БАКАЛАВРСЬКА РОБОТА

БР. ІП - 26.00.00.000 ПЗ

Група ІП-21-2

Коциловська Галина-Анна

2025

Івано-Франківський національний технічний університет нафти і газу

Факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Коциловська Галина-Анна Павлівна

(прізвище, ім'я, по батькові)

УДК 004.4

(індекс)

БАКАЛАВРСЬКА РОБОТА

Автоматизація процесів обробки даних з електронних листів клієнтів

(назва роботи)

Інженерія програмного забезпечення

(назва освітньої програми)

121 - Інженерія програмного забезпечення

(шифр і назва спеціальності)

Робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

Здобувач освітнього рівня Коциловська Г.-А.П.

(підпис, ініціали та прізвище здобувача)

Науковий керівник Мельник Віталій Дмитрович, к.т.н., доцент

(підпис, прізвище, ім'я, по батькові, науковий ступінь, вчене звання керівника)

Допущено до захисту
Завідувач кафедри

доц. Бандура В.В.

(посада) (підпис) (дата) (ініціали та прізвище)

Івано-Франківськ – 2025

Івано-Франківський національний технічний університет нафти і газу

Інститут, факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Освітньо-кваліфікаційний рівень бакалавр

Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Зав. кафедрою **ІПЗ**
доц. В.В. Бандура
“ ____ ” _____2025 р.

ЗАВДАННЯ

НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТОВІ

Коциловській Галині-Анні Павлівні

(прізвище, ім’я, по-батькові)

1. Тема проекту (роботи) “ Автоматизація процесів обробки даних з електронних листів клієнтів”

керівник проекту (роботи) Мельник В.Д., доцент

затверджені наказом закладу вищої освіти від “ 28 ” квітня 2025 р. № 264/7

2. Строк подання студентом проекту (роботи) 11 червня 2025 р.

3. Вихідні дані до проекту (роботи) Результати і матеріали отримані під час проходження переддипломної практики

4. Зміст розрахунково - пояснювальної записки (перелік питань, які потрібно розробити)

1. Дослідження та аналіз предметної області процесів видобування та обробки даних
2. Застосування методології ETL для автоматизованої обробки даних з електронної пошти
3. Алгоритмічна реалізація системи обробки даних з електронних листів клієнтів
4. Проектування діаграми діяльності
5. Програмна імплементація методів ETL для процесів обробки даних з електронних листів

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

1. Основні компоненти системи обробки даних з електронних листів клієнтів (рис. 1.1)
2. Алгоритм ETL процесу (рис. 1.2)
3. Архітектура системи для автоматизованої обробки даних з електронних листів (рис. 1.3)
4. Приклад завантаження файлів с SFTP-сервера в HDFS на основі Apache NiFi (рис. 1.4)
5. Вигляд Talend Data Integration (рис. 1.5)

6. Консультанти розділів проекту (роботи)

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання **28 квітня 2025 р.**

Керівник _____
(підпис)

Завдання прийняв до виконання_____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту	Примітка
1	Дослідження та аналіз предметної області процесів видобування та обробки даних	03.05.2025	виконано
2	Застосування методології ETL для автоматизованої обробки даних з електронної пошти	14.05.2025	виконано
3	Алгоритмічна реалізація системи обробки даних з електронних листів клієнтів	20.05.2025	виконано
4	Проектування діаграми діяльності	27.05.2025	виконано
5	Програмна імплементація методів ETL для процесів обробки даних з електронних листів	02.06.2025	виконано
6	Оформлення пояснювальної записки дипломної роботи завідувачем кафедри	11.06.2025	виконано

Студент – дипломник _____
(підпис)

Керівник роботи _____
(підпис)

АНОТАЦІЯ

Бакалаврська робота містить 77 сторінок, 36 рисунків, список використаних джерел із 36 найменуваннями, 1 додаток.

Метою даної роботи є розробка ефективної архітектури та алгоритмічного забезпечення системи, здатної автоматично вилучати, трансформувати та завантажувати дані з електронних листів клієнтів до корпоративної інформаційної системи.

Об'єкт дослідження - процеси автоматизованого вилучення, трансформації та завантаження даних з електронної пошти.

Предмет дослідження - методи та інструменти реалізації ETL-процесів для обробки даних з електронних листів клієнтів.

В першому розділі проведено всебічний аналіз предметної області обробки електронних листів, визначено сучасні технології та архітектури для реалізації ETL-процесів.

В другому розділі розроблено алгоритмічну модель системи автоматизованої обробки даних, реалізовано UML-діаграми, які деталізують логіку роботи компонентів.

В третьому розділі здійснено програмну імплементацію розробленої системи, створено інтерфейси, реалізовано механізми трансформації, візуалізації та тестування даних.

Висновок: розроблено архітектуру автоматизованої системи, побудовано діаграми варіантів використання, діяльності та послідовностей, а також реалізовано функціональний прототип системи з використанням EWS API, ASP.NET Web API, AngularJS та Microsoft SQL Server.

КЛЮЧОВІ СЛОВА: АВТОМАТИЗАЦІЯ, ОБРОБКА ДАНИХ, ЕЛЕКТРОННА ПОШТА, ETL, ТРАНСФОРМАЦІЯ ДАНИХ, EWS API, ІНФОРМАЦІЙНІ СИСТЕМИ.

ANNOTATION

The bachelor's thesis contains 77 pages, 36 figures, a list of used sources with 36 names, 1 appendix.

The purpose of this work is to develop an effective architecture and algorithmic support for a system capable of automatically extracting, transforming and loading data from customer emails to a corporate information system.

The object of research is the processes of automated extraction, transformation and loading of data from email.

The subject of research is methods and tools for implementing ETL processes for processing data from customer emails.

The first section provides a comprehensive analysis of the subject area of email processing, identifies modern technologies and architectures for implementing ETL processes.

In the second section, an algorithmic model of an automated data processing system is developed, UML diagrams are implemented that detail the logic of the components.

In the third section, the software implementation of the developed system is carried out, interfaces are created, and data transformation, visualization and testing mechanisms are implemented.

Conclusion: the architecture of the automated system was developed, use case, activity, and sequence diagrams were constructed, and a functional prototype of the system was implemented using EWS API, ASP.NET Web API, AngularJS, and Microsoft SQL Server.

KEYWORDS: AUTOMATION, DATA PROCESSING, EMAIL, ETL, DATA TRANSFORMATION, EWS API, INFORMATION SYSTEMS.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

AKS – Сервіс Azure Kubernetes (Azure Kubernetes Service)

API – Програмний інтерфейс застосунків (Application Programming Interface)

FTP – Протокол передачі файлів (File Transfer Protocol)

NUnit – Бібліотека для модульного тестування (testing framework)

OCR – Оптичне розпізнавання символів (Optical Character Recognition)

PDF – Portable Document Format

TDD – Тест-орієнтоване розроблення (Test-Driven Development)

WAF – Брандмауер вебзастосунків (Web Application Firewall)

ВСТУП

У сучасних умовах цифрової трансформації бізнес-процесів ефективна обробка даних стає ключовим фактором конкурентоспроможності підприємств та організацій. Електронна пошта, як один із основних каналів комунікації між компаніями та їх клієнтами, щоденно генерує великі обсяги напівструктурованої інформації, яка часто містить важливі запити, замовлення, документи, звернення та інші форми взаємодії. Відповідно, виникає необхідність в автоматизованих рішеннях, що дозволяють ефективно обробляти, аналізувати й інтегрувати дані з електронних листів у внутрішні інформаційні системи підприємств.

Актуальність роботи

У сучасних умовах стрімкого зростання обсягів електронної комунікації, особливо через електронну пошту, підприємства стикаються з необхідністю ефективного видобування, обробки та інтеграції даних із вхідних повідомлень для подальшої аналітики та прийняття рішень. Ручна обробка таких даних є ресурсоємною, схильною до помилок і неефективною у масштабованих середовищах. Саме тому автоматизація процесів аналізу структурованої інформації в електронних листах із використанням сучасних методів ETL (Extract, Transform, Load) набуває особливої ваги. Створення систем, які дозволяють автоматизовано витягувати дані, трансформувати їх у придатний для подальшого аналізу формат і завантажувати до централізованих сховищ, є необхідною умовою для оптимізації бізнес-процесів і підвищення конкурентоспроможності підприємств.

Традиційні методи ручної обробки електронної пошти є повільними, трудомісткими та схильними до помилок. Крім того, вони не дозволяють оперативно реагувати на запити клієнтів і унеможливлюють масштабування процесів при зростанні навантаження. Впровадження автоматизованих засобів обробки даних забезпечує зниження витрат часу, підвищення

точності та надійності обробки, а також створення передумов для глибшої аналітики клієнтської інформації. Саме тому дослідження у сфері автоматизації процесів обробки електронної пошти є актуальним з наукової та практичної точок зору.

Метою даної роботи є розробка ефективної архітектури та алгоритмічного забезпечення системи, здатної автоматично вилучати, трансформувати та завантажувати дані з електронних листів клієнтів до корпоративної інформаційної системи.

Для реалізації цієї мети у роботі застосовано методологію ETL (Extract, Transform and Load), яка широко використовується у задачах інтеграції даних, а також низку сучасних технологій, серед яких Microsoft Exchange Web Services (EWS API), ASP.NET Web API, AngularJS та Microsoft SQL Server.

Завдання дослідження:

Проаналізувати предметну область обробки структурованих даних із електронних листів.

Вивчити існуючі методології та технології, пов’язані з процесами ETL.

Розробити архітектуру системи автоматизованої обробки листів.

Реалізувати функціональні компоненти системи на основі сучасних веб-технологій.

Розробити засоби трансформації та візуалізації даних.

Провести тестування роботи розробленої системи.

Об’єкт дослідження - процеси автоматизованого вилучення, трансформації та завантаження даних з електронної пошти.

Предмет дослідження - методи та інструменти реалізації ETL-процесів для обробки даних з електронних листів клієнтів.

Методи дослідження:

- аналітичний метод — для дослідження предметної області та технологій;

- системний аналіз — для розробки архітектури системи;

- моделювання — для побудови діаграм використання, діяльності, послідовностей;

- програмна реалізація — для створення інтерфейсів та функціональних модулів.

Наукова новизна

Удосконалено підхід до автоматизованої обробки даних з електронних листів шляхом інтеграції технології EWS API з методологією ETL, що дозволяє забезпечити безперервне отримання, трансформацію та збереження структурованої інформації в режимі реального часу.

Практичне застосування

Розроблена система може бути використана в організаціях для автоматичного оброблення вхідних листів клієнтів з метою вилучення даних для CRM-систем, аналітичних платформ, баз даних тощо.

Таким чином, результати даної роботи спрямовані на вдосконалення процесів цифрової взаємодії з клієнтами, оптимізацію роботи персоналу та підвищення ефективності функціонування інформаційних систем шляхом автоматизації обробки вхідних електронних повідомлень.

Бакалаврська робота містить 77 сторінок, 36 рисунків, 3 розділи список використаних джерел із 36 найменуваннями, 1 додаток.

РОЗДІЛ 1. ДОСЛІДЖЕННЯ ТА АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ПРОЦЕСІВ ВИДОБУВАННЯ ТА ОБРОБКИ ДАНИХ

1.1. Методологія автоматизованої обробки структурованих даних з електронних листів клієнтів

У сучасних бізнес-процесах значний обсяг критично важливих даних передається та зберігається у формі електронних повідомлень. Традиційно, введення та обробка цих даних, що стосуються, наприклад, платіжних документів (рахунків-фактур, рахунків до сплати), умов договорів, деталей замовлень на постачання, транзакційних записів у фінансових установах чи малих підприємствах, або персональних банківських виписок, здійснюється шляхом ручного введення та копіювання. Цей ручний підхід є неефективним, ресурсомістким, схильним до людських помилок та обмежує масштабованість операцій. Зокрема, в контексті взаємодії з клієнтами, електронні листи часто містять запити, замовлення, скарги, інформацію про оплату або інші суттєві дані, які потребують швидкої та точної обробки для забезпечення ефективного обслуговування та ведення бізнесу. Необхідність періодичного вилучення та внесення таких даних вручну створює значне операційне навантаження.

Основною метою даної розробки є створення або впровадження методології та інструментарію для автоматизації трудомістких та повторюваних завдань, пов'язаних з обробкою даних, що надходять через електронну пошту від клієнтів. Зокрема, фокус полягає на автоматизації процесів екстракції, валідації та структуризації реляційних даних, що містяться як у тілі електронних листів, так і у вкладених файлах (наприклад, PDF, електронні таблиці), для подальшого використання в інформаційних системах підприємства (CRM, ERP, бухгалтерське програмне забезпечення тощо). Автоматизація періодичної екстракції та верифікації критичних та

конфіденційних даних з електронних комунікацій спрямована на суттєве скорочення часових витрат, зниження операційних витрат та підвищення загальної продуктивності.

1.1.1. Архітектура та методологія автоматизованої обробки

Реалізація системи автоматизованої обробки даних з електронних листів клієнтів передбачає побудову модульної архітектури, яка включає наступні ключові компоненти та етапи (рис. 1.1):

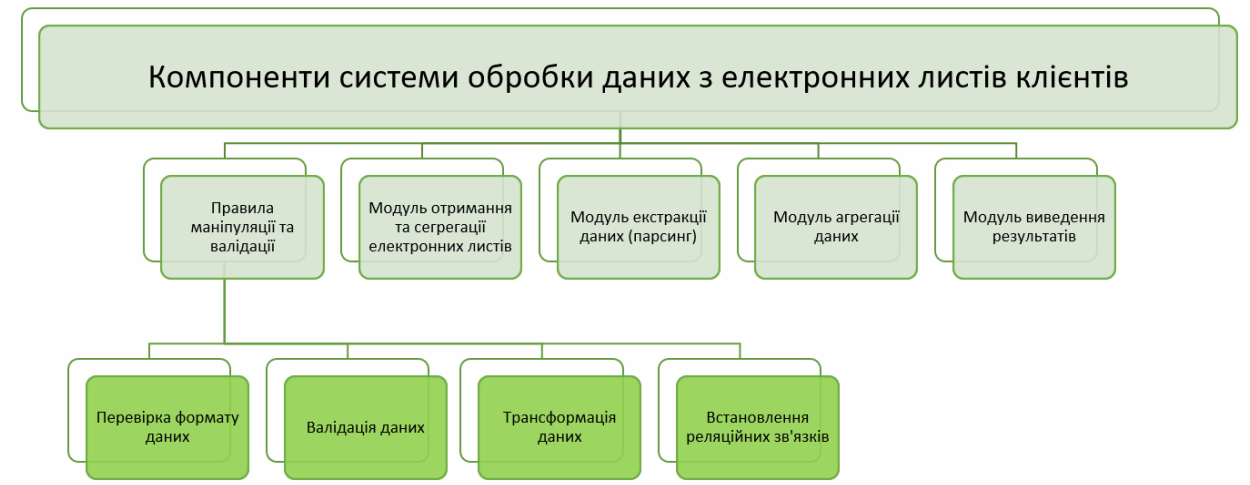


Рисунок 1.1 – Основні компоненти пропонованої системи обробки даних з електронних листів клієнтів

- Модуль отримання та сегрегації електронних листів.

Забезпечує безпечний доступ до визначених поштових скриньок та первинну фільтрацію вхідних повідомлень. Для ефективної сегрегації та управління потоками даних користувачі або адміністратори системи можуть створювати окремі профілі або "облікові записи" в системі автоматизації, кожен з яких асоціюється з конкретною поштовою адресою або функціональним призначенням (наприклад, "Обробка Замовлень", "Служба Підтримки").

- Модуль екстракції даних (парсинг).

Відповідає за вилучення необхідних даних з тіла повідомлення та вкладень. Цей модуль використовує попередньо визначені "шаблони" або правила парсингу, які специфікуються для кожного типу документів або формату даних, що очікуються від клієнтів (наприклад, шаблон для рахунків-фактур, шаблон для форм замовлення). На рівні кожного шаблону визначаються:

- Фільтри даних.

Набір критеріїв (наприклад, ключові слова, регулярні вирази, місцезнаходження тексту відносно інших елементів), що дозволяють ідентифікувати та вилучити конкретні поля даних (наприклад, номер замовлення, дата, ім'я клієнта, перелік товарів, сума).

- Правила маніпуляції та валідації.

Набір логічних умов та операцій, які застосовуються до вилучених даних. Ці правила можуть включати:

- Перевірку формату даних (наприклад, чи є поле числовим, чи відповідає формату дати).

- Валідацію даних проти зовнішніх джерел (наприклад, перевірка існування клієнта в базі CRM за електронною поштою або назвою).

- Трансформацію даних (наприклад, приведення адрес до стандартного формату, конвертація валют).

- Встановлення реляційних зв'язків між вилученими даними (наприклад, зв'язування позицій замовлення з загальною сумою).

- Умови спрацьовування правил.

Логічні вирази, які визначають, коли саме має бути застосоване певне правило (наприклад, "якщо сума замовлення перевищує 10000 грн, надіслати повідомлення менеджеру").

- Модуль агрегації даних. Здійснює консолідацію оброблених даних відповідно до заданих критеріїв. Це може бути агрегація замовлень за певний

період, групування запитів від одного клієнта, підрахунок загальної суми надходжень тощо.

- Модуль виведення результатів.

Забезпечує експорт агрегованих або індивідуальних наборів даних у різні формати (наприклад, XML, TXT, CSV, JSON) або їх безпосереднє завантаження у визначені віддалені бази даних чи інтеграцію з корпоративними інформаційними системами (CRM, ERP, складський облік) через API або інші конектори. Також може бути передбачена генерація спеціалізованих звітів на основі оброблених даних.

1.1.2. Результати застосування пропонованої розробки

Впровадження автоматизованої обробки даних з електронних листів клієнтів дозволить досягти значних переваг, зокрема:

1. Підвищення операційної ефективності: Скорочення часу, необхідного для обробки кожного електронного листа, що містить структуровані дані.

2. Зниження витрат: Мінімізація витрат, пов'язаних з ручним введенням та обробкою даних.

3. Зростання продуктивності: Збільшення обсягу оброблених даних за одиницю часу та вивільнення людських ресурсів для виконання більш складних та аналітичних завдань.

4. Підвищення точності даних: Зменшення кількості помилок, пов'язаних з ручним перенесенням інформації.

5. Покращення швидкості реагування: Прискорення обробки клієнтських запитів та замовлень, що позитивно впливає на якість обслуговування.

6. Систематизація та доступність даних: Забезпечення структурованого зберігання та легкого доступу до даних, отриманих з електронних комунікацій, для подальшого аналізу та прийняття рішень.

Таким чином, автоматизація процесів обробки даних з електронних листів клієнтів є критично важливим кроком для оптимізації бізнес-операцій, підвищення конкурентоспроможності та забезпечення ефективної взаємодії з клієнтською базою.

1.2. Метод Extract, Transform and Load (ETL) для обробки та інтеграції даних

Extract, Transform, Load (ETL) — це фундаментальний триетапний процес інтеграції даних, який використовується для переміщення інформації з однієї або декількох систем-джерел до централізованого сховища даних (наприклад, сховища даних, вітрини даних або іншої цільової бази даних). Цей метод є наріжним каменем у побудові систем бізнес-аналітики, звітності та прийняття рішень, забезпечуючи доступність чистих, узгоджених та структурованих даних для аналізу.

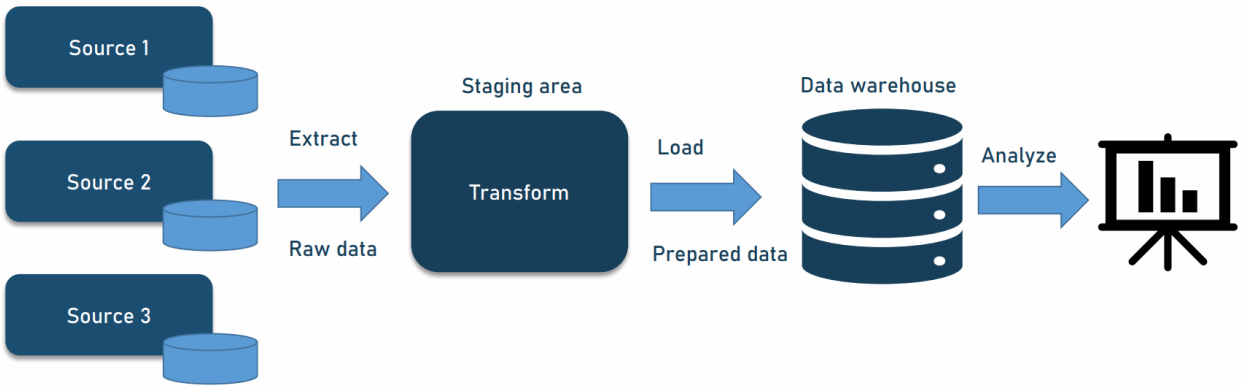


Рисунок 1.2 – Алгоритм ETL процесу

Розглянемо детальніше кожен етап:

1. Extract (Екстракція або Вилучення)

- На цьому етапі дані витягуються з різних систем-джерел. Це можуть бути реляційні бази даних (наприклад, SQL Server, Oracle, PostgreSQL), нереляційні бази даних (NoSQL), плоскі файли (CSV, XML,

JSON), різні додатки (CRM, ERP), API, файли електронних таблиць, дані з вебсайтів тощо.

- Процес екстракції має бути ефективним та надійним, щоб не навантажувати надмірно системи-джерела. Він може включати:

- Повну екстракцію (вилучення всіх даних).

- Інкрементальну екстракцію (вилучення лише даних, які змінилися з моменту останнього запуску процесу).

- Вилучені дані зазвичай розміщуються у тимчасовій області (staging area) перед початком етапу трансформації.

2. Transform (Трансформація або Перетворення)

- Цей етап є найважливішим і часто найбільш складним. Тут дані очищаються, стандартизуються та перетворюються, щоб відповідати вимогам цільової системи та потребам бізнесу.

- Типові операції трансформації включають:

- Очищення даних: Обробка відсутніх значень, корекція помилок, видалення дублікатів, стандартизація форматів (дат, адрес, номерів телефонів).

- Нормалізація або денормалізація: Перетворення структури даних відповідно до моделі цільового сховища (наприклад, перехід від нормалізованих транзакційних таблиць до денормалізованих таблиць для аналізу).

- Агрегація: Підсумовування даних до більш високого рівня деталізації (наприклад, об'єднання транзакцій за день, місяць або клієнтом).

- Об'єднання даних (Joining/Merging): Комбінування даних з різних джерел на основі спільних ключів або ідентифікаторів.

- Фільтрація: Вибір тільки тих даних, які є релевантними для цільової системи або аналізу.

- Застосування бізнес-правил: Обчислення нових значень, категоризація даних, реалізація складної бізнес-логіки.

- Перетворення типів даних: Зміна формату даних (наприклад, з тексту в число, з одного формату дати в інший).

- Мета трансформації — забезпечити високу якість даних, їх узгодженість та готовність до аналізу.

3. Load (Завантаження)

- На фінальному етапі трансформовані дані завантажуються в цільове сховище даних.

- Існують різні стратегії завантаження:

- Повне завантаження (Full Load): Видалення всіх існуючих даних у цільовій таблиці та завантаження нового набору даних. Використовується рідше для великих обсягів.

- Інкрементальне завантаження (Incremental Load): Додавання лише нових записів або оновлення існуючих, які змінилися з моменту останнього завантаження. Це найпоширеніший підхід для регулярних ETL-процесів.

- Обробка повільно змінних вимірів (Slowly Changing Dimensions, SCD): Спеціальні техніки для відстеження змін у вимірних даних (наприклад, як змінилася адреса клієнта або приналежність товару до категорії з часом), зберігаючи історію цих змін.

- Етап завантаження має бути оптимізований для продуктивності, оскільки він часто працює з великими обсягами даних.

ETL є ключовим компонентом в:

- Побудові корпоративних сховищ даних.

- Міграції даних між різними системами.

- Інтеграції даних з різних джерел для консолідованої звітності та аналізу.

- Підготовці даних для систем бізнес-аналітики (BI) та інструментів візуалізації.

- Процесах майнінгу даних та машинного навчання (як етап підготовки даних).

1.3. Застосування методології ETL для автоматизованої обробки даних з електронної пошти

Даний проєкт сфокусований на реалізації автоматизованої системи для обробки структурованих та реляційних даних, що містяться в електронній кореспонденції, зокрема, у вкладених файлах. Електронна пошта визнається як критично важливий канал комунікації як на індивідуальному, так і на корпоративному рівні, забезпечуючи ефективний та надійний обмін інформацією. Значна кількість операційних даних реляційної структури (наприклад, транзакційні записи, форми замовлень, дані рахунків) передається саме у форматі вкладень.

Розроблений інструментарій реалізує парадигму Extract, Transform, Load (ETL) і надає функціонал для:

- Екстракції (Extract): Система передбачає можливість конфігурації умов фільтрації вхідних електронних листів та критеріїв пошуку у вкладених файлах для ідентифікації релевантних даних. Вилучення даних здійснюється згідно з визначеними шаблонами структури вкладених файлів.

- Трансформації (Transform): Реалізація етапу трансформації включає:

Визначення мапування (відповідності) вилучених атрибутів (полів) з вкладених файлів до стовпців цільової таблиці бази даних.

Специфікацію правил валідації та очищення даних для кожного атрибута із зазначенням відповідних умов їх застосування.

Можливість розширення стандартного набору трансформацій шляхом інтеграції користувацьких функцій для виконання складних перетворень даних.

- Завантаження (Load): Трансформовані та валідовані дані завантажуються до визначеної цільової реляційної бази даних.

Після успішного завершення операції завантаження даних до цільового сховища, система забезпечує функціонал агрегації даних та представлення результатів у формі спеціалізованих звітів.

Ключовою перевагою даної системи є вимога лише одноразової параметризації шаблону обробки для кожного нового типу вхідних файлів або структури даних. Після початкової конфігурації процес обробки даних функціонує в повністю автоматичному режимі, що забезпечує значне підвищення ефективності та зниження операційних витрат у порівнянні з ручними методами обробки даних з електронної пошти.

1.4. Архітектура та функціональні компоненти системи автоматизованої обробки даних з електронної пошти

Функціонування даної системи базується на роботі фонового сервісу (background service), який відповідає за ініціацію та виконання ключових операцій. Цей сервіс здійснює періодичний моніторинг (опитування) визначених поштових скриньок з використанням стандартного інтерфейсу програмування застосунків, такого як Exchange Web Services (EWS API), з метою виявлення та ідентифікації нових вхідних електронних повідомлень, що підлягають обробці.

Для взаємодії з користувачем та конфігурації параметрів системи передбачено графічний інтерфейс користувача (ГІК/GUI). Основне призначення цього інтерфейсу – забезпечення можливості налаштування підключень до поштових облікових записів та визначення комплексних шаблонів обробки даних для різних типів вхідної інформації (наприклад, шаблонів екстракції та трансформації даних зі специфічних форматів вкладень).

Крім функціоналу конфігурації, ГІК забезпечує візуалізацію статусу виконання автоматизованих операцій екстракції та трансформації даних, надаючи користувачеві інформацію про поточний стан процесу. Також реалізовано механізм реєстрації та відображення системних помилок чи винятків, що виникають під час виконання завдань, що полегшує моніторинг та діагностику проблем.

Додатково, через ГІК користувачеві доступна функціональність генерації користувацьких наборів даних та аналітичних звітів на основі інформації, яка була успішно екстрагована, трансформована та завантажена до цільової бази даних. Це дозволяє використовувати оброблені дані безпосередньо для аналізу та прийняття рішень.

Таким чином, система поєднує фоновий автоматизований процес з інтерактивним графічним інтерфейсом для ефективного управління життєвим циклом обробки даних, що надходять через електронну пошту.

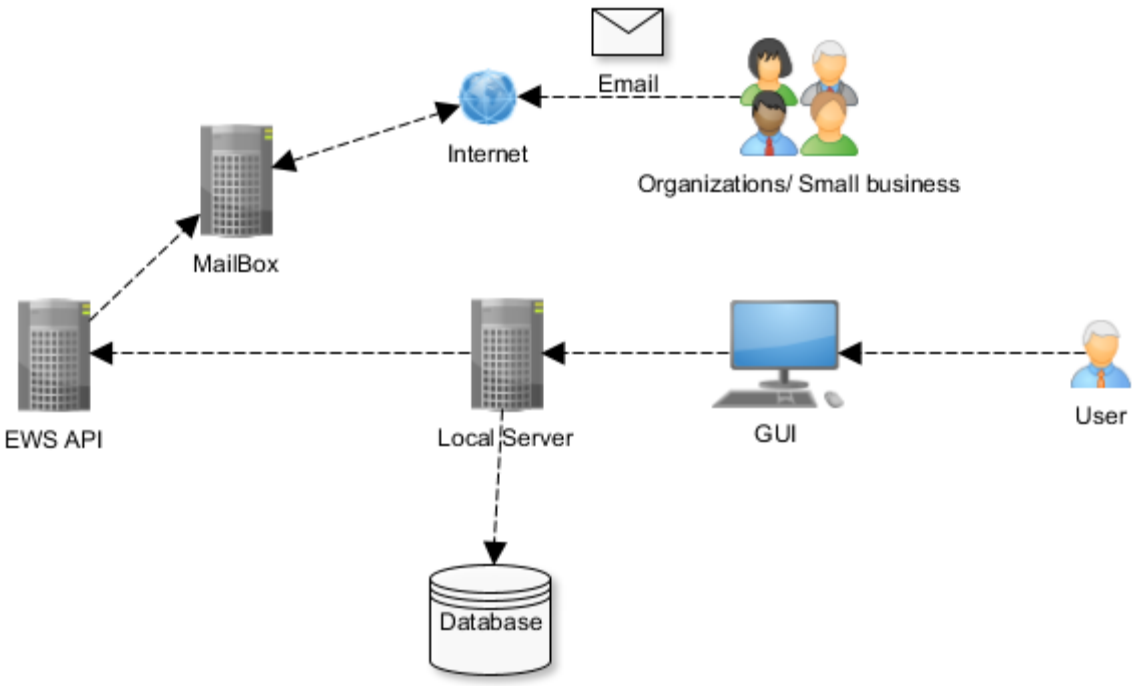


Рисунок 1.3 – Архітектура системи для автоматизованої обробки даних з електронних листів

На рисунку 1.3 наведено архітектуру системи для автоматизованої обробки даних з електронних листів клієнтів як розподілену багаторівневу архітектуру (multi-tier distributed architecture), що включає наступні ключові компоненти та їх взаємодію:

1. Джерело даних (Source Layer):

- Організації/Малі підприємства (Organizations/Small business): Представляють сутності, які генерують та надсилають електронні листи, що містять дані для обробки.

- Електронна пошта (Email) / Інтернет (Internet): Канал та середовище передачі даних від джерел.

- Поштова скринька (MailBox): Сервер електронної пошти, який приймає та зберігає вхідні повідомлення. Є первинним сховищем неопрацьованих даних.

2. Рівень доступу до даних (Data Access Layer):

- EWS API (Exchange Web Services API): Програмний інтерфейс, що забезпечує доступ до даних у Поштовій скриньці. Цей компонент відповідає за реалізацію етапу Екстракції (Extract), дозволяючи системі програмно отримувати доступ до електронних листів та їх вмісту (включаючи вкладення). Діаграма показує прямий зв'язок між Поштовою скринькою та компонентом EWS API, а також зв'язок EWS API з Локальним сервером, що підтверджує його роль як шлюзу для отримання даних.

3. Рівень обробки та зберігання (Processing and Storage Layer):

- Локальний сервер (Local Server): Центральний компонент архітектури, що виконує основні функції обробки. На цьому сервері, ймовірно, розгортається фоновий сервіс, який здійснює моніторинг поштової скриньки через EWS API. Локальний сервер відповідає за етапи Трансформації (Transform) отриманих даних (застосування правил очищення, валідації, мапування) та ініціацію етапу Завантаження (Load). Діаграма показує, що Локальний сервер отримує дані від EWS API.

- База даних (Database): Цільове сховище даних. Підключена до Локального сервера, що вказує на те, що Локальний сервер здійснює операції запису (завантаження) трансформованих даних у цю базу даних. База даних зберігає оброблену інформацію, яка готова для подальшого використання (звітність, аналіз).

4. Рівень представлення (Presentation Layer):

- GUI (Graphical User Interface): Графічний інтерфейс користувача, який надає можливість взаємодії оператору з системою. Він підключений до Локального сервера, що дозволяє користувачеві через інтерфейс виконувати конфігурацію (налаштування облікових записів, шаблонів), моніторинг статусу виконання завдань (активності та помилок) та ініціювати генерацію звітів на основі даних, що зберігаються у Базі даних.

- Користувач (User): Кінцевий оператор системи, який використовує GUI для управління та взаємодії з процесом обробки даних.

Таким чином, представлена архітектура відображає потік даних від зовнішніх джерел (клієнтів через електронну пошту) через рівень доступу (EWS API), централізований рівень обробки та зберігання (Локальний сервер та База даних), до рівня взаємодії з користувачем (GUI), забезпечуючи автоматизований ETL-процес для даних з електронної пошти.

1.5. Огляд популярних сучасних платформ для обробки та видобування даних

Проведемо огляд популярних сучасних платформ для обробки та видобування даних із використанням методології ETL (Extract, Transform, Load). Ці платформи широко застосовуються в аналітичних системах, бізнес-інтелекті (BI) та обробці великих обсягів даних (Big Data).

1. Apache NiFi

Apache NiFi — це інструмент для автоматизації потоків даних, розроблений для переміщення, відстеження та трансформації даних. Він має зручний графічний інтерфейс, який дозволяє налаштовувати складні ETL-процеси без потреби в програмуванні.

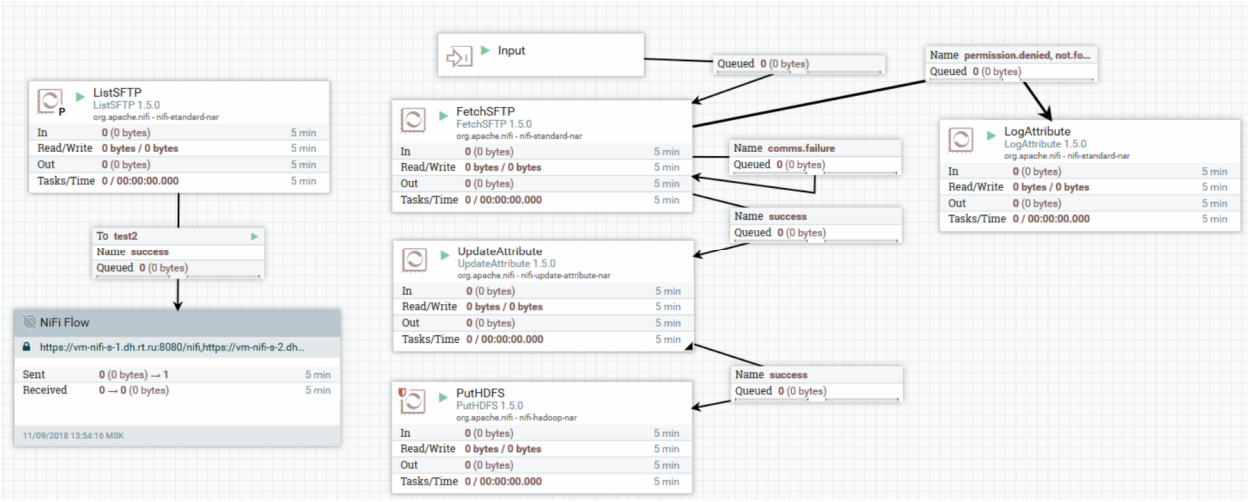


Рисунок 1.4 – Приклад завантаження файлів с SFTP-сервера в HDFS на основі Apache NiFi

Особливості:

- Підтримка потоків у реальному часі.
- Просте підключення до різних джерел (бази даних, FTP, API, email).
- Можливість маршрутизації даних та зберігання історії змін.
- Вбудована безпека та аудит.

2. Talend Data Integration

Talend — потужна open-source платформа для інтеграції даних, що надає повноцінну ETL-функціональність. Має версії як для локального, так і хмарного використання.

Особливості:

- Великий набір готових конекторів до систем (email, БД, REST, SOAP, Hadoop).

- Гнучкий дизайнер потоків даних.

- Підтримка перетворення, очищення та об'єднання даних.

- Можливість створення розкладу та автоматизації задач.

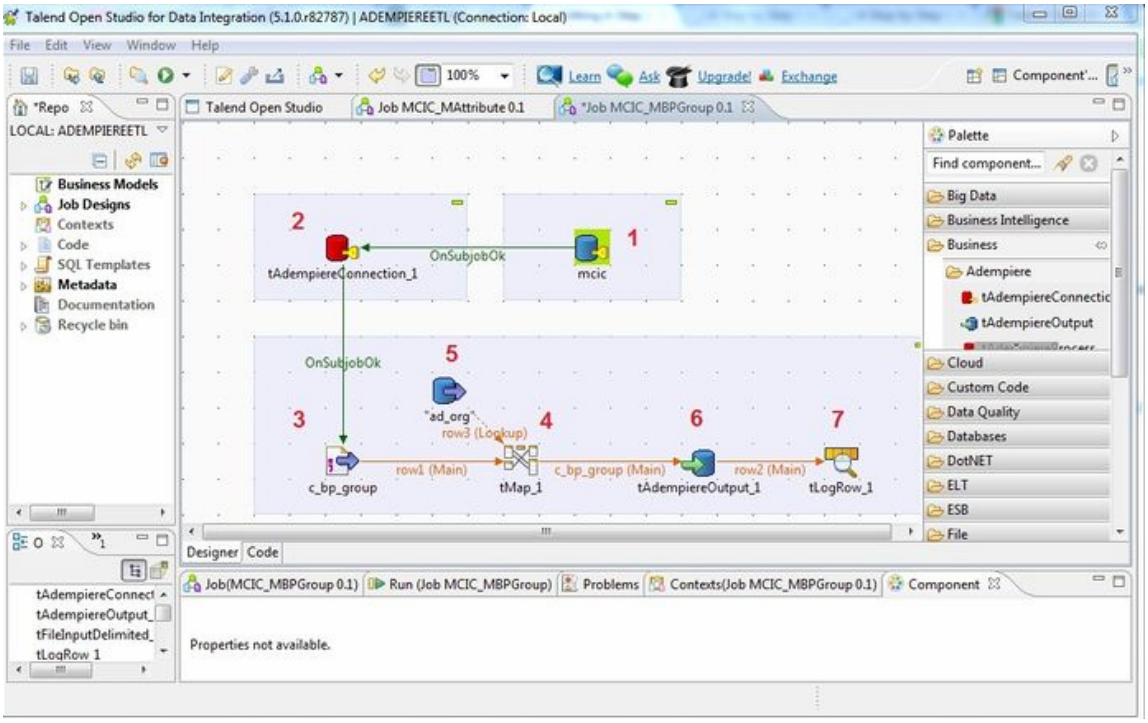


Рисунок 1.5 – Вигляд Talend Data Integration

3. Microsoft SQL Server Integration Services (SSIS)

SSIS — це компонент SQL Server, який надає засоби для імпорту, експорту, трансформації та завантаження даних у середовищі Microsoft.

Особливості:

- Щільна інтеграція з SQL Server.

- Зручний дизайнер пакетів (в середовищі Visual Studio).

- Підтримка email, FTP, Excel, CSV, XML тощо.

- Можливість створення сценаріїв на C# або VB.NET.

4. Apache Airflow

Apache Airflow — це платформа для оркестрації ETL-процесів на основі DAG (Directed Acyclic Graph). Вона не містить вбудованих

трансформацій, але дозволяє організувати складні пайплайни ETL за допомогою Python.

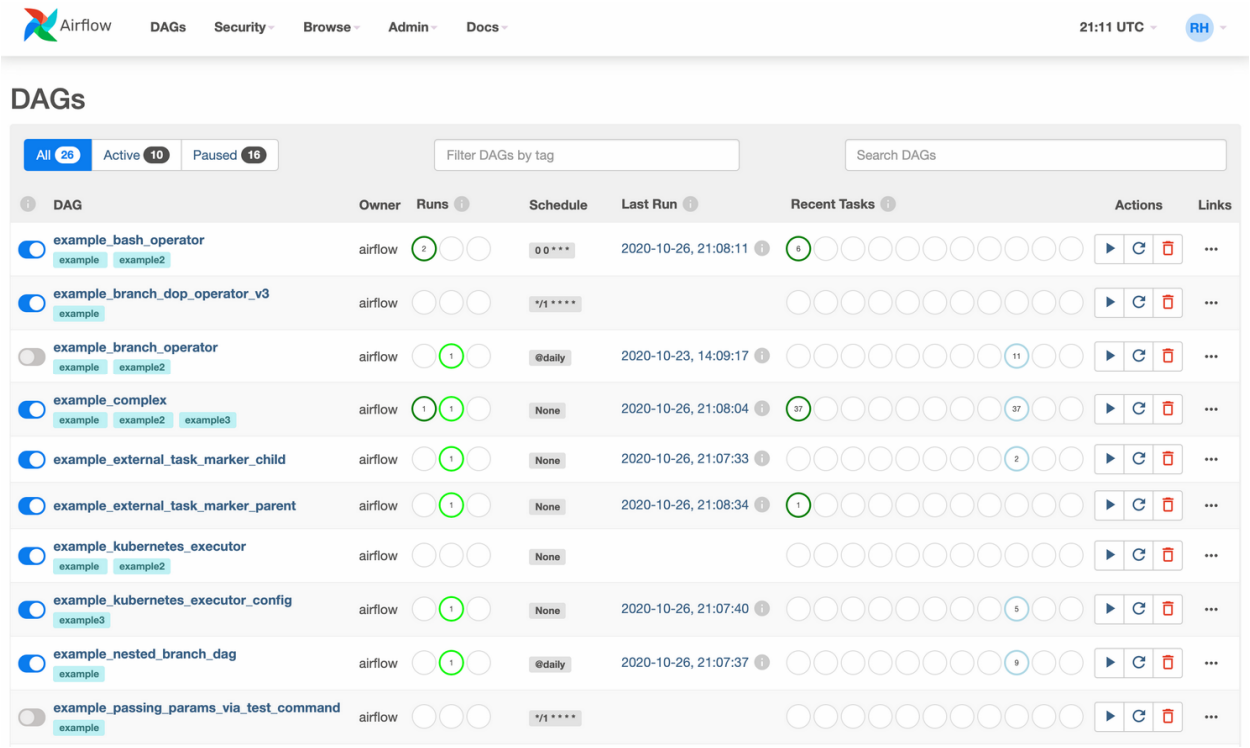


Рисунок 1.6 - Apache Airflow

Особливості:

- Підходить для складних задач ETL з великою кількістю кроків.
- Чіткий контроль за залежностями та виконанням задач.
- Підтримка масштабування через Celery, Kubernetes.
- Інтеграція з email, API, базами даних.

5. Informatica PowerCenter

Informatica — одна з найстаріших та найвідоміших ETL-платформ для корпоративного сегмента, що забезпечує високий рівень продуктивності та безпеки.

Особливості:

- Сильна підтримка великих обсягів даних.
- Потужна система трансформацій і профілювання даних.

- Інтеграція з ERP, CRM, email-системами.

- Підтримка як локальних, так і хмарних джерел.

6. AWS Glue

AWS Glue — керована ETL-служба від Amazon Web Services. Призначена для інтеграції даних у хмарних середовищах.

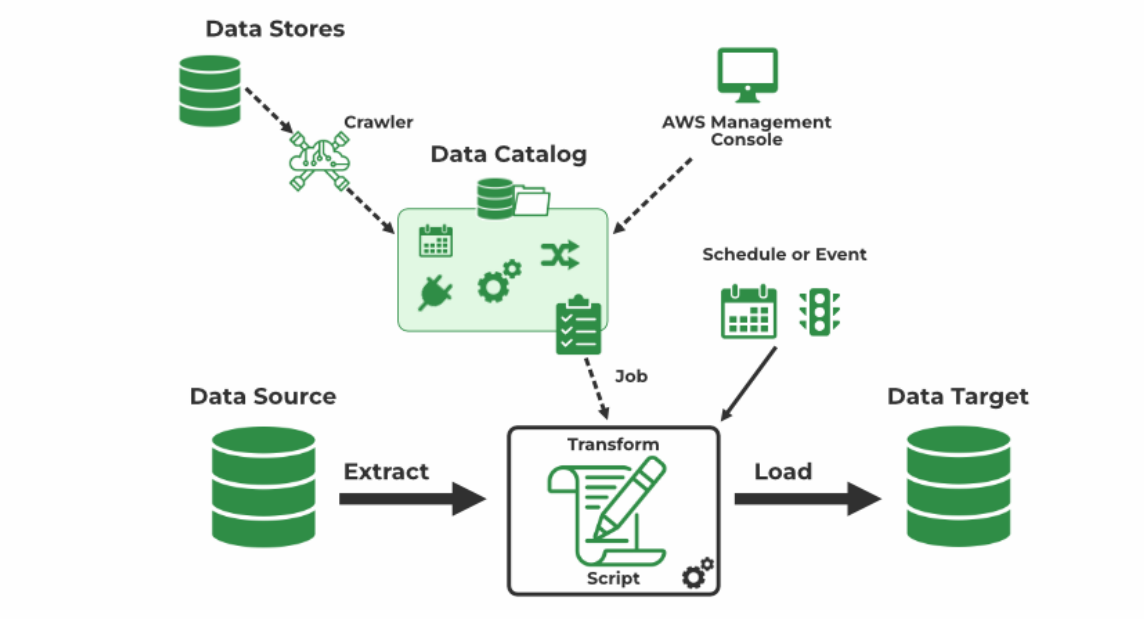


Рисунок 1.7 – Архітектура процесу ETL на основі AWS Glue

Особливості:

- Автоматичне виявлення схем даних.

- Підтримка Python і Spark для трансформацій.

- Інтеграція з Amazon S3, RDS, Redshift, а також з email через API.

- Висока масштабованість.

7. Pentaho Data Integration (PDI)

Платформа з відкритим кодом, яка забезпечує візуальне середовище для побудови ETL-процесів.

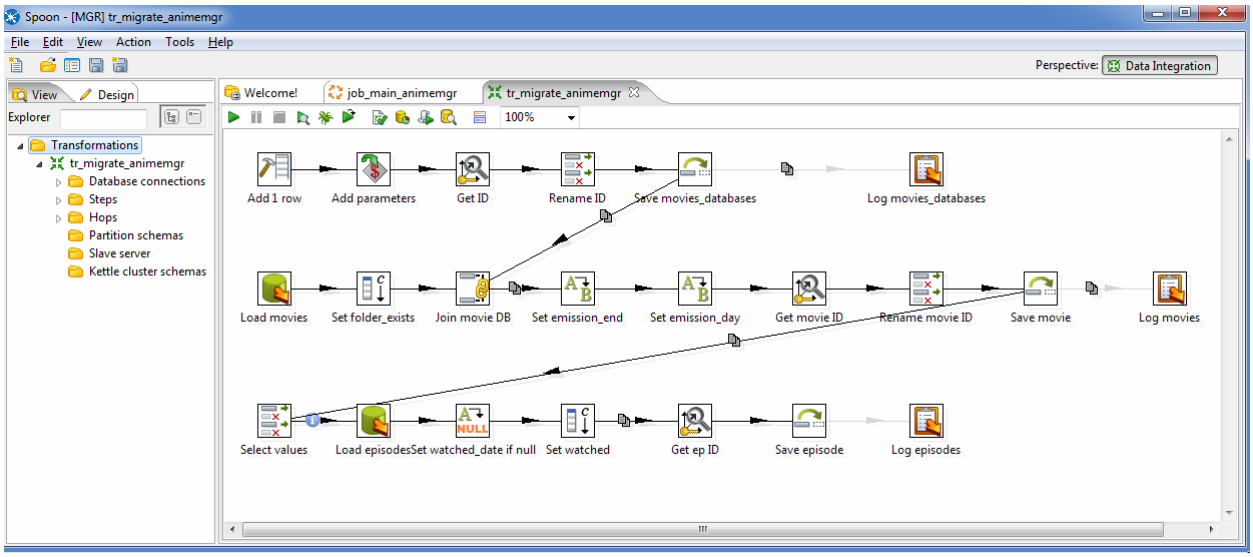


Рисунок 1.8 - Pentaho Data Integration (PDI)

Особливості:

- Просте створення потоків даних через drag-and-drop інтерфейс.
- Підтримка великої кількості джерел.
- Можливість зчитування даних з email (через POP3/IMAP).
- Інтеграція з аналітикою та візуалізацією.

Якщо система зосереджена на обробці електронних листів, то особливо цінною є підтримка поштових протоколів (POP3, IMAP, EWS) або API, а також можливість налаштування трансформацій даних для нестандартних форматів (наприклад, вкладених документів, HTML або таблиць у листах).

Висновки до першого розділу

У першому розділі проведено комплексний аналіз предметної області, що стосується процесів автоматизованого вилучення та обробки даних з електронних листів. Визначено основні джерела, типи вхідних даних, а також вимоги до їх структуризації. Особливу увагу приділено методології ETL (Extract, Transform and Load) як ключовій концепції для побудови системи обробки інформації.

Досліджено архітектурні підходи та методи інтеграції поштових сервісів у бізнес-процеси, проаналізовано варіанти реалізації ETL-процесів у контексті роботи з напівструктурованими текстами електронних листів. Запропоновано структурну модель автоматизованої системи, яка дозволяє поєднувати доступ до поштового середовища, обробку текстового вмісту, трансформацію даних і подальше збереження у БД.

Загалом, результати цього розділу створили методологічне підґрунтя для проєктування та реалізації ефективної програмної системи автоматизованої обробки даних з електронної пошти.

РОЗДІЛ 2. АЛГОРИТМІЧНА РЕАЛІЗАЦІЯ СИСТЕМИ ОБРОБКИ ДАНИХ З ЕЛЕКТРОННИХ ЛИСТІВ КЛІЄНТІВ

2.1. Ключові технології реалізації системи

Функціональність та архітектура описаної системи автоматизованої обробки даних з електронної пошти базуються на інтеграції наступних ключових технологій:

2.1.1. Exchange Web Services Managed API (EWS API)

Exchange Web Services (EWS) Managed API є критично важливим компонентом, що забезпечує програмний доступ до функціоналу Microsoft Exchange Server, зокрема до поштових скриньок та їх вмісту. В контексті даної системи, EWS API виконує фундаментальну функцію на етапі Екстракції (Extract) ETL-процесу, дозволяючи фоновому сервісу системи здійснювати моніторинг (опитування) визначених поштових скриньок, застосовувати критерії фільтрації до вхідних повідомлень та надійно вилучати релевантні електронні листи разом з їх вкладеними файлами.

Exchange Server надає базову інфраструктуру, необхідну для роботи системи обміну повідомленнями. Exchange Server надає базу даних для зберігання даних електронної пошти, інфраструктуру транспортування для переміщення даних електронної пошти з одного місця в інше та точки доступу для доступу до даних електронної пошти через кілька різних клієнтіВ. Це підкреслює вибір Exchange Server як надійної платформи-джерела даних та EWS API як ефективного механізму доступу до цих даних для цілей автоматизованої обробки. Типова архітектура взаємодії клієнтського додатку з Exchange Server через EWS із використанням протоколу Simple Object Access Protocol (SOAP) ілюструється на рисунку 2.1.

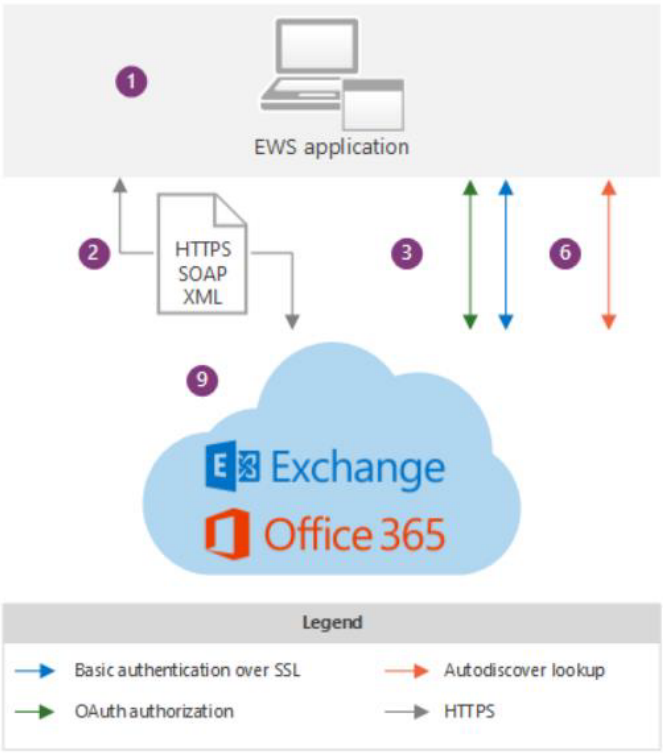


Рисунок 2.1 - Архітектура додатку, що використовує Exchange Web Service, та Exchange Online

2.1.2. AngularJS

Для реалізації рівня представлення (Presentation Layer) системи використовується JavaScript-фреймворк AngularJS. Ця технологія забезпечує розробку динамічного односторінкового графічного інтерфейсу користувача (ГІК/GUI), який слугує основним засобом взаємодії оператора із системою. Архітектурні особливості AngularJS, такі як ін'єкція залежностей, двостороннє зв'язування даних (що спрощує синхронізацію даних між моделлю та представленням), а також надання абстракцій у вигляді контролерів, сервісів та механізмів маршрутизації, сприяють створенню модульного, розширюваного та легкого в підтримці фронтенду. Високорівневі абстракції AngularJS спрощують маніпуляції з об'єктною моделлю документа (DOM) HTML, що є важливим для реалізації функціоналу конфігурації параметрів обробки даних, візуалізації статусу

виконання завдань, відображення системних помилок та інтерактивного формування та представлення звітів на основі оброблених даних.

2.1.3. Web API (ASP.NET Web API)

На рівні обробки та сервісного шару (Processing and Service Layer) системи, ймовірно, використовується ASP.NET Web API, що є частиною фреймворку .NET Framework.

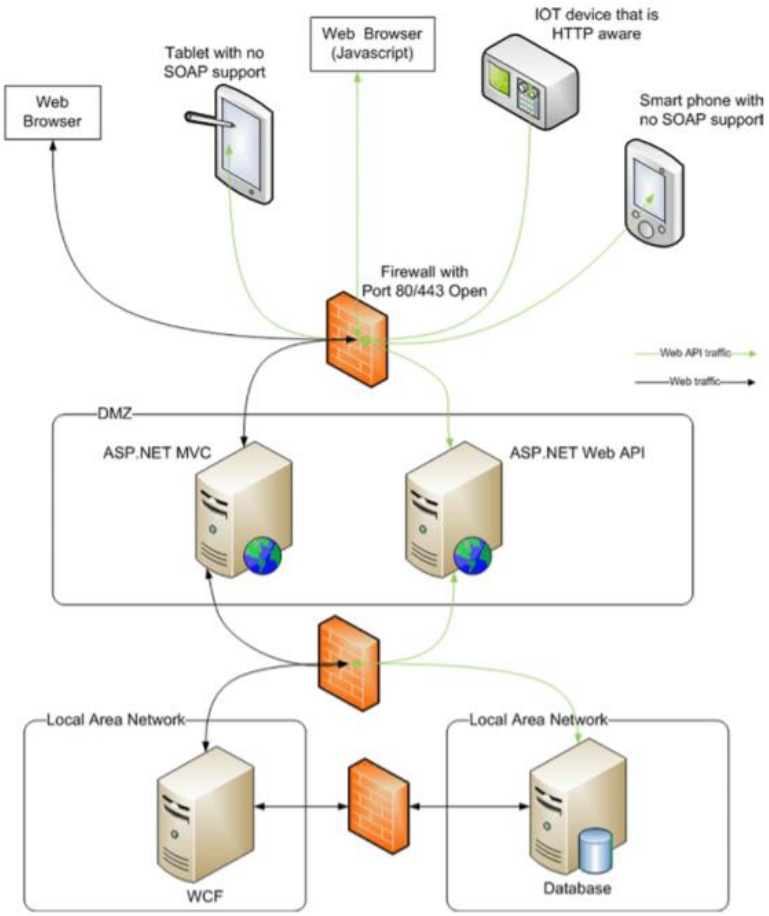


Рисунок 2.2 - Типова діаграма розгортання з Web API

Ця технологія дозволяє розробляти Representational State Transfer (RESTful) сервіси, що взаємодіють по протоколу HTTP. Web API слугує як проміжний шар (backend service) між графічним інтерфейсом користувача (фронтендом) та базовою логікою обробки даних (фоновим сервісом) і

доступу до бази даних. Він обробляє запити від GUI, що стосуються конфігурації системи, моніторингу статусу, управління завданнями та запитів на отримання даних для звітів. Web API надає високу гнучкість у контролі над форматами та вмістом HTTP-запитів і відповідей, що є критично важливим для забезпечення надійної та стандартизованої взаємодії між компонентами системи. Типове розгортання додатку ASP.NET Web API ілюструється на рисунку 2.2.

2.1.4. База даних Microsoft SQL Server

База даних Microsoft SQL Server виконує роль цільового сховища даних (Target Data Repository) у даній архітектурі, реалізуючи етап Завантаження (Load) ETL-процесу. Ця реляційна система управління базами даних обрана для надійного та структурованого зберігання оброблених даних, вилучених з електронної пошти.

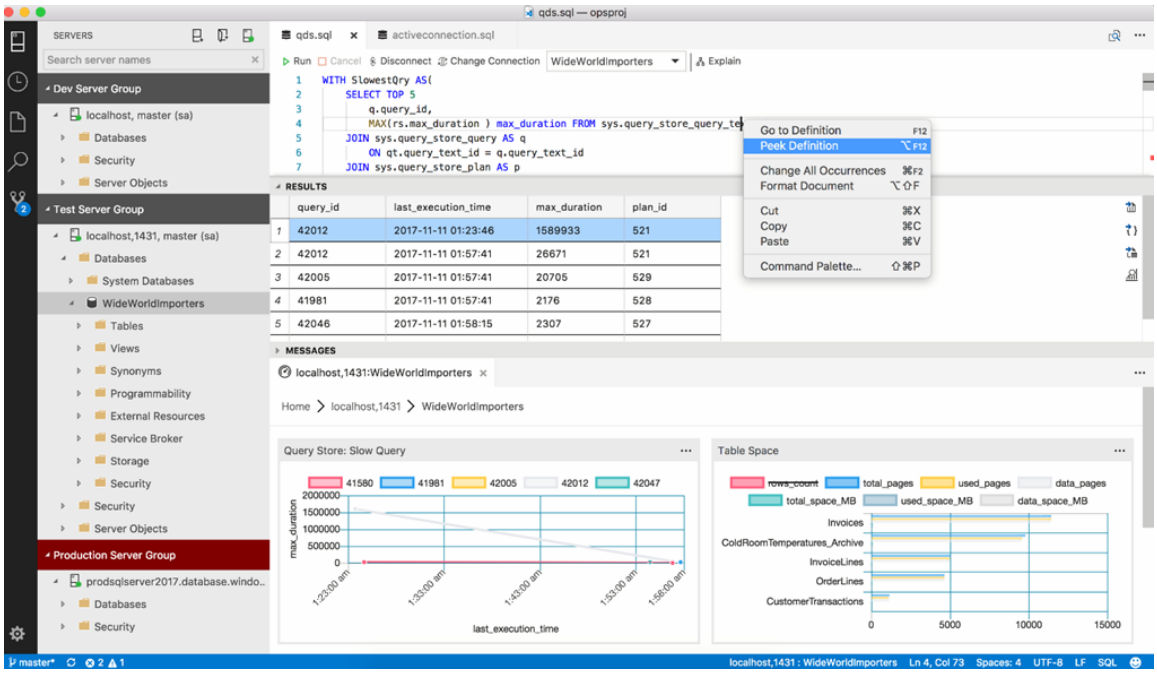


Рисунок 2.3 - Microsoft SQL Server

Крім функцій зберігання, певні операції масового завантаження (bulk loading) та, можливо, частини трансформації даних (data transformation)

можуть ефективно виконуватися безпосередньо на рівні серверу бази даних, використовуючи його оптимізовані механізми. Для досягнення високої продуктивності під час завантаження значних обсягів даних з проміжного шару (staging area) або результатів трансформації до фінальних таблиць активно використовуються спеціалізовані механізми, такі як клас SQLBulkCopy, який забезпечує швидкий та ефективний потоковий запис даних до таблиць SQL Server, мінімізуючи накладні витрати.

Використання цих технологій дозволяє створити надійну, масштабовану та ефективну систему для автоматизованої обробки структурованих даних, що надходять через електронну пошту, забезпечуючи повний цикл ETL від вилучення до завантаження та подальшого використання даних.

2.2. Процес зіставлення атрибутів

На цьому етапі заголовки вихідних стовпців з шаблону вхідного файлу відображаються у вигляді сітки, як показано на рисунку 2.4. Стовпці бази даних відображають списки користувацьких стовпців, які підтримують п'ять типів даних:

1. Boolean;
2. DateTime;
3. Decimal (24,10), довжина 24 і може зберігати до 10 знаків після коми;
4. Number;
5. String.

Різні типи даних у стовпці бази даних дозволяють користувачеві правильно зіставити відповідний стовпець. Подальші операції над стовпцями виконуються на основі типу даних стовпця, наприклад, над стовпцем з

числовим/десятковим типом даних можуть виконуватися всі математичні операції тощо.

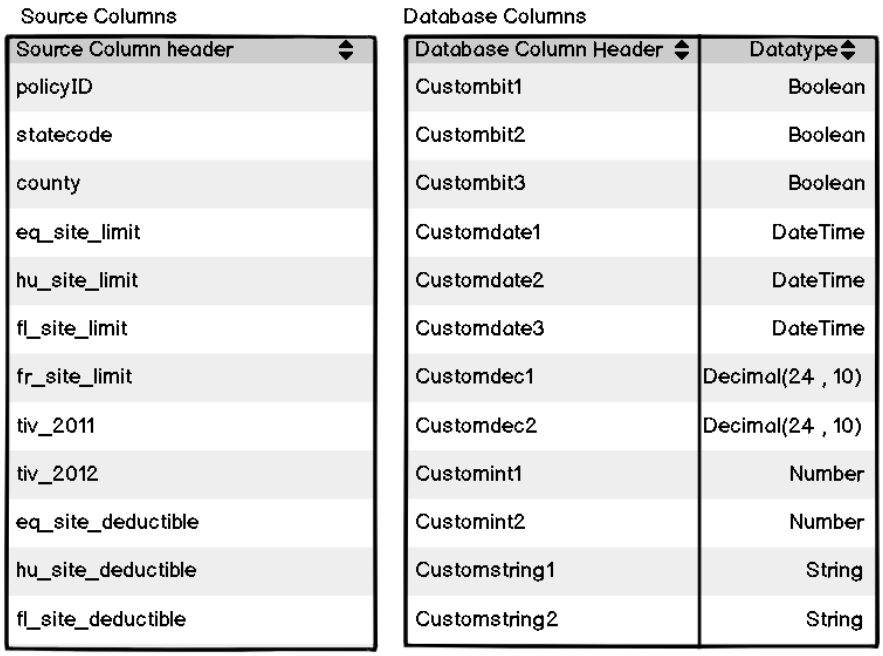


Рисунок 2.4 - Сітка вихідних стовпців та стовпців бази даних зі сторінки зіставлення атрибутів

Після того, як вихідний стовпець (з вхідного джерела даних або файлу) та відповідний цільовий стовпець бази даних зіставлено (тобто визначено відповідність між ними для імпорту/обробки даних), ця встановлена пара відображається у сітці зіставлених атрибутів, показаній на рисунку 2.5. З цією сітки можна вибирати стовпці, які, ймовірно, містять унікальні значення (наприклад, ідентифікатори записів, первинні ключі), що може бути важливим для подальшої обробки даних, індексації або перевірки цілісності. Вибір здійснюється шляхом встановлення відповідного прапорця навпроти назви стовпця. Функціонал також передбачає можливість видалення зіставлення – скасування встановленого зв'язку між вихідним і цільовим стовпцями за допомогою відповідної опції.

Source Column header	Database Column	Datatype	Uniqu	Action
policyID	Custombit1	Boolean	☑	(o)Delete
statecode	Custombit2	Boolean	☐	(o) Delete
county	Custombit3	Boolean	☐	(o) Delete
eq_site_limit	Customdate1	DateTime	☐	(o) Delete
hu_site_limit	Customdate2	DateTime	☐	(o) Delete
fl_site_limit	Customdate3	DateTime	☐	(o) Delete
fr_site_limit	Customdec1	Decimal(24 ,	☐	(o) Delete
tiv_2011	Customdec2	Decimal(24 ,	☐	(o) Delete
tiv_2012	Customint1	Number	☐	(o) Delete
eq_site_deductible	Customint2	Number	☐	(o) Delete
hu_site_deductible	Customstring1	String	☐	(o) Delete
fl_site_deductible	Customstring2	String	☐	(o) Delete

Рисунок 2.5 - Сітка зіставлених атрибутів

У цьому варіанті додано уточнення щодо "вихідного стовпця" (звідки він береться), "цільового стовпця бази даних", пояснення суті зіставлення ("визначено відповідність"), технічний термін "сітка зіставлених атрибутів", уточнення щодо "унікальних значень" (чому вони важливі, приклади використання) та деталей процесу "видалення зіставлення" (скасування зв'язку).

2.3. Розробка діаграми варіантів використання

На рисунку 2.6 зображено діаграму варіантів використання (Use Case Diagram). Вона ілюструє функціональні вимоги до системи з точки зору її акторів та варіантів використання.

Наведемо опис функціональності.

1. Актори:

- User (Користувач): Це основний актор, який представляє людину, що безпосередньо взаємодіє з користувацьким інтерфейсом системи.

- EmailPollingService: Цей актор (позначений як `<<Actor>> EmailPollingService`) представляє собою автоматизовану службу або компонент системи, який діє як зовнішня сутність або ініціатор певних процесів, пов'язаних з обробкою даних, що надходять електронною поштою.

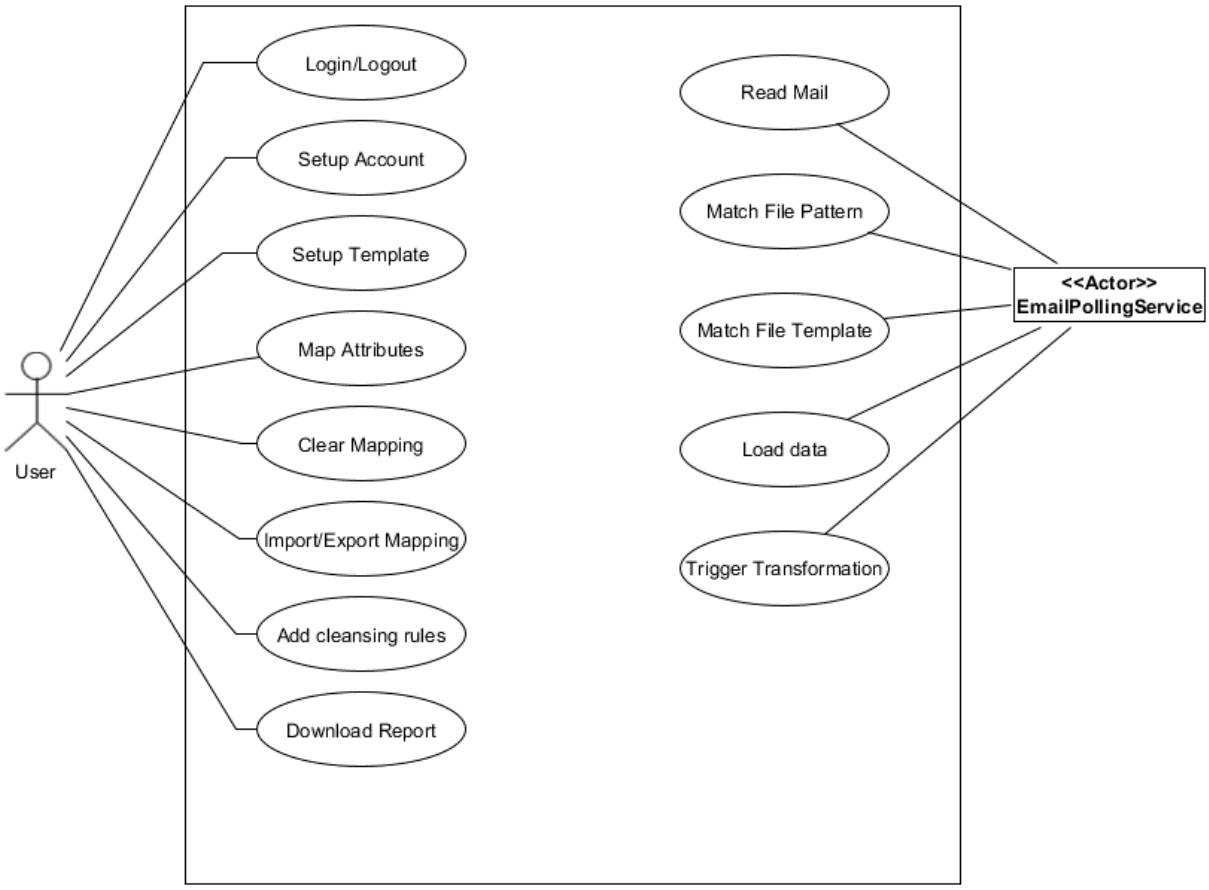


Рисунок 2.6 – Діаграма варіантів використання системи обробки даних

2. Варіанти використання (Функціональність системи):

- Функціональність, доступна Користувачеві (User): Діаграма показує, що Користувач може виконувати такі основні дії:

- Login/Logout (Вхід/Вихід): Функції керування сесією користувача в системі.

- Setup Account (Налаштування облікового запису): Дозволяє користувачеві створювати або конфігурувати власний обліковий запис.

- Setup Template (Налаштування шаблону): Налаштування шаблонів, ймовірно, для обробки вхідних файлів або даних.

- Map Attributes (Зіставлення атрибутів): Визначення відповідності між полями у вхідних даних та полями в системі чи базі даних (як обговорювалося в попередньому тексті).

- Clear Mapping (Очищення зіставлення): Скасування існуючих правил зіставлення.

- Import/Export Mapping (Імпорт/Експорт зіставлення): Можливість збереження або завантаження правил зіставлення.

- Add cleansing rules (Додавання правил очищення): Налаштування правил для очищення або валідації даних перед їх обробкою.

- Download Report (Завантаження звіту): Отримання звітів про оброблені дані або діяльність системи.

- Функціональність, пов'язана з EmailPollingService: Цей актор взаємодіє з варіантами використання, які описують автоматизовані процеси:

- Read Mail (Читання пошти): Отримання електронних листів із визначеної поштової скриньки.

- Match File Pattern (Зіставлення з шаблоном файлу): Ідентифікація та перевірка вхідних файлів на відповідність певним патернам або форматам.

- Match File Template (Зіставлення з шаблоном файлу - ймовірно, те ж саме або пов'язане з попереднім): Можливо, більш конкретне зіставлення виявленого файлу з налаштованим шаблоном обробки.

- Load data (Завантаження даних): Процес завантаження даних з виявленого та ідентифікованого файлу в систему (наприклад, до тимчасового сховища або для подальшої обробки).

- Trigger Transformation (Запуск трансформації): Ініціація процесу трансформації завантажених даних, використовуючи налаштовані правила (операції та умови).

Отже, діаграма варіантів використання ефективно демонструє основні можливості системи, розділяючи їх на функції, доступні безпосередньо користувачеві через інтерфейс (налаштування, керування, звітність) та автоматизовані процеси, що ініціюються або виконуються фоновою службою EmailPollingService (отримання та первинна обробка даних з пошти, завантаження та запуск трансформації).

2.4. Проектування діаграми діяльності

На рисунку 2.7 подано діаграму діяльності (Activity Diagram) яка деталізує потік виконання певного бізнес-процесу або операції в системі, показуючи послідовність дій, точки ухвалення рішень та компоненти системи, відповідальні за виконання цих дій.

Наведемо опис функціональності, зображеної на діаграмі діяльності.

Діаграма описує автоматизований процес імпорту даних, які надходять до системи як вкладення до електронних листів. Процес включає взаємодію між зовнішнім відправником, поштовим сервером, внутрішніми службами обробки електронної пошти, класами імпорту та базою даних.

1. Ініціація процесу (Business/Organization's Mailbox):

- Процес починається, коли Бізнес/Організація надсилає електронний лист із даними як вкладенням.

2. Отримання та нотифікація (Exchange Email Server -> Exchange Web Server API -> EmailPollingService):

- Лист отримується Поштовим сервером Exchange.

- Сервер надсилає нотифікацію підписникам цієї поштової скриньки.

- Ця нотифікація запускає зворотний виклик (callback) через Exchange Web Server API.

- Служба EmailPollingService на Веб-сервері обробляє цей зворотний виклик.

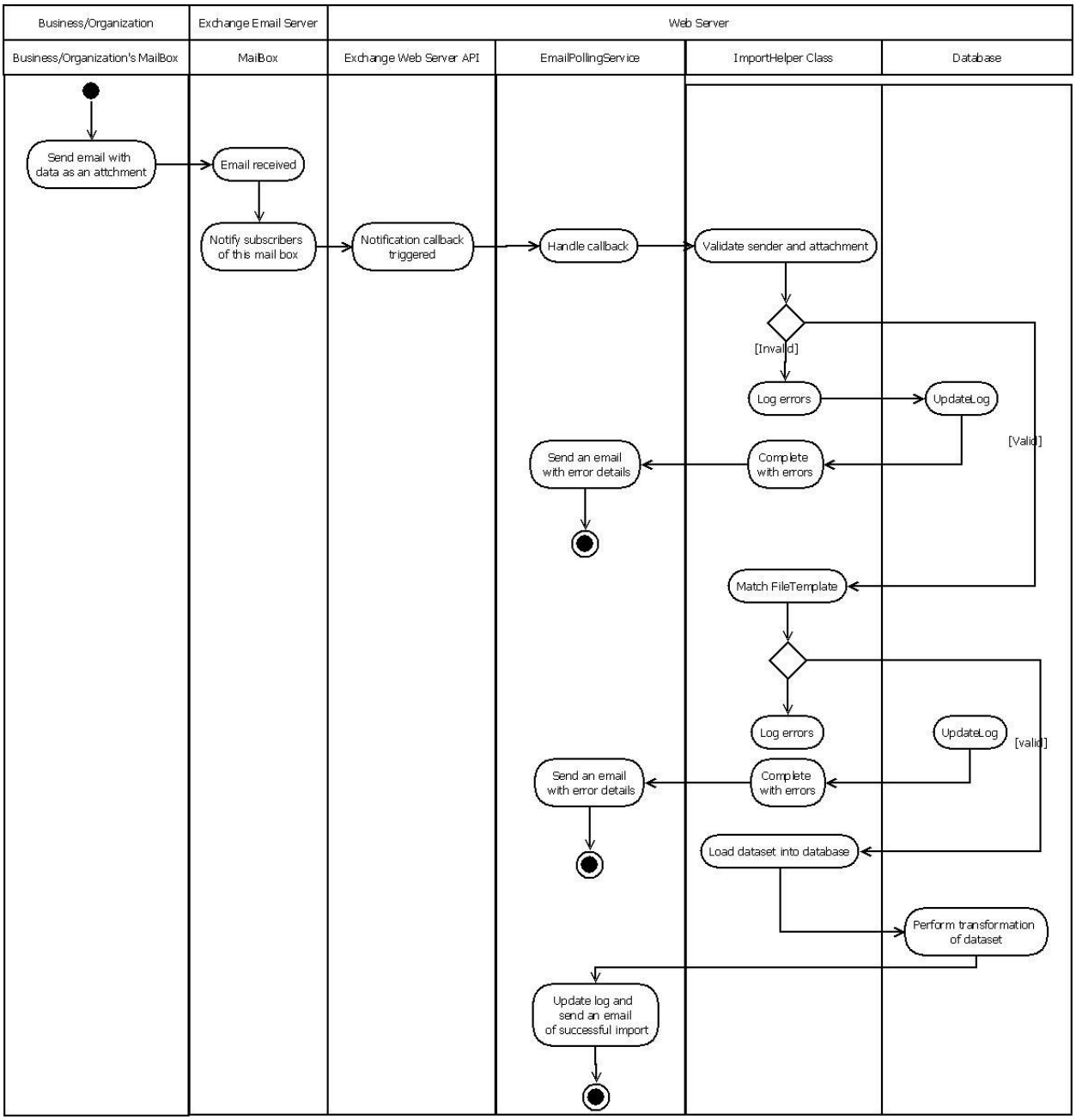


Рисунок 2.7 – Діаграма діяльності

3. Валідація відправника та вкладення (EmailPollingService -> ImportHelper Class):

- EmailPollingService передає управління компоненту ImportHelper Class для валідації відправника та вкладення.

4. Точка ухвалення рішення: Результат валідації:

- [Invalid] (Недійсний): Якщо валідація не проходить (наприклад, невірний відправник, пошкоджений файл, невірний формат):

- Логуються помилки в системі.

- Журнал оновлюється в Базі даних.

- EmailPollingService надсилає електронний лист із деталями помилки (ймовірно, відправнику або адміністратору).

- Процес завершується для цього листа (досягається кінцевий вузол).

- [Valid] (Дійсний): Якщо валідація успішна:

- Виконується спроба зіставлення шаблону файлу (Match File Template).

5. Точка ухвалення рішення: Результат зіставлення шаблону:

- У випадку помилок: Якщо при зіставленні шаблону виникають помилки:

- Логуються помилки.

- Журнал оновлюється в Базі даних.

- EmailPollingService надсилає лист із деталями помилки.

- Процес завершується для цього листа.

- [Valid] (Без помилок): Якщо зіставлення шаблону успішне:

- Набір даних із вкладення завантажується до Бази даних.

6. Трансформація даних (Database):

- Після завантаження в Базі даних виконується трансформація набору даних.

7. Завершення процесу та нотифікація про успіх (EmailPollingService):

- Після завершення трансформації EmailPollingService оновлює журнал з відміткою про успішний імпорт.

- Надсилається електронний лист про успішне завершення імпорту (ймовірно, відправнику).

- Процес завершується.

Ця діаграма діяльності описує автоматизований конвеєр обробки даних, що ініціюється надходженням електронного листа. Вона охоплює

етапи від отримання листа та валідації вкладення до завантаження даних у базу даних, їх трансформації та фінального сповіщення про результат, включаючи обробку помилок на проміжних етапах. Це ключова функція системи, що дозволяє інтегрувати дані із зовнішніх джерел через механізм електронної пошти.

2.5. Розробка діаграм послідовностей системи

2.5.1. Діаграма послідовності для налаштування облікового запису

На рисунку 2.8 зображено діаграму послідовності (Sequence Diagram), яка моделює взаємодії між об'єктами або компонентами системи та їхню послідовність у часі для процесу налаштування облікового запису (Account Setup).

Діаграма показує три основні лінії життя (lifelines), що представляють учасників взаємодії:

1. Window:UI: Представляє користувацький інтерфейс або вікно програми, з яким взаємодіє користувач.
2. WebServer: Представляє серверний компонент, що обробляє запити від інтерфейсу користувача та взаємодіє з базою даних.
3. Database: Представляє шар збереження даних.

Діаграма деталізує такі основні функціональні послідовності під час налаштування облікового запису:

1. Створення облікового запису (CreateAccount):

- Процес починається з дії користувача в інтерфейсі (`Window:UI`), яка ініціює надсилання повідомлення `CreateAccount()` до `WebServer`.

- `WebServer` отримує запит і першим кроком здійснює валідацію даних облікового запису, надсилаючи повідомлення `Validate()` до `Database`. `Database` повертає результат валідації (`Result`).

- Якщо валідація успішна, `WebServer` надсилає повідомлення `Save()` до `Database` для збереження даних нового облікового запису. `Database` підтверджує збереження, повертаючи `Result`.

- Альтернативний сценарій (alt fragment): Якщо валідація не вдається (`[ValidationFailed]`), `WebServer` надсилає код помилки (`ErrorCode`) назад до `Window:UI`. `Window:UI` потім відображає цю помилку користувачеві (`DisplayError()`).

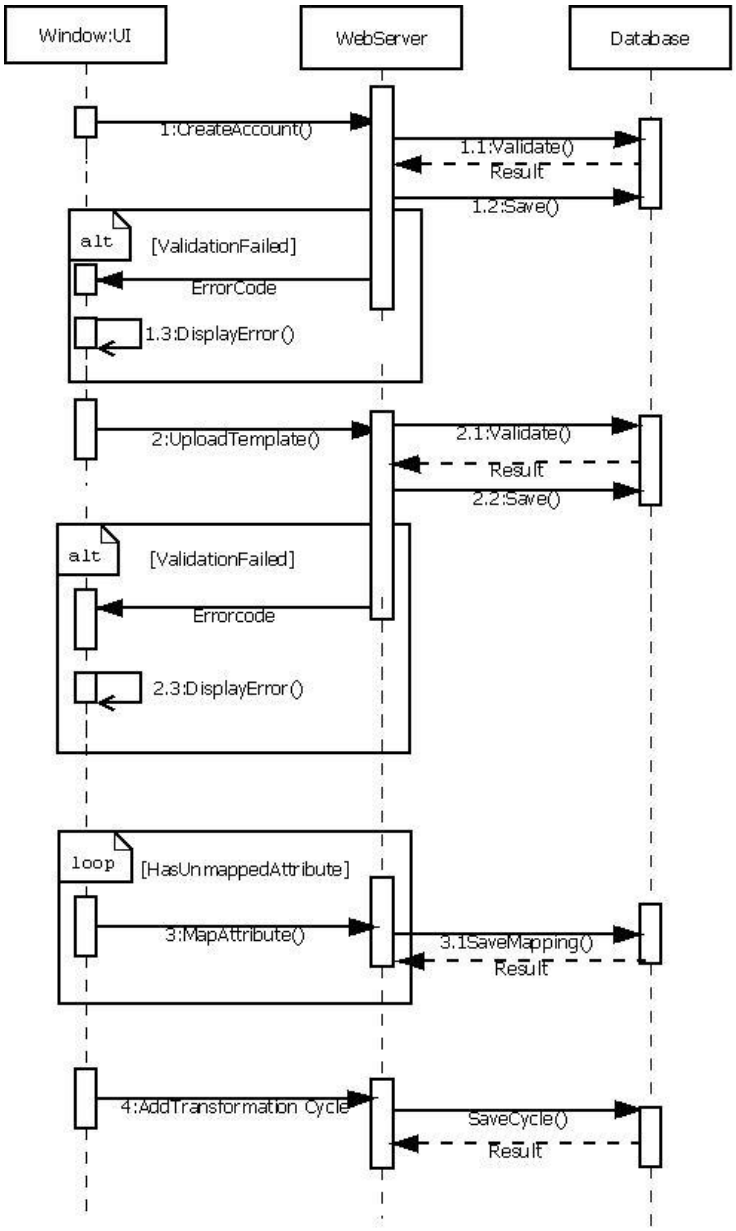


Рисунок 2.8 - Діаграма послідовності для налаштування облікового запису

2. Завантаження шаблону (UploadTemplate):

- Наступним кроком користувач через інтерфейс ініціює `UploadTemplate()`, надсилаючи запит до `WebServer`.

- Як і при створенні облікового запису, `WebServer` спочатку виконує валідацію шаблону, взаємодіючи з `Database` (`Validate()`, `Result`).

- Якщо валідація шаблону успішна, шаблон зберігається в `Database` за допомогою повідомлення `Save()`. `Database` повертає `Result`.

- Альтернативний сценарій (alt fragment): Якщо валідація шаблону не вдається (`[ValidationFailed]`), `WebServer` повертає `ErrorCode` до `Window:UI`, і інтерфейс відображає помилку (`DisplayError()`).

3. Зіставлення атрибутів (MapAttribute):

- Цей етап представлений у вигляді циклу (loop) з умовою `[HasUnmappedAttribute]` (поки є незіставлені атрибути).

- У кожній ітерації циклу користувач виконує дію `MapAttribute()` через інтерфейс, надсилаючи повідомлення до `WebServer`.

- `WebServer` обробляє запит на зіставлення і зберігає правило зіставлення в `Database` за допомогою повідомлення `SaveMapping()`. `Database` повертає `Result`.

- Цикл повторюється для кожного атрибута, що потребує зіставлення.

4. Додавання циклу трансформації (AddTransformationCycle):

- Після етапів зіставлення, користувач може додати правила трансформації, ініціюючи `AddTransformationCycle()` з `Window:UI` до `WebServer`.

- `WebServer` отримує запит і зберігає конфігурацію циклу трансформації в `Database` за допомогою повідомлення `SaveCycle()`. `Database` повертає результат збереження (`Result`).

Представлена діаграма послідовності для налаштування облікового запису демонструє трирівневий патерн взаємодії: клієнтський інтерфейс

ініціює дії, веб-сервер обробляє бізнес-логіку (включаючи валідацію) та виступає посередником, а база даних відповідає за збереження та керування даними конфігурації (облікового запису, шаблонів, зіставлень, правил трансформації). Діаграма чітко показує порядок викликів методів, передачу даних та обробку помилок під час різних етапів процесу налаштування.

2.5.2. Діаграма послідовності процесу імпорту

На рисунку 2.9 подано діаграму послідовності яка ілюструє потік взаємодій між компонентами системи під час процесу імпорту даних, що ініціюється отриманням електронного листа.

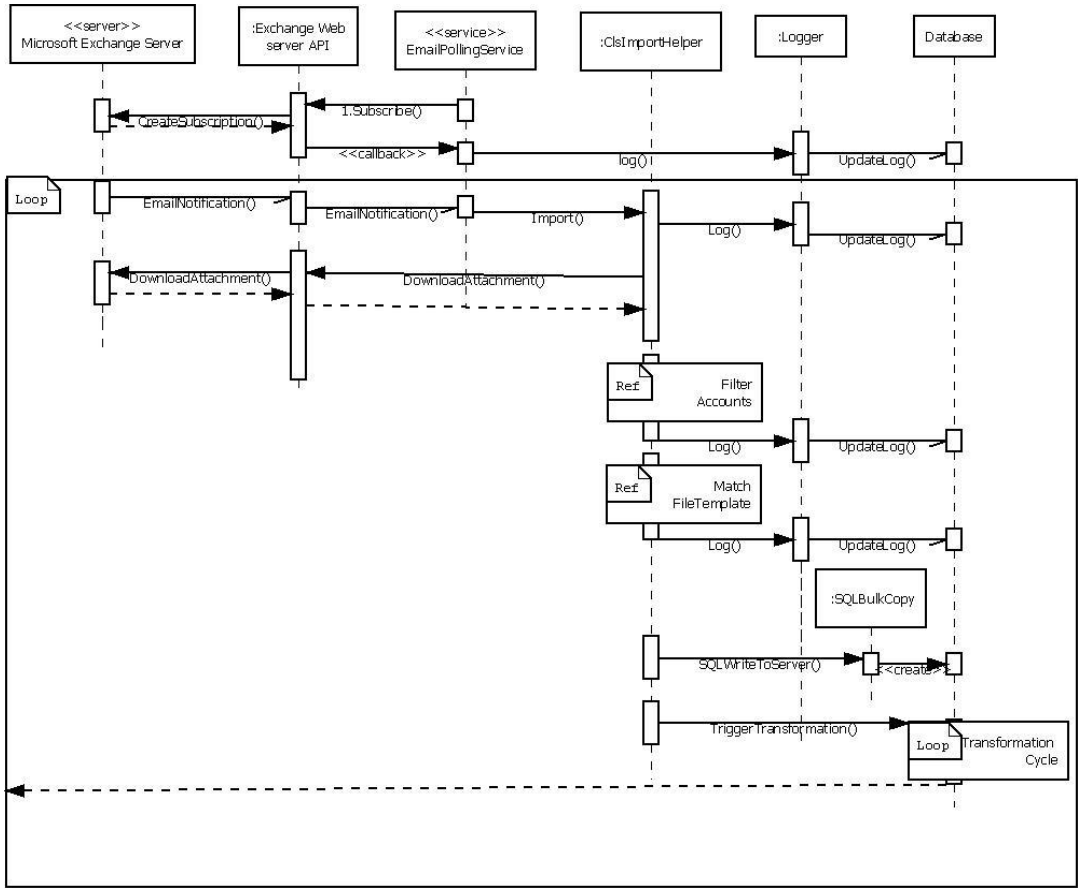


Рисунок 2.9 - Діаграма послідовності процесу імпорту

Діаграма показує послідовність обміну повідомленнями між такими основними учасниками:

1. `<<server>> Microsoft Exchange Server`: Сервер електронної пошти Microsoft Exchange.

2. `:Exchange Web server API`: Програмний інтерфейс для доступу до сервісів Exchange.

3. `<<service>> EmailPollingService`: Сервіс опитування електронної пошти, що працює на веб-сервері.

4. `:ClsImportHelper`: Клас або компонент, що містить допоміжну логіку для процесу імпорту.

5. `:Logger`: Компонент системи для реєстрації подій.

6. `Database`: База даних, де зберігаються дані та логінформація.

Наведемо опис послідовності взаємодій (Функціональність процесу імпорту):

1. Ініціалізація та підписка (Loop на початку):

- На початку процесу (можливо, при старті служби `EmailPollingService`) відбувається підписка на нотифікації про нові листи. `EmailPollingService` надсилає повідомлення `CreateSubscription()` до `Microsoft Exchange Server` через `Exchange Web server API`.

- Коли на сервері Exchange надходить новий лист, спрацьовує механізм зворотного виклику (`<<callback>>`). Нотифікація передається через `Exchange Web server API` до `EmailPollingService`.

- `EmailPollingService` фіксує отримання нотифікації, надсилаючи повідомлення `Log()` компоненту `:Logger`, який оновлює журнал у `Database` (`UpdateLog()`).

2. Обробка вхідного листа (Після початкового Loop):

- Отримавши нотифікацію (`EmailNotification()`), `EmailPollingService` ініціює процес імпорту, викликаючи метод `Import()` у `:ClsImportHelper`.

- Логування: `:ClsImportHelper` розпочинає свою роботу з логування початку процесу (`Log()`, `UpdateLog()`).

- Отримання вкладення: `EmailPollingService` (або делегуючи `ClsImportHelper`) надсилає запит `DownloadAttachment()` до `Microsoft Exchange Server` для завантаження вмісту листа. Сервер повертає дані вкладення.

- Фільтрація облікових записів (Ref: Filter Accounts): Виконується референтний процес фільтрації, ймовірно, для перевірки, чи відправник належить до дозволених облікових записів або пов'язаний з конкретними правилами обробки. Результати логуються.

- Зіставлення шаблону файлу (Ref: Match File Template): Виконується референтний процес зіставлення вкладення з визначеними шаблонами (як описано в попередніх діаграмах). Результати також логуються.

- Завантаження даних до бази даних (SQLBulkCopy, SQLWriteToServer, \<\<create\>\>): Якщо файл відповідає шаблону, відбувається безпосереднє завантаження даних. Діаграма показує використання механізмів масового копіювання даних (SQLBulkCopy, SQLWriteToServer) для ефективного запису в `Database`. Створюються тимчасові структури даних (`<<create>>`).

- Запуск трансформації даних (Trigger Transformation, Loop Transformation Cycle): Після успішного завантаження даних, `:ClsImportHelper` ініціює процес трансформації даних, викликаючи `Trigger Transformation()` у `Database`. Це запускає виконання одного або кількох циклів трансформації (Transformation Cycle), які, згідно з попередніми описами, виконуються безпосередньо на рівні бази даних.

3. Завершення

- Передбачається, що після завершення трансформації відбувається оновлення статусу в журналі танадсилання листа про успішне завершення. Пунктирна лінія, що повертається до початкової частини обробки листа, вказує на те, що `EmailPollingService` продовжує працювати в циклі, очікуючи наступних нотифікацій про нові листи.

Ця діаграма послідовності деталізує автоматизований, керований подіями (надходженням листа) процес імпорту даних. Вона показує, як служба опитування взаємодіє з поштовим сервером для отримання даних, делегує обробку допоміжному класу, який виконує валідацію, зіставлення, завантаження та ініціює трансформацію на рівні бази даних, при цьому весь процес супроводжується логуванням дій.

Висновки до другого розділу

У другому розділі зосереджено увагу на формалізації програмної логіки, яка забезпечує роботу системи автоматизованої обробки листів. Було здійснено вибір і обґрунтування ключових технологій реалізації: EWS API для взаємодії з поштовим сервером, AngularJS — для реалізації інтерфейсу користувача, ASP.NET Web API — як інструмент серверної логіки, а також MS SQL Server для збереження результатів обробки.

Змодельовано основні сценарії функціонування системи шляхом побудови діаграм варіантів використання, діяльності та послідовностей. Реалізовано процес зіставлення атрибутів листа зі структурою бази даних, що стало важливим етапом адаптації системи до різноманітних шаблонів електронних повідомлень.

Таким чином, розділ сформував основу алгоритмічної реалізації, яка забезпечує системну взаємодію всіх компонентів та чітке розмежування функціональних зон у межах архітектури

РОЗДІЛ 3. ПРОГРАМНА ІМПЛЕМЕНТАЦІЯ МЕТОДІВ ETL ДЛЯ АВТОМАТИЗАЦІЇ ПРОЦЕСІВ ОБРОБКИ ДАНИХ З ЕЛЕКТРОННИХ ЛИСТІВ КЛІЄНТІВ

3.1. Проектування макетів інтерфейсу і конфігурації системи

Ефективне функціонування системи автоматизованої обробки даних з електронної пошти вимагає попередньої конфігурації ключових параметрів, що визначають джерела даних та правила їх обробки. Цей процес включає створення та налаштування облікових записів та шаблонів файлів.

3.1.1. Створення облікового запису

В архітектурі системи концепція облікового запису (Account) представляє логічний контейнер або репозиторій даних для агрегації та обробки інформації, що надходить від конкретної зовнішньої організації або бізнес-суб'єкта. Обліковий запис асоціюється з певною поштовою скринькою або набором критеріїв для фільтрації вхідної кореспонденції.

Система підтримує можливість створення множинних облікових записів, що може бути використано для гнучкої сегрегації даних навіть у межах одного суб'єкта-джерела (наприклад, для розділення даних за типами документів чи підрозділами організації-партнера), забезпечуючи таким чином полегшення подальшого аналізу та управління даними. За замовчуванням, користувачам надається доступ до облікових записів, створених ними особисто.

Передбачено функціонал спільного доступу (шарингу) до облікових записів між авторизованими користувачами, що оптимізує можливості колективної роботи та моніторингу процесів обробки. Інтерфейс для створення нового облікового запису представлений на рисунку 3.1.

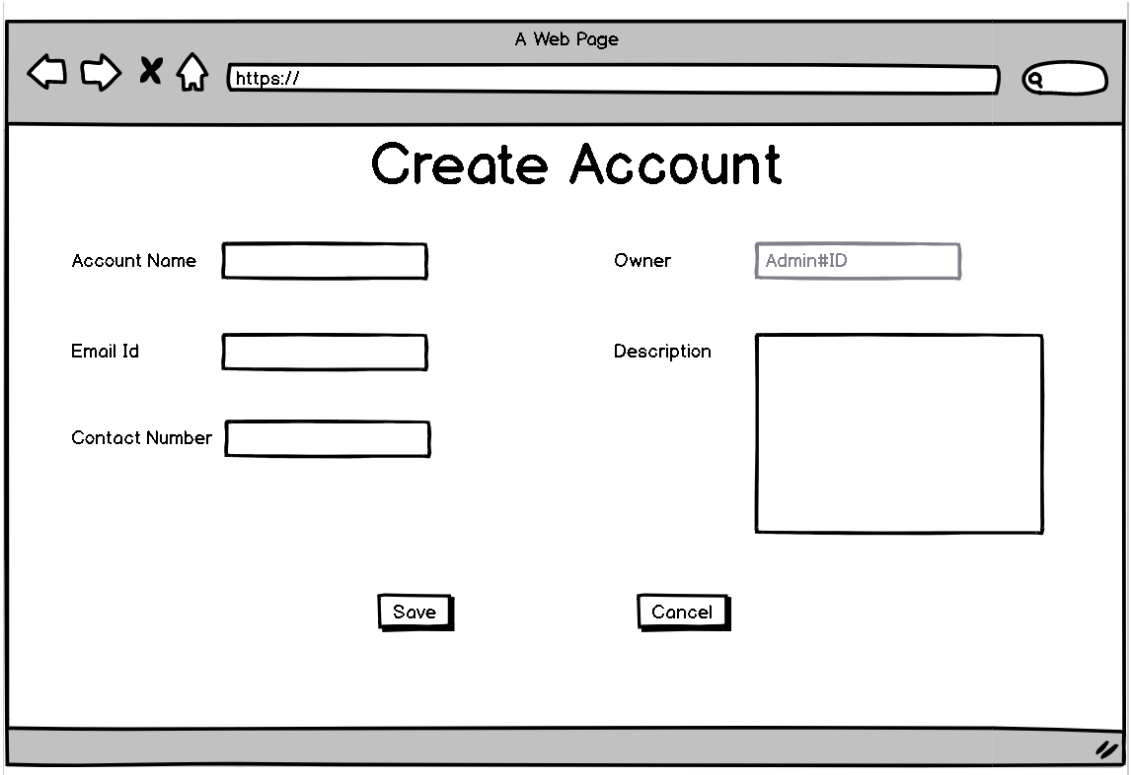


Рисунок 3.1 - Форма додавання облікового запису

Для кожного конфігурованого облікового запису може бути вказаний ідентифікатор електронної пошти (поле "Email ID field for each account"), на який надсилатимуться автоматизовані сповіщення щодо статусу та прогресу виконання операцій обробки даних, пов'язаних з цим обліковим записом. Це забезпечує оперативне інформування користувачів про результати роботи системи або виникнення проблем.

3.1.2. Налаштування шаблону файлу

Шаблон файлу (File Template) є об'єктом метаданих, що слугує для специфікації очікуваної структури вхідних даних, які асоційовані з конкретним обліковим записом. Цей шаблон визначає, як саме слід екстрагувати та попередньо обробляти дані з вкладених файлів певного формату. Процедура конфігурації Шаблону файлу для Облікового запису є одноразовою операцією, яка виконується лише при першому налаштуванні

нового типу вхідних файлів або у випадку зміни їх структури. Рисунок 3.2 ілюструє інтерфейс конфігурації шаблону файлу в межах Облікового запису.

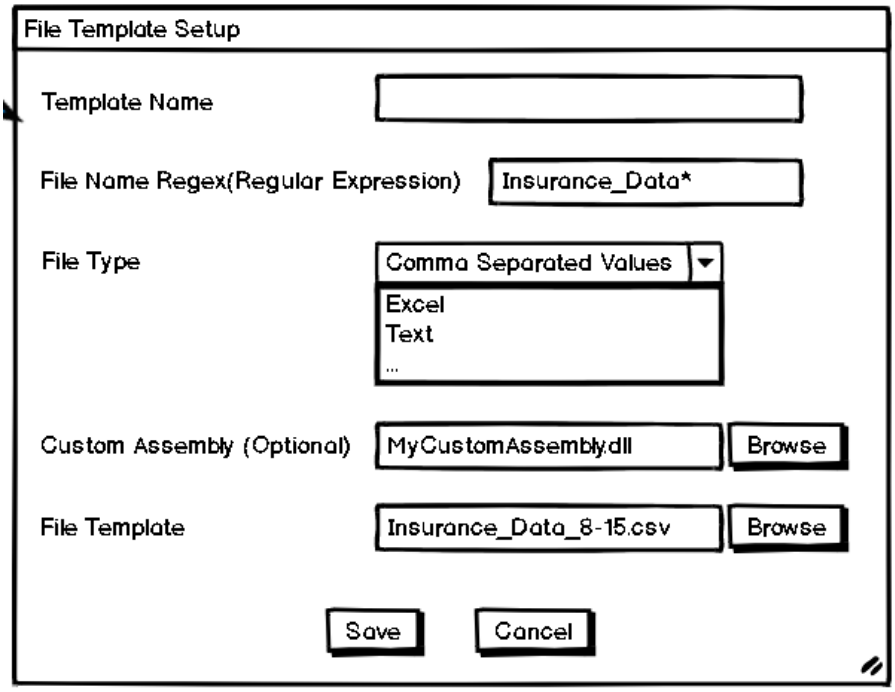


Рисунок 3.2 - Форма налаштування Шаблону файлу

Поля форми налаштування Шаблону файлу включають:

- Назва Шаблону (Template Name): Унікальний текстовий ідентифікатор, що використовується для розрізнення та ідентифікації специфічного шаблону обробки.

- Регулярний вираз для імені файлу (File name Regular expression): Критично важливий параметр, що містить регулярний вираз. Він використовується системою для автоматичного зіставлення імен вхідних файлів (вилучених з електронних листів, пов'язаних з даним Обліковим записом) з відповідним Шаблоном файлу. Наприклад, для файлів з іменами HSInsurance_CA_01-01-19.xlsx, HSInsurance_CA_01-08-19.xlsx, HSInsurance_CA_01-16-19.xlsx від страхової фірми можна використати вираз "HSInsurance_.-" або "HSInsurance_CA_.-", якщо шаблон застосовується лише до даних штату Каліфорнія. Валідність цього виразу є ключовою для

коректного автоматичного вибору шаблону обробки. Вхідні файли, імена яких не відповідають жодному регулярному виразу, визначеному в шаблонах відповідного Облікового запису, будуть відхилені від подальшої обробки.

- Тип файлу (File Type): Вибір зі стандартизованого переліку підтримуваних форматів вхідного файлу (наприклад, файли, розділені комами - .csv; електронні таблиці Excel - .xlsx, .xls; або текстові файли з будь-яким розширенням). Цей параметр визначає парсер, який буде використано для початкової екстракції даних з файлу.

- Користувацька збірка (Custom Assembly): Надає можливість інтеграції користувацької логіки для етапу Трансформації (Transform) даних. Вказується шлях до скомпільованого програмного модуля (збірки), який має реалізувати визначений інтерфейс, наприклад, ITransformationProvider. Ця користувацька збірка буде автоматично виконана для кожного набору даних, що відповідає даному Шаблону файлу, дозволяючи застосовувати складні та специфічні для предметної області правила трансформації. Детальний опис розробки та інтеграції користувацьких збірок представлено в Розділі 4 (згідно з оригінальним текстом).

- Зразок файлу (File Template - Sample File): Поле, що вимагає надання прикладу вхідного файлу для даного Облікового запису та Шаблону. Система може використовувати цей зразок для попереднього аналізу структури файлу, візуалізації даних та полегшення процесу мапування атрибутів та визначення правил трансформації в інтерфейсі конфігурації.

3.2. Налаштування операцій трансформації даних

Цей ключовий етап процесу обробки даних структурований на дві взаємопов'язані складові: операції трансформації та умови їх застосування. Операція визначається як функціональний блок або алгоритм, призначений для перетворення значень у межах певного стовпця набору даних. Умова

являє собою логічний предикат або критерій, який накладає обмеження на значення стовпця, визначаючи, чи буде застосована асоційована операція. Наприклад, рисунок 3.3 ілюструє специфічну операцію заміни рядкових значень, де трансформація відбувається лише для рядків, що є еквівалентними константному значенню "FL".

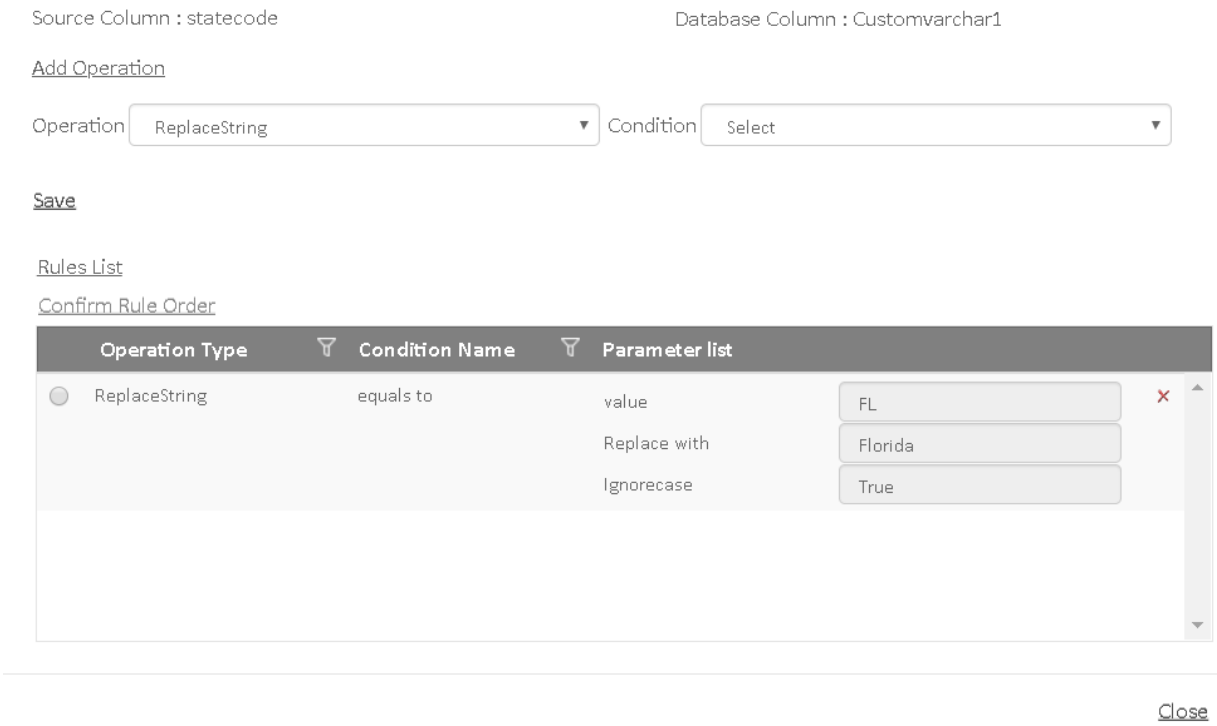


Рисунок 3.3 - Екран налаштування трансформації

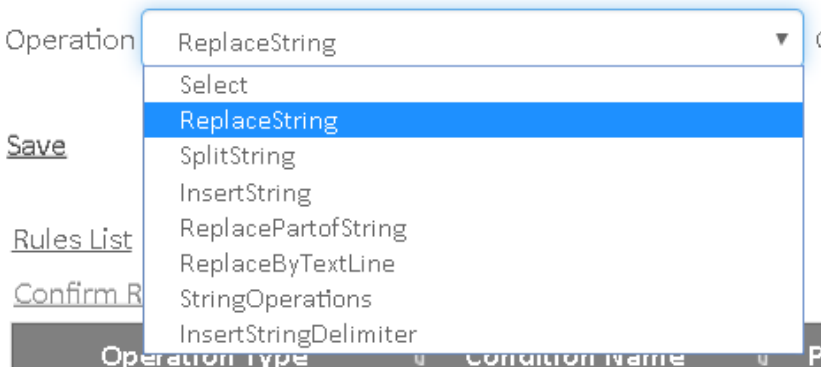


Рисунок 3.4 - Список операцій для рядкового типу

Рисунок 3.4 демонструє каталог доступних операцій, які можуть бути застосовані до стовпців з типом даних 'рядок', тоді як на рисунку 3.5 представлено перелік умов, релевантних для того ж типу даних.Комбінація однієї обраної операції та однієї умови формує одиничний цикл трансформації. До одного й того ж стовпця може бути послідовно застосовано множину таких циклів, що дозволяє реалізувати складніші ланцюжки перетворень.

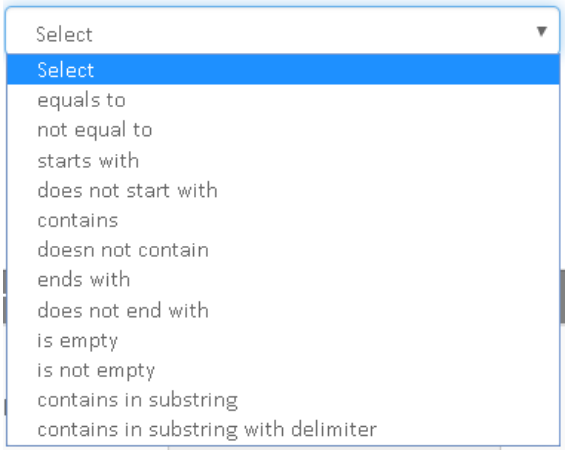


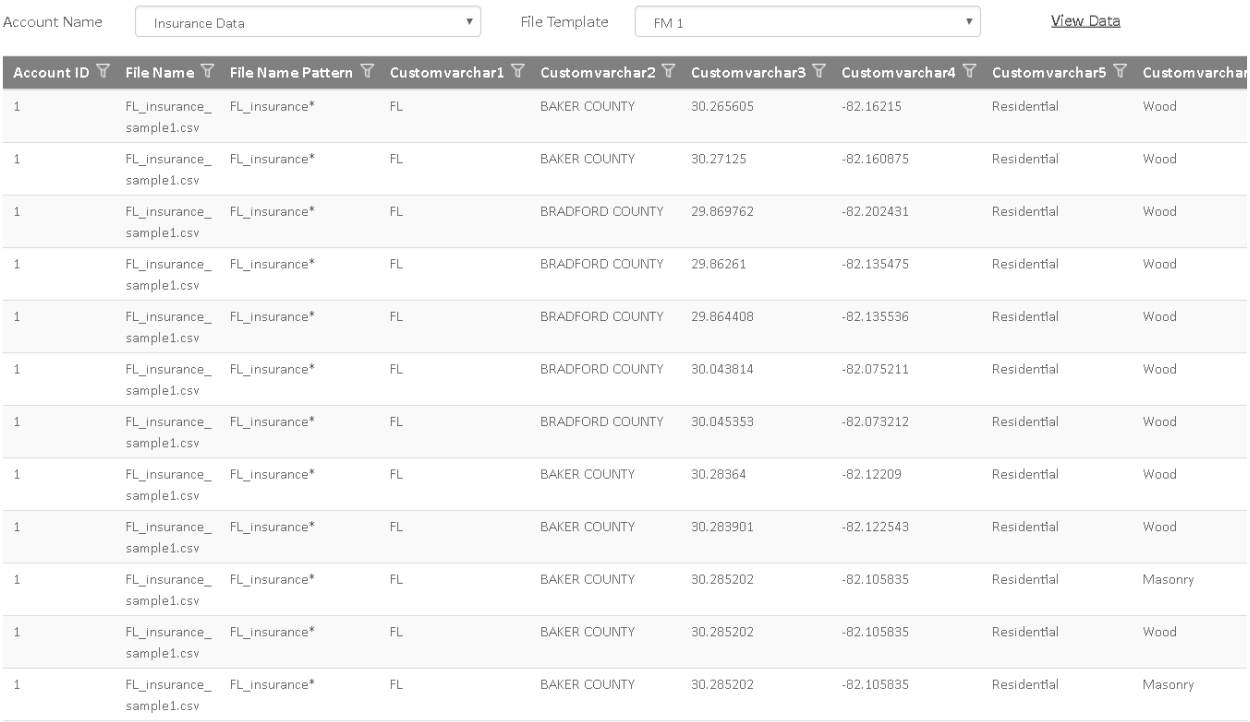
Рисунок 3.5 - Список умов для рядкового типу

Рисунок 3.6 наводить приклад комплексного циклу трансформації для стовпця типу 'рядок', який виконує заміну всіх входжень рядка "FL" на "Флорида", при цьому порівняння вихідного рядка здійснюється без урахування регістру символів для забезпечення гнучкості.

Рисунок 3.6 - Одна трансформація для стовпця рядкового типу

3.2.1. Візуалізація результуючого набору даних (Сітка даних)

Після успішного завершення етапів завантаження та трансформації даних здійснюється візуалізація результуючого набору даних у вигляді інтерактивної сітки даних. Ця сітка відображає фінальний стан даних, який було інтегровано до цільової бази даних.


Account Name Insurance Data File Template FM 1 View Data

Account ID	File Name	File Name Pattern	Customvarchar1	Customvarchar2	Customvarchar3	Customvarchar4	Customvarchar5	Customvarchar
1	FL_insurance_sample1.csv	FL_insurance*	FL	BAKER COUNTY	30.265605	-82.16215	Residential	Wood
1	FL_insurance_sample1.csv	FL_insurance*	FL	BAKER COUNTY	30.27125	-82.160875	Residential	Wood
1	FL_insurance_sample1.csv	FL_insurance*	FL	BRADFORD COUNTY	29.869762	-82.202431	Residential	Wood
1	FL_insurance_sample1.csv	FL_insurance*	FL	BRADFORD COUNTY	29.86261	-82.135475	Residential	Wood
1	FL_insurance_sample1.csv	FL_insurance*	FL	BRADFORD COUNTY	29.864408	-82.135536	Residential	Wood
1	FL_insurance_sample1.csv	FL_insurance*	FL	BRADFORD COUNTY	30.043814	-82.075211	Residential	Wood
1	FL_insurance_sample1.csv	FL_insurance*	FL	BRADFORD COUNTY	30.045353	-82.073212	Residential	Wood
1	FL_insurance_sample1.csv	FL_insurance*	FL	BAKER COUNTY	30.28364	-82.12209	Residential	Wood
1	FL_insurance_sample1.csv	FL_insurance*	FL	BAKER COUNTY	30.283901	-82.122543	Residential	Wood
1	FL_insurance_sample1.csv	FL_insurance*	FL	BAKER COUNTY	30.285202	-82.105835	Residential	Masonry
1	FL_insurance_sample1.csv	FL_insurance*	FL	BAKER COUNTY	30.285202	-82.105835	Residential	Wood
1	FL_insurance_sample1.csv	FL_insurance*	FL	BAKER COUNTY	30.285202	-82.105835	Residential	Masonry

Рисунок 3.7 - Сітка даних

На рисунку 3.7 представлено приклад такої сітки даних, конфігурованої для облікового запису "Дані страхування" та асоційованої з шаблоном "FM 1". Важливою технічною характеристикою цієї сітки є її режим функціонування тільки для читання (read-only), що означає неможливість безпосереднього редагування або модифікації даних через цей інтерфейс.

3.2.2 Система журналювання подій

В архітектурі даного автоматизованого інструменту процеси реєстрації та документування подій є критично важливими для забезпечення прозорості виконання та ефективної діагностики. Система журналювання (logging) надає

деталізоване розуміння внутрішнього потоку виконання додатку, послідовності операцій та дозволяє оперативно виявляти й локалізувати джерела будь-яких помилок або відмов. Інструмент реалізує механізм журналювання у двох основних функціональних розділах:

1. Журналювання процесу імпорту файлів.

Цей розділ системи журналювання відповідає за фіксацію ключових етапів виконання процедури імпорту вхідних файлів. Кожен запис містить точну часову мітку, документуючи прогрес процесу від його початку до фінального статусу, що може бути або успішне завершення імпорту даних, або відхилення файлу з реєстрацією специфічної причини помилки.

2. Журналювання активності користувача.

Цей розділ присвячений документуванню всіх інтерактивних дій, виконаних користувачем через графічний інтерфейс програми (User Interface - UI). Тут фіксуються такі операції, як створення, модифікація або видалення правил зіставлення атрибутів (attribute mapping), а також конфігурування, оновлення чи видалення визначених циклів трансформації даних. Це забезпечує можливість трасування дій користувача та аудиту змін у налаштуваннях обробки даних.

3.3. Архітектура розгортання системи

Відповідно до представленої діаграми розгортання системи (рисунок 3.8), функціонування даного програмного комплексу передбачає розгортання на мінімальній інфраструктурі, яка включає веб-сервер (web server) та сервер бази даних (database server). Доступ кінцевих користувачів до функціоналу системи здійснюється через стандартний веб-браузер, що функціонує на будь-якому мережевому пристрої з підключенням до глобальної мережі Інтернет. З'єднання між клієнтським веб-браузером та веб-сервером

встановлюється за допомогою стеку протоколів TCP/IP, використовуючи уніфікований покажчик ресурсу (URL) для ідентифікації сервісу.

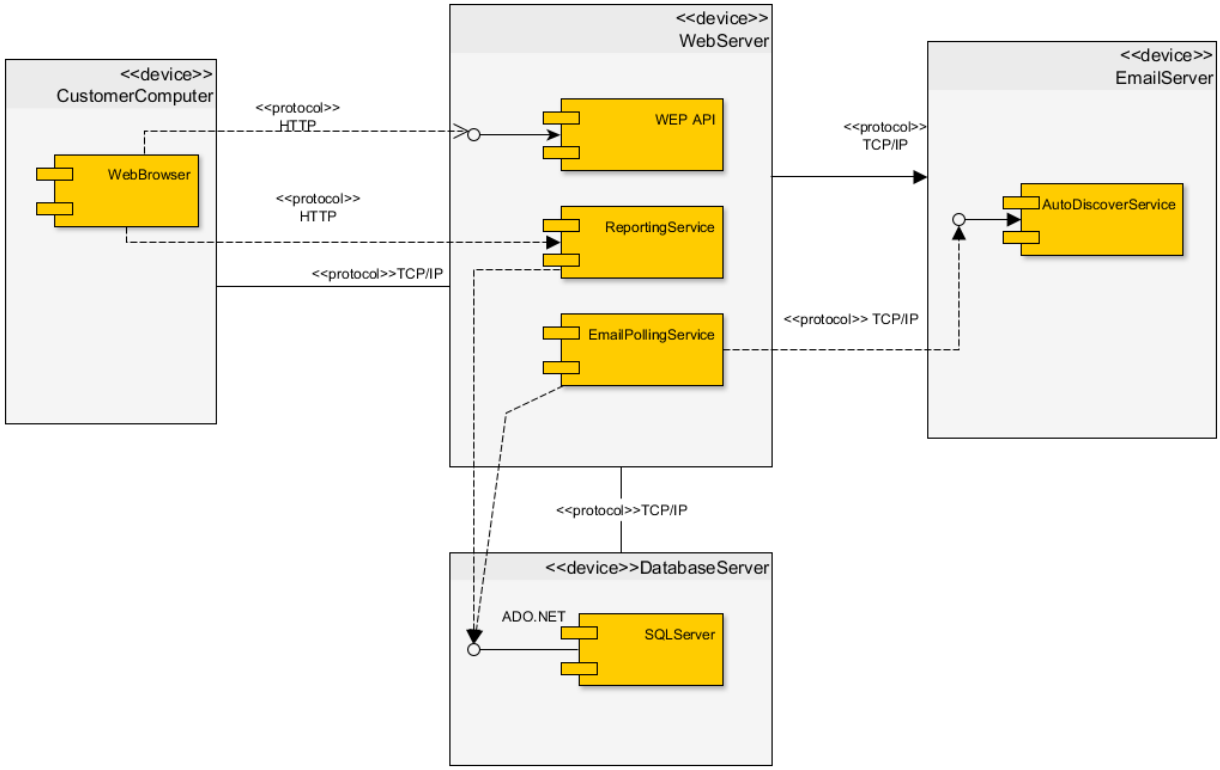


Рисунок 3.8 – Діаграма розгортання

На рівні веб-сервера реалізовано кілька ключових компонентів, що забезпечують специфічну функціональність системи:

- EmailPollingService.

Цей компонент реалізовано як фонову службу Windows (Windows Service). Його основним призначенням є асинхронне та безперервне опитування (polling) визначеної електронної поштової скриньки. Служба відповідає за автоматизоване отримання вхідних повідомлень електронної пошти, вилучення прикріплених файлів та ініціацію їх первинної обробки. Для взаємодії з поштовим сервером EmailPollingService використовує спеціалізований програмний інтерфейс, зокрема, Exchange Web Services (EWS) Managed API, що дозволяє програмний доступ до даних та функцій сервера Microsoft Exchange.

- Компонент Web API.

Цей компонент представляє програмний інтерфейс (API - Application Programming Interface), який надає набір сервісів (services) або кінцевих точок (endpoints) для взаємодії з інтерфейсом користувача (User Interface - UI), що виконується на стороні клієнтського веб-браузера. Web API виступає як шар доступу до серверної бізнес-логіки та даних, дозволяючи клієнтській частині здійснювати запити, отримувати дані та викликати необхідні функції системи через стандартизовані протоколи (наприклад, HTTP/HTTPS).

- Компонент Reporting Service.

Цей компонент відповідає за генерацію та форматування звітів на основі обробленого набору даних, що зберігається у базі даних. Reporting Service здійснює агрегацію, трансформацію та візуалізацію даних згідно з попередньо визначеними шаблонами або конфігурацією користувача. Додатково, він забезпечує функціонал експорту згенерованих звітів у різні формати файлів, вибір яких визначається користувачем (наприклад, PDF, XLSX тощо).

Сервер бази даних слугує централізованим сховищем для всіх даних системи – як сирих (до обробки), так і оброблених, а також конфігураційних параметрів та журналів. Взаємодія між веб-сервером та сервером бази даних здійснюється за допомогою відповідних драйверів та протоколів доступу до даних (наприклад, SQL-протокол). Така архітектура розгортання забезпечує розподіл функціональних обов'язків між компонентами системи та масштабованість рішень.

3.4. Реалізація компоненту сервісу імпорту даних

Компонент Сервісу імпорту даних (Data Import Service) реалізовано як службу Windows (Windows Service). Це визначає його як фоновий програмний процес, здатний виконуватися незалежно у власній сесії

операційної системи Windows, без необхідності активного сеансу користувача. Така архітектура дозволяє сервісу функціонувати постійно у фоновому режимі, забезпечуючи безперервне моніторування та обробку вхідних даних. Для забезпечення програмної взаємодії з функціоналом електронної пошти, зокрема з поштовою скринькою Microsoft Exchange, даний сервіс використовує Exchange Web Services (EWS) Managed API. EWS Managed API є керованою бібліотекою класів, що надає розробникам зручний інтерфейс для доступу до сервісів Exchange.

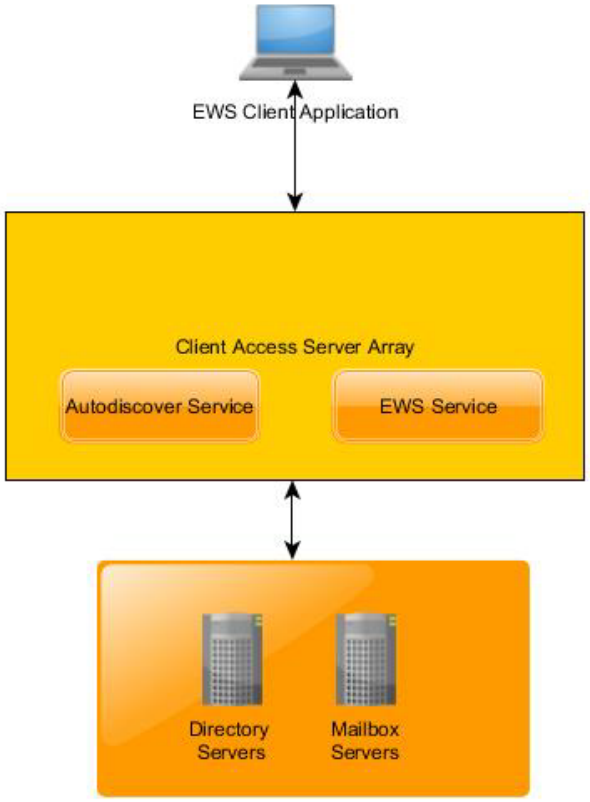


Рисунок 3.9 – Налаштування клієнта Exchange Web Service (EWS)

Як видно з рисунку 3.9, сервіс імпорту даних виступає в ролі клієнтського додатку EWS, ініціюючи запити до сервера Exchange для отримання даних (наприклад, електронних листів з вкладеннями).

Діаграма показує шарувату структуру та ключові компоненти, задіяні в процесі доступу EWS-клієнта до даних поштових скриньок:

1. EWS Client Application (Клієнтський додаток EWS): Це програмне забезпечення (наприклад, Служба імпорту даних), яке потребує доступу до даних користувача в поштових скриньках Exchange (електронні листи, календар, контакти тощо). Клієнтський додаток є ініціатором зв'язку.

2. Client Access Server Array (Масив серверів клієнтського доступу): Це проміжний шар або шлюз, через який клієнтські додатки отримують доступ до Exchange. CAS Array виконує функції автентифікації, авторизації, балансування навантаження та маршрутизації запитів до відповідних серверів поштових скриньок. Цей компонент містить дві ключові служби, важливі для EWS-клієнтів:

- Autodiscover Service (Служба автовиявлення): Як обговорювалося раніше, ця служба дозволяє клієнтському додатку автоматично знайти правильні налаштування для підключення до поштової скриньки користувача, включаючи URL-адресу кінцевої точки EWS, лише на основі електронної адреси користувача. Це значно спрощує процес налаштування клієнта.

- EWS Service (Сервіс EWS): Це безпосередньо веб-сервіс, який надає програмний інтерфейс (API) для виконання операцій з даними поштової скриньки (читання/надсилання листів, керування елементами календаря тощо). Клієнтський додаток взаємодіє з цим сервісом після того, як отримав його адресу через Autodiscover.

3. Directory Servers (Сервери каталогів): Це сервери Active Directory Domain Services. Вони використовуються CAS Array (та іншими компонентами Exchange) для автентифікації користувачів, пошуку інформації про користувачів та їхні поштові скриньки.

4. Mailbox Servers (Сервери поштових скриньок): Ці сервери безпосередньо зберігають дані поштових скриньок користувачів. CAS Array маршрутизує запити, отримані від EWS-клієнтів, до відповідного сервера поштових скриньок для виконання операцій над даними.

Процес налаштування та підключення клієнта EWS виглядає так:

- Крок 1 (Автовиявлення): EWS Client Application ініціює запит до Autodiscover Service (як правило, використовуючи електронну адресу користувача). Метою є автоматичне отримання URL сервісу EWS та інших необхідних налаштувань.

- Крок 2 (Отримання налаштувань): Autodiscover Service взаємодіє з Directory Servers для пошуку інформації про користувача та його поштову скриньку, а потім повертає EWS Client Application необхідні дані для підключення до сервісу EWS.

- Крок 3 (Підключення до EWS Service): Використовуючи отриману від Autodiscover URL-адресу, EWS Client Application встановлює з'єднання безпосередньо з EWS Service на Client Access Server Array.

- Крок 4 (Взаємодія з даними): Після автентифікації (яка може включати взаємодію CAS Array з Directory Servers), EWS Service на CAS Array отримує запити від клієнта (наприклад, "завантажити вкладення з листа"). CAS Array маршрутизує ці запити до відповідного Mailbox Server, де фізично зберігаються дані поштової скриньки користувача.

- Крок 5 (Отримання результатів): Mailbox Server виконує запит, отримує дані та повертає їх через CAS Array до EWS Client Application.

3.5. Механізм автовиявлення (Autodiscover service)

Механізм автовиявлення (Autodiscover service) є критично важливим компонентом архітектури Exchange, що відповідає за автоматичне визначення параметрів підключення клієнтського додатку до поштової скриньки користувача. Його ключова функція полягає у знаходженні оптимальної кінцевої точки (endpoint) сервісу Exchange (включаючи URL для EWS) на основі електронної адреси користувача, що значно спрощує конфігурацію клієнтських застосунків, усуваючи необхідність ручного

введення серверних параметрів. Exchange Web Services (EWS) Managed API надає клієнтським додаткам програмну реалізацію цього механізму, інкапсулюючи складність процесу пошуку. Як ілюструє рисунок 3.9, API містить метод AutodiscoverUrl.

Представлений нижче фрагмент коду демонструє сигнатуру методу AutodiscoverUrl з бібліотеки Exchange Web Services (EWS) Managed API:

```
// Summary:
// Initializes the Url property to the Exchange Web Services URL for the specified
// e-mail address by calling the Autodiscover service.
// Parameters:
// emailAddress:
// The email address to use.
// validateRedirectionUrlCallback:
// The callback used to validate redirection URL.
public void AutodiscoverUrl(string emailAddress, AutodiscoverRedirectionUrlValidat:
```

Рисунок 3.10 - Сигнатура методу Autodiscover

3.6. Механізм підписки на події електронної пошти через EWS

Exchange Web Services (EWS) Managed API [5] надає програмний інтерфейс для реалізації механізму підписки на події (event subscription), що виникають у поштових скриньках користувачів, таких як надходження нового повідомлення. Цей механізм дозволяє клієнтським додаткам отримувати сповіщення про зміни без необхідності постійного опитування сервера (polling). В архітектурі Exchange Web Services реалізовано три основні моделі доставки сповіщень:

- Pull-сповіщення (Pull Notifications). У цій моделі клієнтський додаток періодично надсилає запити до сервера для отримання накопичених сповіщень. Сервер зберігає їх у черзі до моменту запиту клієнта. Цей підхід характеризується простотою реалізації на стороні сервера, але може спричиняти значні затримки у доставці сповіщень та генерувати додаткове навантаження на мережу та сервери через часті запити.

- Push-сповіщення (Push Notifications). У цій моделі сервер виступає ініціатором зв'язку, надсилаючи сповіщення клієнту одразу після виникнення події. Доставка реалізується шляхом виконання HTTP POST-запиту до заздалегідь сконфігурованої URL-адреси на стороні клієнта (функція зворотного виклику). Ця модель забезпечує низьку затримку, але вимагає, щоб клієнтський додаток був доступним з мережі, де знаходиться сервер Exchange.

- Streaming-сповіщення (Streaming Notifications). Ця модель передбачає встановлення та підтримання постійного з'єднання (стриму) між клієнтським додатком та сервером Exchange (як правило, на базі протоколу TCP/IP). Сервер надсилає сповіщення клієнту через це активне з'єднання майже в реальному часі. З'єднання може бути припинене сервером після певного періоду неактивності або через інші мережеві події, вимагаючи від клієнта його поновлення. Ця модель забезпечує баланс між оперативністю Push-сповіщень та простотою розгортання клієнта порівняно з Push-моделлю, оскільки не потребує відкритого для вхідних з'єднань порту на клієнті.

Даний програмний інструмент використовує модель streaming-сповіщень для оперативного отримання нотифікацій про надходження нових електронних листів. Це дозволяє системі швидко реагувати на появу вхідних даних для обробки. Підпис методу з EWS Managed API, що використовується для конфігурації підписки на streaming-сповіщення, показано на рисунку 3.11.

```
// Summary:
// Subscribes to streaming notifications. Calling this method results in a call
// to EWS.
// Parameters:
// folderIds:
// The Ids of the folder to subscribe to.
// eventTypes:
// The event types to subscribe to.
// Returns:
// A StreamingSubscription representing the new subscription.
public StreamingSubscription SubscriptionToStreamingNotifications(IEnumerable<Folde
```

Рисунок 3.11 - Сигнатура методу потокових сповіщень

Згідно з цією сигнатурою, метод SubscribeToStreamingNotifications приймає колекцію ідентифікаторів папок (folderIds), за якими здійснюється моніторинг подій, та масив типів подій (eventTypes) (наприклад, EventType.NewMail, EventType.Created, EventType.Modified). Виклик цього методу призводить до взаємодії з сервісом EWS на сервері Exchange, результатом якої є створення підписки. Метод повертає об'єкт типу StreamingSubscription, який репрезентує встановлену активну підписку, що може бути використана для керування життєвим циклом потоку сповіщень.

3.7. Модуль трансформації даних користувача

Даний програмний інструмент реалізує механізм розширення функціональності трансформації даних шляхом інтеграції користувацьких збірок (custom assemblies). Користувацька збірка – це файл компільованого коду (зазвичай у форматі .NET DLL), що містить спеціалізовану логіку перетворення даних, написану користувачем або розробником. Ця функціональність дозволяє адаптувати процеси трансформації до специфічних вимог, які не можуть бути задоволені стандартними, вбудованими операціями трансформації. Рисунок 3.2 ілюструє користувацький інтерфейс для конфігурації шаблону обробки файлу, який включає опцію для завантаження файлу такої користувацької збірки.

Для забезпечення коректної інтеграції та можливості динамічного виклику з ядра інструменту, користувацька збірка повинна містити клас, що реалізує заздалегідь визначений програмний інтерфейс. Цей інтерфейс виступає як контракт, стандартизуючи спосіб взаємодії між ядром системи та користувацьким модулем трансформації. Структура необхідного інтерфейсу представлена на рисунку 3.12 та у фрагменті коду нижче. Інтерфейс має назву ITransformProvider. Він містить єдиний метод Run, який визначає вхідні та вихідні параметри для користувацької логіки трансформації.

```
using System.Data; // Необхідний для використання типу DataTable
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ETLAutomation.ExtensionMethods
{
    /// <summary>
    /// Implement interface to add custom transformation
    /// tasks
    /// </summary>
    public interface ITransformProvider
    {
        // Summary:
        // Runs the custom transformation logic.
        // Parameters:
        // _ds:
        // The input DataTable containing the dataset to transform.
        // Returns:
        // A DataTable object representing the transformed dataset.
        DataTable Run(DataTable _ds);
    }
}
```

Рисунок 3.12 - Наданий інтерфейс

Метод Run приймає на вхід один параметр типу DataTable, що представляє набір даних (у пам'яті), отриманий після початкового завантаження або попередніх етапів трансформації. Клас DataTable є стандартною структурою даних в .NET, що використовується для маніпуляцій з табличними даними. Очікується, що реалізація методу Run виконає необхідні операції перетворення над цим вхідним об'єктом DataTable і поверне новий об'єкт типу DataTable, що містить результат трансформації. Таким чином, цей інтерфейс забезпечує стандартизований програмний інтерфейс (API) для інтеграції користувацьких алгоритмів обробки даних у загальний конвеєр трансформації інструменту.

3.8. Аналіз структури документа

У сучасних бізнес-процесах формат PDF широко використовується для обміну даними. Однак, автоматизований аналіз текстового вмісту PDF-файлів є складним через відсутність стандартизованої семантичної структури

та визначених тегів, які б описували тип даних або їх логічні зв'язки. Ця особливість ускладнює автоматизоване вилучення конкретних полів даних або таблиць.

Для подолання цих обмежень та забезпечення можливості автоматизованої обробки документів застосовується аналіз їхньої візуальної структури (макету). Аналіз макету дозволяє ідентифікувати окремі текстові блоки, заголовки, абзаци, таблиці та зображення, а також визначити логічний порядок читання. Інформація про макет може бути збережена та використана для кожного конкретного типу або шаблону документа, що дозволяє адаптувати процес аналізу та підвищити точність вилучення даних.

У випадках, коли PDF-файл містить текст у вигляді зображень (наприклад, скановані документи), необхідним етапом є оптичне розпізнавання символів (OCR). Технологія OCR дозволяє конвертувати зображення тексту у машиночитний текстовий формат. Вилучений за допомогою OCR текст у подальшому може бути переданий на етап аналізу макету для структуризації.

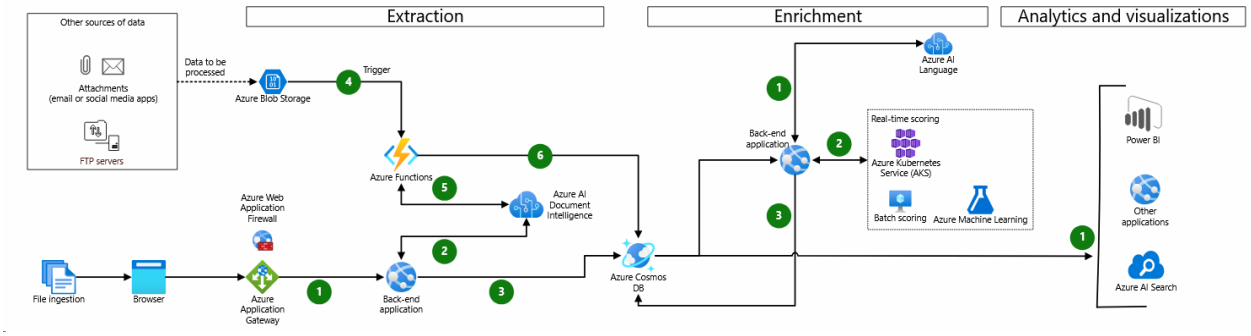


Рисунок 3.13 - Архітектура рішення для обробки та аналізу даних на основі Microsoft Azure

Рисунок 3.13 ілюструє архітектуру рішення для обробки та аналізу даних, побудовану на базі сервісів Microsoft Azure. Архітектура складається з трьох основних етапів:

1. Етап вилучення (Extraction):

- Дані надходять з різних джерел (файли через браузер, вкладення з пошти/соцмереж, FTP сервери).

- Для файлів, завантажених через браузер, вхідний трафік проходить через Application Gateway та Web Application Firewall до Back-end application.

- Інші джерела даних (вкладення, FTP) можуть надходити в Azure Blob Storage.

- Тригери (Trigger) ініціюють Azure Functions або іншу обробку даних з Blob Storage.

- Azure Functions або Back-end application взаємодіють з Azure AI Document Intelligence для інтелектуального вилучення інформації з документів.

- Вилучені дані зберігаються в базі даних Azure Cosmos DB.

2. Етап збагачення (Enrichment):

- Back-end application отримує дані з Cosmos DB.

- Дані можуть надсилатися в Azure AI Language для обробки природною мовою (наприклад, аналіз тональності, вилучення сутностей).

- Для обробки даних можуть використовуватися сервіси, що працюють у Azure Kubernetes Service (AKS) для Real-time scoring (оцінки в реальному часі) або Azure Machine Learning для Batch scoring (пакетної оцінки).

- Результати збагачення також зберігаються в Azure Cosmos DB або можуть передаватися на наступний етап.

3. Етап аналітики та візуалізації (Analytics and visualizations):

- Збагачені дані з Cosmos DB або безпосередньо після вилучення можуть бути доступні для аналітики.

- Використовується Azure AI Search для забезпечення можливості пошуку у даних.

- Візуалізація та аналітика здійснюється за допомогою Power BI або може бути інтегрована в інші застосунки.

Дана архітектура представляє конвеєр обробки даних, що охоплює прийом даних з різних джерел, їх інтелектуальне вилучення, збагачення за допомогою сервісів штучного інтелекту та машинного навчання, зберігання, пошук та візуалізацію.

Інтеграція у інструмент бібліотек оптичного розпізнавання символів з відкритим кодом, таких як Tesseract або інших, дозволить значно розширити діапазон його застосування, включивши обробку сканованих документів та PDF-файлів, що містять інкапсульовані зображення тексту. Tesseract являє собою високоефективний програмний рушій для оптичного розпізнавання символів (OCR), що функціонує як аналітичний інструмент для автоматизованого перетворення зображень тексту у машиночитний текстовий формат.

Архітектура Tesseract реалізує багатоетапний процес, що включає сегментацію зображення на окремі ділянки, ідентифікацію символів (гліфів) та аналіз просторового розташування елементів для реконструкції логічної структури документа (аналіз макету). Рушій підтримує розпізнавання тексту численними мовами завдяки використанню тренованих моделей даних.

Основне призначення Tesseract полягає в обробці зображень, що містять текстову інформацію, таких як скановані документи, фотографії тексту, а також текст, інкапсульований у графічному форматі всередині цифрових документів (наприклад, у PDF-файлах, що є зображеннями).

Завдяки наявності програмних інтерфейсів (API) для багатьох мов програмування, Tesseract легко інтегрується у ширші системи автоматизації обробки документів, системи архівування, пошукові платформи та наукові застосунки, де вимагається вилучення текстової інформації з неструктурованих або напівструктурованих візуальних джерел. Його застосування сприяє підвищенню ефективності обробки великих обсягів документації та автоматизації процесів введення даних.

Такий підхід суттєво підвищить практичну користь інструменту для автоматизації робочих процесів, пов'язаних з маніпуляцією даними з різноманітних джерел. Таким чином, компонент аналізу структури документа, поєднуючи можливості роботи з нативним текстом PDF та результатами OCR, забезпечує трансформацію неструктурованого візуального представлення документа у структуровані дані, придатні для подальшої автоматизованої обробки.

3.9. Модульне тестування

Розроблений інструмент призначений для автоматизації процесу ручної маніпуляції даними. Проектування всіх класів здійснювалося із застосуванням методології тест-орієнтованого розроблення (Test-Driven Development, TDD). Після завершення етапу розробки було проведено системне тестування інструменту. Системне тестування виконувалося вручну шляхом порівняння фактичних результатів з очікуваними.

Протягом процесу розробки було реалізовано модульне тестування. Тестові випадки для окремих методів розроблялися на основі їх специфікацій. Після кожного внесення змін до коду методу виконувався запуск відповідних тестових випадків. Додаткові тестові випадки додавалися за потреби для забезпечення повного покриття та перевірки поведінки. Рефакторинг коду здійснювався після успішного проходження всіх тестів з метою дотримання прийнятних стандартів кодування.

Головною метою модульного тестування є виявлення дефектів на ранніх етапах життєвого циклу розробки програмного забезпечення, що дозволяє зменшити загальні витрати на виправлення помилок. Ефективне модульне тестування сприяє досягненню високого рівня надійності програмного продукту та полегшує процес рефакторингу, оскільки дає змогу

швидко перевірити, чи не порушено цілісність існуючої функціональності після внесення змін.

Згідно з принципами тестування, модульне тестування є одним із найнижчих рівнів у ієрархії тестування ПЗ, однак водночас воно відіграє ключову роль у загальній стратегії забезпечення якості. Результати модульного тестування слугують основою для інтеграційного, системного та приймального тестування.

```
[Test]
public void convertToBoolean_StateUnderTest_ExpectedBehavior()
{
    // Arrange
    var unitUnderTest = this.CreateUtility();
    string s = "2";

    // Act
    var result = unitUnderTest.convertToBoolean(s);

    // Assert
    Assert.IsFalse(result);
}

[Test]
public void GetExcelStrings_StateUnderTest_ExpectedBehavior()
{
    // Arrange
    var unitUnderTest = this.CreateUtility();

    // Act
    var result = unitUnderTest.GetExcelStrings();

    // Assert
    Assert.IsNotEmpty(result);
}
```

Рисунок 3.13 - Зразок модульного тесту

Для виконання модульного тестування застосовувалася бібліотека NUnit. Система NUnit забезпечує генерацію базових тестових випадків для кожного класу. Додаткові (розширені) тестові випадки для детальнішої перевірки функціональності методів розроблялися вручну. Приклади позитивних тестових випадків для методів ConvertToBoolean() та

GetExcelStrings() наведено на рисунку 3.13. Включення нових тестових випадків здійснювалося у відповідь на зміни у функціональній поведінці. Вікно Test Explorer надає інформацію про статус виконання всіх тестових випадків для кожного класу. На рисунку 3.14 продемонстровано частину тестових випадків, що належать до класу Utility.

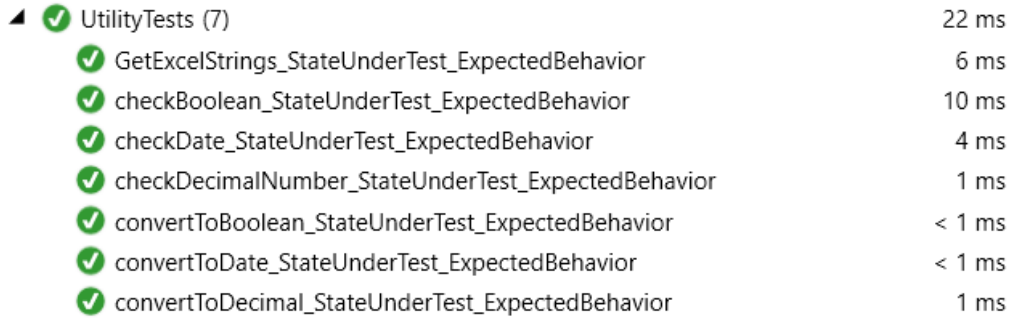


Рисунок 3.14 - Приклад відображення статусу модульних тестів

Отже, даний інструмент призначений для мінімізації помилок у даних та зниження потреби в ручному втручанні в процеси їх обробки. Система забезпечує обробку великих обсягів електронних листів з вкладеннями довільного розміру. Обробка виконується асинхронно фоновим процесом. Моніторинг статусу обробки та інформування користувача здійснюється за допомогою системи ведення журналів (логів) та механізму електронних сповіщень. Результати обробки даних можуть бути використані для генерації аналітичних звітів або експортовані у різні формати для подальшого використання.

Розробка проекту здійснювалася із застосуванням методології тест-орієнтованого розроблення (Test-Driven Development, TDD). Кожна функціональна одиниця проекту забезпечена комплектом модульних тестових випадків, що охоплюють сценарії для перевірки коректної роботи (позитивні випадки), обробки помилок (негативні випадки) та поведінки на граничних умовах (крайні випадки).

Ручне тестування інструменту проводилося з використанням різноманітних наборів даних, типів файлів, типів даних та охоплювало різні сценарії трансформації.

Висновки до третього розділу

У третьому розділі реалізовано повноцінну програмну систему, що втілює раніше розроблену концепцію. Здійснено проєктування макетів інтерфейсу користувача, включно з можливістю створення облікових записів, налаштування шаблонів та операцій трансформації даних. Передбачено зручний механізм візуалізації результатів у вигляді сітки даних, а також систему журналювання подій для забезпечення трасування та відстеження критичних операцій.

Важливою інноваційною складовою стала імплементація механізмів автовиявлення сервера (Autodiscover), підписки на події пошти через EWS, а також модуля трансформації користувацьких даних, що підвищує рівень автономності системи. Проведено модульне тестування, результати якого свідчать про стабільну роботу системи за різних сценаріїв обробки листів.

Таким чином, реалізоване рішення відповідає сучасним вимогам до надійності, масштабованості та зручності в інтеграції з корпоративними середовищами, забезпечуючи ефективну автоматизацію повторюваних задач обробки вхідних повідомлень.

ВИСНОВКИ

У результаті виконання дипломної роботи, присвяченої автоматизації процесів обробки даних з електронних листів клієнтів, було здійснено дослідження предметної області, обґрунтовано вибір методологічних підходів та реалізовано програмне рішення, здатне значно підвищити ефективність і точність обробки клієнтських запитів, що надходять електронною поштою.

На етапі теоретичного аналізу було детально вивчено принципи обробки структурованих даних з електронних повідомлень, а також виявлено ключові виклики, пов’язані з автоматичним вилученням, уніфікацією та збереженням інформації. Особливу увагу було приділено методології Extract, Transform and Load (ETL), яка довела свою ефективність як базова концепція для побудови системи інтеграції даних. Розроблена архітектура автоматизованої системи передбачає багаторівневу обробку повідомлень з урахуванням специфіки електронного документообігу та динамічної зміни форматів вхідних даних.

У процесі реалізації було інтегровано низку сучасних технологій, зокрема Exchange Web Services (EWS API) для доступу до вмісту поштових скриньок, ASP.NET Web API для побудови серверної логіки, AngularJS для реалізації клієнтської частини інтерфейсу, а також Microsoft SQL Server як засобу зберігання та керування даними. Архітектура системи є модульною, масштабованою та придатною до розширення, що забезпечує її адаптивність до змін бізнес-процесів.

Було розроблено повний цикл обробки даних: від виявлення та отримання вхідних електронних листів — до парсингу, трансформації, зіставлення атрибутів та збереження результатів у структурованому вигляді. Проектні діаграми (використання, діяльності, послідовності) відображають

логіку функціонування системи, її основні сценарії взаємодії з користувачем і внутрішні компоненти.

Програмна імплементація ETL-механізмів передбачає підтримку гнучких шаблонів трансформації, інтерактивне налаштування операцій обробки, візуалізацію результатів у вигляді сітки даних, а також механізми журналювання та налагодження. Важливою складовою стала реалізація сервісу автоконфігурації (Autodiscover) та підписки на події поштової системи, що дає змогу системі діяти в режимі реального часу.

На завершальному етапі було проведено модульне тестування, яке засвідчило високу стабільність окремих компонентів, а також відповідність розробленого функціоналу вимогам системного проектування. Результати тестування підтвердили, що система здатна ефективно обробляти вхідні повідомлення з мінімальною участю оператора, забезпечуючи як швидкість, так і достовірність обробки.

Узагальнюючи, можна стверджувати, що розроблена система автоматизованої обробки даних з електронних листів клієнтів становить приклад успішного поєднання класичних принципів інженерії даних та сучасних веб-технологій, дозволяючи значно зменшити час на обробку інформації, знизити ризики помилок ручного вводу та підвищити рівень клієнтського сервісу. Отримані результати можуть бути адаптовані для впровадження в різноманітних галузях бізнесу, де електронна пошта є ключовим каналом взаємодії з клієнтами.

ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Aberdeen, J., et al. (2007). The MITRE Identity and Relationship Extraction System. In Proceedings of the Human Language Technology Conference of the NAACL, Companion Volume: Demonstrations.
2. Alani, H., et al. (2003). Mining customer emails for survey completion. In Proceedings of the fifth international workshop on Web information and data management.
3. Allamanis, M., Sutton, C., & Brockschmidt, M. (2018). Learning to Mine Software Repositories with Graph Neural Networks. arXiv preprint arXiv:1810.00835.
4. Bellifemine, F., et al. (2003). JADE—a third generation FIPA-compliant agent framework. Information Systems Frontiers, 5(1), 7-13.
5. Bengio, Y., Goodfellow, I., & Courville, A. (2016). Deep Learning. MIT Press.
6. Bhardwaj, S., et al. (2010). Cloud computing: From scientific computing to cloud computing. Journal of Grid Computing, 8(2), 149-163.
7. Breuel, T. M. (2003). Automatic layout analysis and OCR for scanned documents. Cambridge University Press.
8. Brown, T. B., et al. (2020). Language Models are Few-Shot Learners. arXiv preprint arXiv:2005.14165.
9. Cao, L. (2017). Data science and analytics: Accelerated data-driven learning. Springer.
10. Cernila, J., et al. (2009). Information Extraction from Unstructured Documents. In Emerging Artificial Intelligence Applications in Computer Engineering. IOS Press.
11. Dean, J., & Ghemawat, S. (2008). MapReduce: Simplified data processing on large clusters. Communications of the ACM, 51(1), 107-113.

12. Duda, R. O., Hart, P. E., & Stork, D. G. (2001). Pattern classification. John Wiley & Sons.
13. Fagin, R., et al. (1995). Querying databases that contain sets. ACM SIGACT News, 26(3), 27-33.
14. Furqan, A., et al. (2018). Document Layout Analysis: A Comprehensive Survey. IEEE Access, 6, 58628-58656.
15. Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley.
16. Gershkovich, M., et al. (2005). Information Extraction from Emails. In Proceedings of the Workshop on Information Extraction from Automatically Acquired Web Pages.
17. Gitman, I. (2011). Communication Networks: An Interconnected Protocol Approach. Elsevier.
18. Gomez-Perez, A., Fernandez-Lopez, M., & Corcho, O. (2004). Ontological Engineering. Springer.
19. Hand, D., Mannila, H., & Smyth, P. (2001). Principles of data mining. MIT Press.
20. Hull, J. J. (1998). Document image layout analysis: A review. Document analysis and recognition, 205-228.
21. Jurafsky, D., & Martin, J. H. (2009). Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition. Prentice Hall.
22. Kleppmann, M. (2017). Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems. O'Reilly Media.
23. Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems, 25.
24. LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. Nature, 521(7553), 436-444.

25. Levin, A., et al. (2007). Learning to extract information from email. International Journal of E-Business Research (IJEBR), 3(1), 50-69.
26. Liu, B. (2012). Sentiment Analysis and Opinion Mining. Morgan & Claypool Publishers.
27. Long, J., Shelhamer, E., & Darrell, T. (2015). Fully convolutional networks for semantic segmentation. IEEE transactions on pattern analysis and machine intelligence, 38(11), 2152-2162.
28. Lowry, P. B., et al. (2013). Automatic semantic data extraction from unstructured text. Journal of Organizational and End User Computing (JOEUC), 25(2), 73-100.
29. Mahmoud, S., et al. (2013). A survey on testing techniques for service-oriented architecture. Journal of Systems and Software, 86(1), 128-150.
30. Mesbah, A., et al. (2012). RESTful Web Services: A State of the Art. ACM Transactions on Internet Technology (TOIT), 12(1), 1-34.
31. Ray, S. (2007). Tesseract OCR Engine. Retrieved from https://github.com/tesseract-ocr/tesseract. Riehle, D. (2000). Test-Driven Development. C++ Report, 12(10), 47-51. (Early paper on TDD).
32. Ries, E. (2011). The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses. Crown Business.
33. Smithies, J. (1998). The development of the Tesseract OCR system. HP Laboratories Technical Report.
34. Sommerville, I. (2015). Software Engineering (10th ed.). Pearson. (Comprehensive textbook covering testing, development methodologies).
35. Wiegers, K. (2002). Peer Reviews in Software: A Practical Guide. Addison-Wesley.
36. Xu, Y., et al. (1999). Document analysis and recognition: A survey. IEEE Transactions on Pattern Analysis and Machine Intelligence, 21(7), 600-624.

ДОДАТКИ

Додаток А

Основні розроблені класи

Клас Email Service

```
internal partial class CronService : ServiceBase
{
    private const string ConstMailInterval = "MailInterval";
    static readonly ILog Logger = LogManager.GetLogger(typeof(CronService));
    private Mailhandler _mailhandler;
    private Timer _timer;

    public CronService()
    {
        log4net.Config.XmlConfigurator.Configure();
        InitializeComponent();
    }

    protected override void OnStart(string[] args)
    {
        try
        {
            EventLog.WriteEntry("Service Starting", EventLogEntryType.Information);
            Logger.InfoFormat("Start Method: {0}", System.Reflection.MethodBase.GetCurrentMetho
            System.Threading.Thread.Sleep(25000);
            _mailhandler = new Mailhandler();
            ConfigurationHelper configHelper = new ConfigurationHelper();
            try
            {
                string strInterval = ConfigurationManager.AppSettings[ConstMailInterval];
                double interval = TimeSpan.FromMinutes(Convert.ToDouble(strInterval)).TotalMill
                _timer = new Timer { Interval = interval };
                _timer.Elapsed += _mailhandler.ReadEmails;
                _timer.Enabled = true;
            }
            catch (Exception ex)
            {
                System.Threading.Thread.Sleep(1000);
                Logger.Error(ex.Message, ex);
                throw new Exception(ex.Message, ex);
            }
            Logger.InfoFormat("End Method: {0}", System.Reflection.MethodBase.GetCurrentMethod(
            EventLog.WriteEntry("EmailService Started", EventLogEntryType.Information);
        }
        catch (Exception ex)
        {
            System.Threading.Thread.Sleep(1000);
            Logger.Error(ex.Message, ex);
            string msg = "EmailService could not start.\r\n" + ex.Message;
            if (ex.InnerException != null && ex.InnerException.Message.Length > 0)
            {
                msg += "\r\n" + ex.InnerException.Message;
            }
            EventLog.WriteEntry(msg, EventLogEntryType.Error);
            throw new Exception("Error while starting service", ex);
        }
    }

    protected override void OnStop()
    {
        EventLog.WriteEntry("EmailService Stopped", EventLogEntryType.Information);
    }
}
```

Фрагмент класу Mailhandler

```
internal class Mailhandler
{
    //System.Configuration.AppSettingsReader reader;
    private static readonly ILog Logger = LogManager.GetLogger(typeof(Mailhandler));
    private static readonly object LockObj = new object();
    private ConfigurationHelper _configHelper;
    private ExchangeService _service;
    private string _sharedLocation;
    private ATEImportBO _boObject;
    private StreamingSubscriptionConnection _connection;
    private bool _connectionClosedInternally;
    private int _counterForEmailInterval;
    private bool _dbUp;

    /// <summary>
    /// Конструктор
    /// </summary>
    internal Mailhandler()
    {
        try
        {
            Init();
        }
        catch (Exception ex)
        {
            Logger.Error(ex.Message, ex);
            //throw ex;
    }

    private void Init()
    {
        var connectionString = ConfigurationHelper.GetConnectionString();
        if (!Helper.CheckConnection(connectionString))
        {
            _dbUp = false;
        }
        else
        {
            _boObject = new ATEImportBO(connectionString);
            XmlConfigurator.Configure();
            var crypt = new Cryptography();
            _service = new ExchangeService(ExchangeVersion.Exchange2010_SP1);
            _configHelper = new ConfigurationHelper();
            _service.Credentials = new WebCredentials(crypt.Decrypt(ConfigurationManager.AppSet
            crypt.Decrypt(ConfigurationManager.AppSettings["password"]));
            _service.AutodiscoverUrl(crypt.Decrypt(ConfigurationManager.AppSettings["username"]
            RedirectionCallback);
            _sharedLocation = Convert.ToString(_configHelper.GetXML(Const.ConstSharedLocation))
            if (!Directory.Exists(_sharedLocation))
                Directory.CreateDirectory(_sharedLocation);
            SetStreamingNotifications(_service);
            _dbUp = true;
        }
    }
```

БІБЛІОГРАФІЧНА ДОВІДКА

Тема дипломної роботи: “Автоматизація процесів обробки даних з електронних листів клієнтів ”

Обсяг пояснювальної записки: 77 аркушів.

Дата закінчення роботи: 11 червня 2025 р.

Підпис студента _____