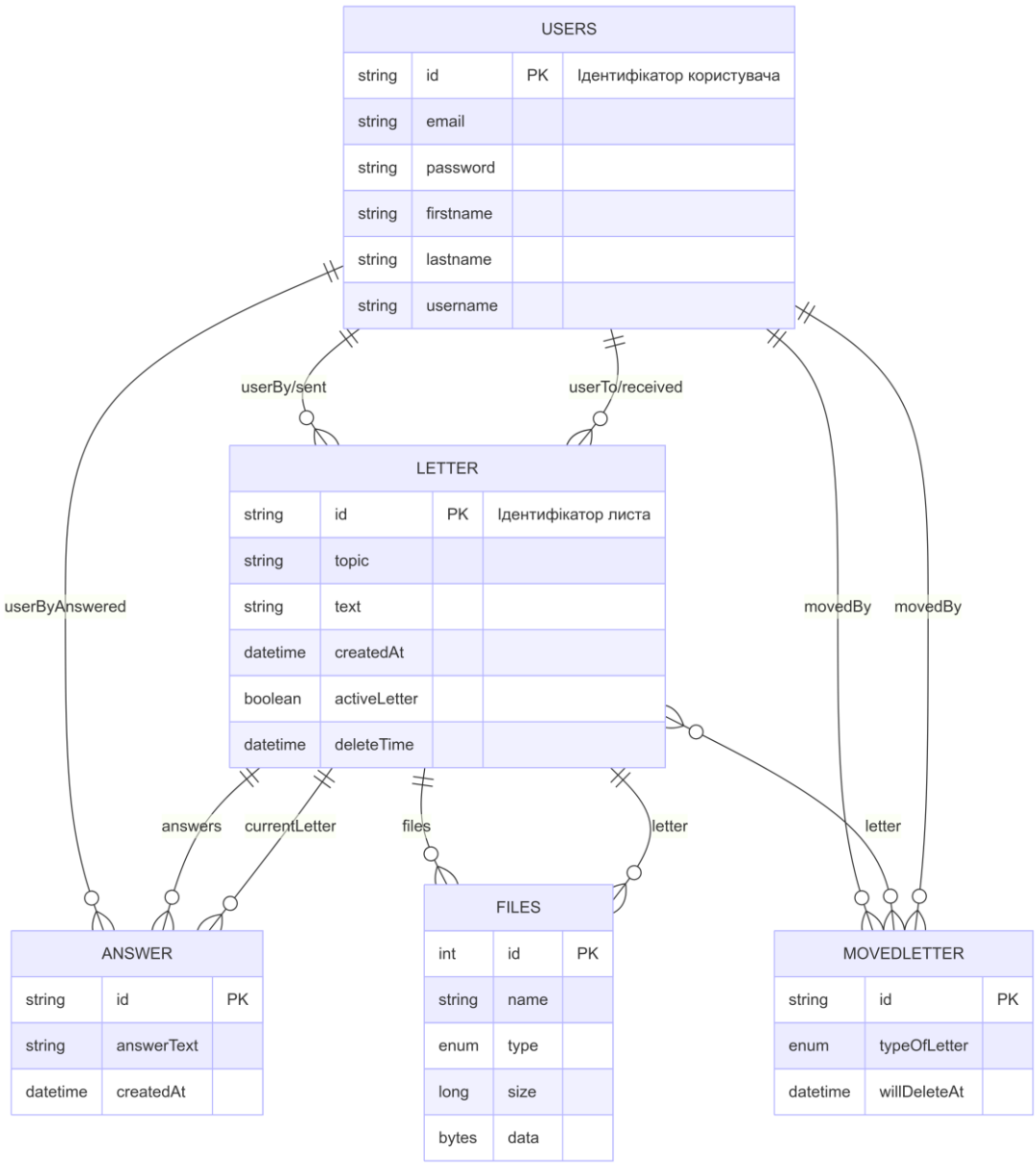

Рисунок 2.1 – Структура бази даних додатка

2.1.2 Архітектура програми

Архітектура програмного комплексу «SeeYaa» дотримується принципів об'єктно-орієнтованого програмування (ООП) та найкращих сучасних практик проектування програмного забезпечення, що забезпечує розширюваність, підтримуваність і повторне використання коду [5].

Модульність та шаблон багаторівневої архітектури (Multi-Layer Pattern) (рис. 2.2). Система побудована за принципами багаторівневої (Layered) архітектури:

- рівень представлення (Presentation Layer): контролери (Controller), які реалізують логіку взаємодії з користувачем, реагують на події, взаємодіють із сервісами та оновлюють інтерфейс користувача (JavaFX);

- сервісний рівень (Service Layer): класи-сервіси інкапсулюють бізнес-логіку (створення, оновлення листів, обробка запитів AI, управління користувачами), ізолюючи її від контролерів і від шару доступу до даних;

- рівень доступу до даних (Data Access Layer): репозиторії (Repository), які працюють напряму з джерелами даних (PostgreSQL через ORM), і моделі, що представляють структуру основних об'єктів;

- інфраструктурний рівень (Infrastructure Layer): допоміжні сервіси (наприклад, аутентифікація, авторизація, робота з файловим сховищем).

ООП, спадкування і шаблони проектування [5]:

- спадкування та абстракція: клас BaseController — це абстрактний базовий клас для всіх контролерів інтерфейсу, що містить загальні методи (handleEvents, initialize). Завдяки спадкуванню кожен спеціалізований контролер (наприклад, SendLetterController, EditController, AIController тощо) розширює базову функціональність, реалізуючи власну поведінку, що відповідає принципу DRY й забезпечує уніфікованість UI-логіки;

- делегування (Delegate Pattern) та ін'єкція залежностей (Dependency Injection);

- контролери взаємодіють із відповідними сервісами, делегуючи їм виконання бізнес-логіки. Це сприяє слабкому зв'язку між компонентами, покращує тестування та розширюваність системи;

- шаблон "Сервіс" (Service Pattern): основна бізнес-логіка, пов'язана з листами (LetterService), користувачами (UsersService), відповідями (AnswerService), переміщенням листів (MovedLetterService), AI (AIController) тощо, винесена в окремі класи-сервіси, що відповідає шаблону сервісного рівня та спрощує підтримку;

- шаблон "Репозиторій" (Repository Pattern): шар доступу до даних інкапсулює всі операції із збереженням й отриманням інформації. Це

дозволяє змінювати спосіб роботи з даними без впливу на верхні рівні (Presentation й Service), що відповідає принципу розділення обов'язків (Separation of Concerns, SoC);

- композиція (Composition): сервіси використовують (компонує) репозиторії для взаємодії з даними й інші сервіси для виконання комплексної логіки.

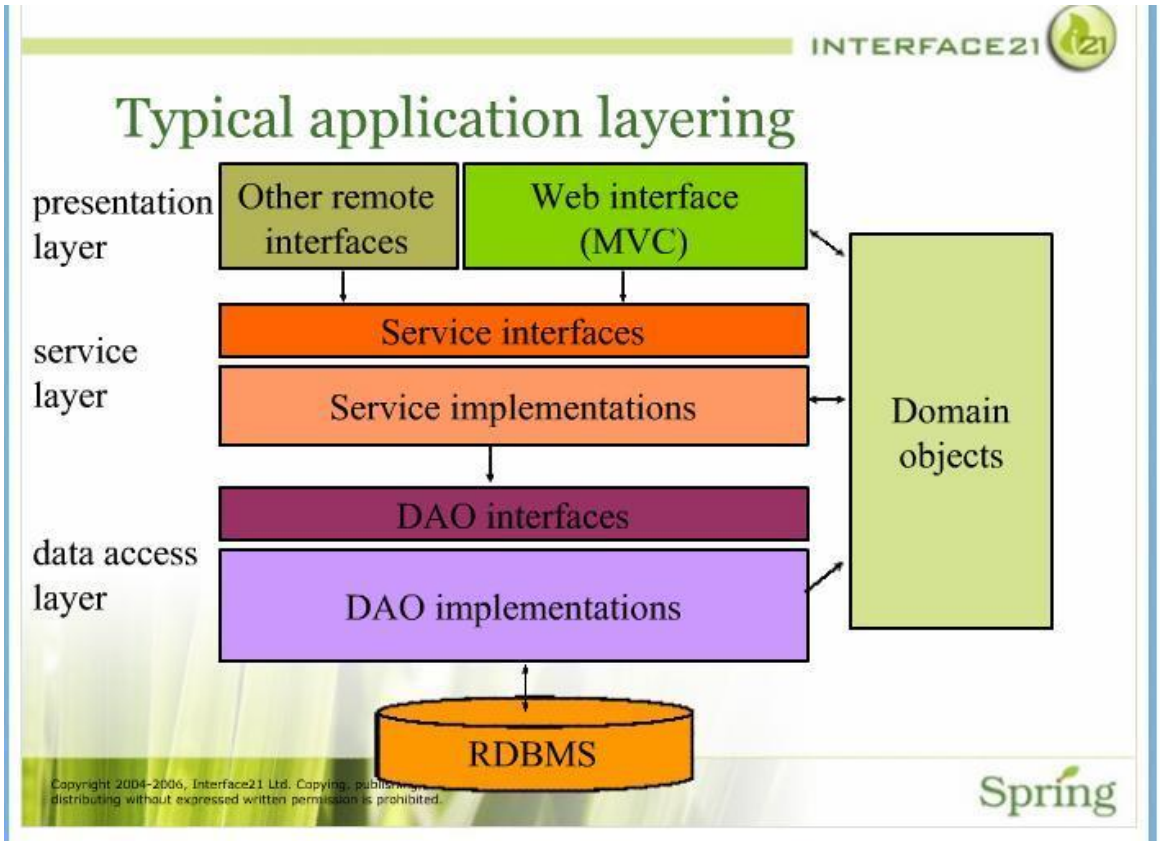

Рисунок 2.2 – Схема архітектури Multi-Layer Pattern

Відповідність принципам SOLID:

- Single Responsibility Principle: Кожен клас відповідає за одну чітко визначену задачу [5];

- Open/Closed Principle: Можливість розширення системи через наслідування чи додавання нових сервісів/контролерів без модифікації існуючих [5];

- Liskov Substitution Principle: Усі нащадки BaseController можна підставити там, де очікується базовий тип, не порушуючи логіки [5];

- Interface Segregation Principle: Контролери взаємодіють лише із тими сервісами/методами, які їм потрібні [5];

- Dependency Inversion Principle: Взаємодія контролерів із сервісами відбувається через інтерфейси та ін'єкцію залежностей, а не через жорстке створення об'єктів [5].

Завдяки такій архітектурі додаток легко масштабувати, додавати нові модулі – наприклад, контракти для інтеграції з іншими зовнішніми сервісами, або окремі контролери для нових UI-можливостей.

Слабкий зв'язок та розділення обов’язків дають змогу писати модульні та інтеграційні тести для різних шарів системи.

Висновок. Дотримання принципів ООП, шаблонів проектування (Layered, Service, Repository, Delegate, Dependency Injection) та SOLID гарантує, що архітектура програми «SeeYaa» (рис 2.3) є гнучкою, масштабованою та легкозмінною, а також слугуватиме взірцем для нових корпоративних чи комерційних IT-рішень.

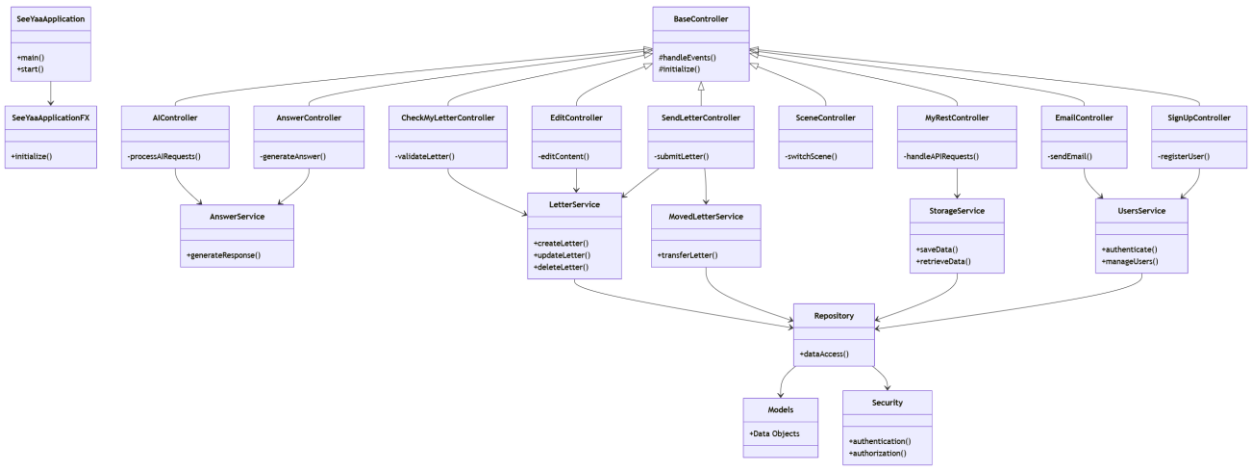

Рисунок 2.3 – Розподіл package у проекті

2.2 Використання JavaFX для UI

Елементи управління – це графічні елементи, які дозволяють користувачам взаємодіяти з додатком або веб-сайтом. До них відносяться кнопки, меню, повзунки, текстові поля, прапорці, перемикачі тощо [3]. У цьому прикладу ми розглянемо різні типи елементів керування інтерфейсом користувача JavaFX.

Розлянемо представлення трьох основних аспектів користувацького інтерфейсу:

- елементи інтерфейсу користувача – це основні візуальні елементи, які користувач в кінцевому підсумку бачить і з якими взаємодіє. JavaFX надає величезний список широко використовуваних і поширених елементів, від простих до складних [3];
- макети – вони визначають, як елементи інтерфейсу повинні бути організовані на екрані і надають остаточний вигляд графічному інтерфейсу користувача (GUI - Graphical User Interface) [3];
- поведінка – це події, які відбуваються, коли користувач взаємодіє з елементами інтерфейсу [3].

Для створення компонентів графічного інтерфейсу (елементів управління) JavaFX підтримує декілька елементів управління, таких як перемикач дат, текстове поле кнопки тощо. Ці елементи управління представлені різними класами пакета javafx.scene.control [3]. Було створено елемент управління, створено екземпляр його відповідного класу (лістинг 2.1).

Лістинг 2.1 – Створення UI файлів

```
public class FileActionImpl implements FileAction {
    private final StorageService storageService;

    public FileActionImpl(StorageService storageService) {
        this.storageService = storageService;
```

```
    }

    @Override
    public HBox createFillRow(Files files, Stage stage) {
        var hbox = new HBox();
        hbox.getStyleClass().add("file-row");
        hbox.setSpacing(10);
        hbox.setAlignment(Pos.CENTER_LEFT);
        hbox.setMinHeight(40);

        var fileNameLabel = new Label(files.getName());
        fileNameLabel.getStyleClass().add("file-name-label");
        fileNameLabel.setWrapText(true);
        HBox.setHgrow(fileNameLabel, Priority.ALWAYS);

        var fileSizeLabel = new
Label(formatFileSize(files.getSize()));
        fileSizeLabel.getStyleClass().add("file-size-label");

        var fileInfoBox = new VBox(5);
        fileInfoBox.getChildren().addAll(fileNameLabel,
fileSizeLabel);
        HBox.setHgrow(fileInfoBox, Priority.ALWAYS);

        final Trio<HBox, Button, ProgressIndicator> object =
createObjects();
        var downloadButton = object.second();
        var progressIndicator = object.third();
        var buttonBox = object.first();

        downloadButton.setOnAction(e -> {
            progressIndicator.setVisible(true);
            downloadButton.setVisible(false);

            Task<Void> downloadTask = new Task<>() {
                @Override
                protected Void call() {
                    downloadFile(files, stage);
                    return null;
                }
            };

            downloadTask.setOnSucceeded(event ->
                    Platform.runLater(() -> {
                        progressIndicator.setVisible(false);
                        downloadButton.setVisible(true);
                    }));

            downloadTask.setOnFailed(event ->
                    Platform.runLater(() -> {
                        progressIndicator.setVisible(false);
                        downloadButton.setVisible(true);
                        showAlert(Alert.AlertType.ERROR,
```

```
"Download Failed",
                                    "Error: " +
downloadTask.getException().getMessage());
                    }));

            new Thread(downloadTask).start();
        });

        hbox.setOnMouseEntered(e -> {
            var pt = new ParallelTransition();

            var scale = new
ScaleTransition(Duration.millis(200), hbox);
            scale.setToX(1.02);
            scale.setToY(1.02);

            var fade = new FadeTransition(Duration.millis(200),
downloadButton);
            fade.setToValue(0.8);

            pt.getChildren().addAll(scale, fade);
            pt.play();
        });

        hbox.setOnMouseExited(e -> {
            var pt = new ParallelTransition();

            var scale = new
ScaleTransition(Duration.millis(200), hbox);
            scale.setToX(1);
            scale.setToY(1);

            var fade = new FadeTransition(Duration.millis(200),
downloadButton);
            fade.setToValue(1);

            pt.getChildren().addAll(scale, fade);
            pt.play();
        });

        hbox.getChildren().addAll(fileInfoBox, buttonBox);
        return hbox;
    }
```

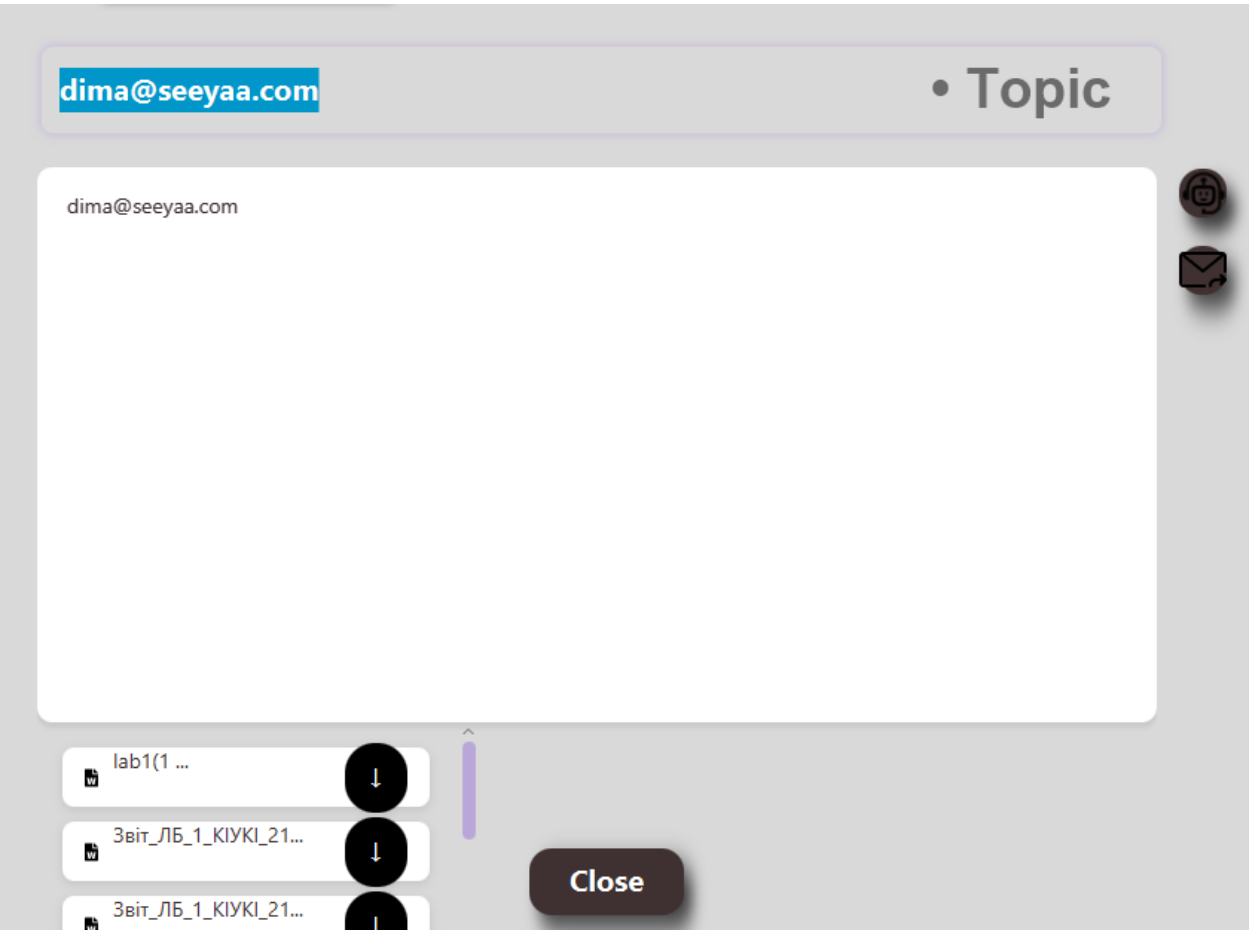

Рисунок 2.4 – Вивід програми

Інтерфейс користувача (рис. 2.4) відображає перелік файлів, які були додані та відправлені користувачем разом із листом. Інформація про вкладення автоматично отримується з бази даних: кожному файлу присвоюється унікальний ідентифікатор (id) та відображається його назва. Це рішення спрощує пошук і вибір конкретного файлу для завантаження чи перегляду, підвищуючи зручність роботи з вкладеннями в додатку.

2.3 Інтеграція Spring Boot

Spring Boot – це популярний фреймворк на базі Java, який спрощує створення веб-додатків і мікросервісів, забезпечуючи готову до використання інфраструктуру. Його основна перевага – зменшення кількості конфігурацій, необхідних для запуску проєкту. Spring Boot дозволяє швидко створювати production-ready додатки з мінімальними зусиллями [8].

Серед ключових переваг Spring Boot:

- Автоконфігурація: автоматично налаштовує залежності, базу даних, сервер тощо.

- Вбудований веб-сервер (Tomcat, Jetty): не потрібно окремо налаштовувати сервер – додаток одразу можна запускати як standalone.

- Spring Security: потужна система аутентифікації та авторизації.

- Підтримка REST API: легко створюються контролери для обробки запитів.

- Інтеграція з базами даних: підтримка JPA, Hibernate, а також можливість використання репозиторіїв через інтерфейси (Spring Data).

- Мікросервісна архітектура: чудово підходить для побудови розподілених систем.

У проєкті «SeeYaa» Spring Boot відіграє роль бекенд ядра, яке обробляє логіку програми, забезпечує взаємодію з базою даних, автентифікацію користувачів, API для обробки листів, файлів, повідомлень, а також використовується для інтеграції з AI-сервісом.

Фрейморк Spring Boot інтегрований з JavaFX. Для забезпечення коректної спільної роботи двох фреймворків – Spring Boot і JavaFX, необхідно реалізувати спеціальний механізм інтеграції. З цією метою створюються окремі стартові класи: один відповідає за ініціалізацію бекенд-складової на основі Spring Boot, інший – запускає графічний інтерфейс користувача, розроблений за допомогою JavaFX (лістинг 2.2).

Лістинг 2.2 – Метод зв’зку Spring boot з JavaFX

```
@SpringBootApplication
@EnableScheduling
public class SeeYaaApplication {

    public static void main(String[] args) {
        Application.launch(SeeYaaApplicationFX.class, args);
    }

}
```

```
public class SeeYaaApplicationFX extends
javafx.application.Application {

    private ConfigurableApplicationContext springContext;

    static {

SecurityContextHolder.setStrategyName(SecurityContextHolder.MODE
_INHERITABLETHREADLOCAL);
    }

    @Override
    public void init() {
        springContext = new
SpringApplicationBuilder(SeeYaaApplication.class).run();
    }

    @Override
    public void start(Stage stage) throws Exception {
        springContext.publishEvent(new StageReadyEvent(stage));

        FXMLLoader fxmlLoader = new
FXMLLoader(getClass().getResource("controller/login.fxml"));
        fxmlLoader.setControllerFactory(springContext::getBean);
        String css =
Objects.requireNonNull(this.getClass().getResource("controller/s
tatic/login.css")).toExternalForm();
        Scene scene = new Scene(fxmlLoader.load());
        scene.getStylesheets().add(css);
        stage.setScene(scene);
        stage.setTitle("SeeYaa");
        stage.show();
    }

    @Override
    public void stop() {
        springContext.close();
        Platform.exit();
    }

    public static class StageReadyEvent extends ApplicationEvent
{
        public StageReadyEvent(Stage stage) {
            super(stage);
        }
        public Stage getStage() {
            return ((Stage) getSource());
        }
    }
}
```

Після цього два фрейморка запрацюють разом та можна робити логіку. У додатку Spring Boot використовується для взаємодії з базою даних та зберігання файлів у вигляді бітів, тому що PostgreSQL не зберігає сам файл, а треба розробити логіку для стріму файлу (лістинг 2.3).

Лістинг 2.3 – Зберігання файлу у вигляді бітів

```
Task<Void> uploadTask = new Task<>() {
    @Override
    protected Void call() {
        LetterRequestDto request = new LetterRequestDto(
                text.getText(),
                topic.getText(),
                toWhom.getText(),
                securityService.getCurrentUserEmail());
        final var savedLetter =
letterServiceImpl.sendLetter(request);

        for (File file : selectedFiles) {
            MultipartFile multipartFile = new
PathMultipartFile(file);
            storageServiceImpl.uploadFile(multipartFile,
savedLetter.id());
        }
        return null;
    }
};

@Override
@PreAuthorize("hasRole('ROLE_USER')")
public void uploadFile(MultipartFile file, String letterId) {
    try {
        final var letter = letterRepo.findById(letterId)
                .orElseThrow(() -> new NotFoundException("Letter
not found with id: " + letterId));

        byte[] fileData;
        try (InputStream inputStream = file.getInputStream()) {
            fileData = inputStream.readAllBytes();
        }

        filesRepo.save(Files.builder()
                .name(file.getOriginalFilename())

.type(FileType.fromFileExtension(file.getOriginalFilename()))
                .size(file.getSize())
                .data(fileData)
                .letter(letter)
                .build());
```

```
    } catch (IOException e) {
        throw new StorageException("Failed to upload file", e);
    }
}

@Override
@PreAuthorize("hasRole('ROLE_USER')")
public InputStream downloadFile(String fileId) {
    Integer id = Integer.parseInt(fileId);
    return filesRepo.findById(id)
            .map(file -> new
ByteArrayInputStream(file.getData()))
            .orElseThrow(() -> new NotFoundException("File not
found with id: " + fileId));
}
```

2.4 Аутентифікація та безпека

Для підвищення безпеки обміну обліковими даними між клієнтським додатком (JavaFX) і сервером (Spring Boot) у процесі авторизації реалізовано механізм асиметричного шифрування на основі RSA. Такий підхід дозволяє мінімізувати ризик перехоплення чутливої інформації при передачі через мережу, навіть у випадку загрози маніпуляції каналом зв'язку.

Основні етапи авторизації (рис 2.5):

- генерація ключів на сервері - при запуску серверної частини генерується пара RSA-ключів (public/private). Публічний ключ зберігається на сервері та видається клієнтському додатку на вимогу;

- отримання публічного ключа на клієнті - JavaFX-клієнт під час ініціалізації отримує публічний ключ від сервера (наприклад, через окремий gRPC-метод);

- шифрування облікових даних на клієнті - перед введенням логіна та пароля користувачем, ці дані об’єднуються, шифруються з використанням RSA-публічного ключа і передаються на сервер у зашифрованому вигляді;

- дешифрування на сервері - на боці бекенду Spring Boot отримані з клієнта дані розшифровуються за допомогою приватного RSA-ключа;

- аутентифікація через AuthenticationManager - розшифровані логін і пароль передаються далі стандартним механізмам Spring Security.

Створюється об’єкт UsernamePasswordAuthenticationToken, який обробляється AuthenticationManager. У разі успішної верифікації користувач зберігається у SecurityContextHolder для подальших дій.

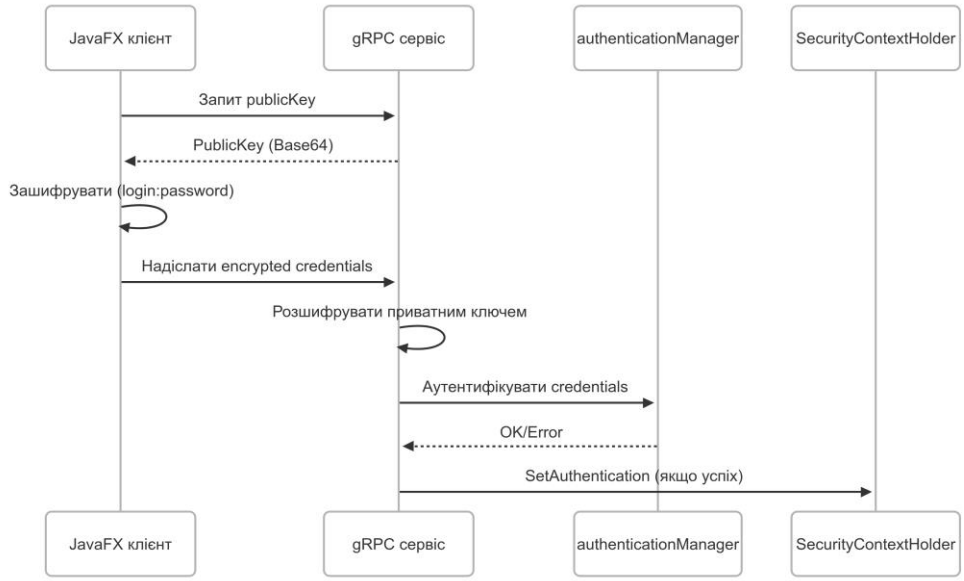

Рисунок 2.5 – Схема роботи RSA

Для авторизованого користувача контекст автентифікації дозволяє використовувати рольову перевірку (наприклад, через @PreAuthorize) і виконання дозволених дій у gRPC-сервісах додатку до завершення сесії.

Переваги такого підходу:

- відсутність передачі пароля у відкритому вигляді між клієнтом та сервером;
- сумісність із стандартною структурою Spring Security та можливість подальшого масштабування (додавання двофакторної автентифікації, зміна схеми сесії тощо);
- логічне відокремлення шифрування даних і верифікації користувача для гнучкого адміністрування безпеки.

2.5 Використання gRPC з Spring Boot (org.springframework.grpc): сучасний підхід до високопродуктивних мікросервісів

У новій версії додатку було вирішено використати сучасний підхід до комунікації між компонентами системи – gRPC у поєднанні зі Spring Boot. Для цього було використано бібліотеку org.springframework.grpc, яка офіційно розвивається спільнотою Spring.

gRPC – це відкритий RPC-фреймворк від Google, який використовує HTTP/2 для взаємодії та Protocol Buffers (protobuf) для серіалізації повідомлень [21]. У результаті це дає:

- швидкість передавання даних – за рахунок бінарного формату protobuf, обсяг даних в мережі мінімізований порівняно з JSON або XML;
- можливість потокової передачі (Streaming) – можна передавати великі файли або великі обсяги даних частинами, що особливо корисно для email-сервісу з вкладеннями;
- багатомовність – gRPC підтримує десятки мов програмування, що у майбутньому спрощує розширення проєкту;
- вбудована підтримка безпеки – завдяки HTTP/2 простіше інтегрувати TLS/SSL, а також гнучко підключати існуючі засоби безпеки Spring Security;
- контроль API контракту – використання .proto-файлів дає єдиний контракт між клієнтом та сервером – це допомагає уникати помилок сумісності;

org.springframework.grpc – офіційна інтеграція.

Бібліотека org.springframework.grpc дозволяє "безшовно" інтегрувати gRPC у екосистему Spring Boot. Деякі з переваг:

- автоматична генерація та реєстрація сервісів як Spring Beans – звичайні анотації, DI, профілі, конфіги;
- Spring Boot autoconfiguration – немає потреби у складному ручному мапінгу чи запуску gRPC-сервера "в обхід" основного застосунку;
- пряма інтеграція з Spring Security – можна додавати аутентифікацію,

авторизацію, а також свої фільтри до gRPC-ендпоінтів так само легко, як і до REST;

- підтримка валідації – накладати обмеження на вхідні дані з допомогою Jakarta Bean Validation (@Valid, @NotBlank і т.д.) навіть у gRPC-контролерах.

В рамках дипломної роботи було виконано міграцію до 4-х модульної архітектури (рисунок 2.6).

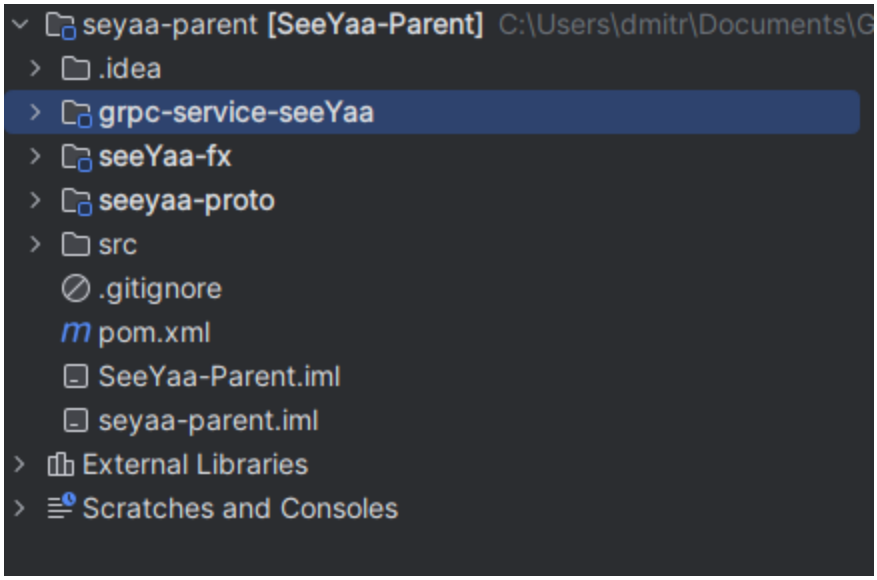

Рисунок 2.6 – Архітектура проекту

Значення модулів:

- seeYaa-fx – клієнтський додаток JavaFX;

- seeYaa-grpc-service – бекенд, який реалізує бізнес-логіку, оперує базою даних і забезпечує gRPC API;

- seeYaa-proto – модуль з .proto-описами (лістинг 2.4) контракту, єдиний для клієнта та сервера;

Лістинг 2.4 – Letter Service

```
syntax = "proto3";

package email.service.letter;

import "google/protobuf/empty.proto";
```

```
import "google/protobuf/timestamp.proto";
import "models.proto";

option java_multiple_files = true;
option java_package        =
"com.seeYaa.proto.email.service.letter";
option java_outer_classname = "LetterServiceProto";

message LetterRequest {
  string user_by_email = 1;
  string user_to_email = 2;
  string topic         = 3;
  string text          = 4;
}

message LetterIdEmailRequest {
  string letter_id = 1;
  string email     = 2;
}

message LetterIdRequest { string id = 1; }

message TopicSearchRequestBy {
  string topic          = 1;
  string user_by_email  = 2;
}

message TopicSearchRequestTo {
  string topic          = 1;
  string user_to_email  = 2;
}

message LetterList { repeated email.models.Letter letters = 1; }

message LetterWithAnswers {
  string                          letterId  = 1;
  repeated email.models.Answer    answers   = 2;
  string                          text      = 3;
}

service LetterService {
  rpc SendLetter            (LetterRequest)          returns
(email.models.Letter);
  rpc SetLetterToSpam       (LetterIdEmailRequest)   returns
(google.protobuf.Empty);
  rpc SetLetterToGarbage    (LetterIdEmailRequest)   returns
(google.protobuf.Empty);

  rpc FindById              (LetterIdRequest)        returns
(email.models.Letter);
  rpc FindAllSentByTopic    (TopicSearchRequestBy)   returns
(LetterList);
  rpc FindAllInboxByTopic   (TopicSearchRequestTo)   returns
```

```
(LetterList);

  rpc DeleteLetterById      (LetterIdRequest)        returns
(google.protobuf.Empty);
}
```

- seeYaa-parent – керуючий модуль-монорепозиторій (Maven parent).

Завдяки gRPC і Spring:

- забезпечено швидку та безпечну передачу листів із файлами-вкладеннями;

- великі файли передаються як окремі gRPC-сесії (рисунок 2.7), із підтримкою лімітів на розмір повідомлення, що недосяжно або менш ефективно у класичному REST;

```
@Configuration
public class GrpcConfig {
    private static final String FILE_SERVER = "0.0.0.0:9090";  1 usage
    private static final String USER_SERVER = "0.0.0.0:9091";  1 usage
    private static final String LETTER_SERVER = "0.0.0.0:9092";  4 usages

    @Bean
    public UsersServiceGrpc.UsersServiceBlockingStub user(GrpcChannelFactory channels) {
        return UsersServiceGrpc.newBlockingStub(channels.createChannel(USER_SERVER));
    }

    @Bean
    public AnswerServiceGrpc.AnswerServiceBlockingStub answer(GrpcChannelFactory channels) {
        return AnswerServiceGrpc.newBlockingStub(channels.createChannel(LETTER_SERVER));
    }

    @Bean
    public LetterServiceGrpc.LetterServiceBlockingStub letter(GrpcChannelFactory channels) {
        return LetterServiceGrpc.newBlockingStub(channels.createChannel(LETTER_SERVER));
    }

    @Bean
    public MovedLetterServiceGrpc.MovedLetterServiceBlockingStub movedLetter(GrpcChannelFactory channels) {
        return MovedLetterServiceGrpc.newBlockingStub(channels.createChannel(LETTER_SERVER));
    }

    @Bean
    public StorageServiceGrpc.StorageServiceBlockingStub storage(GrpcChannelFactory channels) {
        return StorageServiceGrpc.newBlockingStub(channels.createChannel(FILE_SERVER));
    }

    @Bean
    @Primary
    public MovedLetterConfigurationGrpc.MovedLetterConfigurationBlockingStub movedLetterServiceBlockingStub(GrpcChannelFactory channels) {
        return MovedLetterConfigurationGrpc.newBlockingStub(channels.createChannel(LETTER_SERVER));
    }
}
```

Рисунок 2.7 – Розбиття на кілька серверів

- реалізована гнучка перевірка даних та обробка помилок на обох сторонах (протокольний рівень повертає детальні помилки gRPC Status, які легко парсити у клієнті);

- реалізовано розмежування доступу (рольова авторизація) для e-mail-функцій через Spring Security та кастомні gRPC інтеграційні фільтри.

gRPC використовується такими компаніями, як Netflix, Google, Square, Cisco, Docker і багатьма іншими для найкритичніших сервісів. Зокрема, Netflix рекомендував інтеграцію між Spring Boot та gRPC як сучасний стандарт побудови мікросервісів із найменшими затримками та найвищою продуктивністю. Протокольна взаємодія у gRPC виграє до 10-20 разів у пропускній здатності на мережі і CPU-навантаженні при обробці схожих об’ємів даних порівняно зі звичайними REST-сервісами на JSON — це підтверджують open-source бенчмарки.

Spring Boot – це наймасштабованіший Java-фреймворк для створення сучасних production-grade систем. Його співробітниця з gRPC дозволяє отримати максимум з обох світів: знайомість Spring-розробників, безпека, DI + надшвидкий бінарний транспорт та мікросервісний API-контракт.

Висновок: використання gRPC у поєднанні зі Spring Boot дозволило отримати швидкий, надійний та гнучкий бекенд, що задовольняє сучасним вимогам високонавантажених систем з вимогливою клієнтською частиною (JavaFX). При цьому вдалося забезпечити зручний і контрольований API-контракт як для сервісу, так і для будь-яких майбутніх інтеграцій (наприклад, розробка мобільних додатків або зовнішніх мікросервісів), а також вдалося розробити декілька серверів які відповідають за кожний окремий таск, що збільшує у багато разів швидкість додатку.

3 ОПИС ФУНКЦІОНАЛЬНОСТІ EMAIL SERVICE

3.1 Отримання та відправка електронної пошти

Функції надсилання та отримання електронних листів у додатку органічно поєднані з надійними бекенд-сервісами та зручними елементами інтерфейсу, що підвищує функціональність та продуктивність роботи розробника. Відправлення електронного листа запускається на стороні клієнта за допомогою JavaFX SendLetterController, який збирає вхідні дані користувача, вкладення та інформацію про одержувача. Коли натискається кнопка надсилання, фонова задача створює LetterRequestDto і викликає основний сервісний рівень програми. Продуктивність значно підвищується завдяки асинхронному завантаженню файлів і надсиланню листів, підтримці чуйності інтерфейсу та мінімізації затримок для кінцевого користувача.

Логіка відправки бекенду виконується в сервісі LetterService за допомогою методу (лістинг 3.1), що відповідає за валідацію користувачів, перетворення DTO в сутності та зберігання в контексті транзакцій. Дві ключові технології забезпечують значний приріст безпеки та зручності обслуговування: Spring Security обмежує цю операцію користувачами, які увійшли в систему (@PreAuthorize(«hasRole(“ROLE_USER”)»), а Spring's Transactional Management забезпечує узгодженість бази даних навіть у разі збою [9].

Лістинг 3.1 – Процес відправлення листа

```
@Override
@Transactional
@PreAuthorize("hasRole('ROLE_USER')")
public LetterDto sendLetter(@Valid LetterRequestDto
letterRequestDto) {
    final Users usersBy =
usersRepo.findByEmail(letterRequestDto.userBy())
            .orElseThrow(() -> new NotFoundException("User not
found"));
```

```
    return usersRepo.findByEmail(letterRequestDto.userTo())
            .map(userTo ->
LetterMapper.INSTANCE.toLetterDto(letterRepo.save(
                    Letter.builder()
                            .userBy(usersBy)
                            .userTo(userTo)
                            .text(letterRequestDto.text())
                            .topic(letterRequestDto.topic())
                            .activeLetter(Boolean.TRUE)
                            .createdAt(LocalDateTime.now())
                            .build()))
            ).orElseThrow(() -> new NotFoundException("User not
found"));
}
```

Після написання листа, файлові вкладення надсилаються через код (лістинг 3.2), який перевіряє та зберігає кожен файл, пов'язуючи його з відповідною літерою. Продуктивність підвищується завдяки використанню репозиторіїв Spring Data для абстрагування від шаблонної логіки бази даних, а анотації безпеки знижують ризик несанкціонованого доступу.

Лістинг 3.2 – Процес завантаження файлу у додаток

```
@Override
public void uploadFile(UploadFileRequest request,
StreamObserver<Empty> responseObserver) {
    try {
        final var letter =
letterRepository.findById(request.getLetterId())
                .orElseThrow(() -> new NotFoundException("Letter
not found with id: " + request.getLetterId()));


filesRepository.save(org.parent.grpcserviceseeyaa.entity.Files.b
uilder()
                .name(request.getName())
                .type(request.getType())
                .size(request.getSize())
                .data(request.getData().toByteArray())
                .letter(letter)
                .build());


responseObserver.onNext(com.google.protobuf.Empty.getDefaultInst
ance());
        responseObserver.onCompleted();
    } catch (Exception ex) {
        log.error("Error uploading file", ex);
```

```
        responseObserver.onError(
                Status.INTERNAL
                        .withDescription(ex.getMessage())
                        .withCause(ex)
                        .asRuntimeException()
        );
    }
}
```

Система також використовує щоб точно асоціювати надіслані повідомлення з автентифікованими користувачами, запобігаючи видаванню себе за іншого або підвищенню привілеїв та з великою швидкістю використовуючи Thread (лістинг 3.3).

Лістинг 3.3 – Відправлення листа з визначенням автентифікованого користувача

```
@FXML
public void sendLetter(ActionEvent event) {
    final var scene = ((Node) event.getSource()).getScene();
    Platform.runLater(() -> {
        text.setDisable(true);
        attachFile.setDisable(true);
        sendLetter.setDisable(true);
        toWhom.setDisable(true);
        topic.setDisable(true);
        scene.setCursor(Cursor.WAIT);
    });

    Task<Void> uploadTask = new Task<>() {
        @Override
        protected Void call() {
            LetterRequestDto request = new LetterRequestDto(
                    text.getText(),
                    topic.getText(),
                    toWhom.getText(),
                    securityService.getCurrentUserEmail());
            final var savedLetter =
letterServiceImpl.sendLetter(request);

            for (File file : selectedFiles) {
                MultipartFile multipartFile = new
PathMultipartFile(file);
                storageServiceImpl.uploadFile(multipartFile,
savedLetter.id());
            }
            return null;
        }
```

```
    };

    uploadTask.setOnSucceeded(e ->
            Platform.runLater(() -> {
                stage = (Stage) ((Node)
event.getSource()).getScene().getWindow();
                stage.close();
                scene.setCursor(Cursor.DEFAULT);
            }));
    uploadTask.setOnFailed(e -> {
        Platform.runLater(() ->
scene.setCursor(Cursor.DEFAULT));

        showAlert("Upload Failed",
uploadTask.getException().getMessage());
    });

    new Thread(uploadTask).start();
}

public String getCurrentUserEmail() {
    final Authentication authentication =
SecurityContextHolder.getContext().getAuthentication();
    if (authentication == null ||
!authentication.isAuthenticated())
        throw new AuthenticationCredentialsNotFoundException("No
authenticated user found");

    return authentication.getName();
}
```

Цікавим аспектом цієї реалізації є поєднання декларативної безпеки, забезпеченої шляхом використання анотацій, з програмною обробкою контексту в SecurityService. Багаторівнева така стратегія безпеки, разом з розподілом завдань між інтерфейсом, сервісом, сховищем та компонентами безпеки, підтримує ефективний процес розробки та стабільне тестування. Технології, наприклад, Spring Boot, Spring Security і JavaFX, не лише допомагають забезпечувати високомодульні системи, що зручні в обслуговуванні, але також збільшують ефективність команди, визначаючи бізнес-логіку незалежною від інфраструктурних викликів. Цей модульний підхід демонструє сучасні найкращі практики щодо розробки масштабованих та безпечних рішень для електронної пошти в корпоративному та академічному середовищі.

3.2 Обробка вкладень та управління файлами

У наш час керування вкладеннями та файлами стало невід'ємною частиною комунікаційних систем, адже дозволяє користувачам додавати до своїх повідомлень різні типи контенту. У представленому додатку використовується безпечний і модульний метод завантаження конфігурації, збереження та встановлення вкладень для пощаю гнучкість інтерфейсу для користувачів. Коли користувач надсилає лист із вкладеннями, JavaFX-клієнт збирає вибірку файлів і під час надсилання sendLetter (ActionEvent event) перебирає кожний файл до файлу класу MultipartFile для завантаження, що забезпечує правильну сумісність та абстрагування від абстрактного процесу представлення файлу, який виконується.

Завантаження виконується асинхронно за допомогою JavaFx-задачі, щоб підтримувати чуйність інтерфейсу і запобігти блокуванню на потенційно трудомісткі операції для зручності і продуктивності [2].

На сервері сторінці авт спеціально буде оброблено вхідний файл (лістинг 3.4), що зчитає вміст файлу як одне значення байтового масиву і зберігає цей вміст саме з метаданими (назва, тип, розмір, розміром і літерною асоціацією) в базі даних. Таке рішення видаляє всі зовнішні залежності від інформації про файлову систему, зручне резервне копіювання, масштабування та транзакційність, поки що подібні особливості симпліфікуються в хмарних або розподілених розгортах. Анотація @PreAuthorize(“hasRole (“ROLE_USER”)) захищає цей процес - заборона завантаження фактами там із автентифікованими учасниками за допомогою аутентифікації та забезпечує конфіденційність та цілісність даних.

Лістинг 3.4 – Процес зчитання байтів з файлу

```
try (InputStream inputStream = file.getInputStream()) {
    fileData = inputStream.readAllBytes();
}

filesRepo.save(Files.builder()
        .name(file.getOriginalFilename())

.type(FileType.fromFileExtension(file.getOriginalFilename()))
        .size(file.getSize())
        .data(fileData)
        .letter(letter)
        .build());
```

Для процесу завантажування та отримання вже знову використовується шаблон асинхронності, метод FileDownloadServiceImpl.createDownloadTask обгортає функцію пошуку файлів у JavaFX-задачі та забезпечує, що величина діалогів файлів і запис до файлу (copyToFile ()..) не блокують потік інтерфейсу користувача – таким чином, що користувач отримує запит через діалогове вікно FileChooser та файл запишеться на відкритий диск буферизованими фрагментами отримано нижче (лістинг 3.5).

Лістинг 3.5 – Забезпечення не блокування потоку

```
private void copyToFile(byte[] data, File outFile) throws
IOException {
    try (InputStream is = new ByteArrayInputStream(data);
         FileOutputStream fos = new FileOutputStream(outFile)) {
        final byte[] buffer = new byte[1024 * 1024];
        int bytesRead;
        while ((bytesRead = is.read(buffer)) != -1)
            fos.write(buffer, 0, bytesRead);
    }
}
```

Це забезпечує підтримку великих файлів і мінімізує використання пам'яті. Крім того, отримання метаданих (getFileMetadataByLetterId) дозволяє інтерфейсу відображати інформативні списки файлів, включаючи відформатовані розміри та піктограми певних типів (реалізовано у FileUiUtils.createFileTypeIcon), пропонуючи інтуїтивно зрозумілий користувачеві інтерфейс. Використання FileRowFactory.createFileRow

підтримує масштабовані презентації для декількох вкладень, з відображенням прогресу завантаження в реальному часі.

Цікавими моментами та обґрунтуванням цього вибору є рішення зберігати вміст файлів у базі даних (забезпечуючи атомарність за допомогою літерних транзакцій), послідовне застосування перевірок безпеки та суворе розмежування завдань: логіка завантаження/вивантаження відокремлена від презентації та видобування метаданих. Такий вибір покращує зручність обслуговування і полегшує впровадження майбутніх функцій, таких як сканування на віруси, дотримання квот або ведення журналів аудиту. Завдяки використанню асинхронних патернів, практик чистого кодування та моделі безпеки Spring, система відповідає вимогам корпоративного рівня щодо надійності, продуктивності користувачів та безпеки, пропонуючи при цьому багатий та сучасний досвід роботи з вкладеннями.

3.3 Використання AI-асистент

Для покращення аналізу тексту та надання інтелектуального зворотного зв’язку користувачам програма інтегрує розширений помічник штучного інтелекту, що використовує сервіси генеративної мовної моделі Vertex AI Google. Вона генерує підказку можливо курсуючи запитами користувача (наприклад, для глибшого розуміння тексту, короткий вираз про текст чи генерується відповідь самостійно). Щоразу, коли користувач викликає GDeepAnalysisMessage, система спростувати підказку, дефізоряда підказку за повірником та текстом по листу і надсилає його під час генерації підказки, компонент AIController штучного інтелекту Vertex [1].

Для зручності і портрету області конфігурації AIController, штрихова конфігурація проекту полягає в полях анотації Spring @Value. Тому код програми розроблен таким образом у лістинг 3.6.

Лістинг 3.6 – ШІ контролер

```
@Component
@Slf4j
public class AIController {
    @FXML private TextArea responseAi;
    @Setter private String prompt;

    @Value("${vertex.project_id}")
    private String projectId;
    @Value("${vertex.location}")
    private String locationId;
    @Value("${vertex.model_id}")
    private String modelId;

    @FXML
    public void initialize() {
        if (prompt != null) analyzeText();
    }

    private void analyzeText() {
        try (final VertexAI vertexAi = new VertexAI(projectId,
locationId)) {
            final GenerationConfig generationConfig =
                    GenerationConfig.newBuilder()
                            .setMaxOutputTokens(8192)
                            .setTemperature(1F)
                            .setTopP(0.95F)
                            .build();
            final List<SafetySetting> safetySettings =
Arrays.asList(
                    SafetySetting.newBuilder()

.setCategory(HarmCategory.HARM_CATEGORY_HATE_SPEECH)

.setThreshold(SafetySetting.HarmBlockThreshold.BLOCK_MEDIUM_AND_
ABOVE)
                            .build(),
                    SafetySetting.newBuilder()

.setCategory(HarmCategory.HARM_CATEGORY_DANGEROUS_CONTENT)

.setThreshold(SafetySetting.HarmBlockThreshold.BLOCK_MEDIUM_AND_
ABOVE)
                            .build(),
                    SafetySetting.newBuilder()

.setCategory(HarmCategory.HARM_CATEGORY_SEXUALLY_EXPLICIT)

.setThreshold(SafetySetting.HarmBlockThreshold.BLOCK_MEDIUM_AND_
ABOVE)
                            .build(),
                    SafetySetting.newBuilder()
```

```
.setCategory(HarmCategory.HARM_CATEGORY_HARASSMENT)

.setThreshold(SafetySetting.HarmBlockThreshold.BLOCK_MEDIUM_AND_
ABOVE)
                            .build());
            final GenerativeModel model =
                    new GenerativeModel.Builder()
                            .setModelName(modelId)
                            .setVertexAi(vertexAi)

.setGenerationConfig(generationConfig)
                            .setSafetySettings(safetySettings)
                            .build();


            final var content =
ContentMaker.fromMultiModalData(prompt);
            final ResponseStream<GenerateContentResponse>
responseStream =
                    model.generateContentStream(content);

            final StringBuilder fullResponse = new
StringBuilder();


            responseStream.forEach(response -> {
                final String chunk = response.getCandidates(0)
                        .getContent()
                        .getParts(0)
                        .getText();
                fullResponse.append(chunk);

                Platform.runLater(() ->
responseAi.setText(fullResponse.toString()));
            });
        } catch (IOException e) {
            log.info("Error occurred while generating the model:
{}", e.getMessage());
            showAlert("AI didn't response", e.getMessage());
        }

    }
}
```

Отримавши запит, метод analyzeText встановлює безпечне з'єднання зі штучним інтелектом Vertex(плюси/мінуси ШІ на табл. 3.1), налаштовує конфігурацію моделі (наприклад, довжину виводу, температуру та вибірку top-p) та застосовує суворі налаштування безпеки для фільтрації небажаного контенту. Відповідь штучного інтелекту транслюється в режимі реального

часу, динамічно оновлюючи інтерфейс користувача без його зависання завдяки Platform.runLater від JavaFX.

Ця інтеграція використовує найсучасніше розуміння мови Google, щоб надати користувачам швидку та контекстно-залежну аналітику листування, ефективно діючи як розумний помічник, вбудований у робочий процес комунікації. Це не лише підвищує продуктивність та прийняття рішень користувачами, але й демократизує доступ до аналізу тексту на базі штучного інтелекту без необхідності глибоких технічних знань.

Таблиця 3.1 – Аналіз ШІ Vertex

Плюси	Мінуси
Найсучасніше NLP: Використовує найновіші генеративні моделі Google для отримання точних результатів.	Залежність від хмари: Потрібне стабільне підключення до Інтернету та доступ до Google Cloud.
Масштабованість: Обробляє високі навантаження без потреби в локальних обчислювальних ресурсах.	Вартість: Ціна розраховується за використання; інтенсивне використання може призвести до значних поточних витрат.
Проста інтеграція: Добре документовані SDK та API; прості клієнтські бібліотеки Java.	Затримка: Мережеві цикли передачі даних призводять до певної затримки порівняно з локальним виведенням.
Безпека та відповідність: Підтримується функціями безпеки підприємства Google.	Конфіденційність даних: Користувацький контент надсилається на зовнішні сервери, що може викликати занепокоєння.
Автоматичні оновлення: Завжди найновіші версії моделей, немає потреби розгортати/підтримувати локально.	Обмежене налаштування: Менше контролю для низькорівневого налаштування моделей або індивідуальних завдань.
Багаті функції: Вбудовані налаштування безпеки, модерація контенту, великі контекстні вікна.	Регіональні обмеження: Доступність та управління даними можуть залежати від місцезнаходження користувача.

Висновок: Інтеграція Vertex AI забезпечує передовий, гнучкий та масштабований спосіб надання кінцевим користувачам інтелектуальних функцій, але водночас враховує вартість, конфіденційність та залежність від інфраструктури. Такий вибір посилює конкурентну перевагу вашої програми для виконання аналітичних завдань, особливо там, де пріоритетами є гнучкість та мінімальне обслуговування.

3.4 Інтеграція з іншими сервісами

Всім досвідним розробникам відомо, що сучасний розвиток інфраструктури додатків неможливий без тісної інтеграції з зовнішніми хмарними та сторонніми сервісами. Інтеграція розширює функціонал, забезпечує масштабованість та дозволяє зручно додавати нові можливості для користувачів. Проаналізуймо кілька напрямків, у яких система SeeYaa вже може або потенційно буде взаємодіяти з іншими сервісами.

1. Використання AWS Cloud для файлів та бази даних

Файли (Attachments).

На сьогодні більшість сучасних сервісів використовують не локальні файлові системи, а хмарні сховища для безпечного, швидкого та масштабованого зберігання вкладень.

Одним з найпопулярніших рішень сьогодні є Amazon S3. Це просте у використанні, практично безмежне за обсягом та надпотужне щодо швидкодії та надійності рішення для зберігання файлів у хмарі [1]. Інтегрувавши Amazon S3, можна досягти:

- моментальної доставки великих файлів користувачам по всьому світу;
- автоматичного резервного копіювання даних;
- централізованого зберігання вкладень незалежно від основної інфраструктури додатку [1].

База даних.

В якості альтернативи локальній або віртуальній БД PostgreSQL можна використовувати керовані рішення, такі як Amazon RDS (Relational Database Service) [1] (таблиця 3.2). Це дозволяє:

- автоматично масштабувати БД під навантаження;
- отримати високу відмовостійкість і резервування (multi-az redundancy);
- полегшити адміністрування без необхідності самостійно оновлювати чи резервувати базу.

Таблиця 3.2 – Використання AWS

Сервіс	Переваги інтеграції	Приклади використання
Amazon S3	Безмежне сховище, швидкий доступ, резервне копіювання, простий API	Зберігання вкладень (attachments), архів листів, завантаження/завантаження файлів
Amazon RDS	Відмовостійкість, масштабованість, готовність до production	Основна БД додатка для листів, користувачів, лістингов

2. Відправка листів у зовнішні поштові сервіси (наприклад, Gmail)

Реальна інтеграція зі сторонніми email-сервісами – перспективний крок для SeeYaa, що зробить платформу універсальним центром комунікацій.

У майбутньому планується реалізувати функціонал, де користувачі зможуть:

- надсилати листи безпосередньо на Gmail, Outlook й інші загальноприйняті поштові сервіси;
- підключати свої зовнішні поштові скриньки до SeeYaa (імпорт листів);
- синхронізувати контакти, календарі та інші дані через відповідні API (наприклад, Google API, Microsoft Graph API).

Принцип роботи:

- для надсилання листа зовнішнім користувачам використовується стандартний протокол SMTP або API-технології (Gmail API);

- для імпорту або читання листів – IMAP/POP3 протоколи або ті ж офіційні API сервісів;

- система керує авторизацією користувача та захищено працює із зовнішньою поштою через OAuth 2.0;

3. Створення повнофункціонального веб-сайту

Ще один логічний вектор розвитку – реалізація веб-версії додатка (таблиця 3.3). Це забезпечить:

- доступ до пошти та AI-асистента з будь-якого пристрою (браузера, планшета, смартфона);

- можливість інтеграції із зовнішніми сервісами через REST API (наприклад, додавання модулів відеозв’язку, календаря, CRM);

- розширення аудиторії шляхом спрощення входу та реєстрації користувачів;

- зв’язок з іншими веб сайтами і платформами, наприклад, для імпорту контактів, календарів, чи інтеграції з корпоративними порталами та соціальними мережами.

Веб-додаток відкриває можливості для більш ефективної інтеграції з зовнішніми сервісами, зокрема через публічні REST API. Така архітектура дозволяє легко підключати до системи додаткові модулі, наприклад календарі, сервіси відеозв’язку або CRM-рішення, що особливо актуально для корпоративних користувачів. Окремої уваги заслуговує питання реєстрації й автентифікації користувачів у веб-версії. Реалізація сучасних механізмів входу, таких як аутентифікація по e-mail, OAuth або двофакторна перевірка, робить сервіс не лише зручним у використанні, а й безпечним. У перспективі планується підтримка адаптивного дизайну, що гарантуватиме комфортну роботу з інтерфейсом на різних розмірах екрана, а також можливість залучення додаткових інструментів штучного інтелекту та розширення особистих налаштувань для кожного користувача.

Таблиця 3.3 – Потенціал інтеграції

Напрямок	Технологія/сервіс	Мета та переваги	Стадія впровадження
Хмарне зберігання	Amazon S3, Google Cloud Storage	Економія ресурсів, швидкість, резервування	Можлива інтеграція
Керована БД	Amazon RDS, Google Cloud SQL	Надійність, відмова від локальної БД	Можлива інтеграція
Зовнішні email-сервіси	Gmail API, Outlook API, SMTP, IMAP	Розширення аудиторії, універсальність	Перспективно
Веб-платформа	React/Angular + REST API	Мультиплатформеність, інтеграція з іншими модулями	У планах
Інші сервіси	AI, CRM, Payments	Розширення функціоналу, автоматизація	За потреби

4 РОЗГОРТАННЯ ТА ТЕСТУВАННЯ ДОДАТКУ

4.1 Використання докер

Docker – це сучасний стандарт контейнеризації, який дозволяє швидко й просто розгортати серверну частину додатка, незалежно від місця запуску: локальної машини, хостинг-платформи чи хмарного серверу [19]. У проекті «SeeYaa» Docker використовується для деплою серверного застосунку та для спрощення процесу інтеграції з фронтендом (наприклад, майбутнім веб-сайтом) (таблиця 4.1).

Основні переваги використання Docker:

- універсальність – додаток можна запускати на будь-якій ОС без складної ручної установки залежностей;
- ізоляція – усі необхідні інструменти, бібліотеки і налаштування містяться усередині контейнера, що усуває конфлікти із системними пакетами;
- масштабованість – легко піднімати копії серверу для балансування навантаження.

Таблиця 4.1 – Використання Docker

Сцена застосування	Деталі
Сервер додатку (Spring Boot)	Deployment у контейнері для бекенду
Фронтенд-додаток (майбутній)	Окремий контейнер для сайту, швидке підключення до API
Тестова база даних	Локальний чи хмарний контейнер PostgreSQL для випробувань

4.2 Налаштування середовища розробки

Для комфортної роботи над десктопним застосунком на Java застосовується підхід, звичний для класичних Windows/Linux/Mac програм, створюється інсталятор або збірка *(.exe, .msi, .deb*, тощо), який користувач може встановити одним кліком.

Процес налаштування середовища розробки:

Особливості для розробника:

- встановлення JDK (Java Development Kit);
- IDE (наприклад, IntelliJ IDEA, Eclipse чи NetBeans);
- бібліотеки (JavaFX, Spring Boot).

Для кінцевого користувача – це надання інсталятора із простим майстром встановлення (наприклад, за допомогою InnoSetup, Install4j, NSIS) або автоматичне додавання всіх необхідних компонентів у пакет – користувачу не потрібно окремо встановлювати Java чи інші залежності.

Таблиця з налаштуванням до кожного користувача(таблиця 4.2).

Таблиця 4.2 – Середовище розробника для різних користувачів

Група користувачів	Дії для налаштування	Переваги
Розробник	IDE, JDK, запуск у dev-режимі	Повний контроль
Кінцевий користувач	Запуск встановлювача, пару кліків	Легкість, зручність

4.3 Проведення тестування

Якість роботи програмного комплексу перевіряється комплексом тестів для різних частин програми – від роботи з базою даних до продуктивності при обробці листів, зберігання вкладень чи навантаження пам’яті.

Основні напрями тестування:

- тестування продуктивності та використання ресурсів – перевірка,

скільки оперативної пам’яті та процесорного часу споживає додаток під час інтенсивних сценаріїв (надсилання листів, завантаження файлів);

- тестування взаємодії з базою даних – симуляція масового додавання та читання листів і вкладень. Також, перевірка цілісності зв’язків (чи коректно при видаленні очищаються пов’язані дані);
- тестування функціоналу – реєстрація, авторизація, створення/відправка листа, відновлення пароля, робота з AI. (табл. 4.3)

Таблиця 4.3 – Види тестувань

Тип тестування	Мета	Інструменти
Unit/Інтеграційні	Перевірка окремих модулів (сервіси, репозиторії тощо)	JUnit, Mockito, Testcontainers
Стрес-тестування	Вимірювання навантаження, виявлення "вузьких місць" системи	JProfiler, VisualVM
Тест для БД	Перевірка даних при роботі з PostgreSQL, хендлінг винятків	Liquibase, Flyway, тестова БД

Результати тестування (рис. 4.1) показує, що додаток стабільно працює з помірним навантаженням на процесор (22%) та використовує незначний обсяг оперативної пам’яті (22 МБ heap memory). Це свідчить про ефективність і оптимізованість програмної реалізації, а також дозволяє рекомендувати використання додатка навіть на комп’ютерах із невисокими технічними характеристиками.

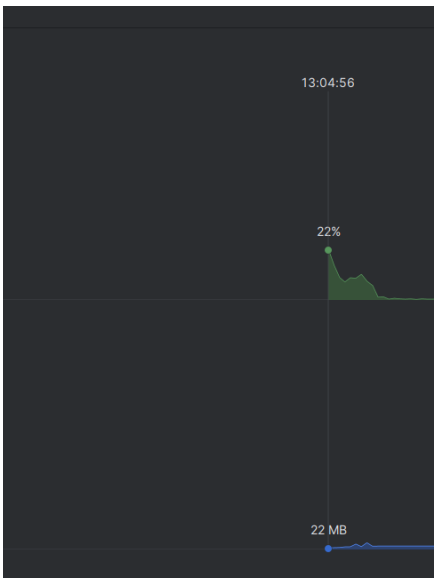

Рисунок 4.1 – Перфоманс додатка під час запуску

Під час реєстрації навантаження на процесор становило лише 3%, а використання оперативної пам'яті складало 94 МБ (рис. 4.2). Це свідчить про низьку ресурсомісткість та ефективну реалізацію даного функціоналу, що забезпечує швидку і стабільну роботу додатка навіть на малопродуктивних системах.

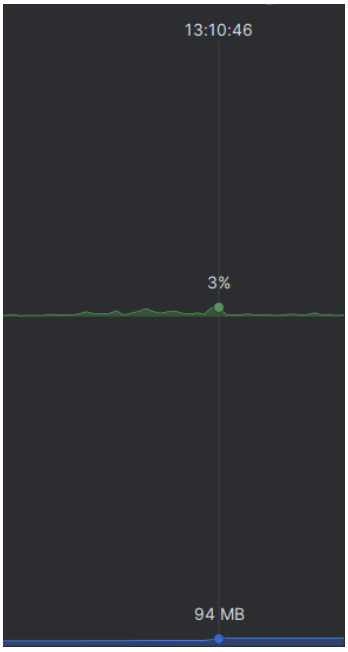

Рисунок 4.2 – Перфоманс додатка під час реєстрації

Під час авторизації додаток споживав лише 1% ресурсів процесора та використовував 109 МБ оперативної пам’яті (heap memory) (рис. 4.3), що свідчить про дуже низьке навантаження на систему й ефективне опрацювання даної операції.

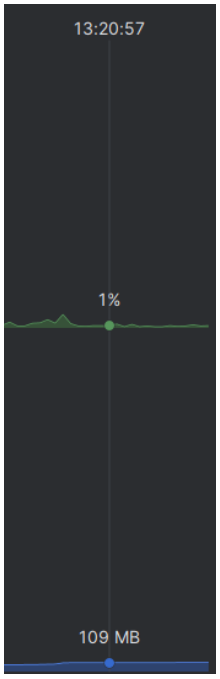

Рисунок 4.3 – Перфоманс додатка під час авторизації

Під час надсилання листа з 35 файлами додаток використовував 4% процесорного часу та 170 МБ оперативної пам’яті (рис. 4.4), що демонструє хорошу оптимізованість і здатність ефективно обробляти масове завантаження вкладень без істотного навантаження на систему.

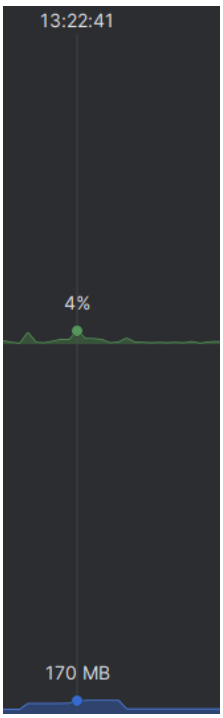

Рисунок 4.4 – Перфоманс додатка під час відправки листа з 35 файлами

Під час надсилання листа з вкладеним файлом розміром 2 ГБ (час надсилання — 2,5 секунди) додаток використовував 10% ресурсів процесора та 2211 МБ оперативної пам’яті (рис. 4.5) для швидкого відправлення. Це свідчить про здатність програми ефективно працювати з великими файлами, забезпечуючи швидке опрацювання.

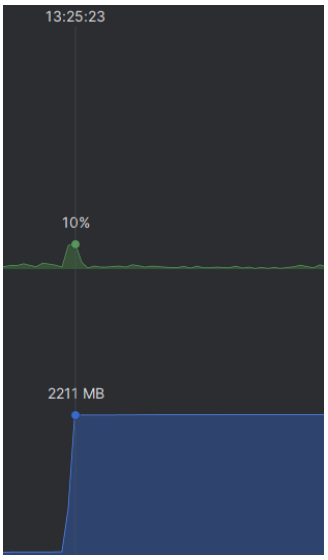

Рисунок 4.5 – Перфоманс додатка під час відправки листа з файлом у розмірі 2 ГБ (2,5 секунди відправки)

ВИСНОВКИ

У основі проекту є модульна структура з чітким розділенням, відповідальності між клієнтом та сервером. Пильну увагу було звернено на створення гарної серверної інфраструктури з використанням Spring Boot та gRPC, інтеграції безпечної системи аутентифікації на базі Spring Security, а також організації зберігання даних у реляційній базі PostgreSQL. Використання JavaFX дозволило зробити простий та функціональний користувацький інтерфейс що підвищує зручність і продуктивність для користувачів.

Особливістю проекту є інтеграція AI-асистента на основі Google Vertex AI, що відкриває нові можливості для автоматизації і підвищення ефективності обробки кореспонденції.

Розроблено та впроваджено ефективні методи тестування, які дозволили протестувати програму на критичні сценарії та забезпечити стабільність їїроботи як під великим навантаженням, так і в звичайному режимі.

Успішно реалізовано підхід до розгортання доповнення з використанням технології Docker, а також підготовлено інсталяційний пакет для зручності кінцевого користувача. Визначено напрямки подальшої роботи - інтеграція з хмарними сервісами AWS та Google, створення веб-версії та розширення взаємодії з іншими поштовими платформами.

Таким чином, отримані результати спільно з розпочатими технологічними рішеннями підтверджують доцільність використання сучасних фреймворків та архітектур для розробки надійних, масштабованих та безпечних IT-продуктів. Реалізований сервіс «SeeYa*a*» діє як конкурентоспроможна альтернатива українським та міжнародним користувачам, який здатен і надалі розвиватися на основі вимог ринку та технологічних тенденцій галузі електронних комунікацій.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. К. Сіва Прасад Редді, Саї Упадх'яюла. Beginning Spring Boot 3: Build Dynamic Cloud-Native Java Applications and Microservices, 2022. – 472 с.

2. OCP Oracle Certified Professional Java SE 21 Developer (Exam 1Z0-830) Java SE 17 Developer (Exam 1Z0-829) Programmer's Guide/ Халід А. Могол, Василь А. Стрельников. 2025. – 7436 с.

3. The Definitive Guide to Modern Java Clients with JavaFX: Cross-Platform Mobile and Cloud Development Updated for JavaFX 21 and 23 / Стівен Чін, Йохан Вос, Джеймс Вівер, Пол Андерсон, Бруно Борхес, Антон Еппл, Вейкі Гао, Джонатан Джайлз, Хосе Переда. 6 грудня 2024 – 648 с.

4. ISBN-10: 1837635641. Learn PostgreSQL: Use, manage, and build secure and scalable databases with PostgreSQL 16 / Лука Феррарі, Енріко Піроцці. 2023. – 744 с.

5. ASIN: B097814WS2. Functional Programming in Java: How functional techniques improve your Java programs / Pierre-Yves Saumont. 2017 – 853 с.

6. Спілке Лауренціу, "Spring Security in Action". – 2024. – С. 480.

7. Spring і Spring Boot: основні відмінності та особливості використання //[Електроний ресурс] https://foxminded.ua/ru/razlichiya-mezhdu-spring-i-spring-boot/ Режим доступу 30.10.24

8. Greg L. Turnquist, "Learning Spring Boot 3.0: Simplify the development of production-grade applications using Java and Spring, 3rd Edition", 2022.

9. Moises Macero Garcia, Tarun Telang, " Learn Microservices with Spring Boot 3: A Practical Approach Using Event-Driven

10. Мартін, Роберт С. "Чиста архітектура: Мистецтво розробки програмного забезпечення" / Роберт С. Мартін ; [пер. з англ. О. Смикалов]. – Фабула, 2019. – 368 с.

11. Наркеде Н., Шапіра Г., Паліно Т. "Apache Kafka. Потокова обробка та аналіз даних" / Неха Наркеде, Гвен Шапіра, Тодд Паліно. – Діалектика,

2019. – 336 с.

12. Лонг Дж., Бастані К. "Java в хмарі. Spring Boot, Spring Cloud, Cloud Foundry" / Джош Лонг, Кеннет Бастані. – 2019. (Оригінальна назва: "Cloud Native Java: Designing Resilient Systems with Spring Boot, Spring Cloud, and Cloud Foundry")

13. Урванов Ф. "Spring и Spring Boot. Разработка облачных приложений на Java". – БХВ-Петербург, 2024. – (Дата написання: кінець 2024 року).

14. Річардс М., Форд Н. "Основи програмної архітектури: Інженерний підхід" / Марк Річардс, Ніл Форд. – (Оригінальна назва: "Fundamentals of Software Architecture: An Engineering Approach") * Ця книга надасть ширший погляд на принципи та практики програмної архітектури, що є важливим для прийняття обґрунтованих проектних рішень.

15. Блох Дж. "Java. Ефективне програмування", 3-тє видання / Джошуа Блох. – (Оригінальна назва: "Effective Java, 3rd Edition") * Класична книга, що містить найкращі практики та поради щодо написання якісного коду на Java. Це буде корисним доповненням до будь-якого списку літератури для Java-розробника.

16. Kasun Indrasiri, Danesh Kuruppu. "gRPC: Up and Running: Building Cloud Native Applications with Go and Java for Docker and Kubernetes". – O'Reilly Media, 2020. – 224 p. (ISBN: 9781492058335). * Це одна з ключових книг по gRPC, яка детально розглядає його основи, архітектуру та практичне використання, зокрема з Java та у зв'язці з Docker і Kubernetes. Дуже релевантно для вашого проєкту.

17. Nigel Poulton. "Docker Deep Dive". – 2023 Edition. – 268 с. (ISBN: 9781916585256).

18. gRPC Documentation [Електроний ресурс] // gRPC Official Website. – Access mode: https://grpc.io/docs/

19. Docker Documentation [Електроний ресурс] // Docker Official Website. – Access mode: https://docs.docker.com/

20. Дмитренко Валерій, Федорченко В.М. // Методи використання

Spring Boot у хмарних обчислень // Матеріали п’ятнадцятої міжнародної науково-технічної конференції «Сучасні напрями розвитку інформаційно-комунікаційних технологій та засобів управління». Том 3: тези доповідей, 24-25 квітня 2025 р. – Харків: ХНУРЕ, 2025 – С. 55 - 56.

21. Pushkar Joglekar. "Practical gRPC with Java: Build Efficient, High-Performance Distributed Applications", 24 – 25 квітня. – Apress, 2024. – (Anticipated, check for final release date and details).