МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ І ПРИКЛАДНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ШЕВЦОВ МИХАЙЛО ВОЛОДИМИРОВИЧ

Допускається до захисту:
в.о. завідувача кафедри
інформаційних технологій,
д-р. техн. наук, професор
_____Наталія ВЕСЕЛОВСЬКА
«____»_____ 2026 р.

РОЗРОБКА КРОСПЛАТФОРМНОГО ПОШТОВОГО КЛІЄНТА З АСИНХРОННОЮ АРХІТЕКТУРОЮ ТА ІНТЕГРАЦІЄЮ ШТУЧНОГО ІНТЕЛЕКТУ ЗАСОБАМИ C++ ТА QT

Спеціальність 122 Комп’ютерні науки

Кваліфікаційна (бакалаврська) робота

Керівник:
Юрій ПОРЕМСЬКИЙ, старший
викладач кафедри інформаційних
технологій, к. т. н.
_____

Оцінка: _____ / _____ / _____
(бали/за шкалою ЄКТС/за національною шкалою)

Голова ЕК: _____
(підпис)

Вінниця - 2026

АНОТАЦІЯ

Шевцов М. В. Розробка кросплатформного поштового клієнта з асинхронною архітектурою та інтеграцією штучного інтелекту засобами C++ та Qt. Спеціальність 122 «Комп'ютерні науки». Донецький національний університет імені Василя Стуса, Вінниця, 2026.

У кваліфікаційній (бакалаврській) роботі створено нативний кросплатформний поштовий клієнт засобами C++23 та Qt 6 за шаблоном Reactor з підтримкою протоколів IMAP та SMTP, OAuth 2.0 та ізольованим ШІ-модулем. Усунено характерні для сучасних клієнтів надмірне споживання ресурсів, перевантаженість інтерфейсу і залежність ШІ-функцій від інфраструктури вендора.

Ключові слова: поштовий клієнт, IMAP, SMTP, OAuth 2.0, асинхронне програмування, C++, Qt, ШІ, кросплатформне програмне забезпечення.

108 с., 28 рис., 15 табл., 65 джерел.

ABSTRACT

Shevtsov M. V. Development of a Cross-Platform Email Client with Asynchronous Architecture and Artificial Intelligence Integration Using C++ and Qt. Specialty 122 "Computer Science". Vasyl Stus Donetsk National University, Vinnytsia, 2026.

In the qualification (bachelor's) thesis, a native cross-platform email client has been developed using C++23 and Qt 6 with the Reactor pattern, supporting IMAP and SMTP protocols, OAuth 2.0, and an isolated AI module. The excessive resource consumption, interface bloat, and AI vendor dependency typical of contemporary clients have been eliminated.

Keywords: email client, IMAP, SMTP, OAuth 2.0, asynchronous programming, C++, Qt, AI, cross-platform software.

108 p., 28 fig., 15 tab., 65 sources.

ЗМІСТ

ВСТУП

Актуальність роботи. Електронна пошта залишається ключовим каналом ділової та академічної комунікації: за період 2006-2018 років середній час, який працівник проводить у пошті щодня, зріс із 29 хвилин до приблизно двох годин, а зафіксований у рецензованих дослідженнях ефект перевантаження поштою суттєво підвищує когнітивне навантаження [1-2]. Звідси прикладна вимога до десктопного клієнта - компактність функціональності і передбачуване споживання ресурсів.

Цьому критерію провідні сучасні клієнти системно не відповідають. Mozilla Thunderbird споживає до 2 ГБ оперативної пам'яті при індексуванні великих скриньок, новий Microsoft Outlook у межах інциденту MO907654 сягав 4 ГБ і був публічно охарактеризований як «не готовий до промислового використання», Apple Mail доступний лише на macOS, а Mailspring успадковує ресурсну вартість стеку Electron [3-7]. Окрему категорію складають десктопні клієнти з інтеграцією штучного інтелекту (Shortwave, Superhuman), у яких повідомлення реплікуються у хмарну інфраструктуру вендора для серверної обробки ШІ-функцій (зокрема через сторонніх субпроцесорів - OpenAI, Anthropic, Pinecone) і не залишаються локально доступними клієнту [8]. Утворена ринкова прогалина - відсутність нативного ресурсо-економного клієнта з архітектурно ізольованим ШІ-модулем - формує предметну нішу цієї роботи.

Запропонована архітектура заповнює цю нішу за рахунок трьох наскрізних рішень. Нативна реалізація мовою C++23 з керуванням ресурсами за ідіомою RAII (Resource Acquisition Is Initialization), без середовища виконання зі збиранням сміття і без вбудованого браузерного рушія дає передбачуване споживання пам'яті; асинхронна модель Reactor поверх механизмів epoll/kqueue/IOCP дозволяє єдиному робочому потоку обслуговувати багато одночасних мережевих операцій без виділення окремого потоку на кожне з'єднання; дворівневий локальний шифрований кеш тіл повідомлень усуває повторні мережеві запити при поверненні до прочитаних листів [9-10]. Автентифікація відповідає чинним нормативним

документам IETF - RFC 8252 «OAuth 2.0 for Native Apps» (BCP 212) і RFC 7636 «Proof Key for Code Exchange», - а ШІ-модуль архітектурно ізольовано в окремий контур, завдяки чому його відмова не є фатальною для системи: решта підсистем продовжують працювати у штатному режимі [11-12].

Мета дослідження - проєктування та програмна реалізація десктопного кросплатформного поштового клієнта з мінімалістичною функціональністю, передбачуваним споживанням ресурсів, асинхронною мережевою архітектурою, безпечною автентифікацією за стандартом OAuth 2.0 із розширенням PKCE, підтримкою кількох облікових записів та опційною інтеграцією зі стороннім сервісом штучного інтелекту, архітектурно ізольованою від основних поштових сценаріїв.

Завдання дослідження. Для досягнення поставленої мети необхідно вирішити такі завдання:

- проаналізувати сучасні десктопні поштові клієнти, виокремити функціональну нішу для розроблюваного застосунку та сформулювати функціональні і нефункціональні вимоги до системи;
- дослідити сучасні технології реалізації нативного кросплатформного програмного забезпечення та обґрунтувати вибір мови програмування, фреймворку графічного інтерфейсу, бібліотек асинхронного вводу-виводу і криптографії;
- розробити багаторівневу архітектуру застосунку з двома незалежними циклами подій, доменну модель даних, дворівневий локальний кеш повідомлень із шифруванням та узгоджену стратегію обробки помилок;
- розробити мережеву підсистему для роботи з сучасними стандартами IMAP та SMTP;
- розробити безпечну автентифікацію за схемою OAuth 2.0 + PKCE з підтримкою кількох облікових записів та збереженням секретів у системному сховищі;

- розробити модуль штучного інтелекту з архітектурною ізоляцією від основних поштових сценаріїв, за якої його відмова не є фатальною і не порушує штатну роботу решти підсистем застосунку;
- розробити програмний прототип та узагальнити результати його статичного аналізу і багаторівневого тестування.

Об’єкт дослідження - процес розроблення десктопного кросплатформного поштового клієнта.

Предмет дослідження - методи та програмні засоби побудови архітектури такого клієнта мовою C++ із використанням бібліотеки Qt, зокрема: асинхронна взаємодія з протоколами IMAP та SMTP, безпечна автентифікація за стандартом OAuth 2.0 із розширенням PKCE та підтримкою кількох облікових записів, локальне зберігання повідомлень із шифруванням, а також відокремлення модуля штучного інтелекту від основних поштових сценаріїв.

Теоретичне та практичне значення одержаних результатів.

Теоретичним результатом є сформульована референтна архітектура нативного кросплатформного поштового клієнта з адаптацією потоку OAuth 2.0 + PKCE (RFC 8252, RFC 7636) до сценарію десктопного клієнта з кількома обліковими записами і схемою ізоляції допоміжного модуля штучного інтелекту; її можна використовувати у наступних інженерних проєктах подібного класу та у навчальному процесі з дисциплін системного й мережевого програмування, прикладної криптографії та сучасної розробки на C++ і Qt.

Практичним результатом є реалізований у прототипі набір самодостатніх інженерних рішень, придатних до перевикористання у споріднених проєктах: асинхронне мережеве ядро на основі шаблону Reactor (Boost.Asio з OpenSSL 3.x) з безпечним мостом до сигнально-слотового механізму Qt 6; дворівневий локальний шифрований кеш повідомлень з AES-256-GCM; реалізація OAuth-входу з PKCE та узагальнене сховище секретів на основі системної ключниці.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ВИМОГИ ДО СИСТЕМИ

1.1.Протоколи та стандарти електронної пошти

Електронна пошта є федеративною системою, побудованою на відкритих стандартах, що публікуються Internet Engineering Task Force у форматі RFC. Сумісність між клієнтами і серверами різних виробників досягається не через спільну реалізацію, а через дотримання цих стандартів; тому проєктування клієнтського застосунку неминуче починається з фіксації переліку протоколів, які він має підтримувати, і з вибору між альтернативами там, де такий вибір передбачений.

Передавання вихідної кореспонденції виконує Simple Mail Transfer Protocol, визначений у RFC 5321 [13]. Для клієнтського сценарію використовується підмножина SMTP submission (RFC 6409): з'єднання встановлюється до спеціалізованого порту поштового сервера, обов'язково проходить автентифікацію і захищається TLS - через окремий порт із неявним TLS або через STARTTLS-апгрейд згідно з RFC 2595 [14-16]. У клієнтських застосунках використовуються порти 587 (для варіанту з використанням STARTTLS) і 465 (для варіанту з неявним TLS, відновлений RFC 8314); порт 25 призначено для міжсерверної передачі [17]. Сесія підпорядкована послідовному протоколу команда-відповідь з ідентифікацією клієнта через команду EHLO, автентифікацією через команду AUTH, передачею конверта і тіла повідомлення; повна послідовність команд для розглянутого сценарію наведена у розділі 2.4.2. Сам SMTP не передбачає засобів для роботи зі вмістом скриньки на стороні сервера; для цього використовуються окремі протоколи доступу до пошти.

Історично так протоколи два: IMAP та POP3. Для отримання пошти прийнято саме перший протокол стандарту RFC 3501; POP3 не розглядається через несумісність із багатопристроєвою синхронізацією [18]. Принципова для

архітектури клієнта особливість IMAP - двонаправлена довготривала сесія, у якій сервер сам надсилає нетеговані повідомлення (* EXISTS, * EXPUNGE, * FETCH). Для пасивного очікування таких подій клієнт переходить у режим IDLE (RFC 2177) і завершує його командою DONE [19]. Узагальнену UML-діаграму послідовності типового циклу IMAP-сесії з фазою IDLE винесено у Додаток А; повна машина станів сесії, реалізована у клієнті, наведена у розділі 2.4.1.

Окремим корисним механізмом в межах протоколу IMAP є так звана інкрементна синхронізація, яку забезпечують розширення CONDSTORE та QRESYNC (RFC 7162) на основі лічильника MODSEQ і значення UIDVALIDITY папки. Підтримка розширень на боці сервера не обов'язкова, тому клієнт має коректно деградувати до звичайних SELECT/FETCH; алгоритм вибору стратегії і механізм деградації деталізовано у розділі 2.4.1.

Формат повідомлень визначає RFC 5322 (конверт і заголовки) у поєднанні з MIME - RFC 2045/2046 для загальних типів і RFC 2387 для multipart/related з inline-зображеннями через Content-ID [21-24]. Транспортний канал захищено TLS 1.3 (RFC 8446); пріоритет - неявний TLS на окремому порту як безпечніший варіант, ніж STARTTLS-апгрейд (RFC 2595) [16, 25].

Безпечну делеговану автентифікацію забезпечує OAuth 2.0 (RFC 6749) з обов'язковим для нативних застосунків розширенням PKCE (RFC 7636); BCP 212 (RFC 8252) фіксує вимоги до відкриття сторінки входу у системному браузері, перенаправлення на IPv4-адресу петлевого інтерфейсу з ефемерним портом і перевірки певних параметрів, витягнутих з URI [11-12][26]. Передавання отриманого токена в IMAP/SMTP виконується механізмом SASL XOAUTH2 - профілем структури SASL (RFC 4422) для bearer-токенів; формат рядка автентифікації специфіковано в документації провідних поштових провайдерів [27-28]. Таблиця 1.1 систематизує перелічені стандарти за призначенням і за роллю в клієнтському застосунку.

Таблиця 1.1 - Порівняння протоколів IMAP та POP3

Характеристика	IMAP4rev1 (RFC 3501)	POP3 (RFC 1939)
Місце збереження повідомлень	На сервері	На клієнтському пристрої
Підтримка папок і ієрархії	Так	Як правило, відсутня
Прапорці й статуси повідомлень	Підтримуються	Обмежено
Часткове завантаження тіл і вкладень	Так	Зазвичай ні
Багатопристроєва синхронізація	Природна	Ускладнена
Серверний пошук	Так (SEARCH)	Ні
Інкрементна синхронізація	RFC 7162 (CONDSTORE/QRESYNC)	Не передбачена

1.2.Архітектурна модель клієнтського застосунку

Реалізація переліченого вище набору протоколів допускає кілька принципово різних архітектурних рішень. Найпростіший - синхронний підхід, у якому мережева операція виконується у тому самому потоці, що обробляє події графічного інтерфейсу. Для повноцінного поштового клієнта він неприйнятний з чотирьох причин: тривала команда IMAP IDLE блокувала б інтерфейс до приходу будь-якої події; TLS-рукостискання і DNS-резолюція у нестабільних мережах займають секунди до початку передавання даних; підтримка кількох облікових записів вимагає одночасних незалежних IMAP-з'єднань; ШІ- та OAuth-сервіси викликаються паралельно з фоновою синхронізацією.

Технічним вирішенням цих обмежень є асинхронна модель введення-виведення з диспетчеризацією завершень через окремий цикл подій - шаблон Reactor: один потік керує демультиплексором ОС (epoll на Linux, kqueue на macOS/BSD, IOCP на Windows) і викликає зареєстровані обробники у відповідь на готовність дескрипторів, завдяки чому єдиний робочий потік може обслуговувати десятки одночасних з'єднань без виділення потоку на кожне з них [10]. Концептуальну різницю між моделями пропонується дослідити, порівнявши рисунки 1.1 та 1.2.

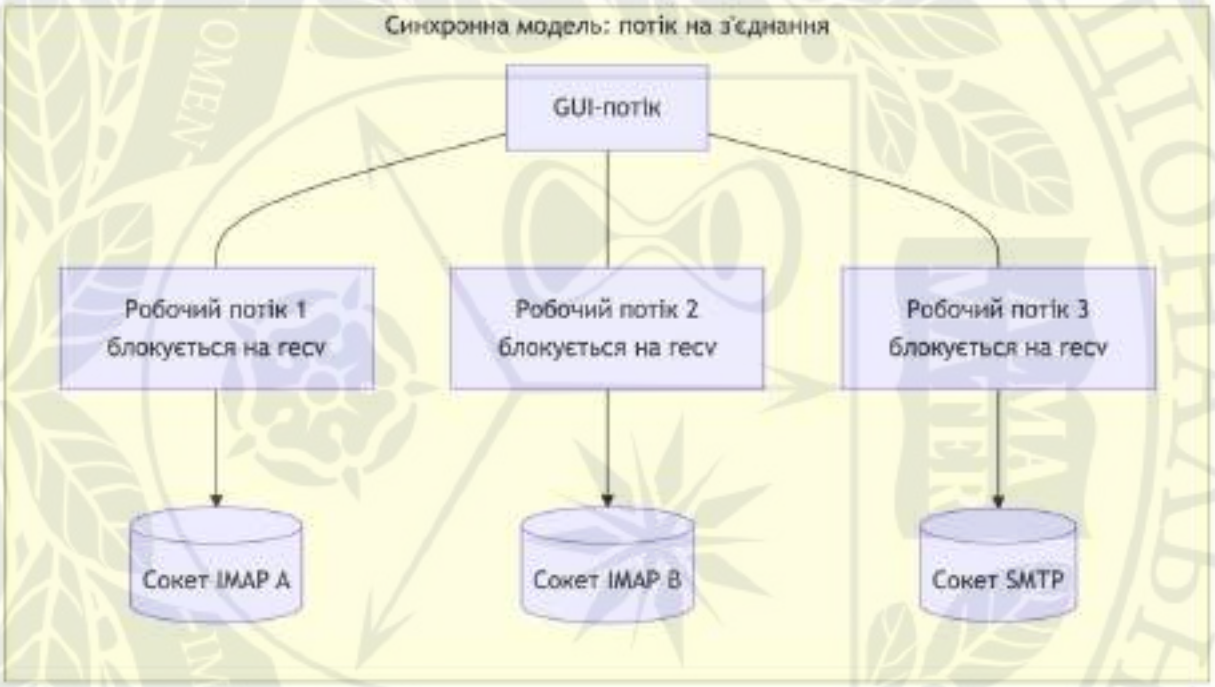

Рисунок 1.1 - Синхронна (потік на з'єднання) моделі мережевого вводу-виводу

Принципова економія - один потік мережевого ядра на N з'єднань (з диспетчеризацією подій від демультиплексора ОС) замість N заблокованих на читанні потоків при синхронній моделі.

Асинхронна модель природно поєднується з трьома суміжними рішеннями: сигнально-слотовим механізмом Qt 6 для безпечної передачі результатів у потік інтерфейсу через черговану доставку, багаторівневою декомпозицією системи для ізоляції протокольних деталей від інтерфейсу і автентифікації від мережевого ядра,

та RAII-керуванням ресурсами C++, що знімає клас помилок витоку сокетів і TLS-сесій [29-32].

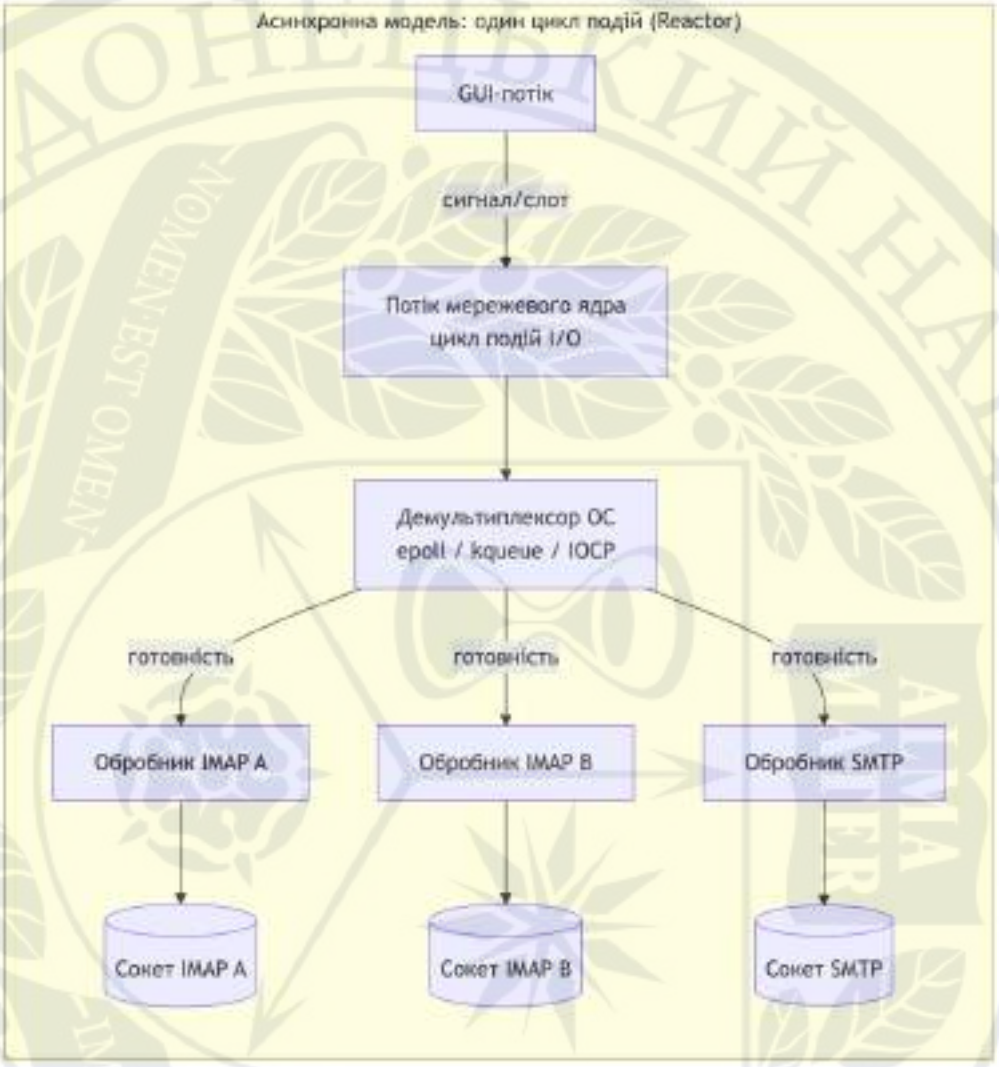

Рисунок 1.2 - Асинхронна моделі мережевого вводу-виводу

Поєднання асинхронної I/O-моделі, чотирирівневого поділу системи і RAII становить мінімально достатній архітектурний каркас для нативного десктопного клієнта, прийнятий за основу проєктування у Розділі 2.

1.3.Огляд архітектури наявних десктопних поштових клієнтів

Характеристику провідних настільних поштових клієнтів подано у двох проєкціях: загальній - за ліцензійною моделлю та цільовими ОС (таблиця 1.2) - і

архітектурній - за основою реалізації, проявом асинхронності та наявністю локального ШІ-модуля (таблиця 1.3).

Таблиця 1.2 - Загальні характеристики розглянутих десктопних клієнтів

Клієнт	Ліцензія	Цільові ОС
Mozilla Thunderbird [33][34]	Open source (MPL)	Windows, Linux, macOS
Microsoft Outlook (новий) [5][6]	Пропрієтарна	Windows, macOS, веб
Apple Mail [3][36][37]	Пропрієтарна	Лише macOS/iOS
Mailspring [7][38][39]	Open core	Windows, Linux, macOS
Geary [40]	Open source (GPL)	Лише Linux/GNOME
Shortwave / Superhuman [8]	Закрита SaaS	Веб-обгортка

Таблиця 1.3 - Архітектурні особливості розглянутих десктопних клієнтів

Клієнт	Основа реалізації	Асинхронність як принцип	ШІ-модуль
Mozilla Thunderbird	XPCOM / C++ / JavaScript	Фрагментарна	Відсутній
Microsoft Outlook (новий)	Гібрид Outlook on the Web	Прихована (веб-основа)	Вбудований (Copilot, не локальний)
Apple Mail	Native Cocoa	Native	Вбудований у платформу

Продовження таблиці 1.3

Mailspring	Electron (Chromium + Node)	Через середовище Node	Відсутній
Geary	Native (Vala / GTK)	Виражена	Відсутній
Shortwave / Superhuman	Веб-стек + тонкий клієнт	Серверно-орієнтована	Через інфраструктуру вендора

Серед розглянутих рішень лише Thunderbird і Mailspring одночасно покривають усі три операційні системи з єдиної кодової бази, проте обидва несуть значне ресурсне навантаження - XPCOM-спадщину і Electron-стек відповідно [4,7]. Жоден із продуктів не поєднує одночасно відкриту нативну реалізацію, виражену асинхронну архітектуру, повну кросплатформеність і архітектурно ізольовану локальну інтеграцію штучного інтелекту - саме така комбінація є архітектурним об'єктом проєктування у поточній роботі

1.4.Цільова аудиторія та сценарії використання

З огляду на сформульовану у вступі нішу мінімалістичного нативного клієнта з архітектурно ізольованим ШІ-модулем виокремлено три цільові групи користувачів (таблиця 1.4) і п'ять базових сценаріїв використання (таблиця 1 Додаток Б), що далі трасуються на функціональні вимоги з розділу 1.5.

Категорія «масовий корпоративний користувач» не розглядається - її покривають наявні промислові рішення.

Базові сценарії використання зібрані у таблицю Б.1 (Додаток Б). Сценарії UC-01 - UC-05 трасують потреби користувачів у формальні технічні результати; кожна

функціональна вимога з розділу 1.5 пов’язана принаймні з одним сценарієм, і навпаки.

Таблиця 1.4 - Цільові групи користувачів

Категорія	Контекст використання	Визначальні потреби
Приватні користувачі	Особисте та академічне листування з кількох облікових записів	Простота інтерфейсу, передбачуване споживання ресурсів, безпечне зберігання облікових даних
Професійні користувачі	Щоденне ділове листування помірного обсягу	Чутливість інтерфейсу під час фонової синхронізації, стабільність на тривалих сесіях
Технічно орієнтовані користувачі	Зацікавлення в архітектурі та поведінці клієнта; чутливість до споживання ресурсів	Прозорість поведінки, можливість контрольованого ввімкнення або вимкнення допоміжних функцій (зокрема ШІ)

1.5.Функціональні та нефункціональні вимоги

Функціональні вимоги формулюються у вигляді переліку із зазначенням сценарію використання та джерела - нормативного документа або обґрунтування у попередніх підрозділах. Класифікація нефункціональних вимог ґрунтується на моделі якості програмного забезпечення ISO/IEC 25010, що охоплює продуктивність, надійність, безпеку, сумісність і супроводжуваність [41]. Перелік функціональних вимог наведено у таблиці Б.2 (Додаток Б), нефункціональні - у

таблиці Б.3 (Додаток Б), - у згрупуванні за категоріями моделі якості ISO/IEC 25010 [41].

Сформульовані функціональні і нефункціональні вимоги є прямою основою для проєктних рішень у розділі 2: кожне з них супроводжуватиметься явним посиланням на конкретну вимогу, яку воно покриває, що забезпечує наскрізну простежуваність від потреб користувача до архітектурної реалізації.

РОЗДІЛ 2. ПРОЄКТУВАННЯ СИСТЕМИ

2.1.Обґрунтування вибору технологічного стеку

Критерії вибору технологій безпосередньо випливають з нефункціональних вимог з розділу 1.5. Додатково враховуються детермінований контроль над ресурсами, зрілість документації та ліцензійна придатність бібліотек для навчально-дослідницького проєкту.

Як основну мову обрано C++23: ідіома RAII забезпечує детерміноване звільнення мережевих ресурсів (TCP-сокети, TLS-сесії, буфери) незалежно від шляху виходу з функції, а нові засоби стандарту - корутини, концепти і std::expected - релевантні для асинхронного програмування і обробки помилок [32]. Аргументи для відхилення альтернативних платформ зведено у таблиці 2.1.

Таблиця 2.1 - Альтернативні технологічні платформи та причини їх відхилення

Альтернатива	Технологія	Причина відхилення
JavaScript /TypeScript	Electron	Вбудований браузерний рушій Chromium у дистрибутиві, що несе за собою значне споживання оперативної пам'яті і ресурсів процесора (NFR-01, NFR-11)
Java / Kotlin	JavaFX, Compose Multiplatform	Накладні витрати JVM на запуск і виконання знижують сприйняття «нативності»
C#	.NET MAUI, Avalonia	Придатна, але не дозволяє предметно показати роботу з низькорівневими механізмами
Python	PyQt/PySide	Втрати продуктивності через інтерпретацію та GIL несумісні з NFR-01 і NFR-02

Серед фреймворків графічного інтерфейсу обрано **Qt 6** як єдиний, що покриває всі три цільові ОС «з коробки» і має сигнально-слотовий механізм, який

природно інтегрується з подієвою моделлю мережевого ядра: завершення мережевої операції передається у потік графічного інтерфейсу через черговану доставку сигналу, унаслідок чого тривала операція I/O не блокує перемальовування вікна (NFR-01, NFR-02) [29]. **Boost.Asio** обрано як кросплатформну реалізацію шаблону Reactor поверх epoll/kqueue/IOCP з підтримкою C++20-корутин і прямою інтеграцією з OpenSSL; альтернативи відхилено через менш ергономічне C-API (libuv), недостатню гнучкість для нестандартних IMAP-сесій (POCO Net) або трудомісткість і ризик безпеки самостійної реалізації поверх select/epoll. Решту компонентів стеку (TLS-бекенд, MIME-обробка, захищене сховище, персистентний кеш, серіалізація JSON, локалізація) обрано за прямою відповідністю окремим функціональним або нефункціональним вимогам; зведене бачення стеку з прив'язкою кожного шару до конкретних вимог подано в таблиці 2.2.

Таблиця 2.2 - Технологічний стек прототипу

Шар	Обрана технологія	Версія / стандарт	Покриває вимоги
Мова	C++	C++23	NFR-03, NFR-07, NFR-08
Графічний інтерфейс	Qt	6.x	NFR-01, NFR-03, NFR-10
Збірка	CMake	3.21+	NFR-08
Асинхронне I/O	Boost.Asio	1.83+	NFR-02, FR-11
TLS	OpenSSL	3.x	NFR-04
HTTP / REST	Мережевий модуль Qt	Qt 6	FR-10
OAuth 2.0 + PKCE	Власна реалізація на Qt + QtKeychain	RFC 6749, RFC 7636	FR-02, NFR-05, FR-12
Сховище секретів	QtKeychain	-	NFR-05, FR-12
MIME	GMime	3.0	FR-04, FR-05

Продовження таблиці 2.2

Персистентний кеш	SQLite	3.x	FR-07, NFR-05, NFR-11
JSON (графічний клієнт)	QJsonDocument	Qt 6	FR-02, FR-10, FR-12
JSON (CLI-утиліта)	nlohmann/json	3.x	-
Локалізація	Qt LinguistTools	Qt 6.x	NFR-10

Кожна вимога з груп безпеки, асинхронності та кросплатформеності закривається у стеку щонайменше одним компонентом-«власником», що знімає неявну залежність від «загальної якості коду» без конкретної реалізаційної опори. Усі обрані технології мають відкриту або сумісну з некомерційним використанням ліцензію, що знімає юридичні ризики для навчально-дослідницького проєкту.

2.2. Багаторівнева архітектура застосунку

Архітектура застосунку базується на чотирьох взаємодоповнювальних принципах: багаторівневий поділ за функціональними відповідальностями, асинхронна неблокуюча модель введення-виведення, поділ виконання на два незалежні цикли подій (графічний і мережевий) із чітко визначеним мостом між ними, та виокремлення підсистеми безпечної автентифікації і модуля ШІ як самостійних компонентів з ізольованими інтерфейсами [43].

Систему поділено на чотири рівні з односпрямованим графом залежностей. **Рівень подання** - головне вікно, діалоги і віджети - жодної мережевої операції безпосередньо не виконує, делегуючи їх нижчим шарам. **Рівень координації**

(фасад сценаріїв) поєднує інтерфейс з доменними сервісами, інкапсулюючи сценарії входу за механізмом OAuth, синхронізації, надсилання та виклику ШІ-функцій і керуючи переходами станів інтерфейсу під час виконання мережевих операцій. **Рівень доменних сервісів** містить три незалежні модулі - авторизації (OAuth, PKCE-сесія, сховище токенів, реєстр облікових записів), пошти (IMAP-сесія, черга команд, MIME-парсер) і штучного інтелекту (REST-клієнт провайдера моделі); зв'язки між цими модулями принципово відсутні, тому відмова одного з них не впливає на стан інших (NFR-09). **Рівень інфраструктури** містить обгортку циклу подій мережевого ядра у фоновому потоці, TLS-контекст, низькорівневі реалізації IMAP/SMTP-протоколів та утиліти безпечної підготовки HTML. Поділ зафіксовано окремими цілями CMake, що технічно унеможливлює циклічні залежності між шарами і підтримує NFR-07. Розкладку рівнів та допустимі напрямки залежностей подано на рисунку 2.1.

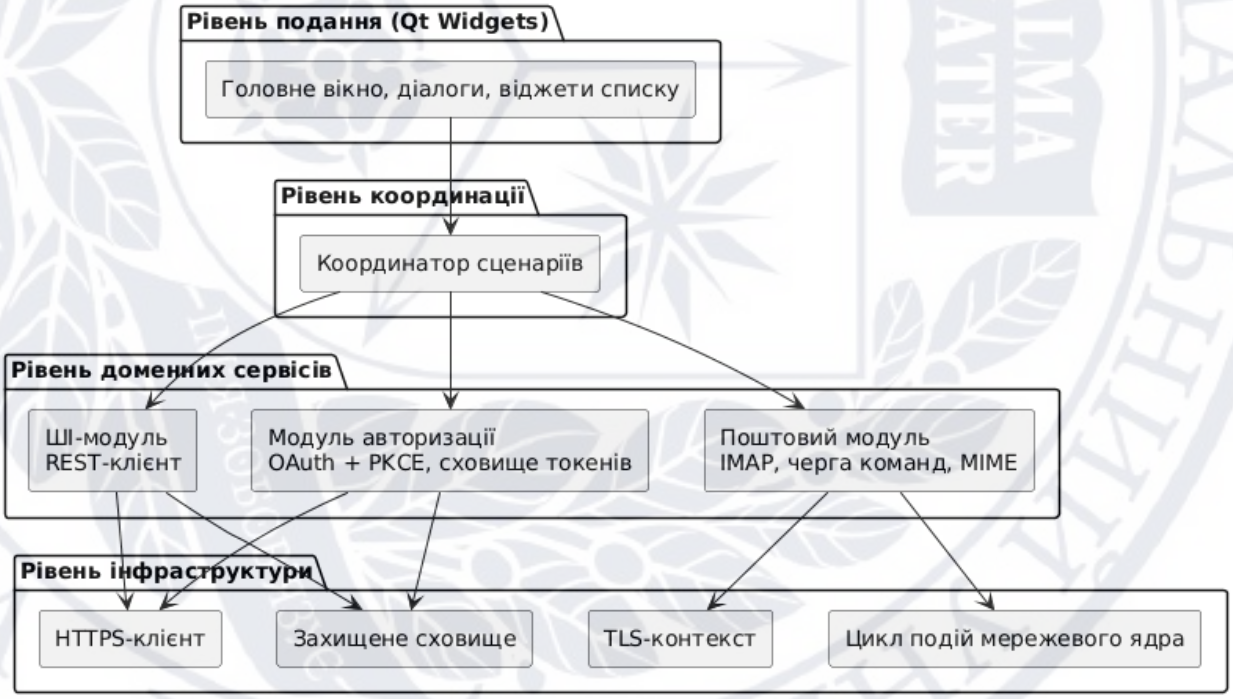

Рисунок 2.1 - Багаторівнева архітектура застосунку

Виконання спирається на два незалежні цикли подій: цикл графічного інтерфейсу обробляє системні події вікон і сигнали Qt, а цикл мережевого ядра за

моделлю Reactor - завершення асинхронних мережевих операцій. Завершення операції в потоці мережевого ядра передається у потік графічного інтерфейсу через сигнали Qt із чергованою доставкою, що серіалізує виклик у черзі подій цільового об'єкта (NFR-01).

Дисципліну потоків і інверсію залежностей часу компіляції через концепти C++20 деталізовано у розділах 2.3 і 3.8 відповідно.

2.3. Декомпозиція на компоненти та модель потоків виконання

Виходячи з раніше сформованих рівнів, застосунок розкладено на близько двох десятків логічних компонентів, що групуються навколо трьох доменних задач - авторизації, роботи з поштою і взаємодії з ШІ; решта виконує допоміжну, інфраструктурну функцію. Ключові з них з прив'язкою до відповідальності наведено в таблиці 2.3.

Реєстр облікових записів виконує вимогу FR-12 і навмисно винесений за межі захищеного сховища: у ньому зберігаються лише неконфіденційні метадані (адреса, метод автентифікації, серверні підказки), що дозволяє керувати переліком акаунтів окремо від автентифікаційних даних.

Таблиця 2.3 - Ключові компоненти системи

Компонент	Відповідальність
Головне вікно і координатор сценаріїв	Кореневий об'єкт інтерфейсу; оркестрація сценаріїв входу/синхронізації/надсилання/виклику ШІ
Модуль авторизації (з PKCE-сесією і локальним приймальником)	Сценарій OAuth, оновлення та валідація токенів
Сховище токенів і реєстр облікових записів	Per-account-серіалізація токенів у системній ключниці; неконфіденційний індекс акаунтів

Продовження таблиці 2.3

Контролер IMAP-сесії і черга операцій	Цикл IDLE, насос команд, коалесценція повторних запитів
Токенізатор і парсер IMAP (zero-copy)	Виокремлення лексем як std::string_view над буфером прийому
Парсер MIME (адаптер на GMime)	Розбір MIME-структури, декодування RFC 2047 і Content-Transfer-Encoding
Дворівневий кеш тіл повідомлень	LRU в оперативній пам'яті + персистентний SQLite з AES-256-GCM
ШІ-сервіс і REST-провайдер	Координація викликів, реалізація протоколу провайдера моделі
Токенізатор і парсер IMAP (zero-copy)	Виокремлення лексем як std::string_view над буфером прийому

Сховище токенів і сховище API-ключа замикаються на той самий захищений бекенд, що зменшує фрагментацію секретів. Зв'язки «доменний сервіс ↔ доменний сервіс» принципово відсутні, що відповідає класичним принципам розв'язаних модулів з каталогу шаблонів проєктування; у тілі розділу обмежимося поданням моделі потоків, оскільки саме вона визначальна для архітектурних інваріантів [30].

У застосунку діють два основні потоки виконання. Потік графічного інтерфейсу, що створюється під час старту, виконує цикл подій вікон і обробляє всі операції з віджетами і HTTPS-запити через мережевий модуль Qt (останній внутрішньо використовує власні робочі потоки, але цю деталь ізолює сам фреймворк). Потік мережевого ядра, створюваний обгорткою циклу подій, відповідає за всі мережеві IMAP- і SMTP-операції - TCP, TLS, читання/запис буферів, тайм-аути. Серіалізацію доступу до IMAP-сесії, що не є потокобезпечною, забезпечує серіалізатор виконання boost:asio::strand - механізм Boost.Asio, який гарантує одночасне виконання не більше ніж одного обробника. Усі IMAP-команди,

з якого б потоку їх не ініціювали, плануються саме на цей серіалізатор, унаслідок чого потреба у явних м'ютексах усувається. Передача результату назад у графічний інтерфейс виконується через черговану доставку сигналу Qt у цільовий потік.

Передача керування асиметрична: потік інтерфейсу не звертається до IMAP/SMTP-сесій безпосередньо, а лише ставить операції на серіалізатор; результати повертаються чергованими сигналами. HTTPS-клієнт є самостійним каналом, керованим внутрішніми потоками Qt, що дає ШІ-сервісу і модулю авторизації незалежний від мережевого ядра шлях до зовнішнього світу і посилює ізоляцію підсистем (NFR-09). Для асинхронних API доменних сервісів передбачено повернення результату через зворотний виклик-параметр у «своєму» потоці, що знімає зі споживачів турботу про потокову безпеку. Узагальнену схему потоків і шляхів передачі керування подано на рисунку 2.2.

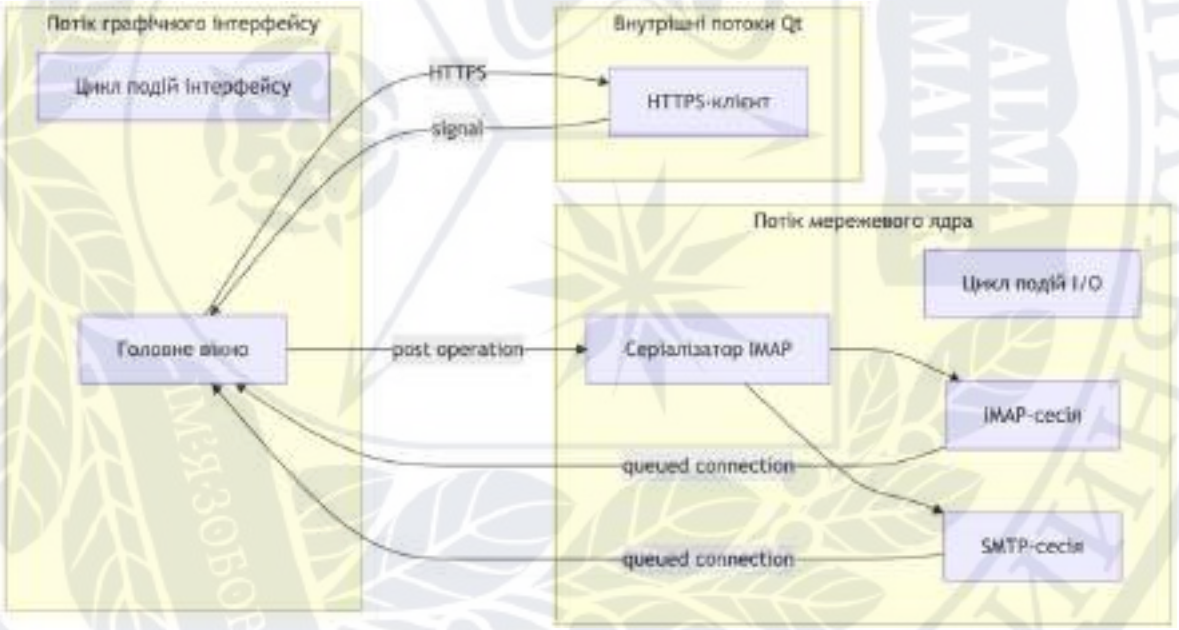

Рисунок 2.2 - Модель потоків і взаємодія через серіалізатор

2.4. Проєктування мережевої підсистеми

Мережева підсистема охоплює три класи з'єднань - IMAP-сесію, SMTP-submission і HTTPS-канал до провайдера OAuth та ШІ-сервісу - усі захищені TLS

1.3 з допустимим відкатом до TLS 1.2 і обов'язковою перевіркою серверного сертифіката (NFR-04) [25]. Стандартні послідовності протокольних команд і типові порти описані у розділі 1.1; у цьому підрозділі фокус - на тому, як саме ці протоколи лягають в асинхронну модель з розділу 2.3.

2.4.1. IMAP-сесія

IMAP-сесія моделюється як взаємодія трьох сутностей. Зовнішнім фасадом виступає сервіс поштової сесії, який створює серіалізатор виконання boost::asio::strand поверх циклу подій мережевого ядра і надає коду інтерфейсу єдину точку входу: будь-яка операція, з якого б потоку її не ініціювали, виконується на серіалізаторі і не перетинається з іншими IMAP-командами. Виконавчим компонентом є контролер IMAP-сесії, що підтримує цикл IDLE, реалізує конвеєр команд, перехоплює нетеговані повідомлення сервера і керує фазами з'єднання. Між фасадом і контролером розташована черга IMAP-операцій з коалесценцією запитів того самого типу: вибір нової поштової скриньки витирає чергу попередніх операцій, а повторні запити на завантаження одного й того самого листа дедублюються до одного запису. Це усуває цілий клас регресій, у якому швидкі повторні натискання користувача утворювали неузгоджені серії звернень до сервера. Алгоритм обробки команди такий: інтерфейс формує опис операції і передає у фасад; фасад додає її в чергу і повідомляє контролер; той за потреби тимчасово зупиняє IDLE надсиланням DONE, послідовно виконує команду через клієнтський рівень IMAP і повертається в IDLE.

Внутрішній стан контролера описується явним переліком фаз, як показано на рисунку 2.4.1; перехід між ними координується атомарними прапорцями і виключає ситуацію, коли команда із черги починає виконуватися до коректного завершення IDLE. Окремої згадки заслуговує тривале очікування у IDLE: штатний таймаут читання поверх захищеного з'єднання (порядок десятків секунд) розриває простій,

породжуючи постійні цикли перепідключення. Цю проблему розв'язано RAII-токеном «довготривале очікування», який тимчасово підвищує таймаут читання потоку до 24 годин і повертає попереднє значення в момент знищення токена. Скасування очікування виконується явним сигналом скасування boost::asio::cancellation_signal, а не зміною таймаутів, що уніфікує реакцію на DONE-команду користувача та на природне закінчення вікна IDLE. Узагальнену машину станів контролера, що поєднує фазу встановлення з'єднання та пост-авторизаційний цикл, наведено на рисунку 2.3.

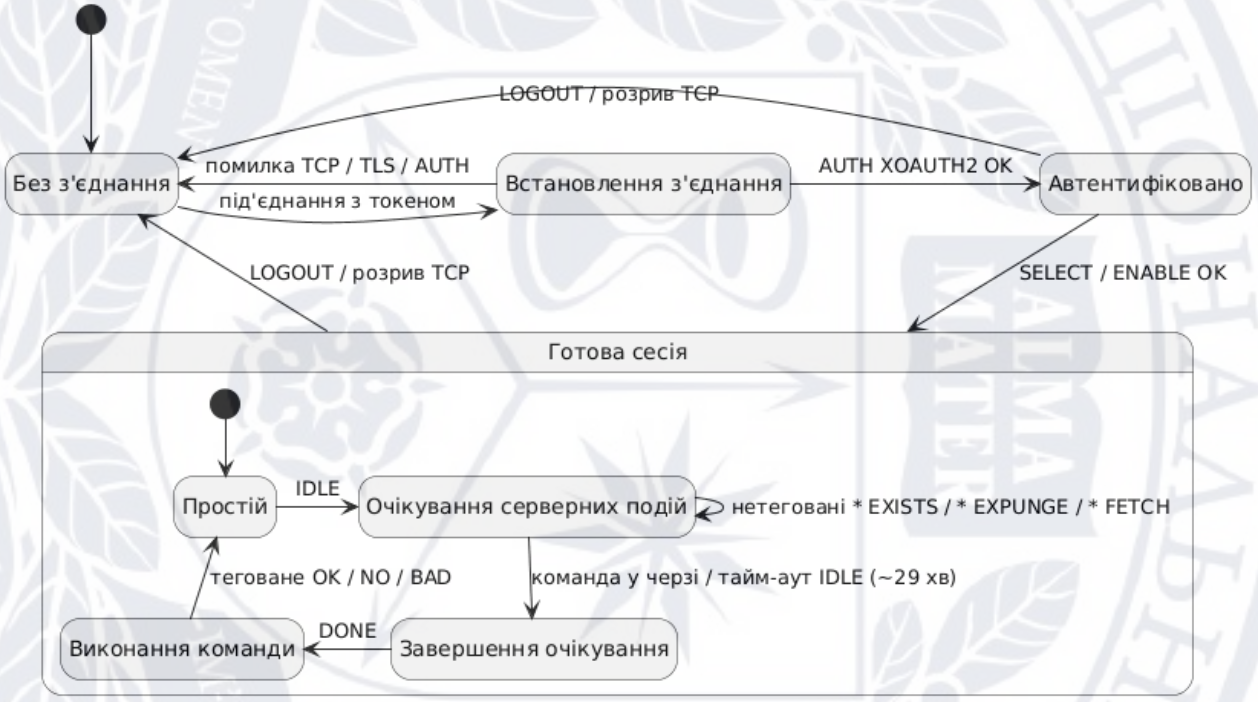

Рисунок 2.3 - Машина станів IMAP-сесії

Зовнішні переходи (втрата з'єднання, припинення дії токена) приводять у стан «Без з'єднання» з будь-якого пост-авторизаційного стану, тоді як внутрішні переходи всередині складеного стану «Готова сесія» керуються лише появою/завершенням команд і нотифікаціями сервера - завдяки такому розділенню «нештатні» ситуації обробляються окремо від звичайного робочого циклу. Завдяки коалесценції черги і RAII-токену IDLE архітектура допускає подальше нарощування функціональності (паралельні активні скриньки на акаунт, фоновий префетч) без зміни зовнішнього контракту.

Інкрементна синхронізація скриньок за розширеннями CONDSTORE/QRESYNC є допоміжною оптимізацією, спрямованою на усунення непотрібного трафіку: для скриньок із тисячами повідомлень повне читання `FETCH 1:*` на кожне відкриття є неприйнятним, тому клієнт зберігає у локальному кеші пару UIDVALIDITY/HIGHESTMODSEQ і запитує лише зміни [20]. Архітектурна інваріанта: локальний кеш є **єдиним джерелом правди** для стану синхронізації, тож рішення про форму команди SELECT (з QRESYNC, з CONDSTORE або без розширень) ухвалюється виключно на основі персистованих у кеші метаданих і оголошених сервером CAPABILITY.

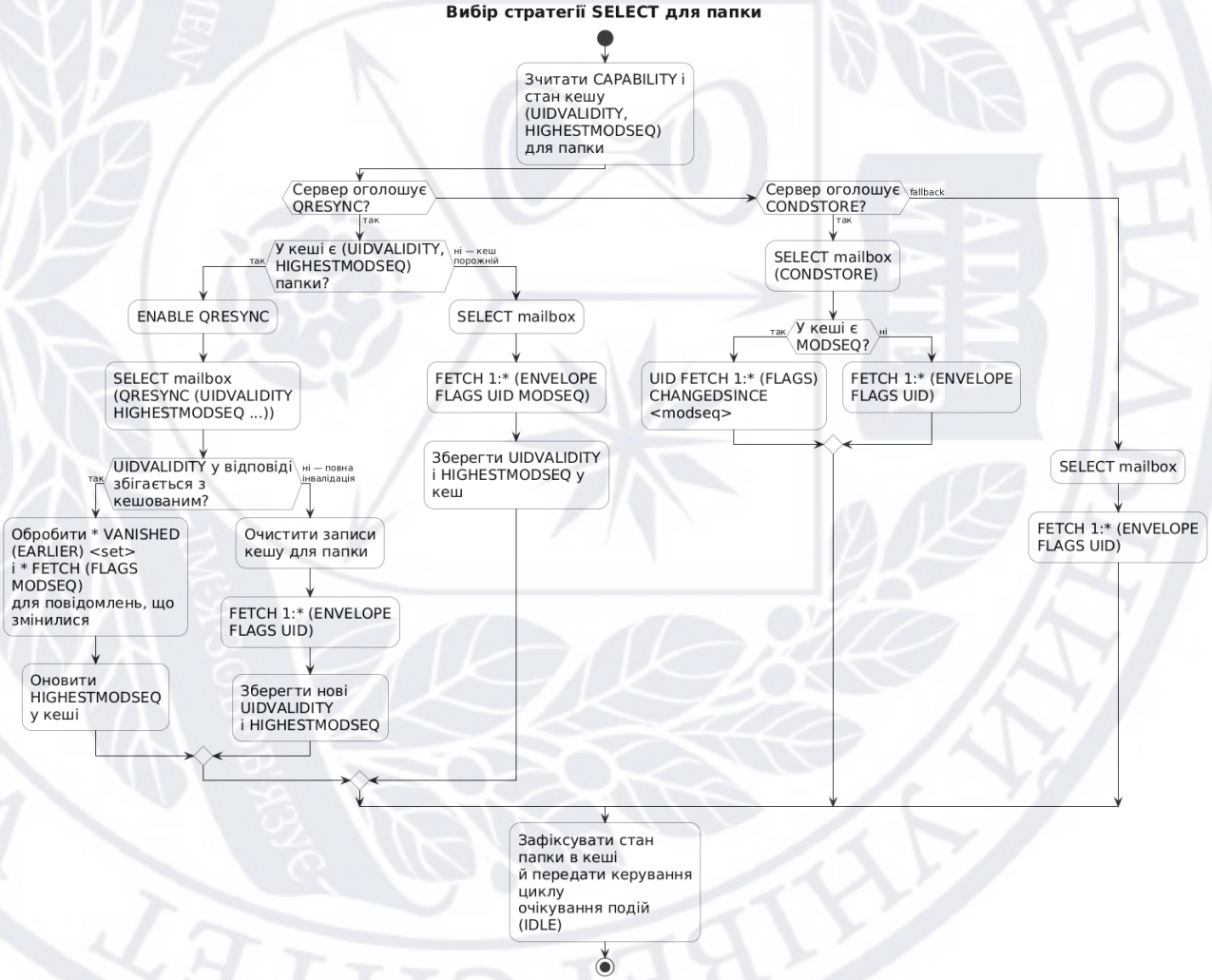

Рисунок 2.4 - Алгоритм вибору стратегії SELECT для папки

Усі граничні умови - відсутність розширень на сервері, втрата метаданих у кеші, зсув UIDVALIDITY - автоматично деградують до повного SELECT/FETCH, що гарантує сумісність із серверами поза Dovecot-родиною. Обробку нетегованих відповідей типу `* VANISHED` реалізовано для поширених форматів, які використовують основні провайдери (зокрема `* VANISHED (EARLIER) <set>` і `* VANISHED <set>` з UID-послідовностями); повна синтаксична сумісність парсера з усіма формами RFC 7162 є напрямком подальшого розвитку (див. розділ 3.13). Узагальнений алгоритм вибору стратегії та обробки VANISHED подано на рисунку 2.4.

2.4.2. SMTP-надсилання та перехресні аспекти

SMTP-надсилання реалізовано як короткоживуча сесія: на кожен лист встановлюється нове TLS-з'єднання на submission-порт, виконується стандартна послідовність команд (див. розділ 1.1) з автентифікацією через AUTH XOAUTH2 зі свіжим токеном доступу. Така схема дає вузьке вікно використання токена, усуває проблему застарілих з'єднань, які SMTP-сервер часто закриває через неактивність, і спрощує обробку помилок: відповіді на MAIL FROM / RCPT TO / DATA однозначно прив'язані до конкретного повідомлення, тож не потрібно зберігати стан між викликами. Повну діаграму послідовності взаємодії клієнта з submission-сервером наведено у Додатку В. Прикладною деталлю є те, що свіжий токен запитується ще до встановлення TCP-з'єднання, а не безпосередньо в момент очікування AUTH: це дозволяє у разі неуспішного оновлення показати модальний діалог повторного входу до того, як буде змарновано час на TLS-рукостискання.

Для всіх асинхронних мережевих операцій встановлюються явні тайм-аути на таймерах циклу подій; при спрацюванні з'єднання коректно закривається, ресурси звільняються через RAII, а у чергу подій інтерфейсу додається подія помилки. Стратегія повторних спроб застосовується лише там, де це безпечно: після

невдалого старту IDLE контролер робить одну фіксовану паузу 5 с і виконує повторний старт; при тихому оновленні токена доступу - одна повторна спроба з оновленим токеном; для SMTP-надсилання і деструктивних дій користувача (видалення, переміщення листа) автоматичне повторення не виконується - натомість пропонується явне підтвердження через інтерфейс.

Усі IMAP- і SMTP-команди логуються через потокобезпечний синхронний логер мережевого ядра. Вибір синхронної моделі - попри асинхронність самого ядра - продиктований природою діагностичного контракту: журнал найбільш корисний саме у момент аварійного завершення, тоді як записи з внутрішньої черги асинхронного логера, ще не скинуті в приймач, разом з процесом гинуть. Принцип нерозкриття секретів витримано на рівні самого протокольного шару: спільний базовий клас мережевих клієнтів пропускає кожен клієнтський і серверний рядок через фільтри редакції, що виявляють і затирають контент чутливих команд LOGIN, AUTHENTICATE, AUTH, аргумент XOAUTH2 та base64-токени у рядках `+ <token>` (IMAP) і `334 <base64>` (SMTP). Це знімає з авторів окремих викликів обов'язок дотримуватися «дисципліни редакції» і робить безпеку журналу властивістю самого транспорту. Додатково всі поля команд, що формуються з даних користувача (пошуковий рядок, ім'я скриньки, прапорці), проходять перевірку відсутності CR/LF, що блокує CRLF-ін'єкції у IMAP/SMTP-команди ще на рівні серіалізатора - раніше, ніж байти потрапляють у мережу. Модульні тести валідатора покривають п'ять негативних кейсів - окремий \r, окремий \n, повну послідовність \r\n (наприклад, INJECTED\r\nMAIL FROM:<x@y> як спробу підмішати додаткову SMTP-команду), вбудований NUL-байт і перевірку формату повідомлення про помилку - у поєднанні з чотирма позитивними кейсами (порожній рядок, друкований ASCII, табуляція і довгі коректні значення).

2.5.Підсистема безпечної автентифікації та підтримка кількох облікових записів

Підсистема автентифікації реалізує сценарій делегованої авторизації за схемою Authorization Code + PKCE і пов'язує її з подальшим використанням токенів у IMAP/SMTP-сесіях через SASL XOAUTH2 [12][26][27]. Структурно вона відокремлена від мережевого ядра і складається з координатора авторизації, PKCE-сесії, локального приймальника перенаправлення OAuth, сховища токенів поверх системної ключниці та реєстру облікових записів, що виконує вимогу FR-12. Ідентифікатор клієнта і необов'язковий клієнтський секрет завантажуються із зовнішнього джерела і навмисно не зберігаються у вихідному коді.

2.5.1. Сценарій авторизації OAuth 2.0 з PKCE

Концептуальний потік авторизації подано на рисунку 2.5. Метод PKCE зафіксовано як S256, а метод plain свідомо не підтримується: plain передає code_verifier без хешування (code_challenge == code_verifier), тож перехоплення авторизаційного запиту негайно розкриває секрет і дозволяє обміняти код на токени, тоді як S256 ховає verifier за односторонньою функцією SHA-256. Сам code_verifier генерується за допомогою КСГПВЧ з довжиною 43 - 128 символів у base64url-алфавіті відповідно до RFC 7636 §4.1, а code_challenge обраховується, як base64url(SHA-256(verifier)) [12]. Параметр state виконує доповнювальну функцію - прив'язує авторизаційну відповідь до конкретної ініціативи цього застосунку - оскільки основний захист від перехоплення на нативному клієнті забезпечує сам механізм PKCE з доведенням володіння секретом code_verifier.

Адресою повернення служить http://127.0.0.1:<port> з ефемерним портом, який ОС видає при створенні сокета у локальному приймальнику - варіант, рекомендований для нативних застосунків у RFC 8252; збіг перенаправлення

виконується не за шляхом URL, а за наявністю у запиті очікуваних параметрів code та state [11]. Сам вхід відбувається у системному браузері, а не у вбудованому вікні: клієнт у принципі не має технічної можливості перехопити облікові дані користувача. Обмін коду на токен виконується POST-запитом з Content-Type: application/x-www-form-urlencoded і параметрами grant_type=authorization_code, code, code_verifier, client_id, redirect_uri - у строгій відповідності з RFC.

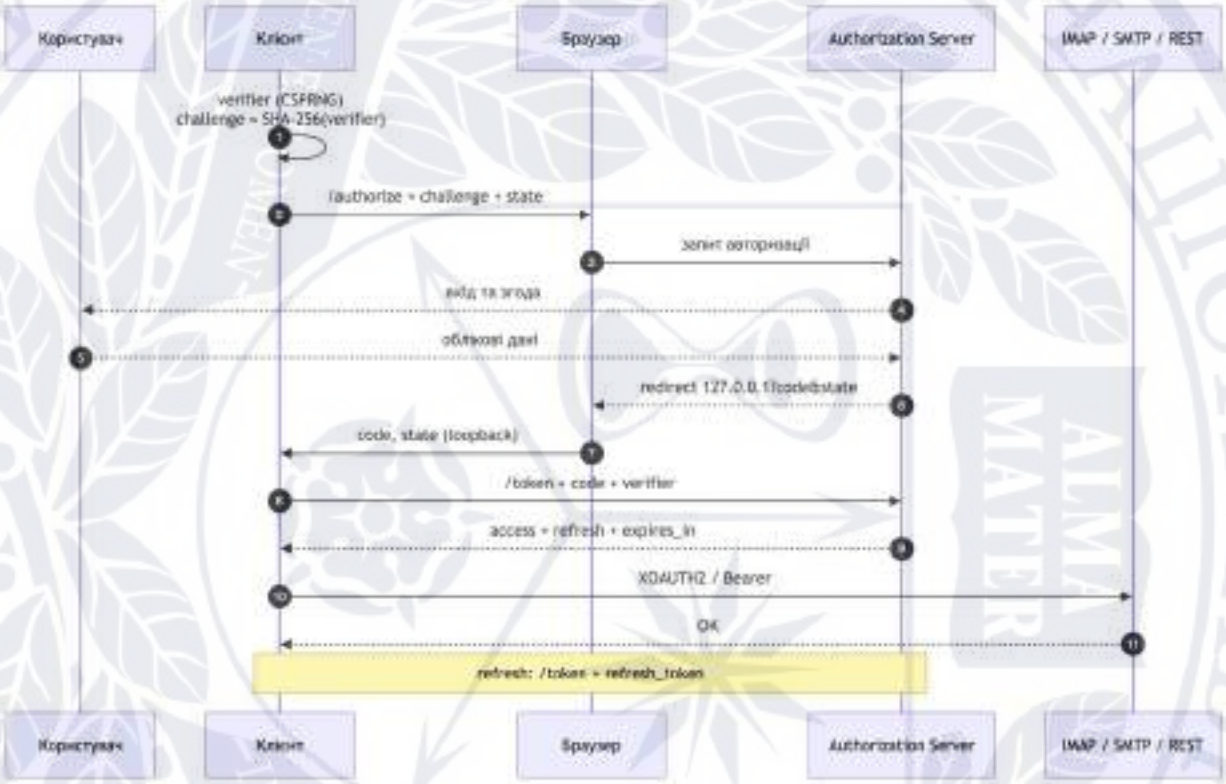

Рисунок 2.5 - UML-діаграма послідовності виконання потоку OAuth 2.0 Authorization Code + PKCE

Логіка отримання дійсного токена побудована як прості правила послідовного вибору: якщо токен доступу присутній і не прострочений - зворотний виклик отримує його без жодних мережевих операцій; якщо токен прострочився, але є токен оновлення - виконується POST запит на кінцеву точку отримання токенів з параметром grant_type=refresh_token, новий запис зберігається у захищеному сховищі; якщо оновлення не вдається - інтерфейс отримує сигнал про необхідність повторного входу. Усі звернення виконуються виключно через TLS: контекст створюється у режимі tls_client, діапазон допустимих версій звужено до

TLS 1.3/1.2 через SSL_CTX_set_min_proto_version, що технічно унеможливлює відкат до SSL/TLS 1.0/1.1. Перевірка ланцюжка сертифікатів через boost::asio::ssl::verify_peer доповнена обов'язковою перевіркою імені хоста (boost::asio::ssl::host_name_verification) разом із SNI; самопідписаних сертифікатів або послаблень перевірки у коді не передбачено.

2.5.2. Зберігання токенів та підтримка кількох облікових записів

Токени зберігаються через узагальнене захищене сховище, що інкапсулює доступ до системного сховища секретів цільової ОС - macOS Keychain, Windows Credential Manager або Linux Secret Service через QtKeychain; токени шифруються самою ОС, а доступ до них контролюється правами процесу і користувача (NFR-05) [46]. Запис у JSON-форматі містить токен доступу, токен оновлення, момент припинення дії, тип, область доступу і адресу електронної пошти користувача. Момент припинення дії клієнт обчислює сам як «поточний час плюс expires_in мінус буфер безпеки 60 с»: цей запас запобігає використанню токена, що ось-ось стане недійсним, біля межі прострочення. Адреса email, потрібну для SASL XOAUTH2, у поточній збірці витягується з ID-токена OpenID Connect (RFC 7519): із трисекційного id_token декодується середній base64url-сегмент, з нього парситься JSON і поле email копіюється у запис токена [64]. Послідовність оновлення токена з 60-секундним буфером безпеки наведена у Додатку Ґ (рисунок Ґ.2).

Для виконання FR-12 ключем розміщення токенів у захищеному сховищі слугує не фіксована константа, а параметризований префікс з адреси користувача (oauth_tokens:<email>); в одному екземплярі сховища без конфліктів співіснують записи довільної кількості акаунтів. Аналогічна стратегія застосована й до сховища паролів для тих провайдерів, де OAuth недоступний.

Тоді як токени і паролі залишаються виключно у захищеному сховищі, супровідні відомості про самі облікові записи - адреса, обраний метод

автентифікації, серверні підказки IMAP/SMTP, мітка активного на даний момент облікового запису - виносяться в окремий, навмисно неконфіденційний реєстр на базі QSettings. Таке розділення зумовлене насамперед мінімізацією наслідків компрометації: зловмисник, що отримав доступ до файлу налаштувань, дізнається лише про факт існування переліку акаунтів та про їхні адреси, але не отримує жодного автентифікаційного матеріалу для жодного з них, що узгоджується з принципом мінімізації даних з літератури з проєктування безпечних систем і з вимогами законодавства про захист персональних даних [45].

Перемикання активного облікового запису потребує коректного закриття поточної сесії, аби запобігти просочуванню даних: зупиняється цикл IDLE, розривається TCP-з'єднання IMAP, очищаються токени в оперативній пам'яті, скидаються локальні кеші, і лише після цього розпочинається повторна ініціалізація сесії для цільового акаунта. Послідовність взаємодії компонентів при перемиканні наведено у Додатку Ґ (рис. Ґ.2).

Явний вихід з облікового запису, на відміну від перемикання, видаляє запис із реєстру і відповідний секрет у захищеному сховищі (секрети інших акаунтів зберігаються); додатково виконується best-effort POST-запит на revocation-кінцеву точку провайдера (RFC 7009) з token_type_hint=refresh_token, аби дозвіл серверного зберігача refresh_token припинявся синхронно з локальним. Запит не блокує вихід з облікового запису і не сповіщає користувача про мережеві помилки, оскільки локальне очищення вже відбулося; для провайдерів без RFC 7009 підтримки (наприклад, Microsoft Identity для refresh_token) ця гілка є no-op, а явний sign-out лишається ефективним лише локально.

Принципово важливо, що перемикання - це не повторне створення екземплярів компонентів, а узгоджене переналаштування вже наявних: сховище токенів змінює лише префікс ключа, модуль авторизації - лише асоційовану адресу, поштова сесія - лише цільовий акаунт. Завдяки цьому FR-12 виконується

локалізованою зміною, що не зачіпає ані мережевої підсистеми, ані машини станів IMAP-сесії з розділу 2.4.1.

2.5.3. Вибір цільового провайдера пошти

Описаний потік OAuth 2.0 з PKCE працює лише за умови, що поштовий провайдер водночас виступає сервером авторизації за RFC 6749 і що сам застосунок попередньо в нього зареєстрований [26]. Серед публічних провайдерів, які надають доступ за OAuth до IMAP/SMTP, реальних альтернатив фактично дві - Google і Microsoft. Як основну цільову платформу обрано Google за чотирма критеріями: поширеність Gmail у поєднанні з Google Workspace; повна підтримка IMAP4rev1 з IDLE і SASL XOAUTH2; документована профільна реалізація RFC 8252 з прийомом відповіді на петлевий інтерфейс без реєстрації фіксованого порту; спільна точка керування з модулем ШІ (розділ 2.6), де і токен XOAUTH2 для пошти, і ключ доступу для Gemini видаються в межах однієї консолі [11][28][47][48]. Підтримку Microsoft збережено як паралельну гілку конфігурації: ті самі C++-структури в коді, але з іншими значеннями кінцевих точок, дозволяють перейти на Azure AD App Registration без змін у коді модуля авторизації, що є наслідком принципу відкритості/закритості [44]. Реєстрація проєкту у Google Cloud Console передбачає три кроки: активацію Gmail API (для поштових сценаріїв) і Generative Language API (для модуля ШІ); налаштування сторінки згоди зі scope https://mail.google.com/ (повний доступ IMAP/SMTP за документацією Google) і списком тестових акаунтів, поки додаток лишається у режимі тестування; створення OAuth 2.0 Client ID типу «Desktop application» [28]. Згенерована консоллю пара client_id/client_secret не зберігається у вихідному коді - значення передаються в бінарник через CMake-кеш або змінні середовища; для типу Desktop application client_secret за RFC 8252 §8.5 не є криптографічним секретом і служить лише ідентифікатором програмного клієнта.

2.6.Інтеграція із сервісом штучного інтелекту

Модуль штучного інтелекту реалізує допоміжне покращення тексту повідомлення (FR-10) і архітектурно повністю ізольований від основних поштових сценаріїв: відмова, недоступність або відсутність налаштованого ключа не впливають на IMAP/SMTP-підсистеми (NFR-09). Сервіс пропонує чотири дії над текстом - перевірку граматики, покращення ясності, підвищення формальності та лаконізацію - усі асинхронні: метод ініціює HTTPS-запит і одразу повертає керування, а результат повертається сигналом про завершення або помилку. Структура результату містить ознаку успіху, оригінальний текст, запропонований варіант, опційне пояснення моделі та повідомлення про помилку - цього достатньо як для порівняння двох варіантів, так і для діагностики у разі збою.

Серед доступних провайдерів LLM (OpenAI GPT, Anthropic Claude, Google Gemini, Mistral, локальна Llama) як стандартний у клієнті обрано Google Gemini у моделі gemini-3-flash-preview, ґрунтованої на архітектурі Transformer; конкретну модель задає користувач у налаштуваннях. Критерії вибору наведено в таблиці 2.4 [49].

Таблиця 2.4 - Критерії вибору ШІ-провайдера

Критерій	Реалізація у Gemini
Безкоштовний рівень доступу	Кілька мільйонів токенів на день для моделей рівня flash; OpenAI і Anthropic безкоштовного API не надають
Простота автентифікації	Єдиний ключ доступу у заголовку x-goog-api-key, без потоку OAuth і керування серверним станом
Контракт REST + JSON без SDK	POST на generateContent з JSON-тілом; достатньо QNetworkAccessManager і QJsonDocument
Передбачувана продуктивність	Моделі flash оптимізовані під короткі трансформації тексту з малою латентністю

Запит формується відповідно до офіційної документації: HTTP POST на кінцеву точку моделі (v1beta/models/gemini-3-flash-preview:generateContent на хості https://generativelanguage.googleapis.com/), заголовки Content-Type: application/json і x-goog-api-key (саме через HTTP-заголовок, не query-параметр), тіло - JSON із системною інструкцією, вхідним текстом і блоком generationConfig (типово temperature 0.3, maxOutputTokens 2000) [50]. Парсинг відповіді виконується QJsonDocument/QJsonObject із захистом від відсутніх полів: будь-яке відхилення трактується як помилка і доводиться до користувача через стандартний механізм діалогу. Ключ API зберігається у тому самому захищеному сховищі, що й OAuth-токени, що уніфікує контракт безпеки; для відображення у діалозі налаштувань використовується маскована форма (видно лише перші чотири і останні чотири символи), а повний ключ ніколи не потрапляє у журнал. Узагальнена діаграма послідовності типового виклику ШІ-функції наведена у Додатку Г.

Архітектура модуля залишається відкритою для розширення: REST/JSON-специфіку зосереджено в одному провайдер-класі, що реалізує принцип відкритості–закритості - додавання альтернативного провайдера зводиться до реалізації трьох методів (побудови URL, тіла запиту і парсингу відповіді) без змін у решті модуля. Діалог покращення реалізовано як трьохстанну машину (Loading → Comparison за успішної відповіді, Loading → Error за збою, з єдиним зворотним переходом Error → Loading через кнопку повторної спроби), що технічно унеможливлює другий паралельний запит, поки виконується перший, і додатково запобігає перевищенню обмежень провайдера на частоту звернень. Якщо ключ не налаштовано, дії з меню ШІ стають неактивними; решта функцій залишається повністю доступною (NFR-09).

2.7.Доменні моделі даних

Внутрішні моделі поділяються на три групи: поштові сутності, сутності автентифікації і конфігураційні структури. Центральною на боці надсилання є вихідне повідомлення з адресами відправника й отримувачів (включно з копією та прихованою копією), темою, текстовим тілом, переліком вкладень та опційними заголовками для ланцюжків листів; серіалізація - за Internet Message Format (RFC 5322) і MIME (RFC 2045/2046). Для відображення у списку використовується значно компактніша структура - прев'ю (UID, тема, відправник, дата, прапорці на зразок \Seen \Answered, ознака наявності вкладень), що будується з відповіді IMAP на FETCH ENVELOPE. При відкритті листа використовується повне тіло з обома частинами (text/plain і text/html) та метаданими вкладень. На верхньому рівні ієрархії - папка з її ім'ям, роздільником, прапорцями, UIDVALIDITY і найбільшим MODSEQ для роботи з QRESYNC. Спрощену UML-діаграму класів доменної моделі у нотації UML 2.5.1 [51][52] подано у Додатку З.

Розбір IMAP- і SMTP-відповідей реалізовано в режимі zero-copy. Токенизатори утримують лише запозичене (borrowed) представлення над буфером прийому; виокремлені атоми, літерали і рядкові поля повертаються як такі ж запозичені підрядки без копіювання вмісту. Винятком є тільки так звані quoted-рядки IMAP з екранованими послідовностями (\\, \") - для них обернене перетворення робить копіювання неминучим. Така диференціація уникає проміжних копій для переважної більшості значень, що особливо важливо для відповідей FETCH BODY[] обсягом у сотні кілобайт. Розбір MIME-структури делеговано GMime через тонкий адаптерний шар, що приймає сирі байти як запозичений підрядок над буфером прийому і повертає декодовані доменні структури; за тим самим принципом працюють і функції допоміжного декодування - заголовків за RFC 2047, тіла з урахуванням Content-Transfer-Encoding, полів Date та адресних полів.

Локальний кеш тіл повідомлень побудовано як **дворівневу структуру за єдиним абстрактним фасадом** - код інтерфейсу не залежить від того, з якого саме рівня прийшла відповідь, і працює з уніфікованими операціями «спробувати дістати», «помістити», «інвалідувати ключ/скриньку/акаунт». Перший рівень - резидентний LRU в оперативній пам'яті з двома незалежно застосованими бюджетами (за кількістю записів і за сумарним обсягом); повідомлення повертається спільним покажчиком типу std::shared_ptr на немутабельний знімок, що дозволяє віддавати запис в інтерфейс без глибокого копіювання під м'ютексом кешу. Другий рівень - шифрований персистентний кеш на основі SQLite, по одному файлу на акаунт, розміщений у захищеному системному каталозі застосунку з обмеженням доступу до власника засобами файлової системи відповідної ОС (NFR-05). Тіло кожного повідомлення зберігається у вигляді AES-256-GCM-шифротексту з унікальним 96-бітним nonce і 128-бітним тегом автентифікації; ключ розміщення в кеші формується кортежем (акаунт, скринька, UIDVALIDITY, UID), а LRU-витіснення обслуговує окремий індекс за моментом останнього звернення.

Майстер-ключ AES-256-GCM генерується КСГПВЧ при першому використанні і зберігається у системній ключниці в окремому просторі імен на кожен акаунт, ізольованому від простору імен OAuth-токенів і паролів. При виході з облікового запису ключ знищується з ключниці, а файл бази даних видаляється з диска - попередньо записаний шифротекст стає невідновним ще до того, як файл фактично перестане існувати на носії. Логіка взаємодії рівнів проста: пошук - від оперативної пам'яті до диска з підняттям запису назад в оперативний кеш при попаданні; запис - синхронно в обидва рівні; інвалідація - з фанаутом на обидва. Завдяки цій структурі кеш одночасно служить двом цілям - швидкому локальному відтворенню раніше переглянутих листів без повторних мережевих запитів (NFR-11) і захисту закешованого вмісту від несанкціонованого читання у разі копіювання файлів профілю користувача (NFR-05). Низькорівневі деталі (режим WAL,

конкретні параметри файлових дозволів, схема таблиці і формат буфера шифротексту) розглядаються у розділі 3.7.

Серед сутностей автентифікації ключовим є запис токенів (токен доступу, оновлення, момент припинення дії, тип, scope, асоційована адреса), що серіалізується у JSON і зберігається у захищеному сховищі (див. розділ 2.5.2). Тимчасовою службовою сутністю є PKCE-сесія, що існує лише на час одного потоку авторизації і знищується одразу після обміну коду на токен. Довгоживучою є конфігурація OAuth-провайдера: кінцеві точки авторизації і отримання токенів, ідентифікатор клієнта, опційний клієнтський секрет, область доступу і шаблон URI перенаправлення. Окремо стоїть запис облікового запису - навмисно неконфіденційна структура з адресою (одночасно ідентифікатор і ключ доступу до захищеного сховища), методом автентифікації (OAuth або Plain) і серверними підказками IMAP/SMTP; сукупність таких записів, разом з адресою активного, утворює реєстр у QSettings.

Перетворення вихідного повідомлення у MIME-байти виконує модуль-серіалізатор з обмеженою схемою: на верхньому рівні формується multipart/mixed із єдиною текстовою частиною (text/plain; charset=utf-8), за якою йдуть прямі частини того ж рівня з Content-Disposition: attachment для кожного файлу. Багатші конструкції (multipart/alternative, multipart/related з Content-ID за RFC 2387) у поточній реалізації не формуються, оскільки композер приймає лише простий текст. У зворотному напрямку модуль-десеріалізатор виконує рекурсивний обхід дерева MIME з обробкою multipart/alternative, multipart/related й inline-частин і вибирає найбагатшу доступну альтернативу (за замовчуванням HTML, у разі відсутності - plain). Така асиметрія є свідомою: толерантний прийом будь-якої коректної структури з мережі поєднується з мінімальним діалектом на виході. Перед відображенням HTML проходить санітарну обробку відповідно до рекомендацій OWASP: видаляються <script>, <iframe> та «on*»-обробники подій, обмежуються

схеми у href/src безпечним переліком (http, https, cid, data:image/*); виконання JavaScript і завантаження зовнішнього вмісту в переглядачі апріорі неможливі [53].

2.8. Обробка помилок та сценаріїв відмов

Узгоджена стратегія обробки помилок впливає на FR-09, NFR-06, NFR-09 і NFR-10 та ґрунтується на чотирьох наскрізних принципах: відсутності аварійного завершення роботи, коли всі винятки на межі асинхронного зворотного виклику перехоплюються, а стан компонента переводиться у безпечний режим; прозорості для користувача, за якої повідомлення формуються звичайною мовою з можливістю перегляду технічних деталей; локалізації відмов, коли збій ШІ-модуля або сервера авторизації не впливає на IMAP-з'єднання; а також контролі ресурсів через RAII, що гарантує закриття сокетів, звільнення TLS-сесій та скасування таймерів незалежно від шляху виходу з функції.

Мережеві помилки класифікуються за типом взаємодії та відповідною реакцією системи. До окремої категорії належать транспортні відмови, спільні для всіх мережевих підсистем. До них відносяться помилки резолюції DNS, відмова TCP-з'єднання, перевищення часу очікування або помилка TLS-рукостискання. У всіх таких випадках система закриває поточне з'єднання. Для IMAP-підсистеми після п'ятисекундної паузи виконується перезапуск лише циклу IDLE, тоді як для SMTP, ШІ-сервісу та сервісу авторизації користувачеві відображається повідомлення про помилку без автоматичного повторення операції.

Для IMAP-протоколу окремо обробляються серверні відповіді типу NO або BAD, а також випадки отримання нерозбірливої відповіді. Такі ситуації призводять до переходу клієнта в стан «Без з'єднання» з подальшим перезапуском IMAP-сесії. У SMTP-підсистемі серверні помилки класу 5xx, зокрема коди 550 або 552, не спричиняють автоматичного повторного надсилання повідомлення, оскільки це

створювало б ризик дублювання листа. Натомість користувач отримує повідомлення із зазначенням коду серверної відмови.

Під час взаємодії з ШІ-сервісом можуть виникати HTTP-помилки класів 4xx, наприклад через перевищення квоти або використання невалідного ключа доступу, помилки класу 5xx, а також ситуації, коли отримана відповідь не відповідає очікуваній схемі даних. У таких випадках користувачеві відображається повідомлення з можливістю повторити запит. На час виконання повторної спроби відповідна кнопка замінюється індикатором прогресу.

Помилки взаємодії із сервісом авторизації обробляються залежно від HTTP-статусу відповіді. Якщо під час виконання API-запиту отримано HTTP 401 або його функціональний еквівалент через прострочений access token, система автоматично виконує оновлення токена за допомогою refresh token та повторює початкову операцію без залучення користувача. Такий інцидент не відображається в інтерфейсі.

У випадку отримання відповіді HTTP 400 на кінцевій точці /token через помилки invalid_grant (відкликаний або прострочений токен) чи invalid_request (некоректний code_verifier) запис токенів видаляється зі сховища, очищуються локальні кеші сесії, після чого користувачеві пропонується виконати повторний вхід до системи. Аналогічне очищення виконується і при отриманні HTTP 401 на /token з причиною invalid_client, що може свідчити про блокування або зміну client_id. Додатково користувач отримує повідомлення про помилку конфігурації та рекомендацію повторно пройти процедуру авторизації.

Якщо сервер повертає HTTP 403 через недостатні повноваження (insufficient_scope) або блокування OAuth-додатка адміністрацією провайдера, користувачеві пропонується повторний вхід із запитом повного набору необхідних дозволів, а також відображається підказка щодо можливого адміністративного обмеження на стороні провайдера. Для випадків перевищення ліміту запитів на оновлення токена (HTTP 429) застосовується експоненційна затримка з джиттером

тривалістю 1, 2 та 4 секунди і виконується одна автоматична повторна спроба. Якщо вона також завершується невдачею, користувачеві рекомендується повторити операцію пізніше.

Транспортні відмови винесено в окрему категорію та поширено на всі чотири мережеві підсистеми, оскільки помилки етапу встановлення з'єднання - резолюції DNS, TCP-рукостискання та TLS-рукостискання - не залежать від прикладного протоколу й потребують однакової реакції у вигляді закриття з'єднання з коректним звільненням TLS-сесії та сокета засобами RAII.

Асиметрія між політиками обробки помилок IMAP і SMTP пояснюється тим, що повторний запуск циклу IDLE є ідемпотентною операцією та не створює побічних ефектів, тоді як автоматичне повторне надсилання листа може призвести до його дублювання. Архітектурна декомпозиція цієї підсистеми допускає подальше розширення політики повторних спроб, зокрема впровадження експоненційного збільшення інтервалів або автоматичного перепідключення для окремих класів транспортних помилок, без зміни зовнішнього контракту контролера IMAP-сесії.

Узагальнений цикл життя типового запиту з усіма розгалуженнями по класах помилок зображено на рисунку 2.6.

Найважливішою гілкою рисунка 2.6 є зворотний цикл «Виконання операції → Оновлення токена → Виконання операції»: він реалізує сценарій прозорого оновлення токену доступу без сповіщення користувача, тоді як перехід зі стану «Оновлення токена» до «Запиту повторного входу» настає лише за реальної невдачі оновлення.

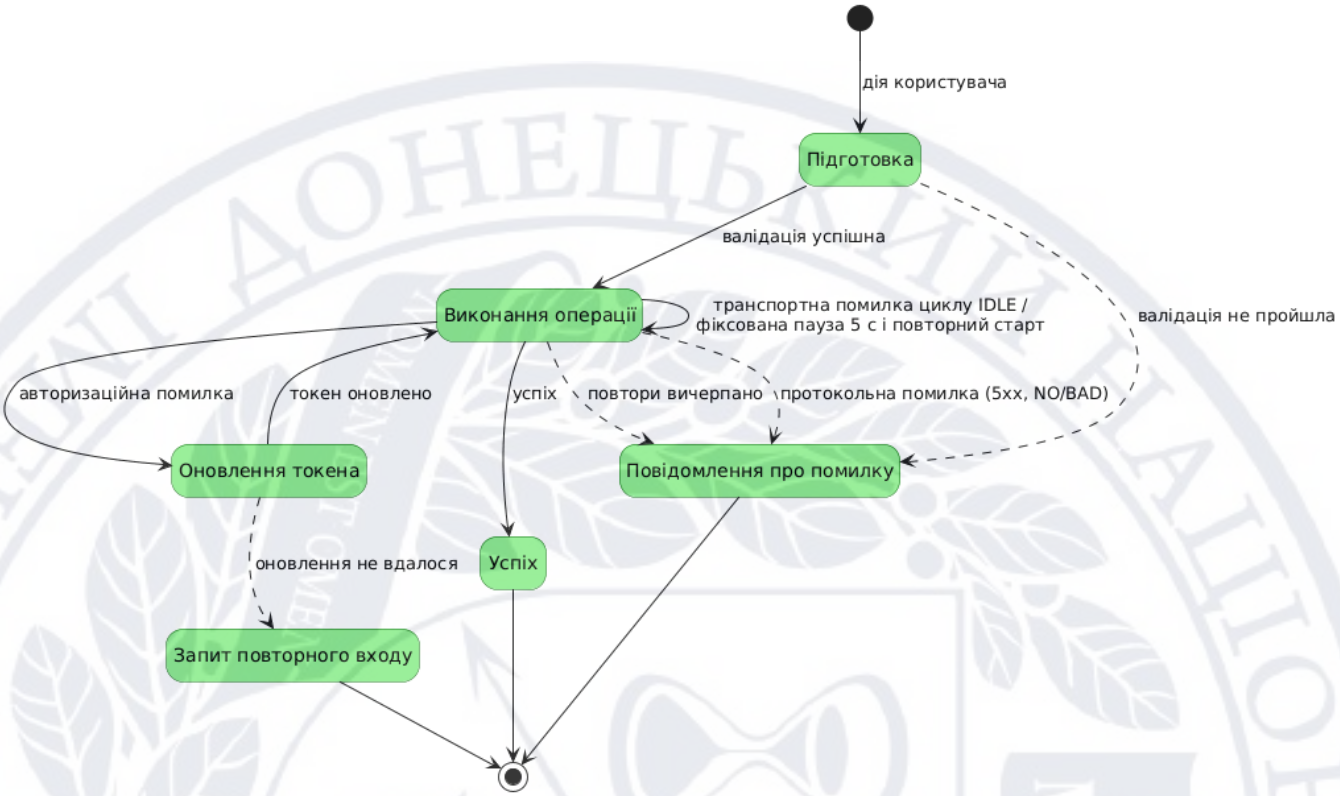

Рисунок 2.6 - Цикл життя запиту з обробкою помилок

Окремо від мережевих категорій розглядаються логічні помилки інтерфейсного походження - типовим випадком є закриття користувачем модального діалогу під час активної мережевої операції, коли результат запиту вже не має кому бути доставлений. Скасування таких довготривалих операцій (синхронізація великої папки, виклик ШІ-функції, оновлення токена) реалізовано двома засобами, які взаємодоповнюють один одного: сигналом скасування мережевого ядра для асинхронних операцій і перериванням активного відповідного об'єкта для HTTPS-запитів Qt. Скасування завжди приводить до коректного звільнення ресурсів і завершується статусом «скасовано», а не «помилка», щоб не вводити користувача в оману щодо причини зупинки.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ, ТЕСТУВАННЯ ТА ДЕМОНСТРАЦІЯ СИСТЕМИ

3.1. Структура проєкту і модульний поділ

Кодову базу організовано як один CMake-проєкт із двох верхньорівневих дерев і набору допоміжних артефактів. Перше дерево - бібліотека поштових протоколів, нейтральна до графічного інтерфейсу: її типи й API оперують стандартною бібліотекою C++, Boost.Asio і OpenSSL, що допускає її повторне використання у фонових сервісах синхронізації або в утилітах командного рядка. Друге дерево - графічний клієнт на Qt 6, який підключає бібліотеку як готовий компонент і додає до неї шар подання, координацію сценаріїв та платформо-залежні інтеграції. До них долучаються каталог скриптів первинного налаштування, каталог розширень CMake з інструментами розробки і два декларативні файли збірки (CMakePresets.json і vcpkg.json), описані у розділах 3.2 - 3.3. Поділ на два дерева реалізує принцип односпрямованих залежностей (бібліотека не знає про застосунок) і покриває NFR-07 у трактуванні Соммервілла, де модуль із єдиною відповідальністю та ациклічним графом включень розглядається як базова умова супроводжуваності великих систем [31,43].

Внутрішня декомпозиція кожного з двох верхньорівневих дерев - бібліотеки поштових протоколів і графічного клієнта - відтворює структурні таблиці розділу 2.3. Бібліотеку поштових протоколів поділено на шари протокольних клієнтів, серіалізаторів команд, розбору відповідей і спільних засобів - кожен зі своєю CMake-ціллю та власним набором юніт-тестів. Графічний клієнт поділено на доменний шар і шар подання зі суворим напрямом залежності «подання → домен». Допоміжне дерево діагностичної REPL-утиліти, що адаптує протокольний шар до інтеграційних Python-сценаріїв, розглянуто у розділі 3.9. Узагальнену структуру репозиторію з показом цих верхньорівневих дерев і їх безпосередніх підкаталогів наведено на рисунку 3.1.

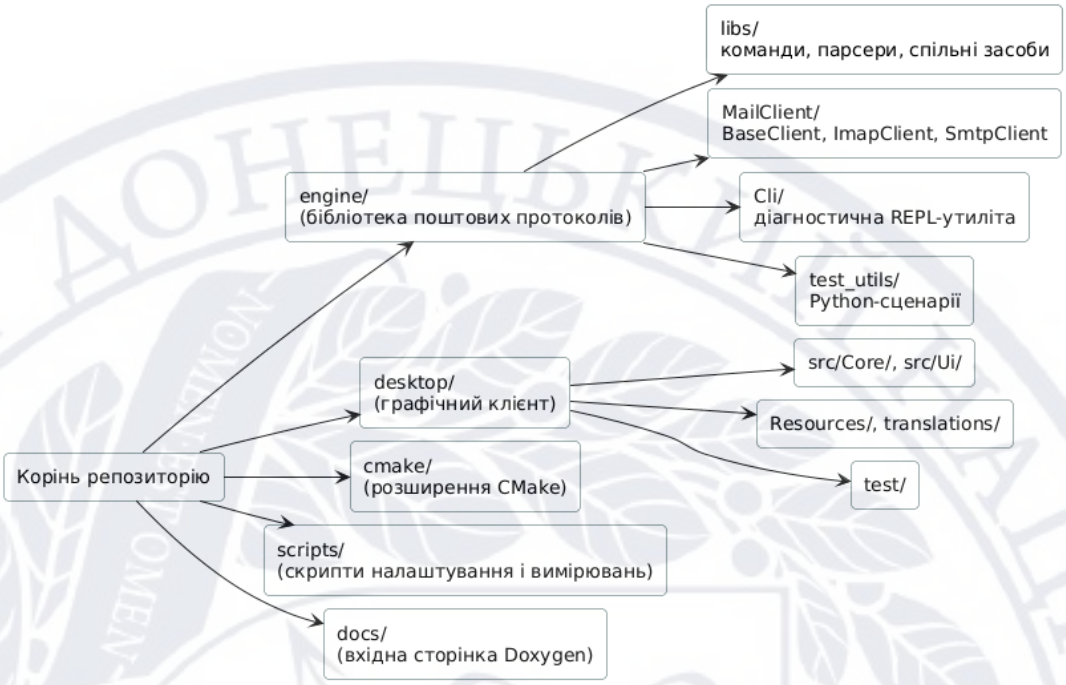

Рисунок 3.1 - Узагальнена структура репозиторію проєкту

Доменний шар графічного клієнта об'єднує підсистеми автентифікації (модуль OAuth, реєстр облікових записів, узагальнене захищене сховище токенів поверх системної ключниці), штучного інтелекту з провайдером Google Gemini, поштової сесії, локального шифрованого кешу повідомлень (детально розглянутого у розділі 3.7), серіалізації конфігурації через JSON-модуль Qt, RAII-обгортки циклу подій асинхронного мережевого ядра (описаної в розділі 3.8) і допоміжних утиліт.

Адаптерний шар розбору MIME піднімає доменні структури бібліотеки до Qt-сумісних типів на основі QString, QByteArray і QList. Без цього прошарку шар подання був би змушений безпосередньо звертатися до API стандартної бібліотеки C++ і вручну виконувати конверсію кодувань на кожному виклику, що порушило б поділ інтерфейсу і доменної логіки. Сам прошарок реалізує шаблон «адаптер» з каталогу шаблонів проєктування Е. Гамми та співавторів [30]. Шар подання структуровано за відповідальностями: реалізація головного вікна задає каркас із трьох вертикальних панелей і обробку меню (декларативна частина в Qt Designer формі компілюється в клас із зовнішнім поданням засобами CMake), а великі за обсягом обов'язки винесено в окремі компоненти - менеджер списку листів,

координатор поштової сесії з його машиною станів переходів «не авторизований → авторизація → активна сесія → переавторизація» і колекція користувацьких віджетів (рендер елемента списку, безпечний переглядач HTML-вмісту, модальне вікно ШІ-асистента і діалог налаштувань ШІ-провайдера).

Структуру проєкту характеризують два незалежні інваріанти. Перший, статичний: граф включень не містить циклів - кожен файл графічного клієнта залежить від бібліотеки поштових протоколів, але не навпаки, а в межах клієнта шар подання залежить від доменного шару, але не навпаки. Це закріплено структурою цілей CMake: бібліотека експортує агрегатну INTERFACE-ціль, а графічний клієнт компонує доменний шар і шар подання у дві окремі бібліотеки, що компонуються у фінальний виконуваний файл. Другий інваріант, динамічний: кожен модуль вкладається у виконавчу схему розділу 2.3 відповідно до своєї ролі - обгортка циклу подій належить мережевому потоку; шар подання, модуль авторизації, кеш повідомлень, конфігурація і адаптерний шар MIME→Qt працюють у потоці графічного інтерфейсу, активуючи мережеві операції через серіалізатори виконання мережевого потоку. Ці інваріанти забезпечують виконання NFR-07: новий модуль додається у відповідний підкаталог із власним файлом збірки без змін у решті проєкту [41].

3.2. Система збирання і керування залежностями

Систему збирання реалізовано на CMake 3.22+ за каноном Modern CMake [54]. Кореневий файл задає стандарт мови (C++23 без GNU-розширень) і вмикає генерацію compile_commands.json для статичних аналізаторів, після чого підключає два піддерева - бібліотеку поштових протоколів і графічний клієнт, - кожне з яких має власний простір імен цілей збірки. Файл збірки бібліотеки створює агрегатну INTERFACE-ціль як єдину точку входу для її споживачів. Декларативним фасадом над конфігурацією служить файл CMakePresets.json: шість іменованих наборів - для

macOS, Linux і Windows у профілях Debug і Release - фіксують умови активації за іменем хостової ОС, генератор (Ninja на POSIX, Visual Studio 17 на Windows) і набір кешових змінних, а спільні налаштування винесено у прихований базовий набір. Такий підхід замінює десятки «-D…»-прапорців на трикомандний цикл «конфігурація → збирання → тестування», уніфікований між трьома операційними системами.

Стратегія керування залежностями адаптивна до платформи. На macOS і Linux застосовуються системні пакетні менеджери (Homebrew і apt) як простіший шлях для операційних систем, де бібліотеки пакетовані супровідниками дистрибутивів. На Windows використовується менеджер пакетів vcpkg від Microsoft у режимі маніфесту [55]: файл vcpkg.json декларує мінімальні версії потрібних модулів Qt 6, QtKeychain, Boost.Asio, OpenSSL 3, GMime і nlohmann/json [56]. На рівні CMake передбачено двоступеневий пошук GMime - через pkg-config із резервним шляхом ручного пошуку, - що покриває всі три цільові операційні системи. Сукупно описані рішення забезпечують виконання NFR-07 на рівні самої збірки.

3.3. Кросплатформне збирання через єдиний скрипт первинного налаштування

Кросплатформеність як вимога NFR-03 передбачає, щоб будь-хто міг відтворити збірку на власній машині - без подвійних інструкцій і без неявних знань про шляхи до Qt. У великих відкритих проєктах цю задачу типово розв'язує безперервна інтеграція; для дипломної роботи такий шлях надмірний. Натомість обрано єдиний скрипт первинного налаштування на сімейство операційних систем, що бере на себе встановлення залежностей за стратегією з розділу 3.2, конфігурування проєкту, збирання і прогін тестів. POSIX-варіант скрипта ідемпотентний; обидва скрипти керуються невеликою кількістю змінних

середовища і аргументів командного рядка, що дозволяють пропустити крок встановлення залежностей або обрати профіль збірки (за замовчуванням - Release; Debug-профіль додатково запускає прогін тестів через CTest).

3.4. Кросплатформний код: умовна компіляція і нативні інтеграції

Скрипт первинного налаштування з попереднього підрозділу - лише зовнішній шар кросплатформності; сам код повинен враховувати відмінності між операційними системами. У проєкті умовну логіку локалізовано на чотирьох рівнях. Препроцесорні гілки в C++/Qt-коді через макроси Q_OS_MACOS, Q_OS_WIN, Q_OS_UNIX присутні лише у чотирьох файлах; решта кодової бази платформо-нейтральна - будь-яка нова платформо-залежна логіка виноситься в окрему функцію з кількома реалізаціями [29]. На рівні CMake умовні блоки додають специфічні модулі і прапорці компіляції - від прапорців компілятору MSVC (без яких кириличні літерали у викликах макросу tr() ламаються в момент сканування lupdate) до Windows-специфічних властивостей цілей із підключенням advapi32 для керування ACL. Властивості цілей CMake реалізують пакування у платформо-специфічні артефакти: MACOSX_BUNDLE TRUE для .app-бандла на macOS згідно з рекомендацією Apple, WIN32_EXECUTABLE TRUE на Windows для перемикання у GUI-підсистему, звичайний ELF файл на Linux [57]. Опційні залежності і функціональні прапорці (USE_SYSTEM_KEYCHAIN, BUILD_TESTS) керують макросами компілятора, що активують бекенд захищеного сховища секретів і збірку модульних тестів.

Платформо-залежну поведінку ізольовано в окремих, чітко означених точках кодової бази. На macOS задано уніфіковану візуальну модель Qt - це усуває розбіжності рендерингу елементів керування між версіями платформи і забезпечує однаковий вигляд інтерфейсу незалежно від цільової ОС. Файлові операції, що формалізують вимогу NFR-05 щодо приватності локального кешу, реалізовано

через нативні API кожної ОС за єдиною семантикою (деталі - у розділі 3.7), а коректне відображення позаалфавітних Unicode-символів забезпечує єдиний кросплатформний ланцюг резервних шрифтів, у якому шрифтовий матчер Qt автоматично обирає першу наявну сім'ю на конкретній ОС - без умовної компіляції чи розгалуження за платформою.

Окремою категорією платформо-специфічних інтеграцій є доступ до системного сховища довіри сертифікатів. POSIX-системи постачають кореневі сертифікати у вигляді файлових бандлів за стандартними шляхами, які OpenSSL знаходить автоматично під час ініціалізації контексту TLS, тому жодного додаткового коду на macOS і Linux не потрібно. На Windows кореневі сертифікати лежать у системному сховищі CryptoAPI і не доступні OpenSSL напряму; для відповідності NFR-04 (обов'язкова перевірка серверного сертифіката) у Windows-збірці передбачено окремий ініціалізаційний адаптер, що під час старту застосунку зчитує кореневі сертифікати зі сховища ОС через CertOpenSystemStoreW і додає їх у X509_STORE довірчого контексту OpenSSL. Це відмінність ОС у моделі дистрибуції коренів довіри, інтегрована за загальною дисципліною - платформна логіка ізольована в одному файлі під #ifdef Q_OS_WIN, а спільний код TLS-клієнта залишається платформо-нейтральним.

Сукупно описані механізми забезпечують виконання NFR-03: близько 95 % файлів проєкту компілюються однаково на трьох цільових ОС, а решта 5 % - локалізовані фрагменти платформо-залежної логіки з чіткою CMake- або препроцесорною межею, що технічно унеможливлює «протікання» платформних деталей у спільну частину коду.

3.5. Реалізація графічного інтерфейсу

Шар графічного інтерфейсу побудовано як комбінацію декларативного і програмного підходів. Декларативна частина - форма головного вікна в Qt Designer,

де задається статичний каркас: три вертикальні панелі (список облікових записів і папок, список листів, перегляд тіла листа), меню і панелі інструментів. Під час збирання система CMake через механізм автоматичної генерації подання викликає утиліту uic для генерації заголовка з класом, що інкапсулює розкладку, після чого програмний код зв'язує згенероване подання з екземпляром вікна. Така комбінація поєднує переваги обох підходів: Qt Designer дозволяє візуально проєктувати складну розкладку, не ризикуючи помилками вирівнювання у коді, а C++-код залишається зосередженим на бізнес-логіці взаємодії [29]. Документація Qt Widgets прямо рекомендує такий підхід для застосунків зі складною розкладкою головного вікна [58].

Програмну частину шару подання побудовано за принципом одноосібної відповідальності: клас головного вікна, попри значний обсяг, делегує спеціалізованим компонентам через сигнали і слоти всю логіку авторизації, запитів до IMAP-сервера і керування переліком облікових записів.

Центральним вузлом шару координації є координатор сесії - самостійний модуль без власних візуальних елементів, що поєднує дві відповідальності. Перша - інкапсуляція машини станів переходів «не авторизований → авторизація → активна сесія → переавторизація» (не плутати з машиною станів IMAP-сесії у розділі 2.4.1: координатор сесії агрегує її як одну зі своїх внутрішніх фаз): він отримує події від модуля авторизації і сервісу поштової сесії та виставляє назовні лише високорівневі сигнали («встановлення сесії», «її завершення», «потреба у повторній автентифікації»). Друга - керування сценаріями кількох облікових записів (авторизація нового, автоматичне відновлення сесії при старті, перемикання активного через попереднє коректне завершення поточної поштової сесії, вихід із повним стиранням пов'язаних секретів), що транслюється у сигнали «вхід завершено», «вихід виконано», «новий активний обліковий запис». Завдяки цьому головне вікно не реалізує сценаріїв входу/виходу самостійно, а лише реагує на ці події. Цей шаблон відомий як «координатор» або «посередник»; його перевага -

додавання нового сценарію (наприклад, тимчасового відключення мережі) розширює лише логіку координатора, не торкаючись модуля авторизації чи головного вікна [30].

Другим прикладом одноосібної відповідальності є виокремлений у власні файли менеджер списку листів. Він відповідає виключно за життєвий цикл лівого і центрального стовпців: завантаження повідомлень із поштової сесії, побудову моделі даних для віджета списку, обробку прокручування і кешування прев'ю. Виокремлення менеджера запобігає перетворенню вікна на клас-«всезнавець», що за класифікацією Мартіна є одним з ключових антипатернів об'єктно-орієнтованого проєктування і призводить до ситуацій, коли модифікація однієї деталі вимагає тестування всього застосунку [35]. Аналогічну роль виконують контролери поштової сесії, які інкапсулюють низькорівневу взаємодію з протокольним шаром і повертають менеджеру списку лише готові доменні об'єкти.

Окрему категорію складових шару подання утворюють спеціалізовані віджети. Віджет елемента списку листів виводить рядок у списку (аватар відправника, ім'я або поштову адресу, тему, прев'ю першого рядка тіла, час отримання, мітку «непрочитано») і підмінює стандартний рендер відповідним механізмом Qt, що дає повний контроль над розкладкою. Ізольований переглядач HTML - підклас вбудованого браузера тексту Qt - інтегрує асинхронне довантаження зовнішніх зображень схем http/https через мережевий менеджер Qt, а будь-які навігаційні переходи делегує штатній функції відкриття URL у системному браузері користувача замість обробки усередині застосунку. Виконання вбудованих сценаріїв і обробників подій у переглядачі вимкнене самим рушієм Qt, незалежно від змісту листа. Базовий рівень очищення тексту винесено в окремий допоміжний модуль доменного шару, який у поточній версії реалізує фільтрацію символів plain-частини за переліком дозволених категорій Unicode і виправлення некоректних RGBA-кольорів у CSS-атрибутах. Обмеженням наявної реалізації є те, що це не повноцінний санітайзер тегів і атрибутів за рекомендаціями довідника OWASP з

протидії XSS; винесення логіки очищення в окремий компонент дозволяє у наступних версіях нарощувати правила без зміни зовнішнього контракту переглядача [53]. Третій і четвертий віджети - модальне вікно ШІ-асистента і діалог налаштувань ШІ-провайдера - закривають функціональність UC-05.

Візуальну стилізацію реалізовано через єдину таблицю Qt Style Sheets: реалізовано темну тему, згруповану за віджетами для зручності супроводу. Файл стилів вкладається у бінарник через механізм Qt-ресурсів, а застосунок завантажує його у точці входу [59].

Шар подання ґрунтується на трьох принципах. Сигнали і слоти Qt є основним механізмом розв'язки модулів: головне вікно не викликає методи модуля авторизації безпосередньо, а отримує його сигнали через механізм сигнально-слотового зв'язку, що дозволяє замінити одну реалізацію на іншу без модифікації коду вікна. RAII-керування через ієрархію parent-child-об'єктів Qt забезпечується передачею вікна як батьківського об'єкта при створенні похідних віджетів, що автоматично гарантує їх знищення при закритті вікна без явних викликів delete. Поділ відповідальностей між шаром подання і доменним шаром забезпечується тим, що кожен сигнал від доменного модуля до подання оперує доменними типами (інформація про обліковий запис, зведення про лист, стан авторизації), а не низькорівневими структурами Boost.Asio чи OpenSSL - це гарантує, що зміна транспортного рівня не призведе до перекомпіляції шару подання.

Узагальнений вигляд шару подання та його зв'язки з доменом подано на рисунку 3.2.

Таблиця стилів компенсує відмінності платформних стилів Qt Widgets (AppKit з обходом через Fusion на macOS, UxTheme на Windows, GTK на Linux) і забезпечує консистентний вигляд застосунку незалежно від цільової ОС.

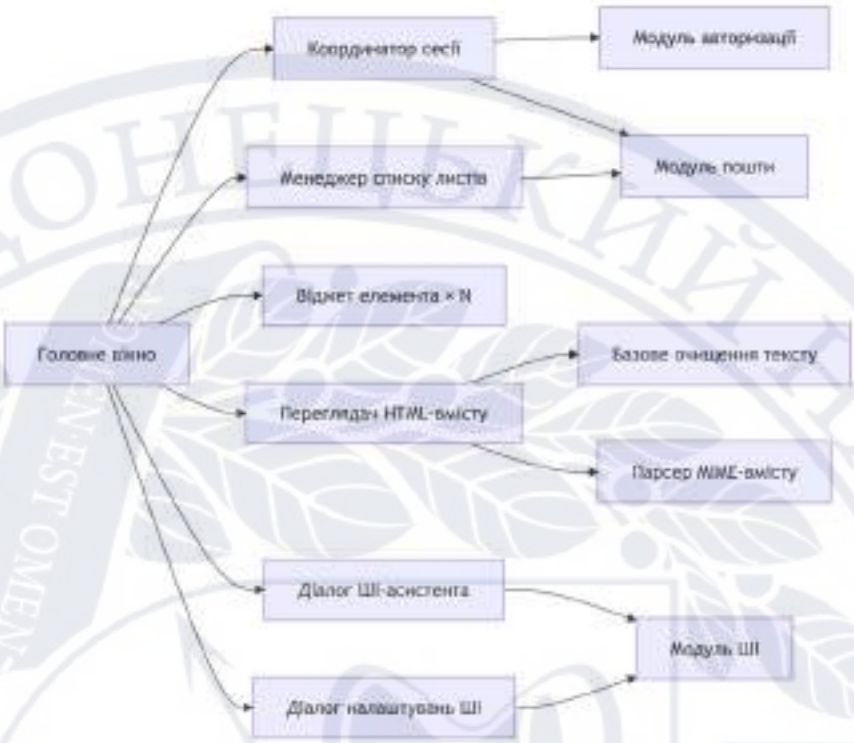

Рисунок 3.2 - Шар подання і його зв'язки з доменним шаром

3.6.Реалізація потоку OAuth: локальний приймач і HTML-сторінки

Архітектурне обґрунтування OAuth 2.0 з PKCE як методу автентифікації подано у розділі 2.5; цей підрозділ присвячено реалізації - як нативний застосунок без власного веб-сервера приймає відповідь від сервера авторизації. Структуру і взаємозв'язки компонентів підсистеми авторизації подано на рисунку 3.3. Координатор авторизації виступає фасадом для шару подання, володіє локальним приймачем перенаправлення і сесією PKCE, делегує рендеринг HTML-сторінок статичному рендеру, а збереження токенів - узагальненому захищеному сховищу, що транслюється у конкретний бекенд системної ключниці. Реєстр облікових записів є самостійним неконфіденційним зберігачем суто індексної ролі і живе у QSettings. Координатор сесії використовує всі ці компоненти за прямим контрактом і не має жодного власного візуального елемента - він міститься у шарі координації

між головним вікном і доменним шаром.

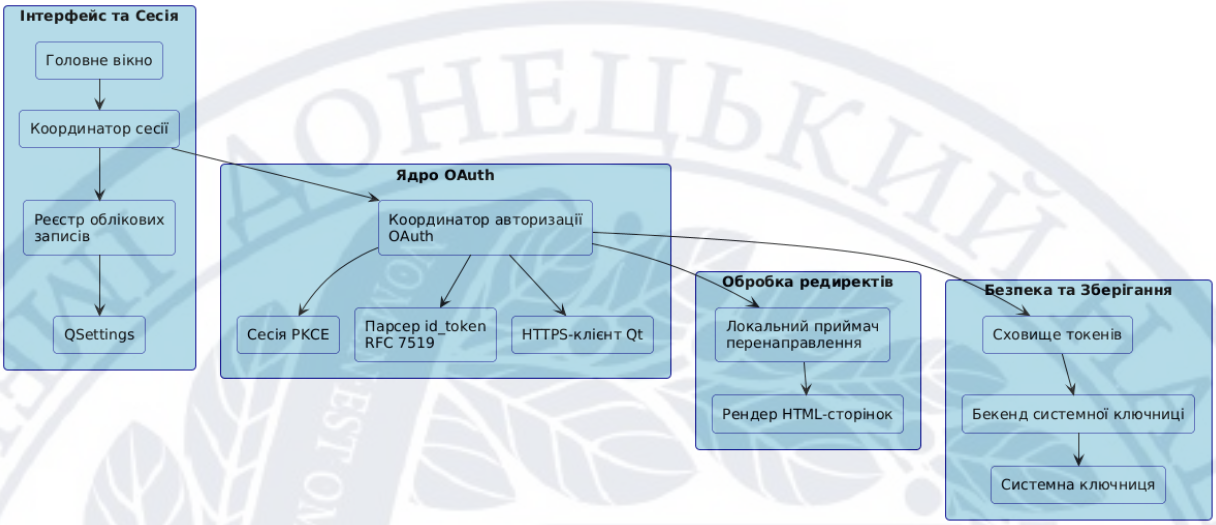

Рисунок 3.3 - Компоненти підсистеми авторизації

Згідно з RFC 8252 (BCP-212), оптимальною стратегією для нативних застосунків є перенаправлення на адресу виду http://127.0.0.1:<port>, де порт обирається динамічно при запуску потоку авторизації [11]. Цей варіант безпечніший за альтернативу з користувацькою URL-схемою, оскільки на більшості платформ сторонній застосунок може зареєструвати ту саму схему і перехопити код перенаправлення, тоді як петлева адреса доступна лише процесам, що працюють на тій самій машині від імені того самого користувача.

Локальний приймач інкапсулює TCP-сервер Qt, що слухає на петлевому інтерфейсі на ефемерному порту; реальне значення порту читається після старту і підставляється у поле redirect_uri як рядок виду http://127.0.0.1:<порт> без path-сегменту. Збіг перенаправлення виконується не за шляхом, а за наявністю у GET-запиті очікуваних параметрів - це свідоме спрощення: код виходить простішим, а провайдери однаково приймають перенаправлення без path-сегменту, якщо саме так його зареєстровано в консолі.

Слот, приєднаний до сигналу нового з'єднання сервера, читає сирий HTTP-запит до завершення першої лінії, розпарсовує рядок запиту, формує і надсилає відповідь HTTP-200 із рендерованим HTML і закриває з'єднання. Розмір вхідного

запиту обмежено вісьмома кілобайтами, а час очікування читання - трьома секундами, що захищає приймач від некоректних або навмисно повільних клієнтів. У разі успіху приймач видає сигнал «отримано код авторизації»; за наявності параметра «error» - сигнал помилки з розпарсованим повідомленням. Якщо запит коректний за форматом, але не містить ні коду, ні помилки, відправляється HTML-сторінка помилки з повідомленням про відсутність очікуваних параметрів, а сигнал не видається, щоб не перервати очікування користувача.

Параметр state генерується у класі сесії PKCE одночасно з code_verifier через криптографічно стійкий генератор випадкових чисел Qt. При отриманні перенаправлення координатор авторизації звіряє повернений state зі збереженим у поточній сесії: невідповідність свідчить або про помилку, або про спробу зовнішньої ін'єкції відповіді - у такому випадку код викидається без обміну на токен. У нативних сценаріях з петлевим інтерфейсом основний захист від перехоплення коду авторизації забезпечує сам механізм PKCE з доведенням володіння секретом code_verifier (RFC 7636); параметр state виконує доповнювальну функцію - прив'язує авторизаційну відповідь до конкретної ініціативи цього застосунку - відповідно до §10.12 RFC 6749 і §8.9 RFC 8252 [11][26]. Для публічного релізу restricted-scope https://mail.google.com/ потребує проходження верифікації Google OAuth (програма CASA Tier 2); поточний прототип лишається у режимі тестування з фіксованим переліком допустимих акаунтів, чого достатньо для верифікації функціональних сценаріїв UC-01 - UC-05, але недостатньо для надання дозволу довільному користувачеві [48].

У відповідь приймач надсилає одну з двох HTML-сторінок - успішної або помилкової авторизації, - які слугують візуальним підтвердженням завершення зовнішнього кроку OAuth і повідомляють користувачеві, що системний браузер можна закрити та повернутися до застосунку. Шаблони сторінок та їхні стилі вкладено у бінарник через механізм ресурсів Qt і рендеряться окремою статичною

утилітою з мінімальною підстановкою плейсхолдерів. Узагальнений потік взаємодії компонентів подано на рисунку 3.4.

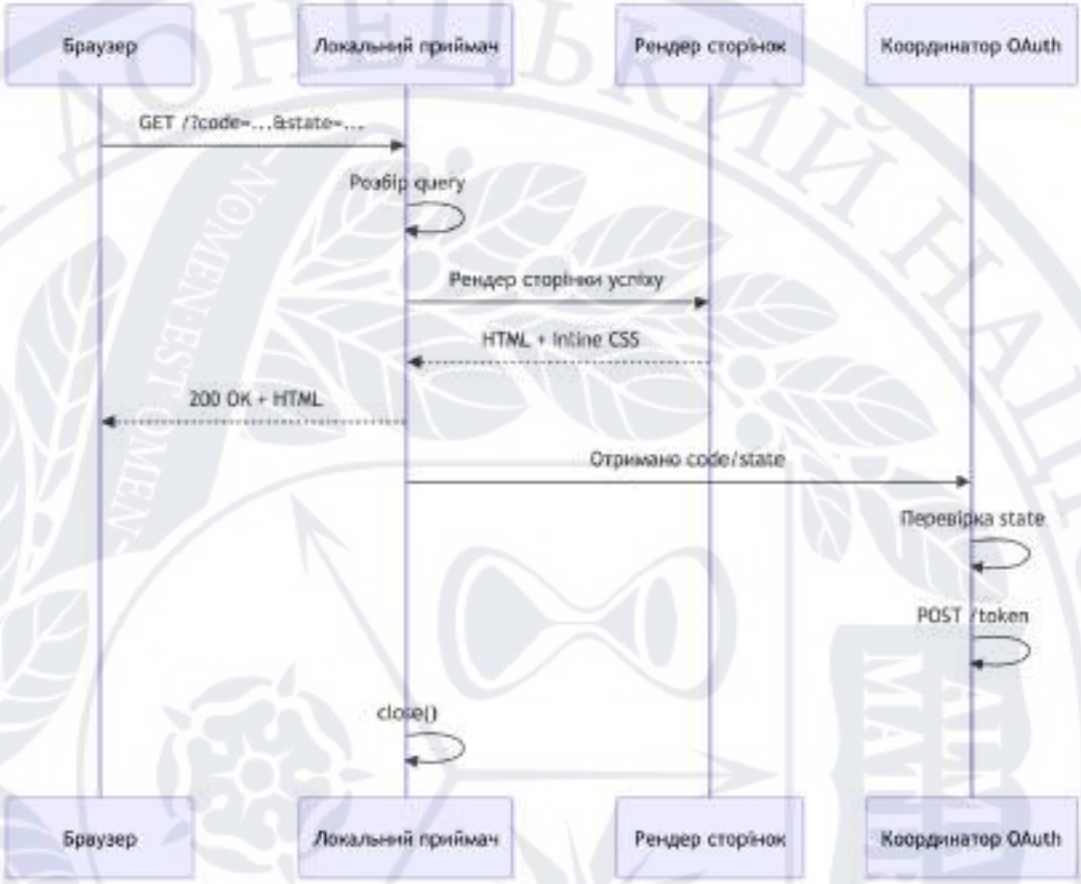

Рисунок 3.4 - Послідовність взаємодії при OAuth-авторизації

Завершальний крок - звільнення ефемерного порту після прийому коду. Це гарантує, що повторний запит на той самий URL із кешу браузера завершиться помилкою з'єднання і не отримає коректної відповіді в обхід перевірки параметру state. Уся механіка - від створення приймача через ієрархію parent-child-об'єктів Qt до автоматичного знищення при закінченні сесії - реалізована через RAII-семантику Qt, без явних викликів delete, що відповідає архітектурному рішенню з розділу 2.5.

3.7. Реалізація локального кешу повідомлень

Цей підрозділ розкриває реалізаційний бік локального кешу повідомлень (вимоги FR-07, NFR-05, NFR-11). Кеш виступає єдиним джерелом правди для стану

інкрементної синхронізації, як обґрунтовано у розділі 2.4.1, тож його схема одночасно зберігає тіла повідомлень у шифрованому вигляді і per-message MODSEQ для потреб CONDSTORE/QRESYNC.

Підсистему кешу повідомлень складають вісім модулів, поділених на чотири логічні групи: інтерфейс і доменні типи, шифрувальний шар, два рівні кешу та об'єднувальний фасад. Сутності кешу і зв'язки між ними наведено на рисунку 3.5.

Метадані папки прив'язані до emailAddress (а не до окремого синтетичного accountId), оскільки адреса одночасно слугує ключем доступу до системної ключниці й гарантовано унікальна в реєстрі акаунтів - це уникає проблеми синхронізації двох ідентифікаторів. Тіла повідомлень шифруються MASTER_KEY_ENTRY, тоді як OAuth-токени зберігаються у власному просторі імен ключниці без перетину з ключовим матеріалом кешу: компрометація БД кешу не дає доступу до токенів, а компрометація токенів - до раніше закешованого вмісту. Інвалідація на зсуві UIDVALIDITY реалізована через первинний ключ MESSAGE_CACHE_RECORD, що включає uidValidity - зміна цього поля у контексті скриньки робить старі рядки сиротами, які прибираються однією DELETE-операцією.

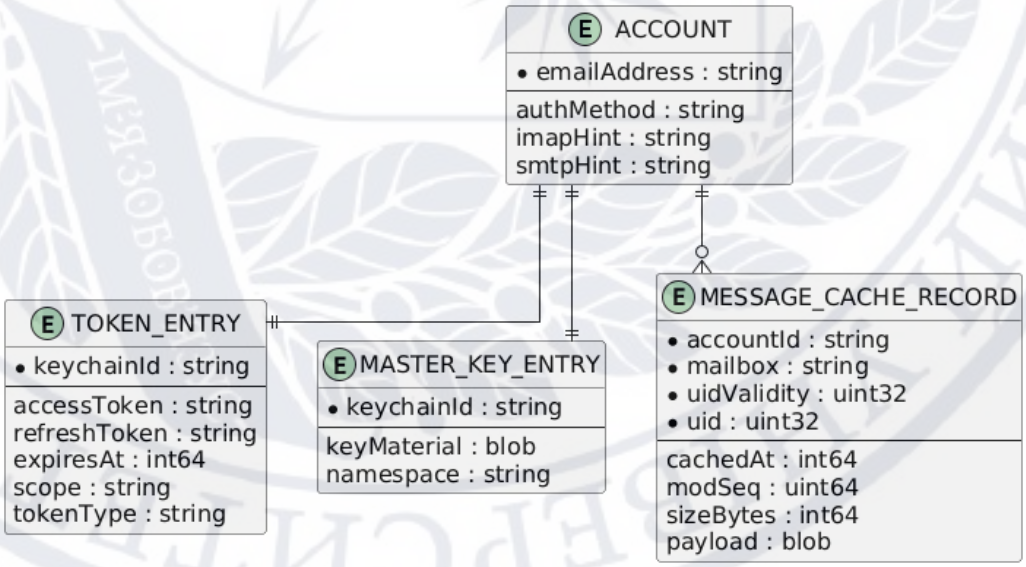

Рисунок 3.5 - ER-діаграма сутностей локального кешу і пов'язаних записів ключниці

Інтерфейс кешу визначає абстрактний контракт із чотирьох операцій - отримання, оновлення, інвалідації окремого запису і повного очищення кешу облікового запису, - а ключем виступає кортеж (ідентифікатор облікового запису, ім'я поштової скриньки, UIDVALIDITY, UID), як того і вимагає RFC 3501 §2.3.1.1 (UID коректний лише у межах фіксованого UIDVALIDITY). Шифрувальний шар інкапсулює взаємодію з EVP-API OpenSSL: ключову інваріанту неповторюваності nonce за умов одного майстер-ключа явно зафіксовано у коді (повторне використання порушує гарантії GCM), а формат буфера на диску - [nonce || ciphertext || tag]. Цілісність ключових полів запису забезпечує первинний ключ SQLite-таблиці у персистентному шарі, а не AAD на рівні AEAD. Майстер-ключ на кожен обліковий запис розміщено у системній ключниці в окремому просторі імен `msgcache:` - ім'я запису формується як 32-символьний шістнадцятковий префікс SHA-256 від accountId, аби уникнути збереження адреси у переліку записів ключниці у відкритому вигляді. Декомпозиція постачальника ключа за концептом захищеного сховища (розділ 3.8) допускає у наступних версіях підключення додаткових джерел ключа без зміни зовнішнього контракту шару кешу.

Реалізація двох рівнів кешу деталізує дворівневу схему з розділу 2.7 на рівні внутрішніх структур даних і параметрів зберігання. Перший рівень (у пам'яті) реалізує LRU-витіснення поверх єдиного асоціативного контейнера типу unordered_map<MessageKey, Entry> і допоміжного списку ключів, упорядкованого за нещодавністю звернення; обидва бюджети (типово 64 записи і 64 МіБ) застосовуються одночасно, а читання повертає розумний вказівник shared_ptr<const CachedMessage>, що дозволяє безпечно віддавати знімок в інтерфейс без копіювання під м'ютексом. Другий рівень (на диску) - SQLite у режимі WAL з єдиною таблицею messages, де колонки cached_at (для byte-budgeted-LRU з типовим бюджетом 256 МіБ) і mod_seq (per-message MODSEQ) розміщено поряд із шифрованим payload; платформо-специфічне обмеження доступу до файлу (POSIX 0600 / Windows DACL) описано у розділі 3.4. Файл бази розміщено у захищеному

системному каталозі застосунку як messagecache/<sha256(accountId)>.db - окрема SQLite база даних на акаунт. Збережена пара UIDVALIDITY/HIGHESTMODSEQ (на рівні папки) і mod_seq (на рівні повідомлення) слугує метаданими для алгоритму вибору стратегії синхронізації, наведеного у розділі 2.4.1 (рисунок 2.4.2).

3.8. Інженерні засади якості коду

Якість коду у проєкті підтримують інструментальні засоби і архітектурні шаблони, що формалізують контракти модулів на рівні системи типів. Базовим стилем коду є Google C++ Style із локальними поправками у файлі .clang-format. Інструментальну підтримку оформлено як окремі CMake-цілі у спеціалізованому модулі діагностичного інструментарію: окремі цілі для перевірки і застосування форматування, цілі для статичного аналізу загального характеру і Qt-специфічних шаблонів [60][61]. Жоден з інструментів не вбудовано в обов'язковий цикл збірки - вони запускаються розробником за потреби. Документацію для класів і ключових функцій оформлюють у форматі Doxygen з двох автономних конфігурацій (по одній на дерево вихідних текстів) зі спільною головною сторінкою; згенерована документація містить опис кожного відкритого класу, ієрархію наслідування і перехресні посилання між типами [62]. Зразок згенерованої сторінки наведено на рисунку 3.6.

Класичну C++-парадигму обробки помилок через викидання винятків відкинуто на користь value-based-підходу через узагальнений тип-результат, типізований як std::expected<T, E> із стандарту C++23 [32]. На користь цього вибору наведено два аргументи. Перший - передбачуваність API: винятки не помітні у сигнатурі функції, тому при читанні коду неможливо з першого погляду встановити, чи може функція завершитися невдало; std::expected робить можливість провалу частиною типу повернення і обов'язковою для перевірки на стороні виклику.

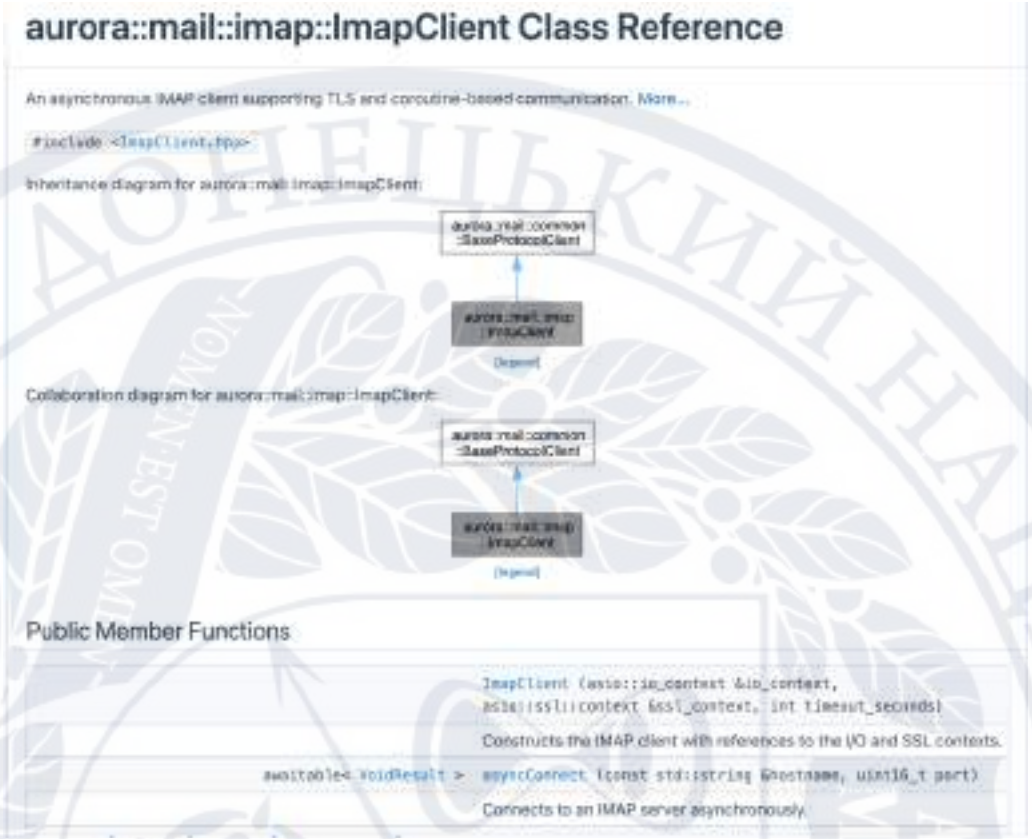

Рисунок 3.6 - Згенерована Doxygen-документація: сторінка ключового класу ImapClient

Другий - продуктивність: викидання винятка у реалізаціях gcc/clang коштує сотні тактів процесора через розкручення стека, що неприйнятно для критичних шляхів виконання, наприклад для парсингу IMAP-відповідей, що відбувається на кожне нез'єднане повідомлення сервера. Спільним типом помилки для всього протокольного шару служить доменна структура з полями коду помилки, повідомлення і прапорця можливості повторної спроби. Винятки у проєкті використовуються виключно як захисний шар на межі з зовнішніми бібліотеками: якщо така бібліотека викидає виняток у зворотному виклику, він перетворюється на коректну невдачу типу-результату.

Архітектурне рішення про два паралельні цикли подій - цикл інтерфейсу і мережевий цикл boost::asio::io_context - зафіксовано у розділі 2.1. З точки зору реалізації це втілено у вигляді RAII-обгортки над контекстом введення-виведення і потоком-власником, що цей цикл виконує. При побудові обгортки створюється

контекст, до нього прикріплюється явний «утримувач роботи» boost::asio::executor_work_guard (щоб виконання не завершилося миттєво при порожній черзі), після чого запускається керований потік std::jthread, який виконує цикл подій у нескінченному циклі з перехопленням усіх винятків: будь-який виняток, що пройшов до межі потоку, логується, а виконання поновлюється, щоб одна помилкова корутина не призвела до аварійного завершення всього застосунку. При знищенні обгортки утримувач знімається, контекст зупиняється, а потік автоматично приєднується завдяки семантиці std::jthread. Концепція двох циклів подій підкріплена засобом серіалізованого виконання boost::asio::strand операцій усередині мережевого циклу (без явних м'ютексів) і відкладеним перекиданням результатів у потік графічного інтерфейсу через стандартний механізм Qt.

Стандарт C++20 ввів концепти - спосіб формалізувати вимоги до шаблонних параметрів на рівні системи типів [32]. У цьому проєкті концепти використовуються як гарантований компілятором варіант принципу інверсії залежностей з SOLID: верхньорівневий модуль не залежить від конкретної реалізації нижчого, а лише від концепта, якому ця реалізація повинна задовольняти, і порушення контракту виявляється помилкою компіляції, а не часу виконання [35]. Той самий підхід реалізовано у підсистемі автентифікації: концепт SecureStorageConcept вимагає від типу-бекенда фіксованого набору операцій із секретами і статичного прапора захищеності, а шаблонне сховище токенів параметризується саме через нього (рис 3.7).

Виконання контракту перевіряється статичним асертом (static_assert(SecureStorageConcept<KeychainBackend>)) у заголовку кожного бекенда, тому невідповідність виявляється на стадії компіляції. У штатних збірках застосунок створює сховище з бекендом на основі системної ключниці операційної системи; для модульних тестів той самий шаблонний клас інстанціюється з мок-бекендом, що дозволяє виконувати тести без зовнішніх залежностей.

```cpp
template<typename T>
concept SecureStorageConcept = requires(
  T storage,
  const T constStorage,
  const QString& key,
  const QString& value
) {
  // Store a secret value
  { storage.store(key, value) } -> std::same_as<void>;

  // Retrieve a secret value
  { constStorage.retrieve(key) } -> std::same_as<QString>;

  // Remove a secret
  { storage.remove(key) } -> std::same_as<void>;

  // Check if storage is secure (compile-time constant)
  { T::isSecure() } -> std::same_as<bool>;
};
```

Рисунок 3.7 – Визначення концепту SecureStorageConcept

Перевага концепта над класичним абстрактним інтерфейсом тут двоступенева: відсутні віртуальні таблиці і динамічна диспетчеризація, а перемикання бекенда відбувається на стадії компіляції - це втілення варіанта інверсії залежностей часу компіляції, описаного Б. Меєром: контракти модулів є частиною системи типів, а не лише документації [44].

3.9. Тестування розробленої системи

Тестування у проєкті побудовано як трирівневу піраміду, прийняту у класичній літературі з якості програмного забезпечення: широкий ярус швидких автоматичних юніт-тестів, тонший ярус інтеграційних тестів проти реальних провайдерів і ручний ярус сценаріїв графічного інтерфейсу [43]. Кожен ярус має власну зону відповідальності і власний інструмент: GoogleTest для юніт-тестів, Python з діагностичною утилітою командного рядка для інтеграційних тестів, чек-лист сценаріїв UC-01 – UC-05 для ручної перевірки. Сукупно вони закривають вимогу NFR-12 і дають об’єктивну основу для висновку про готовність застосунку до використання.

Перший ярус - юніт-тести. Проєкт містить 30 окремих файлів модульних тестів: 23 у бібліотеці поштових протоколів і 7 у доменному шарі графічного клієнта. У бібліотеці тести покривають шість категорій артефактів: серіалізатори IMAP- і SMTP-команд; синтаксичні аналізатори відповідей обох протоколів разом із токенизатором і парсером розширених кодів стану RFC 3463; MIME-стек (читач, записувач, будівельник повідомлень, поштові адреси, вкладення); утиліти (Base64, декодер модифікованого UTF-7 для імен IMAP-папок, валідатор протокольних команд із перевіркою відсутності CR/LF як захист від CRLF-ін'єкцій, генератор тегів IMAP-сесії); сесійні структури (доменна помилка протоколу, журнальне повідомлення і логер, серіалізація стартової конфігурації, абстракція над сокетами) [63]. Особливістю CMake-конфігурації є те, що кожен файл компілюється в окрему виконувану ціль, а не в один моноліт, що дає змогу повторно компілювати лише ті тести, у модулях яких відбулися зміни, і вибірково запускати їх через ctest із regex-фільтром [65]. Сукупно 23 файли містять близько 434 тест-кейсів GoogleTest (за підрахунком макросів `TEST(`/`TEST_F(` у тестових файлах бібліотеки), прохід повного набору на macOS Apple Silicon M3 Pro займає близько чотирьох секунд при холодному старті.

Сім файлів юніт-тестів доменного шару графічного клієнта зібрано за іншою схемою – як єдину виконувану ціль, що компонує всі сім файлів тестів. Причина – необхідність до запуску першого тесту створити єдиний для всього процесу екземпляр базового класу застосунку Qt: цього вимагають статичні фабрики Qt-рядків, серіалізатор JSON, файлова підсистема Qt і модуль баз даних, без яких неможливо протестувати ні конфігурацію, ні парсер MIME→Qt, ні кеш повідомлень. Власна точка входу тестового виконуваного файлу спершу ініціалізує GoogleTest, потім створює core-варіант базового класу застосунку (а не GUI-варіант, аби тест не вимагав активного підключення до віконного сервера) і лише потім запускає всі тести. Сім файлів покривають симетричне шифрування AES-GCM, серіалізацію конфігурації застосунку, модель і ключ повідомлення в кеші, in-

memory-варіант кешу повідомлень, адаптерний шар MIME→Qt і утилітарний санітизатор тексту.

Усі тестові цілі – як 23 окремі виконувані файли бібліотеки, так і консолідований набір модульних тестів графічного клієнта – реєструються через стандартний Cmake-механізм у відповідних файлах збірки. Це дозволяє однією командою прогнати всі тести після збірки з автоматичним підрахунком успішних і провалених. Увімкнення тестів контролюється Cmake-опцією, увімкненою за замовчуванням лише для діагностичних наборів, що знімає обчислювальне навантаження GoogleTest з Release-збірки. На macOS повний прохід усіх 30 тестових цілей займає близько чотирьох секунд при холодному старті, що робить юніт-тести природною частиною кожного циклу розробки.

Другий ярус – інтеграційні тести через діагностичну утиліту командного рядка. Модульні тести покривають парсинг команд, серіалізацію відповідей, побудову MIME-повідомлень, корисні утиліти; вони не покривають реальної поведінки проти живих IMAP- і SMTP-серверів. Цей пробіл закривають Python-сценарії з каталогу інтеграційних тестів поштового ядра: один із них запускає підпроцес діагностичної утиліти у режимі IMAP із реальними обліковими даними тестового акаунта, надсилає послідовність команд (LOGIN, SELECT, FETCH, IDLE, LOGOUT) і перевіряє очікувану структуру відповіді; другий тестує сценарії надсилання – від простого text/plain до multipart/related із вкладеннями. Підключення Python-обгортки замість написання інтеграційного тесту на C++ зумовлене практичними міркуваннями: Python спрощує парсинг JSON-виводу утиліти, керування змінними середовища, запуск підпроцесу з обмеженням за часом, асинхронну координацію «надсилання → очікування → перевірка». Діагностична утиліта, описана у розділі 3.1, тут виступає не як кінцевий продукт, а як адаптер між фреймворком тестування на Python і протокольним шаром C++. Інтеграційні тести не запускаються в обов’язковому циклі безперервної інтеграції чи у звичайному розробницькому циклі: вони вимагають реальних облікових даних,

доступу до Інтернету, можуть провалитися через тимчасові проблеми мережі чи обмеження частоти запитів на стороні провайдера. Їхня роль – періодична димова перевірка перед мажорними релізами, що виконується розробником вручну з контрольованого середовища.

Третій ярус – ручні сценарії графічного інтерфейсу. Найскладніша для автоматизації частина застосунку – взаємодія через інтерфейс: порядок натискань кнопок, момент появи модальних діалогів, текст у дочірніх вікнах. Перед кожним мажорним релізом виконується чек-лист із п’яти сценаріїв, що відповідають варіантам використання UC-01 – UC-05 з розділу 1.4. Кожен сценарій має чіткі передумови, кроки і постумови. Це не повноцінна автоматизація, проте й не вільний пошук, а структурований димовий тест, що дозволяє швидко виявити регресії візуального чи інтерактивного характеру.

Сукупно триярусна структура відповідає принципу тестової піраміди Соммервілла [43]: широка база швидких модульних тестів, середній шар інтеграційних, вузька верхівка ручних. Така стратифікація є компромісом між повнотою покриття і витратами часу, прийнятним для дипломного проєкту з чітко окресленою функціональністю.

3.10. Інструментація для діагностики, відлову помилок і вимірювання витрат ресурсів

Розробка програмного застосунку такого класу і обсягу неминуче супроводжується низкою дефектів, частина з яких виявляється лише на конкретних платформах і за певних послідовностей подій. Цей підрозділ описує засоби діагностики, які дозволяють отримати достатньо інформації про збій навіть після завершення процесу, а також інструментально перевірити заявлені у розділі 1.5 чутливість інтерфейсу (NFR-01), асинхронність введення-виведення (NFR-02) і економічність споживання ресурсів (NFR-11). Послідовно розглянуто

інструментацію аварійної діагностики на трьох цільових ОС, приклади дефектів, виявлених і виправлених під час розробки, а також методику і результати вимірювання витрат пам'яті, процесорного часу і часу запуску.

3.10.1. Інструментація аварійної діагностики

Підтримка аналізу фатальних помилок принципово відрізняється між UNIX-родинними системами і Windows. На macOS і Linux ядро штатно зберігає core-dump за фатальними сигналами SIGSEGV/SIGABRT, який далі завантажується у LLDB або GDB разом із бінарником зі символами; жодного спеціального коду у самому застосунку для отримання аварійного звіту не потрібно. На Windows таких механізмів немає: за невловленого винятку процес у режимі Release-збірки завершується системою без діагностичного виведення стека, а еквівалент core-dump – minidump – не формується автоматично без явного API-виклику з точки збою.

Цей розрив компенсовано Windows-специфічною підсистемою у точці входу графічного клієнта - двоступеневим обробником структурованих винятків (векторний обробник з пріоритетом над фрейм-залежними фільтрами як основний шлях, фрейм-залежний фільтр як резерв) і парним PowerShell-скриптом розкодування відносних адрес minidump-файлів через системну бібліотеку dbghelp.dll із PDB-символами поруч із бінарником, потреба в якому виникла при діагностиці проблеми, описаної у розділі 3.10.2. Спільна процедура формування звіту враховує умови потенційно зруйнованої купи: прапорець реентрантності захоплюється атомарною операцією, використовуються статичні буфери фіксованого розміру, адреса збою перетворюється у представлення «ім'я модуля + відносна адреса», інваріантне до ASLR. Підсистема замінює зовнішні аварійні фреймворки (Crashpad, Breakpad) у межах обсягу, потрібного на етапі розробки.

Поза описаною Windows-машинерією, протягом розробки використовувались стандартні зовнішні засоби діагностики, специфічні для кожної

ОС: на macOS – LLDB і пакет Apple Instruments (Time Profiler, Allocations, Leaks); на Linux – GDB у поєднанні з strace для діагностики мережевих сценаріїв, valgrind –tool=memcheck для перевірки на витоки і perf stat для збору лічильників процесора; на Windows – WinDbg для аналізу minidump-файлів і парний PowerShell-скрипт для швидкого розкодування адрес без запуску повноцінного дебаггера. Для діагностичних збірок на Windows додатково використовується окремий Cmake-набір зі зменшеним рівнем оптимізацій і повним записом PDB-символів, що зберігає переваги Release-конфігурації і повертає видимість кожного кадру стека.

3.10.2. Категорії дефектів, локалізованих за допомогою наявних засобів діагностики

Дисципліна асинхронної моделі, описана в розділі 2.3 - серіалізатор на з’єднання, заборона прямих звернень з потоку інтерфейсу до мережевих сесій, RAII-керування ресурсами - дозволила здебільшого уникнути класичних concurrency-дефектів кросплатформного нативного клієнта. Залишковий пул проблем зведено до трьох категорій, кожна з яких потребувала засобів різного рівня - від штатних core-dump-механізмів до динамічних санітайзерів виконання.

Висячі посилання у колбеку асинхронного запиту. У лямбдах-завершеннях REST-запитів до сервісу штучного інтелекту, а пізніше у частині обробників завершення IMAP-команд, припустилися помилки життєвого циклу: лямбда захоплювала посилання на буфер даних на стеку функції, тоді як HTTPS-запит виконувався асинхронно і обробник спрацьовував уже після того, як викликальний фрейм був звільнений – посилання вказувало у недійсну ділянку пам’яті. Дефект платформонезалежний за природою – ймовірність прояву визначається лише часовими відносинами між мережевою операцією і моментом руйнування викликального фрейму. Найшвидше він проявився саме на Windows: Release-збірка на цій платформі поєднує агресивну оптимізацію часу життя стека з усуненням

охоронних значень налагоджувача, тож звільнена пам’ять одразу перевикористовується наступним виділенням, перетворюючи перевикористання після звільнення на жорсткий збій. Локалізацію виконано шляхом аналізу зібраного minidump через PowerShell-скрипт, описаний у розділі 3.10.1, що однозначно вказав на лямбду-завершення REST-запиту. На UNIX той самий сценарій штатний core-dump відкривав у LLDB/GDB без додаткової машинерії, а AddressSanitizer і valgrind –tool=memcheck у діагностичних збірках підтвердили use-after-free у тій самій точці. Усунуто переходом на захоплення значення копії і відкладене знищення дочірнього об’єкта засобами циклу подій інтерфейсу.

Гонка даних на доступі до кешу повідомлень. Дворівневий кеш повідомлень за початковою реалізацією читався з потоку графічного інтерфейсу (для відкриття листа без блокування), а записи в обидва його рівні виконувалися з потоку мережевого ядра одразу після завершення FETCH BODY[]. Припущення про те, що операції з оперативним рівнем швидші за будь-яке планування і не потребують серіалізації, виявилося помилковим. У діагностичній збірці з ThreadSanitizer (Linux, Clang) було зафіксовано гонитву даних на внутрішньому списку черги доступу LRU при одночасному виконанні операції get зі сторони інтерфейсу і put зі сторони мережевого ядра, що проявлялась як спорадичні зависання інтерфейсу і періодичне повторне завантаження раніше витісненого запису з диска. Допоміжно «strace -f -e trace=futex» підтвердив, що під час інциденту жодних м’ютекс-операцій між двома потоками не виконувалось – отже, доступ був неконтрольованим. Архітектурне рішення – перевести всі мутації обох рівнів кешу на серіалізатор того ж IMAP-клієнта, до якого прив’язана сесія, і обмежити доступ з потоку інтерфейсу читанням, що повертається через черговану доставку сигналу Qt. Це ремонтує проблему за рахунок прийнятої у розділі 2.3 дисципліни «один серіалізатор на з’єднання» без уведення додаткових примітивів синхронізації, що узгоджується із загальною асинхронною моделлю застосунку.

Помилки життєвого циклу і фільтрації результатів, локалізовані довготривалим моніторингом ресурсів. Аналіз часових рядів інструментального моніторингу (розділ 3.10.3) - нетривіальне зростання резидентної пам'яті у 480-секундному вікні інтерактивного прогону і неочікуване зменшення розміру персистентного кешу за той самий прогін - дозволив виявити дві програмні помилки, що не проявлялися у коротких сценаріях UC-01 - UC-05. Перша - нестрога розмежованість сценаріїв виходу з облікового запису і перемикання активного у координаторі сесії: спільна процедура згортання сесії передавала команду на знищення персистентного кешу не лише у разі явного виходу, а й при кожному перемиканні між уже сконфігурованими обліковими записами, чим пояснювалося зменшення розміру кешу за прогін. Друга - відсутність фільтрації застарілих асинхронних відповідей у читачі повідомлень: при швидких послідовних натисканнях кожна команда FETCH BODY[] після завершення оновлювала інтерфейс без перевірки актуальності цільового UID, що частково пояснювало і підвищене процесорне навантаження у фазі стійкого стану. Обидві помилки виправлено явним розрізненням сценаріїв і фільтром прийому за актуальним ключем відображення, без перегляду архітектурної декомпозиції, наведеної в розділі2.

Три категорії об'єднує спільна риса: кожну з них надзвичайно важко виявити статичним аналізом або модульними тестами доменного шару, оскільки причина у кожному випадку лежить у взаємодії застосунку з зовнішнім середовищем - циклом подій Qt, ядром операційної системи у частині життя сокетів, фоновим циклом подій мережевого ядра чи характером користувацького навантаження на тривалому вікні спостереження. Саме для таких випадків і призначена сукупна інструментація, описана в розділах 3.10.1 і 3.10.3 - штатні core-dump-механізми UNIX-родини з LLDB/GDB і санітайзерами, Windows-компенсаційна підсистема minidump на платформі без автоматичного core-dump і довготривале спостереження за часовими рядами споживання ресурсів.

3.10.3. Методика і результати інструментального моніторингу ресурсів

Заявлені у розділі 1.5 нефункціональні вимоги NFR-01 (чутливість інтерфейсу під час фонових мережевих операцій), NFR-02 (асинхронність вводу-виводу у єдиному циклі подій без потоку на з'єднання) і NFR-11 (економічність споживання ресурсів у фазі стійкого стану) потребують інструментального підтвердження. Інструментальний моніторинг ресурсів, описаний у цьому підрозділі, виконує ще одну функцію поза прямим підтвердженням цих вимог: він слугує продовженням діагностичного інструментарію з розділів 3.10.1 - 3.10.2, бо аналіз отриманих часових рядів дозволив виявити дефекти, які не проявлялися на коротких ручних сценаріях UC-01 - UC-05.

Для відтворюваного знімання часових рядів реалізовано POSIX-shell-скрипт зовнішнього моніторингу ресурсів, що визначає момент готовності інтерфейсу за стабілізацією резидентного обсягу пам’яті після холодного старту, а далі періодично фіксує у CSV резидентну пам’ять, утилізацію центрального процесора і кількість потоків, окремо обчислюючи зріз останніх 60 секунд як фазу стійкого стану. Розмір шифрованого персистентного кешу знімається до і після кожного прогону. Скрипт запущено з Release-збіркою на пристрої Apple MacBook Pro M3 Pro з macOS 15.7.5 та 36 ГБ ОЗП з двома зареєстрованими обліковими записами Gmail зі співмірними обсягами вхідних повідомлень для двох робочих профілів. Перший – інтерактивний: холодний запуск, повна автоматична синхронізація обох папок INBOX, безперервна серія ручних дій оператора-тестувальника (послідовне гортання списку зі стабільним темпом, відкриття десятків повідомлень, перемикання між обліковими записами) і фаза спокою з активним IDLE; для усереднення показників цей профіль повторено серією багаторазових запусків тривалістю 8 хвилин кожен, а наведені нижче числа є усередненими значеннями по

серії. Другий - неінтерактивний, уведений для контрольної оцінки чистого стійкого стану без накладеного користувацького навантаження: холодний запуск, Oauth-авторизація, повна автоматична синхронізація INBOX і подальша тривала фаза спокою з активним IDLE, без жодних дій оператора після завершення першої синхронізації.

Зняті метрики охоплюють час холодного запуску, резидентний обсяг пам’яті одразу після старту і у фазі стійкого стану, середню і пікову частку процесора у фазах синхронізації і спокою, кількість потоків (як побічний індикатор стабільності пулу мережевого циклу подій) і розмір шифрованого персистентного кешу на диску. Усереднені показники серії інтерактивних прогонів подано у таблиці 3.1; підсумки контрольного неінтерактивного прогону подано далі у таблиці 3.2 разом із заключною інтерпретацією.

Таблиця 3.1 – Усереднені показники серії інтерактивних прогонів тривалістю 480 секунд кожен

Метрика	Значення
Час холодного запуску до готовності	8 с
Резидентна пам’ять одразу після холодного запуску	71 МБ
Резидентна пам’ять після завершення першої синхронізації (t≈45-75 с)	228 МБ
Пік частки процесора у момент FETCH ENVELOPE (t=37 с)	62.4 % одного ядра
Усереднена резидентна пам’ять за весь 480-с прогін	333.5 МБ
Резидентна пам’ять у фазі стійкого стану (останні 60 с)	мін 403 / сер 408.5 / макс 410 МБ
Середня частка процесора у фазі стійкого стану	9.79 %
Стабільний інтервал кількості потоків	6 – 21 (медіана 12)
Розмір шифрованого кешу на диску до прогону	5.9 МБ
Розмір шифрованого кешу на диску після прогону	3.0 МБ

З усереднених по серії інтерактивних прогонів показників таблиці 3.1 три позиції безпосередньо підтверджують відповідні нефункціональні вимоги. Час холодного запуску у 8 секунд від виклику виконуваного файлу до моменту готовності інтерфейсу значно менший за типовий запуск важких клієнтів - Mozilla Thunderbird на схожому обладнанні стартує за 5 - 15 секунд при холодному кеші підпису і має відомі проблеми з gloda-індексуванням; резидентна пам'ять одразу після запуску - 71 МБ - підтверджує NFR-11 за відсутності користувацьких даних [4]. Резидентна пам'ять після завершення першої синхронізації обох папок INBOX тримається на рівні 228 МБ, що на порядок менше від типових витрат Mozilla Thunderbird з активним gloda-індексуванням (за даними Mozilla Bugzilla - від 800 МБ до 2 ГБ) і нового Outlook у Electron-обгортці (за повідомленнями Microsoft Learn - 400 - 700 МБ і вище) [4][5]. Фактична частка процесора поза піками FETCH ENVELOPE тримається у межах одиниць відсотків одного ядра, що відповідає очікуваній з архітектури (розділ 2.4) неблокуючій поведінці циклу IDLE.

Натомість три інші позиції з тієї самої таблиці на перший погляд не узгоджуються з очікуваннями. Резидентна пам'ять зросла з 228 МБ після першої синхронізації до 410 МБ у кінці прогону, причому навіть формальний зріз останніх 60 секунд показує мінімум 403, середнє 408.5 і максимум 410 МБ - у формально стійкій фазі помітне повільне зростання. Середня частка процесора у тій самій фазі - 9.79 % - суттєво вища за очікувану в режимі спокою. Розмір шифрованого кешу на диску за час прогону зменшився з 5.9 МБ до 3.0 МБ, тобто частина персистентних даних була видалена у штатному режимі. Аналіз цих аномалій у поєднанні з прямою інспекцією коду виявив дві програмні помилки - у координаторі сесії та читачі повідомлень, - систематизовані як третя категорія дефектів у розділі 3.2. Решту розходження пояснює характер самого інтерактивного профілю.

Після виправлення обох помилок залишковий розмах зростання резидентної пам'яті за 8 хвилин (+180 МБ) і медіанна частка процесора у фазі стійкого стану (9.79 % одного ядра) для інтерактивного режиму узгоджуються з характером

сценарію і не є ознакою дефекту. Профіль прогону свідомо інтерактивний: упродовж усього вікна спостереження тестувальник у лиці автора кваліфікаційної роботи безперервно гортав список повідомлень і відкривав нові листи, чим примушував поштову сесію виконувати додаткові FETCH ENVELOPE / FETCH BODY[], а багаторівневий кеш - приймати у свій оперативний шар нові тіла повідомлень до досягнення меж LRU-обмежень. Таким чином, спостережувані метрики відображають штатну реакцію кешу на безперервне навантаження читанням - нові повідомлення підвантажуються швидше, ніж LRU встигає витіснити старі для конкретного 480-секундного вікна.

Для контрольної перевірки цієї інтерпретації виконано окремий неінтерактивний прогін із зафіксованим робочим профілем «холодний запуск + OAuth-авторизація + автоматична синхронізація INBOX + тривала фаза `IDLE`», без жодних дій оператора після першої синхронізації. Підсумкові показники прогону подано у таблиці 3.10.2.

Таблиця 3.2 – Показники контрольного неінтерактивного прогону

Метрика	Значення
Час холодного запуску до готовності	8 с
Резидентна пам'ять одразу після холодного запуску	72 МБ
Резидентна пам'ять після завершення першої синхронізації INBOX (t ≈ 18 с)	216 МБ
Пік утилізації центрального процесора у момент FETCH ENVELOPE (t = 15 с)	9.3 % одного ядра
Тривалість фази стійкого стану у прогоні	≈ 360 с (від t ≈ 41 с до t ≈ 399 с)
Резидентна пам'ять у фазі стійкого стану (мін / сер / макс)	215 / 215 / 215 МБ (нульова дисперсія)
Середня утилізація центрального процесора у фазі стійкого стану	≈ 0 % одного ядра

Стабільна кількість потоків у фазі стійкого стану	6 (медіана; рідкі сплески до 7)

Контрольний прогін інструментально підтверджує наведену вище інтерпретацію інтерактивного режиму. По-перше, час холодного старту (8 с) і резидентна пам'ять одразу після нього (72 МБ) збігаються з відповідними показниками інтерактивної серії з точністю до похибки вибірки, що свідчить про детермінованість фази ініціалізації незалежно від робочого профілю. По-друге, після завершення першої синхронізації INBOX резидентна пам'ять стабілізується на рівні 216 МБ - на 12 МБ нижче аналогічного показника інтерактивного прогону (228 МБ), що відповідає очікуваній різниці на обсяг тіл повідомлень, не довантажених у кеш через відсутність відкривань листів. По-третє, і це ключовий результат, упродовж приблизно 360 секунд фази стійкого стану резидентна пам'ять залишається абсолютно сталою на рівні 215 МБ (нульова варіація між мінімумом, середнім і максимумом за зрізом останніх 60 секунд), а утилізація центрального процесора у тій самій фазі тримається на рівні близько 0 % одного ядра (одиничні зчитування 0.1 % на тлі стабільного нуля). Кількість потоків стабілізується на нижній межі діапазону інтерактивного прогону (6 проти медіани 12 у режимі активного гортання), що відображає відсутність контекстів, породжуваних активною інтерактивною взаємодією. Зведене порівняння двох робочих профілів - інтерактивного (таблиця 3.1) і неінтерактивного (таблиця 3.2) - однозначно показує, що зростання +180 МБ і базове навантаження 9.79 % у «фазі стійкого стану» інтерактивного прогону повністю пояснюються безперервним користувацьким навантаженням FETCH ENVELOPE/FETCH BODY[] і не свідчать ні про витік пам'яті, ні про надмірну фонову активність ядра у режимі IDLE. Зафіксована нульова дисперсія резидентної пам'яті за 360 секунд спостереження є інструментально найсильнішим підтвердженням стабільності, яке може дати

поодинокий прогін: подальше серійне повторення у тій самій конфігурації не додає статистичної цінності, бо варіація відсутня.

Інструментальний моніторинг ресурсів, отже, виконав дві узгоджені функції: підтвердив відповідність NFR-01, NFR-02 і NFR-11 у двох взаємодоповнювальних робочих профілях (де неінтерактивний прогін додатково ізолював режимі IDLE багаторівневого кешу як абсолютно стабільне плато 215 МБ) і надав часові ряди, аналіз яких локалізував дві програмні помилки, систематизовані як третя категорія дефектів у розділі 3.2 - у дусі рекомендації Соммервілла про безперервне спостереження за поведінкою застосунку у виробничих умовах [43].

3.11. Локалізація і функціональна демонстрація

Локалізацію реалізовано через стандартну механіку Qt LinguistTools [42]. Усі рядки інтерфейсу обгорнуто у штатний макрос перекладу Qt, що при зборі утилітою lupdate потрапляють до українського каталогу перекладів графічного клієнта - XML-файла, де кожен запис містить вихідний рядок, його контекст-клас, переклад і за потреби коментар-підказку для перекладача. Утиліта lrelease компілює цей каталог у бінарний файл, який вкладається у бінарник застосунку через механізм ресурсів Qt. Активна мова визначається з конфігурації застосунку і підвантажується у точці входу через стандартний завантажувач перекладу Qt. Додавання нової мови зводиться до додавання нового файла перекладу і одного рядка у файлі ресурсів; жодних модифікацій логічного коду не потрібно, що відповідає вимозі NFR-10.

Функціональну демонстрацію подано через серію знімків, що документують типові користувацькі сценарії на трьох цільових ОС. Знімки розподілено між операційними системами для одночасного покриття NFR-03 і ключових функціональних груп - автентифікації, читання повідомлення, надсилання листа з підтвердженням наскрізної доставки, роботи ШІ-асистента у двох режимах. Щоб не розривати безперервний виклад розділу серією зображень підряд, повний набір

знімків (рисунки Ґ.1 - Ґ.8 з підрисунками виду Ґ.х.у там, де один сценарій документується кількома суміжними кадрами) фізично розміщено у Додатку Д; у таблиці 3.3 подано лише навігаційний каталог із прив'язкою кожного знімка до сценарію, вимоги і цільової ОС.

Таблиця 2.3 - Каталог знімків функціональної демонстрації

№ рис.	ОС	Сценарій / вимога	Що зображено
Д.1, Д.2	Windows 11	UC-01	Послідовні екрани згоди Google OAuth після перенаправлення з застосунку із PKCE-параметрами: вступний екран згоди (Д.1) і деталізований перелік запитуваних scope (Е.2)
Д.3	macOS 15	UC-01 (завершальна фаза)	HTML-сторінка успішної автентифікації, рендерована локальним приймачем (розділ 3.6)
Д.4	Ubuntu 24.04	FR-12	Список облікових записів і папок із кількома підключеними акаунтами
Д.5	macOS 15	UC-02, FR-01, FR-10, FR-11	Головне вікно з трьома панелями і відкритим листом у безпечному переглядачі HTML
Д.6	macOS 15	UC-03 (підготовча фаза)	Діалог редактора нового листа з полями отримувача, теми, тіла

Продовження таблиці 3.3

Д.7, Д.8	Windows 11; стороннє веб-вікно поштового провайдера	UC-03, FR-03	Підтвердження наскрізної доставки: лист з'являється в колонці «Надіслані» застосунку (Д.7), а на стороні отримувача той самий лист спостерігається у вхідних повідомленнях стороннього клієнта (Д.8)
Д.9, Д.10	Ubuntu 24.04	UC-05, FR-08, NFR-09	Модальне вікно ШІ-асистента з результатом від Gemini у двох режимах виклику: перевірка граматики (Д.9) і перетворення тону на формальніший (Д.10)
Д.11	Windows 11	UC-05	Діалог налаштувань ШІ-провайдера зі зберіганням API-ключа у Credential Manager

3.12. Верифікація вимог

Підсумкову відповідність вимогам з розділу 1.5 формалізовано у вигляді двох трасувальних таблиць Додатку Ж. Кожному рядку відповідає одна вимога; колонка «Реалізаційний артефакт» вказує конкретний модуль або CMake-таргет із перехресним посиланням на основний підрозділ розділу 2 або 3, де артефакт обґрунтовано, а колонка «Спосіб перевірки» - застосовний контрольний механізм за трирівневою стратегією тестування з розділу 3.9, з додатковим посиланням на інструментальний моніторинг з розділу 3.10.3 для характеристик, що не мають сенсу як точкова перевірка (продуктивність, стабільність, відсутність витоків), і на Додаток Д - для тих, що верифікуються візуально. Така матриця слугує стандартним

інструментом верифікації за моделями ISO/IEC/IEEE 12207 та ISO/IEC 25010 [31][43].

За результатами трасування 23 з 24 вимог реалізовано повністю. Нефункціональні NFR-01, NFR-02 і NFR-11 додатково підтверджено інструментальним моніторингом, описаним у розділі 3.10.3, у двох взаємодоповнювальних робочих профілях, де контрольний неінтерактивний прогін зафіксував нульову дисперсію резидентної пам'яті за 360 секунд фази стійкого IDLE-стану. Кросплатформну збірку у режимі повного циклу інструментального моніторингу емпірично перевірено на macOS Apple Silicon, Windows 11 та Ubuntu 24.04 LTS, що завершує підтвердження NFR-03 на повному наборі цільових платформ. Єдину позицію з частковою реалізацією - FR-04 у частині безпечного рендерингу HTML - і похідні з обсягових обмежень напрями подальшого розвитку систематизовано у розділі 3.13.

3.13. Невиконані вимоги та напрями подальшого розвитку

Частково реалізована вимога - FR-04 у частині безпечного рендерингу HTML. Базовий шар очищення тексту (розділ 3.5) реалізує лише фільтрацію символів за дозволеними категоріями Unicode і виправлення некоректних RGBA-значень у CSS; повноцінного санітайзера тегів і атрибутів за рекомендаціями довідника OWASP з протидії XSS не виконано. Вимкнення виконання сценаріїв у переглядачі Qt лише частково компенсує ризик і не замінює санітайзера, тому пасивні вектори (атрибути-обробники подій, посилальні таблиці стилів зі сторонніх URI, `data:`-зображення) у поточній реалізації не нейтралізуються.

Напрями подальшого розвитку. Жодна з перелічених нижче позицій не блокує штатної роботи прототипу:

- повноцінний HTML-санітайзер тегів і атрибутів за рекомендаціями OWASP [53] з покриттям модульними тестами на стандартні XSS-вектори, що закриває залишок FR-04;
- повна синтаксична сумісність парсера відповідей типу `* VANISHED` з усіма формами RFC 7162: поточна реалізація (розділ 2.4.1) надійно обробляє формати `* VANISHED (EARLIER) <set>` і `* VANISHED <set>` з UID-послідовностями, достатні для основних публічних провайдерів, але не охоплює рідкісні синтаксичні варіанти;
- паралельна підтримка протоколу JMAP (RFC 8620/8621) як сучасної альтернативи IMAP у провайдерів, що його надають;
- автоматизація сценаріїв графічного інтерфейсу на основі Qt Test або Squish, що замінить наявний ручний чек-лист з розділу 3.9 і дозволить ефективніше виявляти UI-регресії;
- безперервна інтеграція з прогоном всіх видів наявних в проєкті тестів на трьох цільових ОС;
- зберігання майстер-ключа кешу у захищеному апаратному середовищі (Secure Enclave на Apple, TPM 2.0 на Windows, Strongbox на сумісних Linux-збірках) замість програмного ключа у системній ключниці;
- доопрацювання інтерфейсних рішень головного вікна, редактора і діалогу налаштувань за результатами фокус-групових випробувань UC-01 - UC-05 без зміни функціонального обсягу.

ВИСНОВКИ

У роботі вирішено комплексне завдання проєктування та програмної реалізації нативного кросплатформного десктопного поштового клієнта з підтримкою IMAP і SMTP, безпечною автентифікацією за OAuth 2.0 з розширенням PKCE та архітектурно ізольованою інтеграцією зі стороннім сервісом штучного інтелекту. На основі аналізу сучасних десктопних клієнтів виокремлено функціональну нішу - ресурсо-економний когнитивно-розвантажений застосунок з ізольованим ШІ-модулем - і сформульовано набір функціональних та нефункціональних вимог за моделлю якості ISO/IEC 25010.

Обрано технологічний стек C++23 і Qt 6 з керуванням ресурсами за ідіомою RAII, без середовища виконання зі збиранням сміття і без вбудованого браузерного рушія, що дає передбачуване споживання пам'яті. Систему побудовано як багаторівневу архітектуру з двома незалежними циклами подій - потоком графічного інтерфейсу і фоновим мережевим ядром за шаблоном Reactor поверх epoll/kqueue/IOCP. До неї додано доменну модель даних, дворівневий локальний шифрований кеш повідомлень з AES-256-GCM і узгоджену стратегію обробки помилок.

Мережеву підсистему реалізовано у двох незалежних модулях: IMAP-сесія підтримує режим `IDLE` і опційну інкрементну синхронізацію, а SMTP-надсилання виконується через submission-порт зі STARTTLS і SASL XOAUTH2. Безпечну делеговану автентифікацію реалізовано за OAuth 2.0 з розширенням PKCE відповідно до RFC 8252 і RFC 7636 з підтримкою кількох одночасних облікових записів і збереженням токенів у системному сховищі ключів. Модуль штучного інтелекту виконано як ізольований REST-клієнт без зворотних залежностей із поштовим та авторизаційним модулями, завдяки чому його збій або недоступність не впливають на штатну роботу решти підсистем застосунку.

Програмний прототип побудовано як єдиний кросплатформний проєкт зі збіркою на macOS, Linux і Windows; до нього застосовано статичний аналіз та

багаторівневу стратегію тестування. Інструментальний моніторинг ресурсів підтвердив виконання нефункціональних вимог щодо стабільності споживання пам'яті у режимі тривалого очікування пошти.

Теоретичним результатом роботи є референтна архітектура нативного кросплатформного поштового клієнта з адаптацією потоку OAuth 2.0 + PKCE до сценарію кількох облікових записів і схемою ізоляції допоміжного ШІ-модуля; її можна використовувати у наступних інженерних проєктах подібного класу та у навчальному процесі з дисциплін системного й мережевого програмування, прикладної криптографії і сучасної розробки на C++ і Qt.

Практичним результатом є набір самодостатніх інженерних рішень, придатних до перевикористання у споріднених проєктах: асинхронне мережеве ядро на основі шаблону Reactor з безпечним мостом до сигнально-слотового механізму Qt 6, дворівневий локальний шифрований кеш повідомлень з AES-256-GCM, а також реалізація OAuth-входу з PKCE поверх узагальненого сховища секретів на основі системної ключниці.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Tarafdar M., Wenninger H., Stich J.-F. Email Overload: Investigating Technology-fit Antecedents and Job-related Outcomes. The Data Base for Advances in Information Systems. 2022. URL: https://www.jfstich.com/data/medias/papers/tarafdar-et-al-2023-email-overload.pdf (дата звернення: 04.05.2026).
2. Friedrichs J., Kern M., Ohly S., Ďuranová L. Drowning in emails: investigating email classes and work stressors as antecedents of high email load and implications for well-being. Frontiers in Psychology. 2024. Vol. 15. Article 1439070. URL: https://www.frontiersin.org/journals/psychology/articles/10.3389/fpsyg.2024.1439070 (дата звернення: 04.05.2026).
3. Apple. Mail user guide for Mac. URL: https://support.apple.com/guide/mail/ (дата звернення: 15.03.2026).
4. Mozilla Bugzilla. Bug 526568: Consumes too much memory, nearly 2 GB with gloda indexing on, 800 MB with gloda indexing off. URL: https://bugzilla.mozilla.org/show_bug.cgi?id=526568 (дата звернення: 04.05.2026).
5. Microsoft Learn. New Outlook memory issues: incident MO907654. 2024. URL: https://learn.microsoft.com/en-us/answers/questions/4674369/how-to-fix-outlook-(new)-memory-leak (дата звернення: 04.05.2026).
6. Sams B. Microsoft is wrong: the new Outlook for Windows is not ready for prime time. Windows Central. 2024. URL: https://www.windowscentral.com/software-apps/windows-11/microsoft-is-wrong-the-new-outlook-for-windows-is-not-ready-for-prime-time (дата звернення: 04.05.2026).
7. Cruz L., Malavolta I., Lago P. Electron vs. web: a comparative analysis of energy and performance in communication apps. Proceedings of the 17th International Conference on the Quality of Information and Communications Technology (QUATIC 2024). Vrije Universiteit Amsterdam, 2024. URL:

https://research.vu.nl/en/publications/electron-vs-web-a-comparative-analysis-ofenergy-andperformance-in/ (дата звернення: 04.05.2026).

8. Shortwave, Inc. Everything Shortwave shipped in 2024. 2024. URL: https://www.shortwave.com/blog/2024-new-shortwave-features-ai-email-calendar-business-teams (дата звернення: 04.05.2026).
9. Stevens W. R., Fenner B., Rudoff A. M. UNIX Network Programming. Vol. 1: The Sockets Networking API. 3rd ed. Boston: Addison-Wesley, 2004. 991 p.
10. Schmidt D. C. Reactor: an Object Behavioral Pattern for Demultiplexing and Dispatching Handles for Synchronous Events. Pattern Languages of Program Design / eds. J. O. Coplien, D. C. Schmidt. Boston: Addison-Wesley, 1995. P. 529 - 545.
11. Internet Engineering Task Force. RFC 8252: OAuth 2.0 for Native Apps: Best Current Practice 212. October 2017. URL: https://www.rfc-editor.org/rfc/rfc8252 (дата звернення: 04.05.2026).
12. Sakimura N., Bradley J., Agarwal N. Proof Key for Code Exchange by OAuth Public Clients: RFC 7636. IETF, 2015. URL: https://datatracker.ietf.org/doc/html/rfc7636 (дата звернення: 15.03.2026).
13. Klensin J. C. Simple Mail Transfer Protocol: RFC 5321. IETF, 2008. URL: https://datatracker.ietf.org/doc/html/rfc5321 (дата звернення: 15.03.2026)
14. Hoffman P. SMTP Service Extension for Secure SMTP over Transport Layer Security: RFC 3207. IETF, 2002. URL: https://datatracker.ietf.org/doc/html/rfc3207 (дата звернення: 04.05.2026).
15. Gellens R., Klensin J. Message Submission for Mail: RFC 6409. IETF, 2011. URL: https://datatracker.ietf.org/doc/html/rfc6409 (дата звернення: 04.05.2026).
16. Newman C. Using TLS with IMAP, POP3 and ACAP: RFC 2595. IETF, 1999. URL: https://datatracker.ietf.org/doc/html/rfc2595 (дата звернення: 07.04.2026).
17. Moore K., Newman C. Cleartext Considered Obsolete: Use of Transport Layer Security (TLS) for Email Submission and Access: RFC 8314. IETF, 2018. URL: https://datatracker.ietf.org/doc/html/rfc8314 (дата звернення: 04.05.2026).

18. Crispin M. Internet Message Access Protocol - Version 4rev1: RFC 3501. IETF, 2003. URL: https://datatracker.ietf.org/doc/html/rfc3501 (дата звернення: 15.03.2026).
19. Leiba B. IMAP4 IDLE command: RFC 2177. IETF, 1997. URL: https://datatracker.ietf.org/doc/html/rfc2177 (дата звернення: 04.05.2026).
20. Melnikov A., Cridland D. IMAP Extensions: Conditional STORE Operation or Quick Flag Changes Resynchronization (CONDSTORE) and Quick Mailbox Resynchronization (QRESYNC): RFC 7162. IETF, 2014. URL: https://datatracker.ietf.org/doc/html/rfc7162 (дата звернення: 15.03.2026).
21. Resnick P. Internet Message Format: RFC 5322. IETF, 2008. URL: https://datatracker.ietf.org/doc/html/rfc5322 (дата звернення: 15.03.2026).
22. Freed N., Borenstein N. Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies: RFC 2045. IETF, 1996. URL: https://datatracker.ietf.org/doc/html/rfc2045 (дата звернення: 26.04.2026).
23. Freed N., Borenstein N. Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types: RFC 2046. IETF, 1996. URL: https://datatracker.ietf.org/doc/html/rfc2046 (дата звернення: 26.04.2026).
24. Levinson E. The MIME Multipart/Related Content-type: RFC 2387. IETF, 1998. URL: https://datatracker.ietf.org/doc/html/rfc2387 (дата звернення: 26.04.2026).
25. Rescorla E. The Transport Layer Security (TLS) Protocol Version 1.3: RFC 8446. IETF, 2018. URL: https://datatracker.ietf.org/doc/html/rfc8446 (дата звернення: 07.04.2026).
26. Hardt D. The OAuth 2.0 Authorization Framework: RFC 6749. IETF, 2012. URL: https://datatracker.ietf.org/doc/html/rfc6749 (дата звернення: 15.03.2026).
27. Melnikov A., Zeilenga K. Simple Authentication and Security Layer (SASL): RFC 4422. IETF, 2006. URL: https://datatracker.ietf.org/doc/html/rfc4422 (дата звернення: 04.05.2026).
28. Google. Sign in to your mail client using XOAUTH2 (Help me write emails in Gmail). URL: https://support.google.com/mail/answer/6337898 (дата звернення: 15.03.2026).

29. The Qt Company. Qt 6 Documentation. URL: https://doc.qt.io/qt-6/ (дата звернення: 15.03.2026).
30. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. Boston: Addison-Wesley, 1994. 395 p.
31. Bass L., Clements P., Kazman R. Software Architecture in Practice. 3rd ed. Boston: Addison-Wesley, 2012. 624 p.
32. ISO/IEC 14882:2024. Programming languages - C++. International Organization for Standardization, 2024. 1900 p.
33. Mozilla Foundation. Thunderbird - free your inbox. URL: https://www.thunderbird.net/ (дата звернення: 15.03.2026).
34. Mozilla Wiki. Thunderbird:IMAP. URL: https://wiki.mozilla.org/Thunderbird:IMAP (дата звернення: 15.03.2026).
35. Thunderbird is getting a major redesign - this is what it looks like. OMG! Linux. URL: https://www.omglinux.com/major-thunderbird-redesign-early-look/ (дата звернення: 18.04.2026).
36. Gallagher W. How to replace Apple Mail on the Mac, and why you might want to switch. AppleInsider. 2018. URL: https://appleinsider.com/articles/18/08/13/how-to-replace-apple-mail-on-the-mac-and-why-you-might-want-to-switch (дата звернення: 18.04.2026).
37. Best email app for Macs. Macworld. URL: https://www.macworld.com/article/668716/best-email-app-for-macs.html (дата звернення: 18.04.2026).
38. Mailspring. Mailspring - the best free email app. URL: https://getmailspring.com/ (дата звернення: 15.03.2026).
39. Erz H. On Electron, the Bloated Web, and Trade-Offs. 2021. URL: https://www.hendrik-erz.de/post/electron-bloated-web-and-trade-offs (дата звернення: 04.05.2026).

40. GNOME Project. Geary email client. URL: https://wiki.gnome.org/Apps/Geary (дата звернення: 15.03.2026).
41. ISO/IEC 25010:2011. Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models. International Organization for Standardization, 2011. 34 p.
42. The Qt Company. Qt Linguist Manual: writing source code for translation; lupdate, lrelease and the .ts/.qm catalogue format. URL: https://doc.qt.io/qt-6/qtlinguist-index.html (дата звернення: 05.05.2026).
43. Sommerville I. Software Engineering. 10th ed. Boston: Pearson, 2016. 768 p.
44. Meyer B. Object-Oriented Software Construction. 2nd ed. Upper Saddle River, NJ: Prentice Hall, 1997. 1296 p.
45. Про захист персональних даних: Закон України від 01.06.2010 № 2297-VI. Відомості Верховної Ради України. 2010. № 27. Ст. 273. URL: https://zakon.rada.gov.ua/laws/show/2297-17 (дата звернення: 07.04.2026).
46. Apple Inc. Keychain services. Apple Developer Documentation. URL: https://developer.apple.com/documentation/security/keychain_services (дата звернення: 04.05.2026).
47. Litmus. Email Client Market Share - обзор: розподіл часток поштових клієнтів та інфраструктури повідомлень. URL: https://www.litmus.com/email-client-market-share (дата звернення: 09.05.2026).
48. Google. Using OAuth 2.0 for desktop apps; OAuth API verification FAQs (incl. CASA Tier 2/3 assessment for restricted scopes). Google Identity Platform Documentation. URL: https://developers.google.com/identity/protocols/oauth2/native-app (дата звернення: 09.05.2026).
49. Vaswani A., Shazeer N., Parmar N., Uszkoreit J., Jones L., Gomez A. N., Kaiser Ł., Polosukhin I. Attention is all you need. Advances in Neural Information Processing Systems. 2017. Vol. 30. P. 5998–6008.

50. Google. Gemini API documentation. URL: https://ai.google.dev/gemini-api/docs (дата звернення: 15.03.2026).
51. Object Management Group. OMG Unified Modeling Language (OMG UML), Version 2.5.1: formal/2017-12-05. December 2017. URL: https://www.omg.org/spec/UML/2.5.1/ (дата звернення: 07.04.2026).
52. Fowler M. UML Distilled: A Brief Guide to the Standard Object Modeling Language. 3rd ed. Boston: Addison-Wesley Professional, 2004. 175 p.
53. OWASP Foundation. Cross Site Scripting Prevention Cheat Sheet. OWASP Cheat Sheet Series. URL: https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html (дата звернення: 04.05.2026).
54. Kitware, Inc. CMake Reference Documentation: cmake-presets(7), find_package(), add_subdirectory(). URL: https://cmake.org/cmake/help/latest/ (дата звернення: 06.05.2026).
55. Microsoft. vcpkg Manifest Mode Documentation. URL: https://learn.microsoft.com/en-us/vcpkg/concepts/manifest-mode (дата звернення: 06.05.2026).
56. Lohmann N. JSON for Modern C++ (nlohmann/json), version 3.12. URL: https://github.com/nlohmann/json (дата звернення: 06.05.2026).
57. Apple Inc. Bundle Programming Guide. Apple Developer Documentation. URL: https://developer.apple.com/library/archive/documentation/CoreFoundation/Conceptual/CFBundles/Introduction/Introduction.html (дата звернення: 06.05.2026).
58. The Qt Company. Qt Widgets Module Documentation. URL: https://doc.qt.io/qt-6/qtwidgets-index.html (дата звернення: 06.05.2026).
59. The Qt Company. Qt Style Sheets Reference. URL: https://doc.qt.io/qt-6/stylesheet.html (дата звернення: 06.05.2026).

60. LLVM Project. Clang-Tidy: a clang-based C++ "linter" tool, and ClangFormat: a tool for formatting C/C++ code. URL: https://clang.llvm.org/extra/clang-tidy/ (дата звернення: 06.05.2026).
61. KDAB. Clazy: Qt-oriented static code analyzer based on Clang framework. URL: https://github.com/KDE/clazy (дата звернення: 06.05.2026).
62. van Heesch D. Doxygen: a documentation system for C++. URL: https://www.doxygen.nl/ (дата звернення: 06.05.2026).
63. Vaudreuil G. Enhanced Mail System Status Codes: RFC 3463. IETF, 2003. URL: https://datatracker.ietf.org/doc/html/rfc3463 (дата звернення: 09.05.2026).
64. Jones M., Bradley J., Sakimura N. JSON Web Token (JWT): RFC 7519. IETF, 2015. URL: https://datatracker.ietf.org/doc/html/rfc7519 (дата звернення: 09.05.2026).
65. Google. GoogleTest: Google's C++ test framework. URL: https://google.github.io/googletest/ (дата звернення: 06.05.2026).

ДОДАТОК А

Послідовність типового циклу IMAP-сесії з фазою IDLE

Рисунок 1.А - UML-діаграма послідовності типового циклу IMAP-сесії з фазою IDLE

ДОДАТОК Б

Сценарії використання та вимоги

Таблиця Б.1 - Базові сценарії використання

ID	Назва	Стислий опис	Очікуваний технічний результат
UC-01	Авторизація користувача	Додавання облікового запису через OAuth 2.0 + PKCE з відкриттям системного браузера	Отримання та безпечне збереження пари токенів доступу та оновлення у системному сховищі
UC-02	Синхронізація скриньки	Оновлення стану папки IMAP, у тому числі за допомогою CONDSTORE/QRESYNC, якщо сервер їх підтримує	Інкрементне завантаження лише нових або змінених метаданих у фоновому потоці
UC-03	Читання повідомлення	Вибір листа зі списку, завантаження тіла та розбір MIME-структури	Неблокуюче відображення тексту і вкладень у графічному інтерфейсі
UC-04	Надсилання листа	Створення повідомлення з текстовим тілом і довільною кількістю файлових вкладень та його передача через SMTP	Надсилання без блокування інтерфейсу, з прозорим звітом про результат

UC-05	ШІ-асистування	Звернення до стороннього провайдера моделі для покращення тексту листа	Асинхронний REST-запит і відображення альтернативної редакції із збереженням оригіналу

Таблиця Б.2 - Функціональні вимоги до системи

ID	Назва	Стислий опис	Сценарій	Джерело
FR-01	Підключення до IMAP-сервера	Перегляд переліку папок та повідомлень у вибраній папці	UC-02, UC-03	RFC 3501
FR-02	Делегована автентифікація OAuth 2.0 + PKCE	Безпечний вхід із відкриттям сторінки автентифікації у системному браузері	UC-01	RFC 6749, RFC 7636, RFC 8252
FR-03	Надсилання через SMTP-submission	Передавання повідомлень через submission-порт із обов'язковим TLS і автентифікацією SASL XOAUTH2	UC-04	RFC 5321, RFC 2595, специфікація SASL XOAUTH2
FR-04	Розбір MIME-структури отриманих повідомлень	Підтримка multipart/alternative, multipart/related, multipart/mixed та inline-вкладень через Content-ID	UC-03	RFC 5322, RFC 2045/2046/2387

FR-05	Формування MIME-структури вихідного повідомлення	Генерація multipart/mixed із текстовим тілом і довільною кількістю файлових вкладень	UC-04	RFC 5322, RFC 2045/2046
FR-06	Перегляд списку повідомлень із прапорцями	Відображення \Seen, \Answered, \Flagged та збереження стану між сесіями	UC-02	RFC 3501
FR-07	Локальне дворівневе кешування тіл повідомлень	LRU-рівень в оперативній пам'яті і шифрований персистентний рівень (SQLite + AES-256-GCM з окремим ключем на обліковий запис)	UC-02, UC-03	NFR-11, обґрунтування у вступі
FR-08	Структуроване журналювання	Логування поштових і авторизаційних подій без розкриття значень токенів і вмісту листів	усі	NFR-08, ISO/IEC 25010
FR-09	Узгоджена обробка помилок	Реакція на мережеві, протокольні та авторизаційні збої з можливістю відновлення сесії	усі	ISO/IEC 25010
FR-10	Інтеграція зі сервісом	REST-доступ до моделі провайдера для покращення тексту листа	UC-05	документація провайдера

	штучного інтелекту			
FR-11	Асинхронна неблокуюча взаємодія з мережею	Жодна з протокольних операцій не виконується у потоці графічного інтерфейсу	усі	розділ 1.2, шаблон Reactor
FR-12	Одночасна підтримка кількох облікових записів	Незалежне зберігання секретів окремо для кожного облікового запису	UC-01, UC-02, UC-04	розділ 1.4

Таблиця Б.3 - Нефункціональні вимоги до системи

ID	Категорія	Опис вимоги
NFR-01	Чутливість інтерфейсу	Графічний інтерфейс залишається чутливим до дій користувача протягом будь-якої мережевої операції; перемальовування не блокується тривалими IMAP-, SMTP- чи HTTPS-запитами
NFR-02	Асинхронність введення-виведення	Мережеве ядро обслуговує одночасні з'єднання у єдиному циклі подій без виділення окремого потоку на з'єднання, що забезпечує масштабування при кількох активних облікових записах і фоновому циклі IDLE
NFR-03	Переносність	Застосунок збирається і функціонує на Windows, Linux та macOS з єдиної кодової бази; платформо-специфічна логіка ізольована у мінімальній кількості модулів із чіткою CMake- або препроцесорною межею

NFR-04	Безпека транспорту	Усі мережеві з'єднання виконуються через TLS 1.3 із допустимим відкатом до TLS 1.2; обов'язковими є перевірка серверного сертифіката і явна верифікація імені хоста
NFR-05	Безпека секретів	Токени, паролі і ключі шифрування локального кешу зберігаються виключно у системному сховищі облікових даних цільової ОС; вихідний код і конфігураційні файли не містять вшитих секретів
NFR-06	Стійкість	Застосунок коректно обробляє мережеві, протокольні та авторизаційні збої без аварійного завершення; усі ресурси (сокети, TLS-сесії, таймери) звільняються незалежно від шляху виходу з функції за ідіомою RAII
NFR-07	Супроводжу-ваність архітектури	Архітектура є шаровою з односпрямованим графом залежностей між модулями; нові реалізації стратегічних компонентів (сховище секретів, ШІ-провайдер) додаються через інтерфейси без змін у клієнтському коді
NFR-08	Якість коду	Кодова база має узгоджений стиль коду і регулярно перевіряється статичними аналізаторами; правила оформлення зафіксовано в репозиторії і застосовуються однаково до всіх дерев проєкту
NFR-09	Локалізація відмов	Збій або відсутність допоміжних модулів (зокрема стороннього ШІ-сервісу) не впливає на роботу основних поштових сценаріїв;

		недоступність одного облікового запису не блокує інших
NFR-10	Зручність використання та локалізація	Видимі рядки інтерфейсу проходять через стандартний механізм перекладу Qt і надаються щонайменше двома мовами; повідомлення про помилки сформульовано у природному стилі з можливістю розгорнути технічні деталі
NFR-11	Економічність споживання ресурсів	Резидентна пам'ять і навантаження на процесор у фазі стійкого стану утримуються на рівні, помітно нижчому, ніж у рішень, що вбудовують у дистрибутив повний браузерний рушій (Chromium / Electron)
NFR-12	Тестованість	Доменні сервіси доступні для модульного тестування у відриві від графічного інтерфейсу і зовнішніх служб; стратегічні залежності замінюються тестовими реалізаціями через інтерфейси без зміни клієнтського коду

ДОДАТОК В

Послідовність надсилання повідомлення за SMTP

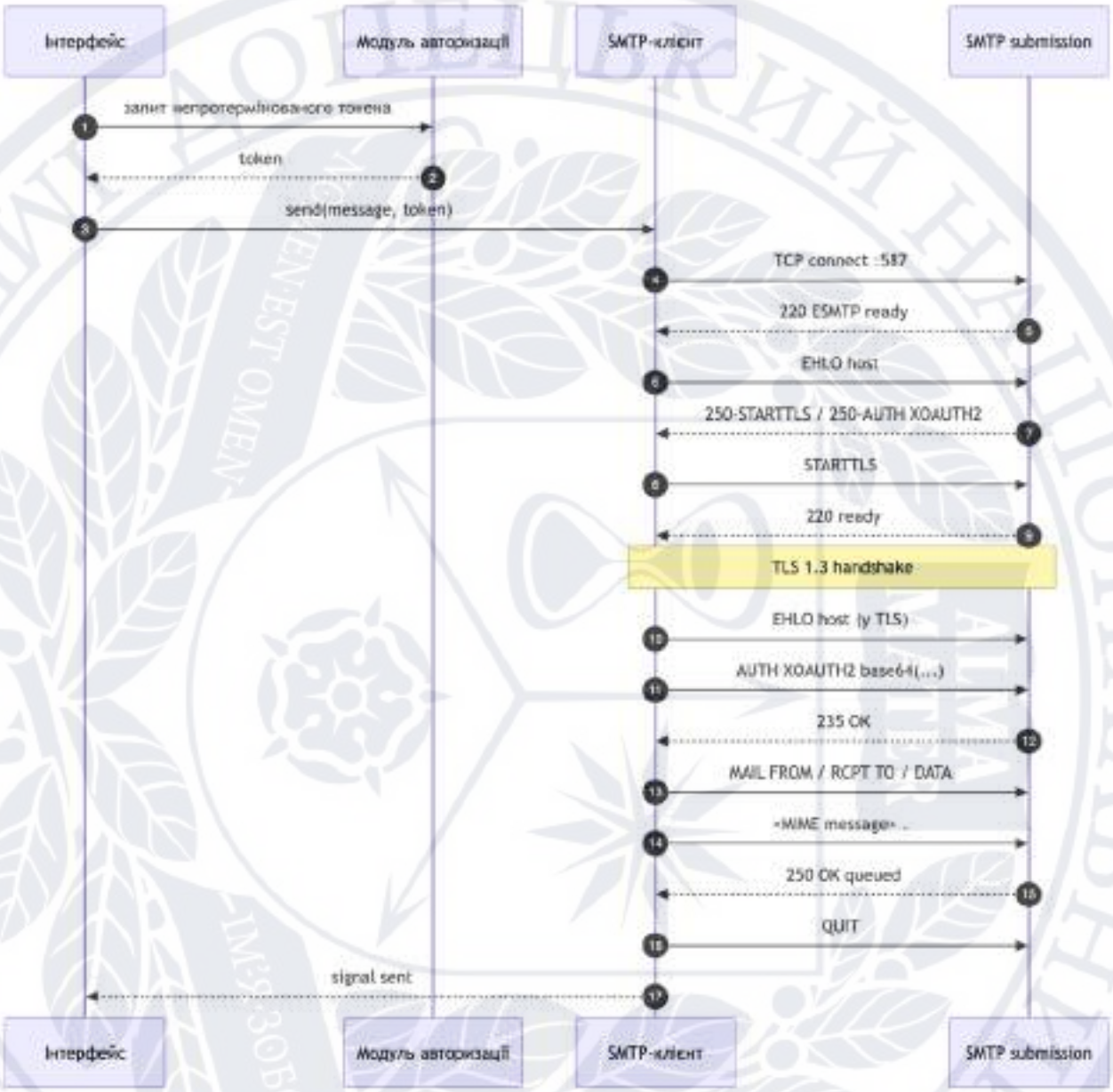

Рисунок В.1 - UML-діаграма послідовності надсилання повідомлення через SMTP

ДОДАТОК Ґ

UML-діаграми механізму OAuth 2.0

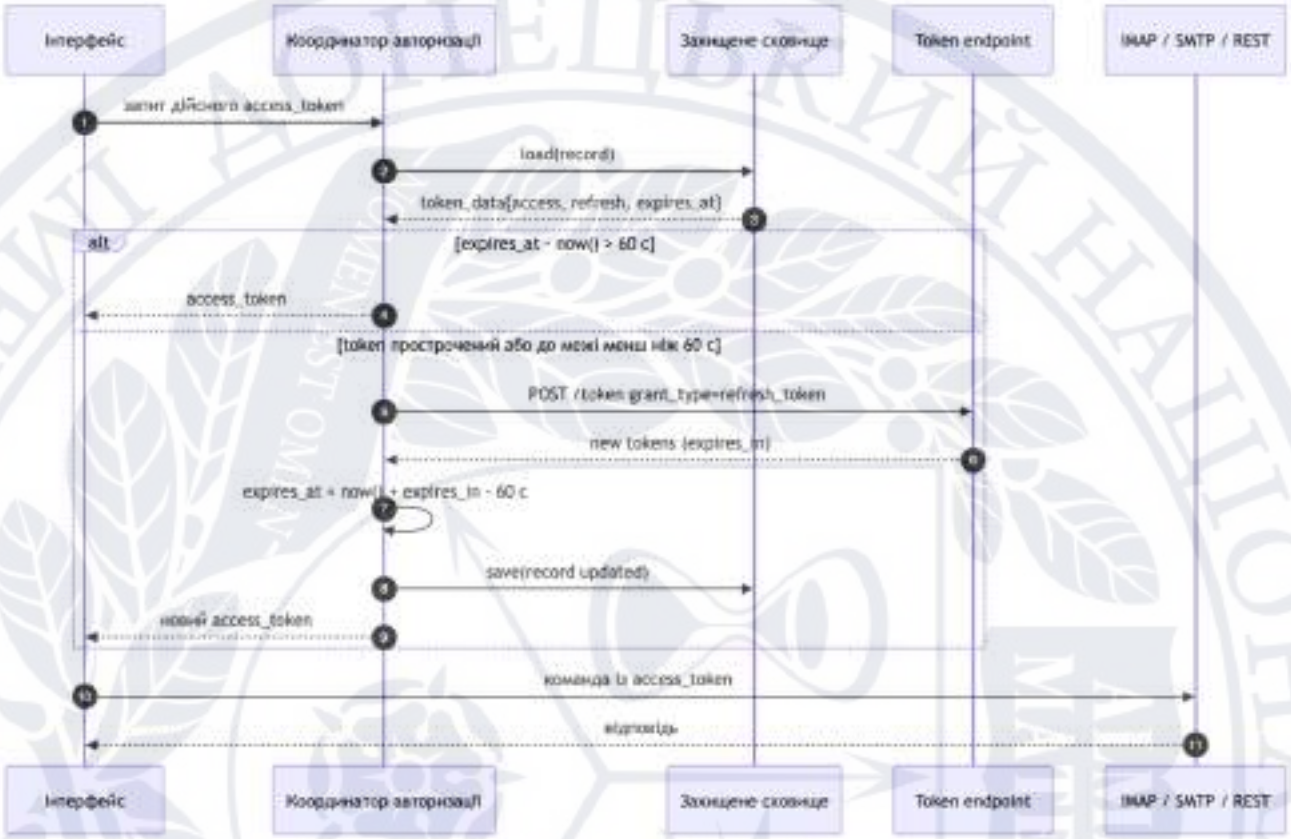

Рисунок Ґ.1 - UML-діаграма послідовності оновлення токена з 60-секундним буфером безпеки на token endpoint

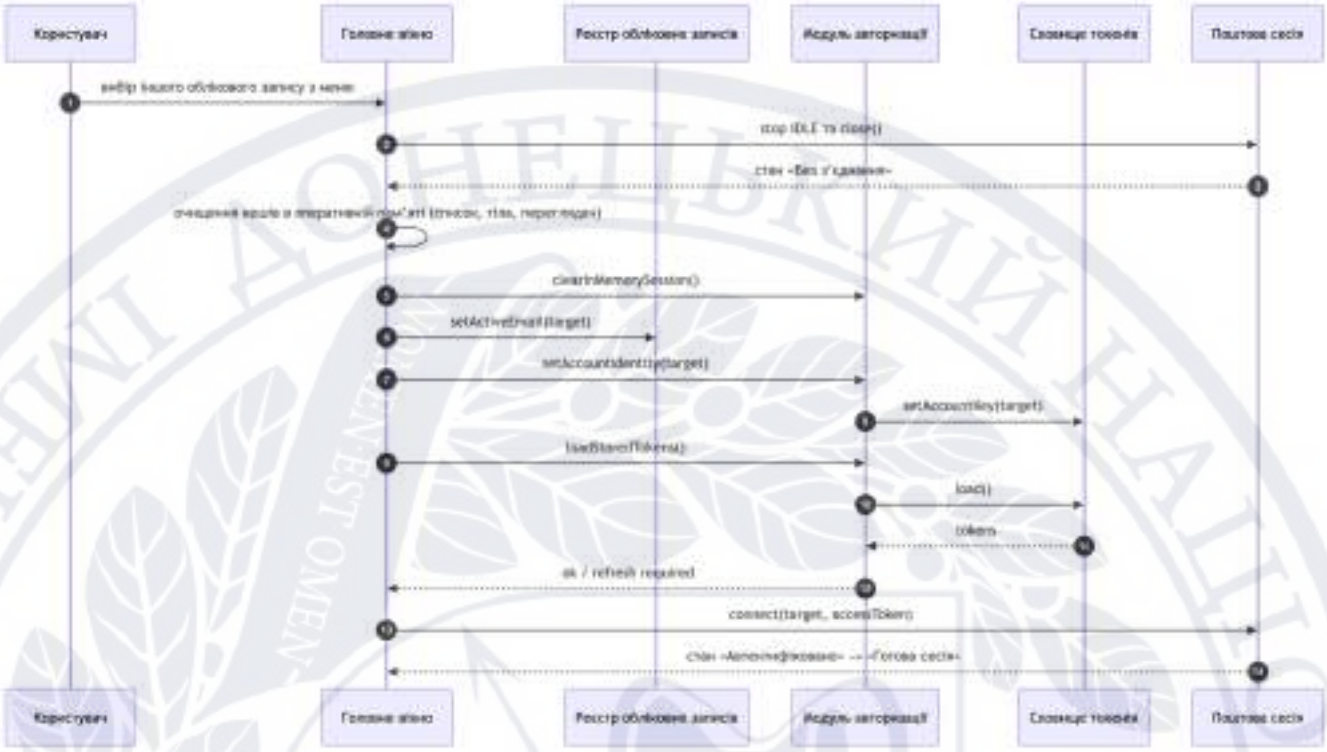

Рисунок Ґ.2 - UML-діаграма послідовності перемикання активного облікового запису

ДОДАТОК Д

Знімки застосунку

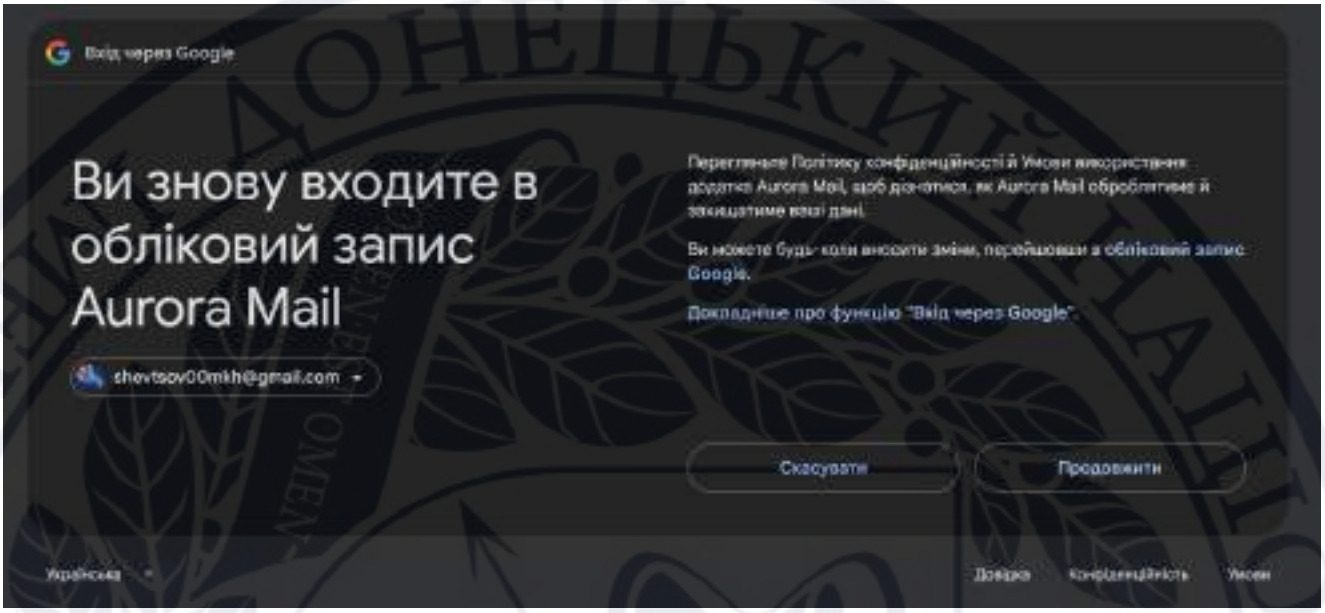

Рисунок Д.1 - Вступний екран згоди Google OAuth після перенаправлення з застосунку (Windows 11; UC-01)

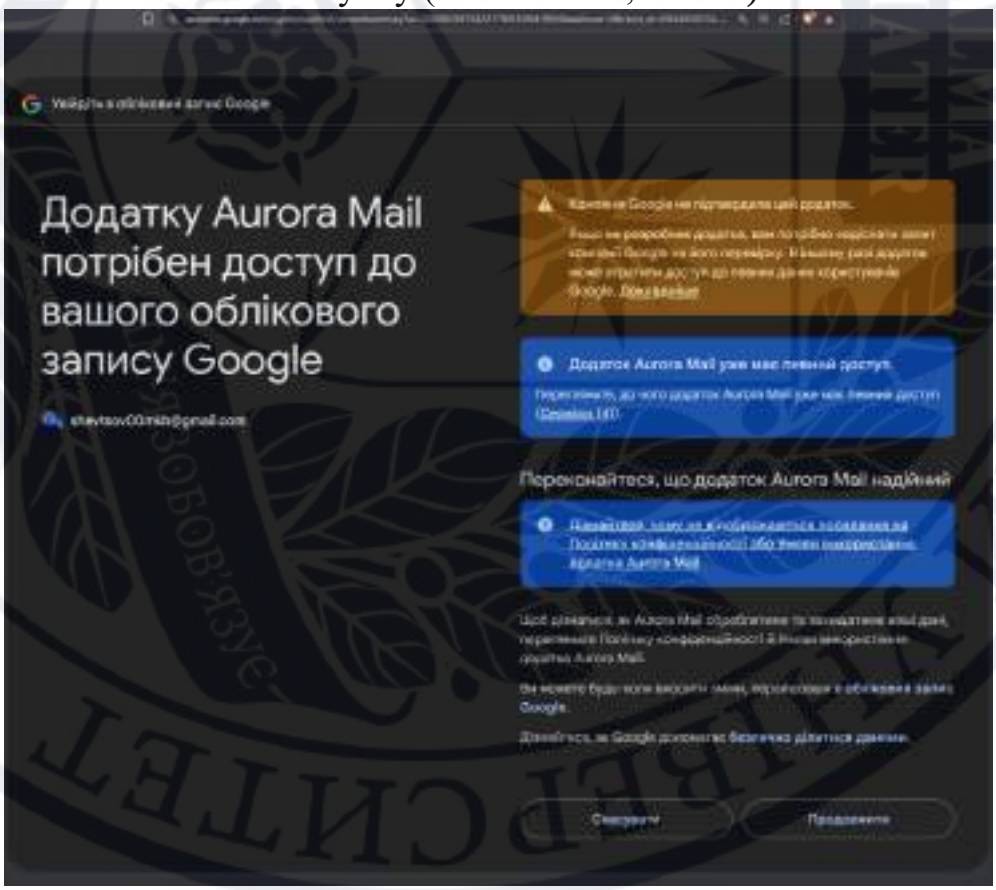

Рисунок Д.2 - Вступний екран згоди Google OAuth після перенаправлення з застосунку (Windows 11; UC-01)

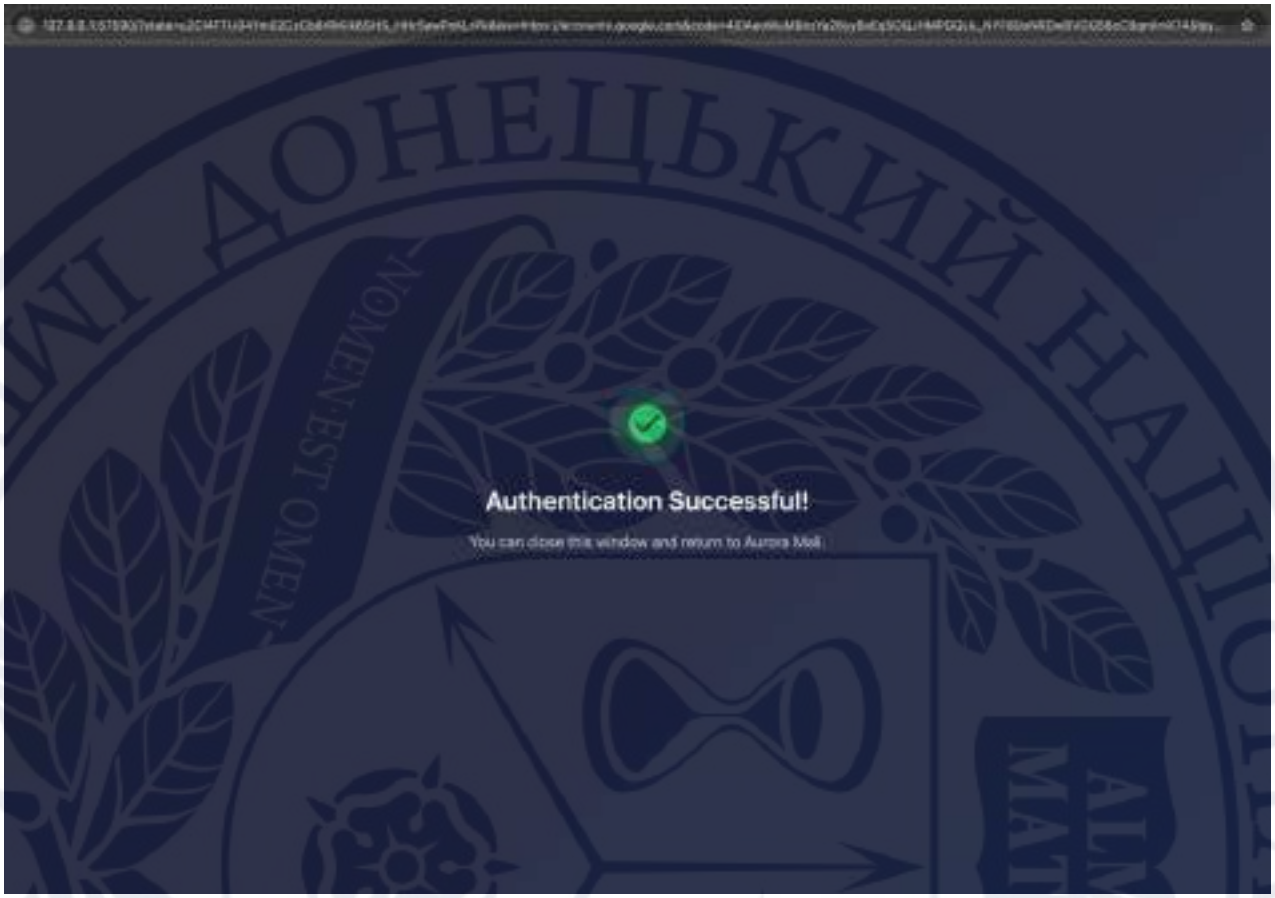

Рисунок *Д*.3 - HTML-сторінка успішної автентифікації, рендерована локальним приймачем (macOS 15; UC-01, розділ 3.6)

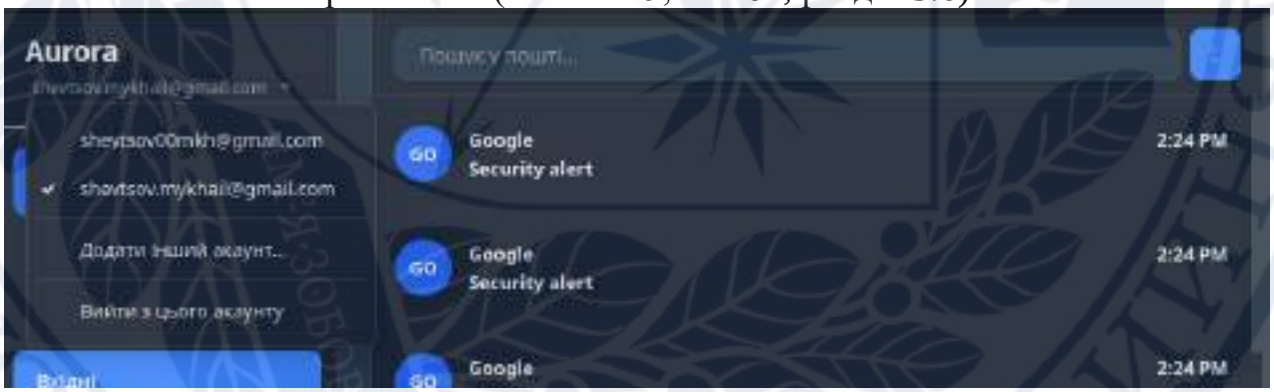

Рисунок Д.4 - Список облікових записів із кількома підключеними акаунтами (Ubuntu 24.04; FR-12)

Рисунок Д.5 - Головне вікно з трьома панелями і відкритим листом у безпечному переглядачі HTML (macOS 15; UC-02, FR-01, FR-10, FR-11)

Рисунок Д.6 - Діалог редактора нового листа з полями отримувача, теми і тіла (macOS 15; UC-03, підготовча фаза)

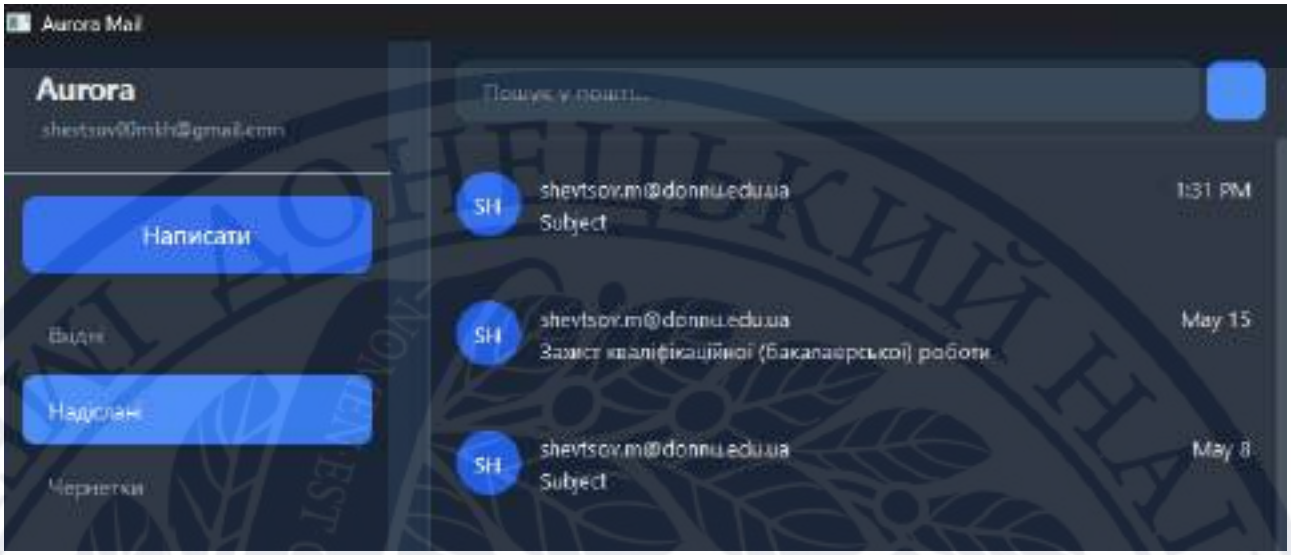

Рисунок Д.7 - Бік клієнта: лист з'являється в колонці «Надіслані» одразу після завершення SMTP-сесії (Windows 11; UC-03, FR-03)

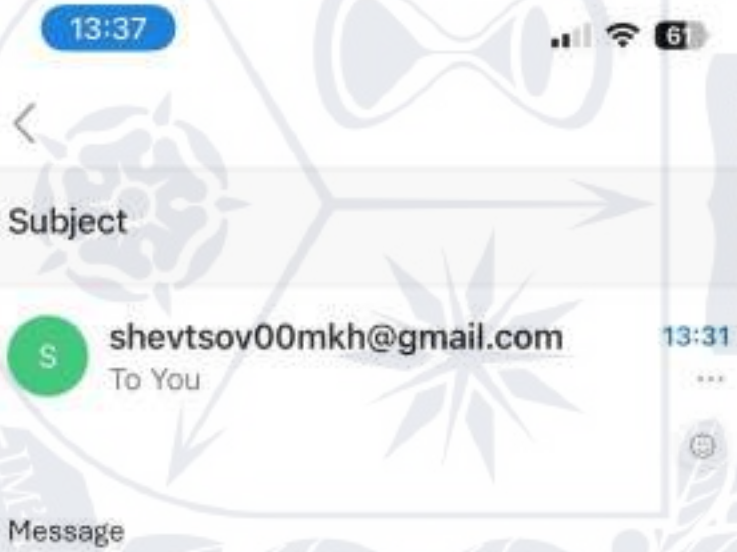

Рисунок Д.8 - Бік отримувача: той самий лист у вхідних повідомленнях стороннього клієнта, що підтверджує наскрізну доставку (UC-03, FR-03)

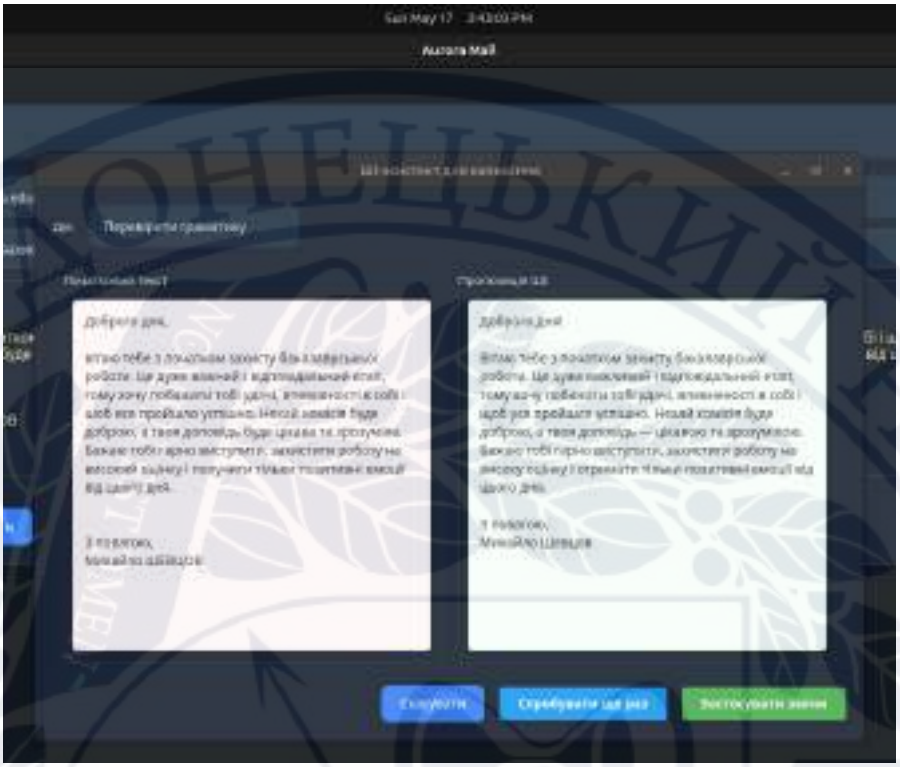

Рисунок Д.9 - Модальне вікно ШІ-асистента з результатом від Gemini у режимі перевірки граматики (Ubuntu 24.04; UC-05, FR-08, NFR-09)

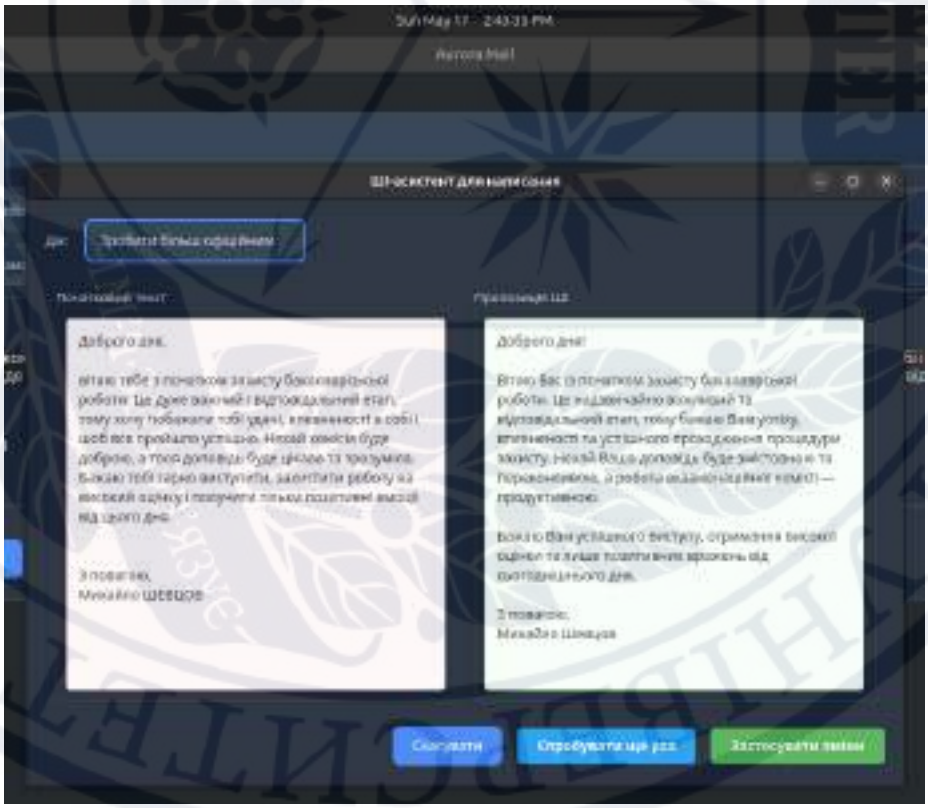

Рисунок Д.10 - Модальне вікно ШІ-асистента з результатом від Gemini у режимі перетворення тону на формальніший (Ubuntu 24.04; UC-05, FR-08, NFR-09)

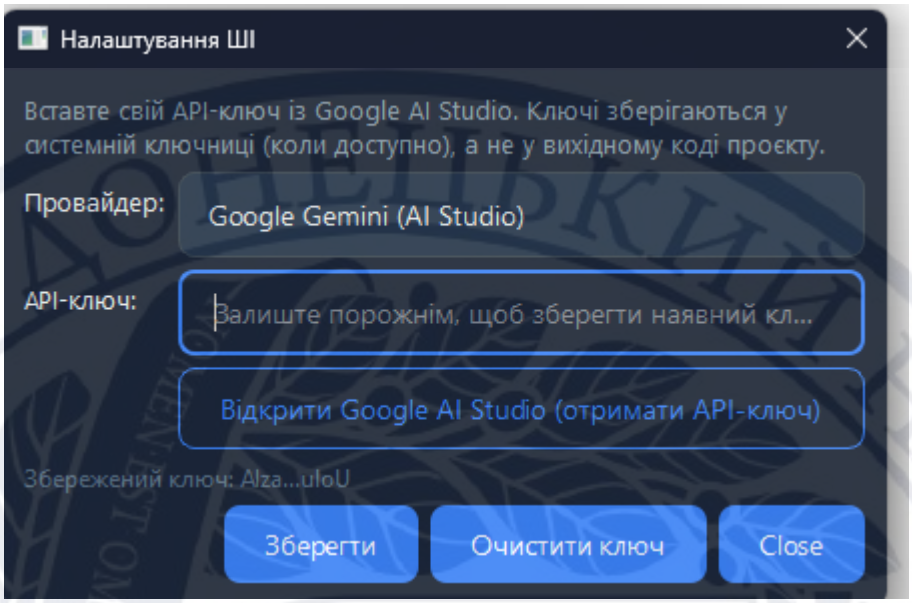

Рисунок Д.11 - Діалог налаштувань ШІ-провайдера зі зберіганням API-ключа у системному сховищі облікових даних (Windows 11; UC-05)

ДОДАТОК Ж

Трасування виконання вимог

Таблиця Ж.1 - Трасування функціональних вимог

ID	Реалізаційний артефакт	Спосіб перевірки
FR-01	Модуль Core/Mail/Imap (RFC 3501-сесія, підрозділ 2.4); черга команд із RAII-токеном IDLE	UC-02, UC-03 (ручні); інтеграційний Python-сценарій LOGIN/SELECT/FETCH/IDLE поверх діагностичної CLI
FR-02	Модуль Core/Auth (OAuthManager + Internal/PkceSession + Internal/OAuthRedirectServer, підрозділи 2.5, 3.6)	UC-01 (ручний); юніт-тести GoogleTest на генерацію code_verifier/code_challenge і валідацію state
FR-03	Модуль Core/Mail/Smtp із SASL XOAUTH2 (підрозділ 2.4)	UC-04 (ручний); інтеграційний Python-сценарій EHLO/AUTH/MAIL FROM/RCPT TO/DATA поверх CLI
FR-04	Обгортка GMime у Core/Mail/Mime; базова фільтрація HTML на боці UI (підрозділ 3.5)	UC-03 (ручний, перевірка вкладень і inline-зображень); юніт-тести GoogleTest на multipart/alternative, multipart/related, multipart/mixed. Часткова реалізація - див. підрозділ 3.13
FR-05	Будівельник вихідного MIME у Core/Mail/Mime	UC-04 (ручний); юніт-тести GoogleTest на multipart/mixed із вкладеннями
FR-06	Прив'язка Core/Mail/Imap ↔ UI-моделей повідомлень із	UC-02 (ручний); знімок E.4 (Додаток Е)

	підтримкою \Seen, \Answered, \Flagged	
FR-07	Модуль Core/Mail/Cache: MemoryMessageCache (LRU) + PersistentMessageCache (SQLite + AES-256-GCM, розділи 2.7, 3.7)	Юніт-тести GoogleTest на витіснення LRU, виведення ключа кешу, цикл nonce/tag; інструментальний моніторинг (нульова дисперсія резидентної пам'яті у фазі стійкого стану)
FR-08	Інфраструктура структурованого журналювання з редакцією токенів LOGIN/AUTH/XOAUTH2	Перегляд журналів після UC-01 - UC-05; модульні тести GoogleTest на редакційний фільтр
FR-09	Уніфікована стратегія std::expected<T, ProtocolError> у Core/Mail/Imap, Core/Mail/Smtp, Core/Auth, Core/Ai (підрозділ 2.8)	Юніт-тести GoogleTest на сценарії BAD, NO, тайм-аутів і HTTP-статусів 400/401/403/429
FR-10	Модуль Core/Ai (AiClient + адаптер Gemini, підрозділи 2.6, 3.8); UI-діалог AIAssistantDialog	UC-05 (ручний); інтеграційний Python-сценарій до Gemini-API
FR-11	io_context мережевого ядра у фоновому потоці зі strand-серіалізацією; чергована доставка сигналів Qt (підрозділ 2.3)	Інструментальний моніторинг в розділі 3.10.3 (відсутність блокувань UI у фазі активного FETCH); прогін під ThreadSanitizer на UC-02, UC-04

FR-12	Core/Auth/AccountRegistry + параметризований префікс oauth_tokens:<email > у захищеному сховищі (підрозділ 2.5.2)	UC-01 з двома обліковими записами Gmail (ручний); знімок E.3

Таблиця Ж.2 - Трасування нефункціональних вимог

ID	Реалізаційний артефакт	Спосіб перевірки
NFR-01	Розділення циклів подій UI / мережевого ядра, чергована доставка сигналів Qt (підрозділ 2.3)	Інструментальний моніторинг в розділі 3.10.3 (стабільні 60 fps у профілі активного гортання)
NFR-02	Boost.Asio io_context як єдиний цикл для IMAP/SMTP/HTTPS-з'єднань без поток-на-з'єднання (підрозділ 2.3)	Інструментальний моніторинг в розділі 3.10.3 (≈ 0 % CPU у фазі стійкого IDLE-стану на 360-секундному вікні)
NFR-03	Кросплатформний CMake-граф; ізоляція платформенної логіки у підкаталогах Platform/{Mac,Win,Linux} із препроцесорними Q_OS_*-межами (підрозділи 2.2, 3.4)	Емпіричний прогін збірки і UC-сценаріїв на macOS Apple Silicon, Windows 11 і Ubuntu 24.04 LTS (підрозділи 3.10.3, 3.11)
NFR-04	OpenSSL 3.x із системним сховищем довіри і верифікацією імені хоста для всіх IMAP/SMTP/HTTPS-з'єднань (підрозділ 2.4)	Юніт-тести GoogleTest на відмову при підставленому сертифікаті; перегляд коду TLS-контексту

NFR-05	QtKeychain (macOS Keychain, Windows Credential Manager, Linux Secret Service) для токенів і паролів; AES-256-GCM-ключ кешу зберігається у ключниці (підрозділи 2.5.2, 2.7)	Перегляд коду; статичний аналіз clang-tidy/clazy на правила запобігання вшитим секретам
NFR-06	RAII-обгортки для сокетів, TLS-сесій і токена IDLE; уніфіковане value-based-обробляння помилок (підрозділи 2.4, 2.8)	Юніт-тести GoogleTest на коректне звільнення ресурсів при ранньому виході; перегляд коду виходів за всіма гілками
NFR-07	Шарувата архітектура з односпрямованим CMake-графом; інверсія залежностей через концепти C++20 (підрозділ 2.2)	Перевірка ациклічності CMake-цілей; перегляд коду на відсутність зворотних включень UI → Core
NFR-08	clang-format (репозиторійні правила), clang-tidy, clazy як обов'язкові статичні аналізатори (підрозділи 3.2, 3.8)	CI-пресет CMake запускає форматування і аналізатори перед збіркою; нульовий діагностичний бюджет на критичних правилах
NFR-09	Архітектурна ізоляція ШІ-модуля (відсутність зворотних залежностей з поштового й авторизаційного модулів, підрозділ 2.6); незалежний	UC-05 з вимкненим ШІ-сервісом (поштові сценарії продовжують працювати); UC-01 при недоступності одного облікового запису з двох (ручні)

	OAuthManager на обліковий запис (підрозділ 2.5.2)	
NFR-10	Qt LinguistTools з українським і англійським каталогами перекладів; завантаження активної мови у точці входу (підрозділ 3.11)	Перегляд знімків інтерфейсу на трьох ОС у Додатку Е; прохід UC-01 - UC-05 з активним українським каталогом (ручний)
NFR-11	Дворівневий шифрований кеш з обмеженнями LRU обох рівнів замість вбудованого браузерного рушія (підрозділи 2.7, 3.7)	Інструментальний моніторинг в розділі 3.10.3 (плато 215 МБ резидентної пам'яті на macOS у неінтерактивному прогоні)
NFR-12	Інверсія залежностей через концепти C++20: бібліотечні таргети Core/Mail/*, Core/Auth, Core/Ai зібрані без UI-залежностей (підрозділ 2.2)	Юніт-тести GoogleTest для кожного Core/*-таргета у відриві від графічного інтерфейсу; CI-пресет збирає тестову ціль ізольовано