НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

(назва навчального закладу)

Кафедра ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

Дисципліна «Технології розроблення програмного забезпечення»

Курс 3 Група ІА-12 Семестр 5

ЗАВДАННЯ

на курсову роботу студента

Глухова Івана Юрійовича

(прізвище, ім'я, по батькові)

1. Тема роботи Email-crawler

2. Строк здачі студентом закінченої роботи 30.12.2023

3. Вихідні дані до роботи:

Емейл сканер дає змогу здійснювати пошук ключових слів усередині листів .eml.

Так само здійснює пошук у вкладених файлах будь-якого типу: архіви, csv, pdf і так далі. Результати пошуку зберігає в зручному форматі: текстовий документ, html-сторінка.

4. Зміст розрахунково – пояснювальної записки (перелік питань, що підлягають розробці):

Огляд існуючих рішень, загальний опис проєкту, вимоги до застосунків системи, функціональні вимоги до системи, нефункціональні вимоги до системи, сценарії використання системи, концептуальна модель системи, вибір бази даних, вибір мови програмування та середовища розробки, проєктування розгортання системи, структура бази даних, архітектура системи, специфікація системи, вибір та обґрунтування патернів реалізації, інструкція користувача.

Додатки:

Додаток 1 - Ключовий програмний код бекенд-частини додатку, Додаток 2 - Ключовий програмний код фронтенд-частини додатку

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

UML-діаграма класів, UML-діаграма прецедентів, UML-діаграма послідовностей

6. Дата видачі завдання 6.10.2023

КАЛЕНДАРНИЙ ПЛАН

№, п/п	Назва етапів виконання курсової роботи	Строк виконання етапів роботи	Підписи або примітки
1.	Дата видачі завдання	6.10.2023	
2.	Огляд існуючих рішень	9.10.2023	
3.	Загальний опис проєкту	14.10.2023	
4.	Реалізація компонентів	19.10.2023	
5.	Вимоги до застосунків системи	27.10.2023	
6.	Сценарії використання системи	01.11.2023	
7.	Концептуальна модель системи	07.11.2023	
8.	Вибір бази даних	12.11.2023	
9.	Вибір мови програмування та середовища розробки	17.11.2023	
10.	Проєктування розгортання системи	03.12.2023	
11.	Структура бази даних	06.12.2023	
12.	Архітектура системи	07.12.2023	
13.	Інструкція користувача	11.12.2023	
14.	Захист курсової роботи	5.01.2023	

Студент____Глухов_____ **Іван Юрійович**_____
(підпис) (Ім'я ПРІЗВИЩЕ)

Керівник_____ Валерій КОЛЕСНІК_
(підпис) (Ім'я ПРІЗВИЩЕ)

«_5_»_____січня_____2024_ р.

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
Кафедра інформаційних систем та технологій

Тема____Email-crawler_____

Курсова робота
З дисципліни «Технології розроблення програмного забезпечення»

Керівник
доц. Колеснік В.М.
«Допущений до захисту»
_____
(Особистий підпис керівника)
« »_____2023р.
Захищений з оцінкою
_____
(оцінка)

Виконавець
ст. Глухов І.Ю._____
залікова книжка №____– _____
гр. ІА-12_____
_____Глухов_____
(особистий підпис виконавця)
«5»_____січня_____2024р.

Члени комісії:

_____ _____
(особистий підпис) (розшифровка підпису)

_____ _____
(особистий підпис) (розшифровка підпису)

Київ – 2023

ЗМІСТ

ВСТУП

Сучасна IT-сфера надзвичайно популярна у всьому світі, і кількість нових компаній, зокрема в цифровому просторі, щодня зростає. Усі підприємства, незалежно від їхнього масштабу - від малих стартапів до великих корпорацій, знаходяться перед схожою задачею: ефективним управлінням завданнями та ресурсами між працівниками. Це питання особливо актуальне для великих підприємств, але й малі бізнеси не можуть обійтися без ефективної стратегії розподілу робочих ресурсів.

Однак вирішення цієї проблеми не обмежується лише однією стратегією; воно вимагає використання різноманітних інструментів. З урахуванням різноманіття галузей бізнесу існує безліч інструментів, придатних для різних типів підприємств.

Попит на програмне забезпечення, пов'язане з управлінням проектами, завжди залишатиметься високим, оскільки технології розвиваються разом із бізнесом. Це призводить до виникнення нових ідей та механізмів для класифікації та розподілу завдань і обов'язків. Однак важливо враховувати, що вибір конкретного програмного забезпечення для управління проектами може призвести до ряду наслідків, таких як проблеми з безпекою, складнощі в адаптації до конкретного ПЗ та високі витрати. Це особливо важливо в контексті теми нашого дослідження - Email-crawler.

1 ПРОЄКТУВАННЯ СИСТЕМИ

1.1. Огляд існуючих рішень

При розгляді існуючих рішень для Email-crawler важливо враховувати різноманіття програм, кожна з яких має свої переваги та обмеження [2]. Серед популярних критеріїв порівняння можна виділити:

- Функціонал: Різноманіття інструментів для ефективної обробки та аналізу електронних листів.
- Зручність використання: Інтуїтивний інтерфейс та легкість використання для зручного взаємодії користувача з програмою.
- Інтеграції: Можливість взаємодії та інтеграції з іншими інструментами та сервісами.
- Ціна: Вартість використання програми та різні тарифні плани.
- Здатність витримувати великий об’єм даних: Можливість ефективно обробляти та зберігати значну кількість електронних листів та даних.
- Швидкість роботи: Продуктивність та ефективність при обробці та аналізі інформації.

Один з яскравих прикладів в цій області - Hunter.io, яке спеціалізується на зборі та аналізі інформації з електронних листів для визначення контактної інформації компаній.

Іншим рішенням є Snov.io, яке не лише допомагає в знаходженні інформації про електронні адреси, але також надає можливість взаємодії з цими адресами, реалізуючи функції відправки листів та взаємодії зі знайденими контактами.

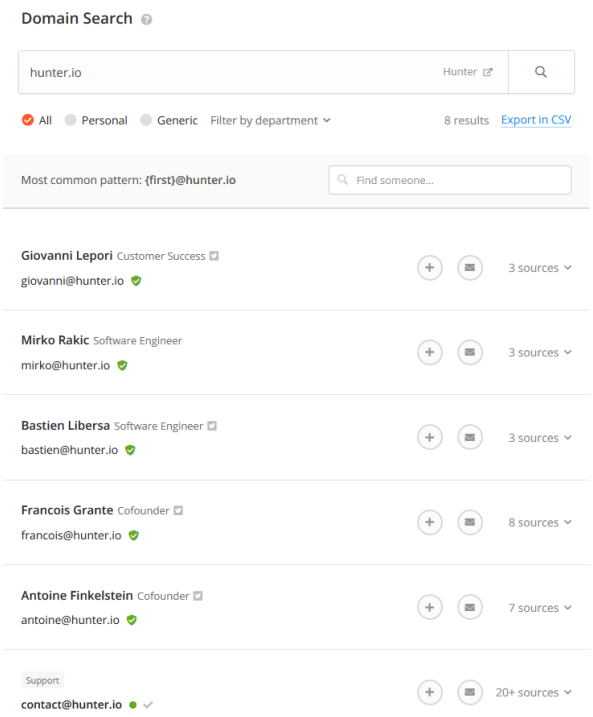
Іншим цікавим варіантом є Clearbit, яке використовується для автоматизованої обробки електронних адрес та виведення з них важливих бізнес-даних.

Кожен з цих інструментів має свої унікальні особливості та можливості, що важливо враховувати при виборі рішення для конкретних потреб у сфері Email-crawler.

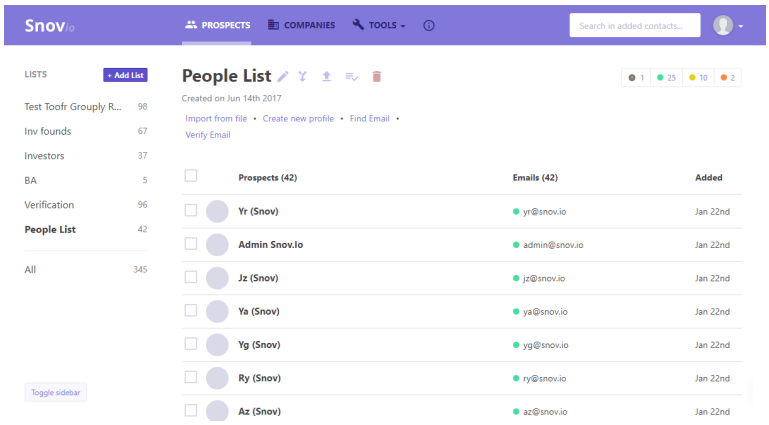
1. Hunter.io:
    - Функціонал: Hunter.io спеціалізується на пошуку інформації про електронні адреси компаній. Він забезпечує можливість знаходження та перевірки електронних адрес для визначення контактної інформації підприємств.
    - Зручність використання: Простий та зрозумілий інтерфейс, що дозволяє користувачам швидко та ефективно здійснювати пошук необхідної інформації.
    - Інтеграції: Підтримує інтеграції з іншими сервісами та інструментами для полегшення роботи зі знайденою інформацією.
    - Ціна: Має тарифні плани з різними можливостями та ціновими пакетами для різних типів користувачів.
    - Здатність витримувати великий об'єм даних: Здатний ефективно обробляти великі обсяги електронних адрес та надавати користувачам необхідну інформацію.
2. Snov.io:
    - Функціонал: Snov.io надає можливості не тільки для знаходження електронних адрес, але й для взаємодії з ними, включаючи відправку листів та взаємодію зі знайденими контактами.
    - Зручність використання: Інтуїтивний інтерфейс, що полегшує взаємодію користувача з різними функціями.
    - Інтеграції: Має можливість інтеграції з іншими інструментами та сервісами для забезпечення повноцінного використання.
    - Ціна: Надає різні тарифи, які враховують різні потреби користувачів, включаючи безкоштовні та платні плани.
    - Здатність витримувати великий об'єм даних: Ефективно працює з великою кількістю електронних адрес та надає можливості для їх обробки.
3. Clearbit:
    - Функціонал: Clearbit використовується для автоматизованої обробки електронних адрес та виділення важливих бізнес-даних з них.
    - Зручність використання: Програма, яка пропонує зручний та легко засвоюваний інтерфейс для користувачів.

- Інтеграції: Підтримує інтеграції з іншими інструментами та сервісами для максимальної взаємодії.
- Ціна: Має різні тарифні плани, включаючи безкоштовні та платні варіанти.
- Здатність витримувати великий об’єм даних: Забезпечує ефективну обробку та аналіз великої кількості електронних адрес.

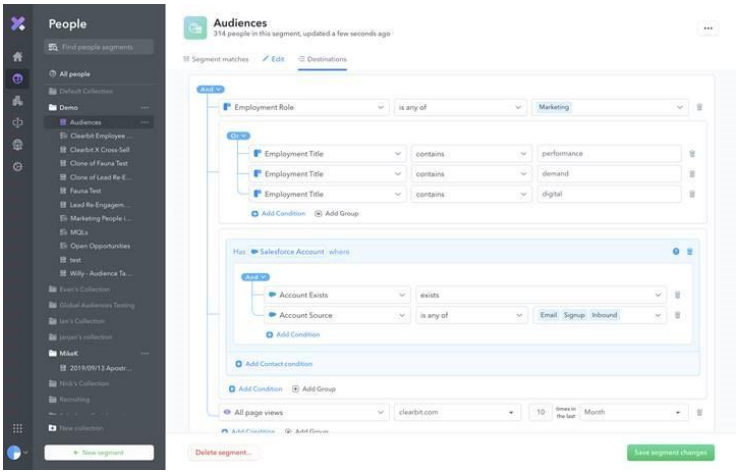
Hunter.io



Snov.io



Clearbit



1.2. Загальний опис проєкту

Проєкт "Email-crawler" має на меті використання email-crawler для швидкого та ефективного пошуку необхідної інформації та обробки електронних листів. Це програмне забезпечення дозволить автоматизувати процеси аналізу та збору даних з електронних листів, забезпечуючи команді зручний та швидкий доступ до необхідної інформації.

Основні характеристики проєкту:

Швидкий пошук інформації: Використання email-crawler дозволить ефективно сканувати та аналізувати великі обсяги електронних листів для швидкого пошуку необхідної інформації.

Обробка електронних листів: Програмне забезпечення буде здатне обробляти отримані електронні листи, витягуючи ключові дані та виконуючи необхідні завдання згідно з вимогами проєкту.

Автоматизація процесів: Використання email-crawler дозволить автоматизувати процеси збору та обробки інформації, що значно підвищить ефективність роботи команди.

Зручний доступ до даних: Команда матиме можливість зручно та швидко отримувати доступ до обробленої інформації через інтерфейс програмного забезпечення.

Захист даних: Забезпечення високого рівня конфіденційності та безпеки даних, оброблених за допомогою email-crawler.

Моніторинг та Звітність: Програмне забезпечення буде обладнане засобами моніторингу та звітності, що дозволить команді отримувати детальну статистику щодо використання та результативності email-crawler.

Скорочення Часу та Зусиль: Застосування email-crawler дозволяє великою мірою автоматизувати задачі, пов'язані з обробкою електронних листів, що призведе до зменшення часових та людських ресурсів, необхідних для виконання цих завдань.

Проєкт "Email-crawler" спрямований на створення потужного інструменту для оптимізації робочих процесів, забезпечуючи команді зручний та ефективний інструмент для роботи з електронними листами.

1.3. Вимоги до застосунків системи

1.3.1.1 Функціональні вимоги до системи [10]

Сканування Емейлів (.eml Файлів):

Система повинна мати змогу сканувати вміст електронних листів у форматі .eml.

Користувач повинен мати можливість вибирати конкретні .eml файли або цілі папки для сканування.

Пошук Ключових Слів:

Система має надавати можливість вводити одне або декілька ключових слів для пошуку.

Пошук має відбуватися як у тексті емейлу, так і у вкладених файлах.

Підтримка Різних Форматів Файлів:

Система повинна підтримувати пошук у вкладених файлах різних форматів, включаючи архіви, CSV, PDF тощо.

Збереження Результатів Пошуку:

Користувач повинен мати змогу зберігати результати пошуку у вигляді текстового документа або HTML-сторінки.

Система має надавати опції для вибору формату збереження результатів.

Інтерфейс Користувача:

Система має мати інтуїтивно зрозумілий графічний інтерфейс для вибору файлів, введення ключових слів і перегляду результатів.

1.3.1. Нефункціональні вимоги до системи [10]

Система має відповідати наступним функціональним вимогам:

- Для збереження постійності даних, система має використовувати базу даних для зберігання та читання цих даних
- Система повинна надавати можливість зручної взаємодії з серверною частиною за допомогою консольного застосунку
- Система повинна ефективно розпоряджатись ресурсами
- Система повинна використовувати новітні технології для максимальної відмовостійкості
- Система має бути відкрита до подальших модифікацій та додавання функціоналу
- Код має бути організовано використовуючи ООП та шаблони проєктування
- Система має використовувати протокол HTTPS для взаємодії клієнта та серверної частини

1.4. Сценарії використання системи

Unified Modeling Language (UML) — уніфікована мова моделювання. Розшифруємо: modeling передбачає створення моделі, що описує об'єкт. Unified (універсальний, єдиний) — підходить для широкого класу проектованих програмних систем, різних областей додатків, типів організацій, рівнів компетентності, розмірів проектів. UML описує об'єкт в єдиному заданому синтаксисі, тому де б ви не намалювали діаграму, її правила будуть зрозумілими для всіх, хто знайомий з цією графічною мовою — навіть в іншій країні.

Одне із завдань UML — бути засобом комунікації всередині команди та при спілкуванні з замовником.

Нижче наведено можливі варіанти використання дігарам:

- Проектування. UML-діаграми стануть у пригоді при моделюванні архітектури великих проектів, в якій можна зібрати як великі, так і дрібніші деталі і намалювати каркас (схему) програми. По ньому пізніше буде будуватись код.
- Реверс-інжиніринг — створення UML-моделі з існуючого коду додатку, зворотна побудова. Может застосовуватися, наприклад, на проектах підтримки, де є написаний код, але документація неповна або відсутня.
- З моделей можна витягувати текстову інформацію і генерувати відносто читабельні тексти — документувати. Текст і графіка будуть доповнювати один одного.

На рис. 1.3 наведено “Use case” діаграму:

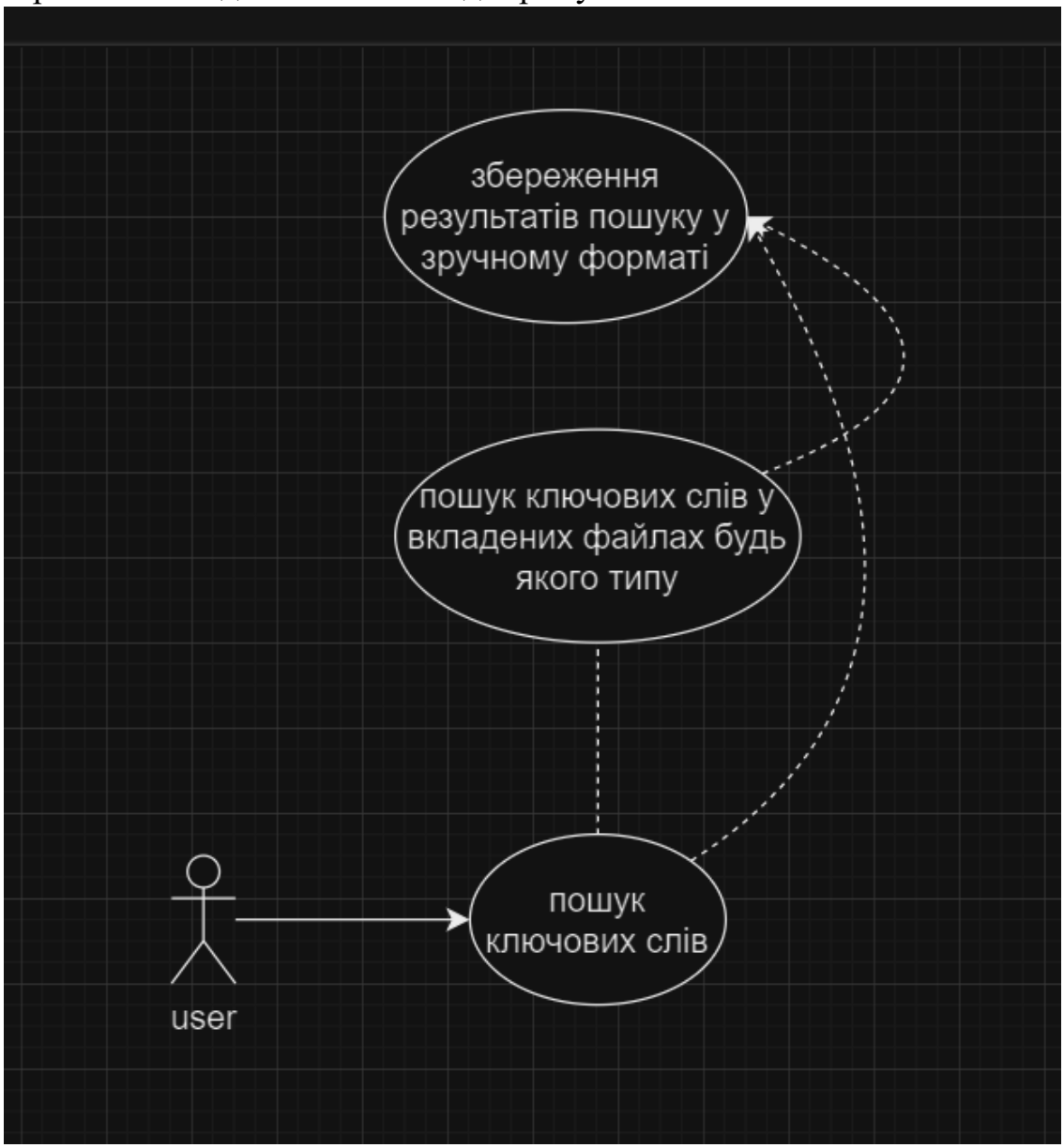


Рис. 1.3 – “Use case” діаграма

Прецедент 1: Обробка листів електронної пошти та збереження результатів у текстовий документ

Передумови:

- Користувач має папку із файлами .eml, які містять електронні листи.
- Користувач має ключові слова для пошуку.

Актори: Користувач, сервер.

Опис:

1. Користувач запускає програму та вводить шлях до папки із файлами .eml та ключові слова для пошуку.

2. Система створює об'єкт EmailProcessor, передаючи йому шлях до папки, ключові слова та інші параметри.
3. EmailProcessor рекурсивно обходить файли .eml у вказаній папці та її підпапках.
4. Для кожного електронного листа EmailProcessor витягує його вміст, декодує та шукає в ньому ключові слова.
5. Якщо знайдено ключове слово, EmailProcessor реєструє це у лог-файлі разом із інформацією про файл.
6. Завершено обробку всіх листів у папці, програма повідомляє користувача про кількість оброблених файлів та завершує виконання.

Основний хід подій:

1. Користувач вводить шлях до папки та ключові слова.
2. Система створює об'єкт EmailProcessor.
3. EmailProcessor обходить файли .eml у вказаній папці та її підпапках.
4. Для кожного листа EmailProcessor витягує вміст та шукає в ньому ключові слова.
5. Якщо знайдено ключове слово, EmailProcessor реєструє це у лог-файлі.
6. Завершено обробку всіх листів, програма повідомляє користувача та завершує виконання.

Прецедент 2: Обробка вкладених файлів у листах електронної пошти та збереження результатів у текстовий документ та файли

Передумови:

- Користувач має папку із файлами .eml, які містять електронні листи.
- Користувач має ключові слова для пошуку.

Актори: Користувач, сервер.

Опис:

1. Користувач запускає програму та вводить шлях до папки із файлами .eml та ключові слова для пошуку.
2. Система створює об'єкт EmailProcessor, передаючи йому шлях до папки, ключові слова та інші параметри.
3. EmailProcessor рекурсивно обходить файли .eml у вказаній папці та її підпапках.
4. Для кожного електронного листа EmailProcessor перевіряє наявність вкладень.
5. Якщо вкладення знаходиться, EmailProcessor витягує його вміст, декодує та шукає в ньому ключові слова.
6. Якщо знайдено ключове слово, EmailProcessor реєструє це у лог-файлі разом із інформацією про файл.
7. Завершено обробку всіх вкладень у листах, програма повідомляє користувача про кількість оброблених файлів та завершує виконання.

1.5. Концептуальна модель системи

Концептуальна модель містить наступні сутності:

- core2: основна логіка системи
- server: виконання операція та парсинг
- client: обробка та вивід інормації

На рис. 1.4 наведена UML діаграма класів:

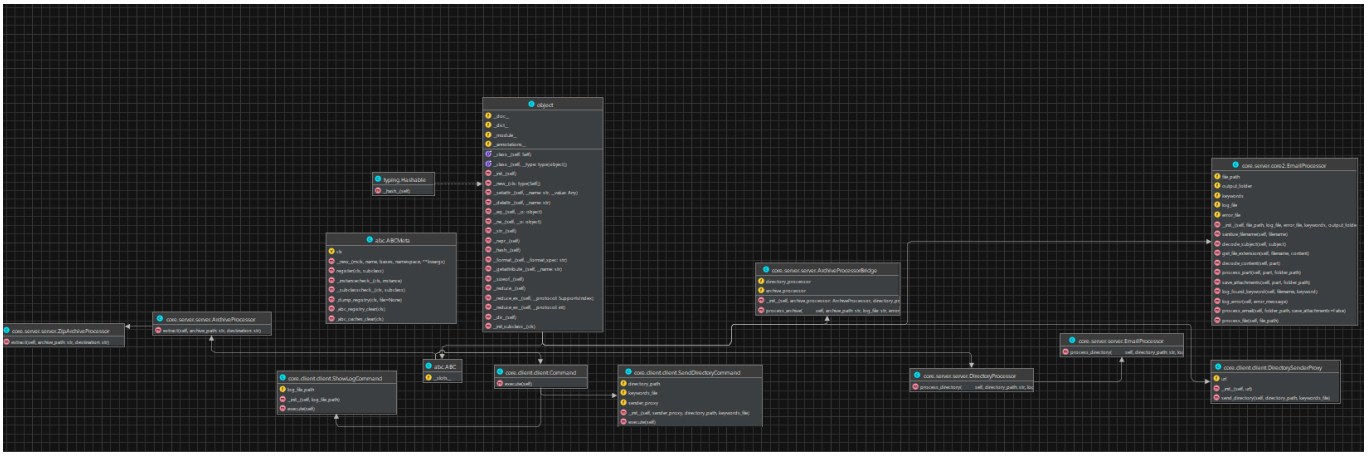


Рис. 1.4 - UML діаграма класів

Ця діаграмма використовуються для візуалізації взаємодії між об'єктами або компонентами в системі впродовж певного проміжку часу. Вони дозволяють показати, як об'єкти взаємодіють між собою, обмінюючи повідомленнями або виконуючи методи, і як ця взаємодія впливає на стан системи.

Об'єкти (Objects): Об'єкти або сутності, які беруть участь у взаємодії, представлені у верхній частині діаграми. Кожен об'єкт позначається ім'ям і може мати життєвий цикл, який відображається на діаграмі зліва направо.

Лінії життєвого циклу (Lifelines): Лінії життєвого циклу об'єкта відображають, як триває його існування під час взаємодії. Вони позначаються вертикальними лініями, на яких розташовані повідомлення, інтервали активації та інші моменти життєвого циклу об'єкта.

Повідомлення (Messages): Повідомлення показують передачу інформації або виконання дій між об'єктами. Існують різні типи повідомлень, такі як синхронні (з блокуванням), асинхронні (без блокування), відповіді на повідомлення тощо.

На рис. 1.5 наведена діаграма послідовності взаємодії клієнта, сервера та бази даних:

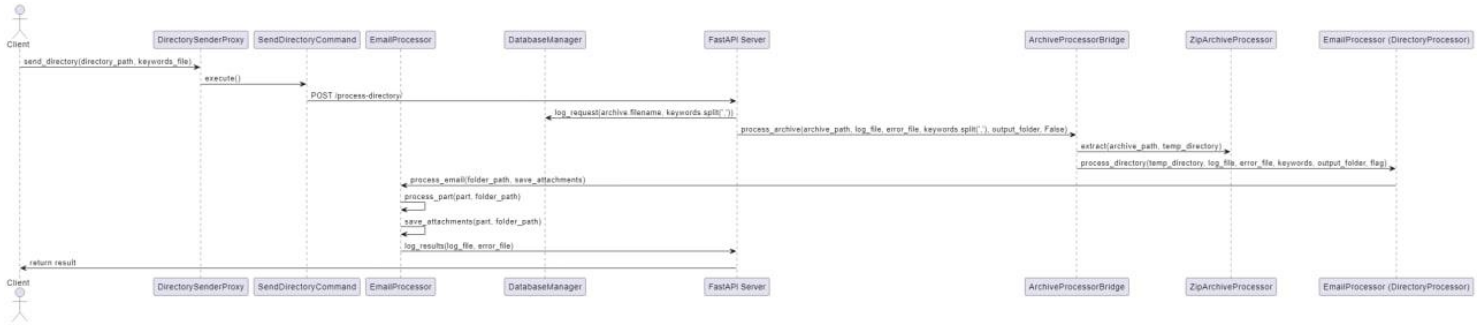


Рис. 1.5 - діаграма послідовності взаємодії клієнта, сервера та бази даних

1.6. Вибір бази даних

Вибір бази даних є важливим етапом розробки будь-якого програмного забезпечення. В даному випадку, цьому етапу розробки варто приділити особливої уваги, адже відбуватиметься інтенсивна комунікація між клієнтом, сервером та БД.

База даних — це певний набір даних, які пов'язані між собою спільною ознакою або властивістю, та впорядковані, наприклад, за алфавітом. Об'єднання великої кількості даних в єдину базу дає змогу для формування безлічі варіації групування інформації — особисті дані клієнта, історія замовлень, каталог товарів та будь-що інше. Головною перевагою БД є швидкість внесення та використання потрібної інформації. Завдяки спеціальним алгоритмам, які використовуються для баз даних, можна легко знаходити необхідні дані всього за декілька секунд. Також в базі даних існує певний взаємозв'язок інформації: зміна в одному рядку може спричинити зміни в інших рядках — це допомагає працювати з інформацією простіше і швидше.

Бази даних для сайтів дають змогу зберігати інформацію, що виглядає як зв'язані між собою таблиці. Саме в БД зберігаються вся необхідна та корисна інформація для функціонування сайту (клієнтські дані, прайс-лист, список товарів). Щоб створити запит до бази даних часто використовують Structured Query Language. SQL дає змогу додавати, редагувати та видаляти інформацію, що міститься у таблицях.

Існує багато різних типів баз даних, таких як реляційні, документо-орієнтовані, графові та інші, і кожна з них має свої переваги і недоліки. Наприклад, реляційні бази даних дуже добре підходять для роботи з структурованими даними, але можуть бути неефективними при роботі з неструктурованими даними. Також слід врахувати масштабування, надійність та швидкодію бази даних, які можуть вплинути на продуктивність та стабільність роботи проєкту. Обираючи базу даних, слід урахувати усі ці аспекти і вибрати ту, яка найкраще задовольняє потреби та вимоги проекту.

Під час прийняття рішення щодо вибору бази даних, було розглянуто наступні варіанти:

SQL Server

База даних SQL Server, розроблена Microsoft, є однією з найпопулярніших баз даних у світі. Спочатку запущений у 1989 році та написаний на C, C++, SQL Server зараз широко використовується серед великих компаній. SQL Server також є частиною хмари Microsoft Azure як Azure SQL Server. Поточна версія SQL Server – SQL Server 2019. Подібно до Oracle і MySQL, SQL Server також є системою керування реляційною базою даних (RDBMS).

PostgreSQL

PostgreSQL — це реляційна база даних із відкритим вихідним кодом, яка набирає популярності в останні роки. PostgreSQL — це система керування об'єктно-реляційною базою даних, яку можна встановити на будь-які популярні операційні системи, включаючи Windows, Unix і Linux. PostgreSQL також доступний у більшості публічних хмар, включаючи AWS, Azure і Google Cloud.

SQLite

(Structured Query Language - Lite) - це вбудована реляційна база даних, яка зазвичай використовується для локального зберігання даних в додатках. Основні риси та характеристики SQLite:

- Легкість використання: SQLite визначається своєю простотою та легкістю використання. Вона не вимагає окремого серверу та управління, що робить її ідеальною для вбудованих систем та мобільних додатків.
- Самостійність: SQLite - це самостійна база даних, яка не потребує зовнішніх залежностей чи конфігураційних файлів. Вся база даних зберігається в одному файлі на диску.
- Вбудована в більшість мов програмування: SQLite має багато бібліотек та API для різних мов програмування, таких як Python, Java, C#, PHP, і багатьох інших, що робить його дуже популярним у різних галузях розробки.
- Низький обсяг пам'яті: База даних SQLite може працювати в обмеженому

обсязі пам'яті, що дозволяє їй ефективно використовуватися на обмежених пристроях, таких як мобільні телефони та вбудовані системи.

- Транзакції та ACID властивості: SQLite підтримує транзакції, і вона гарантує ACID-властивості (Atomicity, Consistency, Isolation, Durability), що робить її надійним рішенням для зберігання даних.
- Безкоштовна та відкрита ліцензія: SQLite поширюється під ліцензією Public Domain, що означає, що ви можете використовувати, модифікувати і розповсюджувати код без обмежень.
- Підтримка стандартів SQL: SQLite підтримує багато стандартних функцій SQL, таких як SELECT, INSERT, UPDATE, DELETE, JOIN, і багато інших.
- Наявність GUI-інструментів: Існують різноманітні інструменти, такі як DB Browser for SQLite або SQLiteStudio, які надають графічний інтерфейс для роботи з базою даних SQLite.
- SQLite є популярним вибором для проектів з невеликим обсягом даних, а також там, де необхідна легка та портативна база даних.

MySQL

MySQL — вільна система керування реляційними базами даних, яка була розроблена компанією «TcX» для підвищення швидкодії обробки великих баз даних. Ця система керування базами даних з відкритим кодом була створена як альтернатива комерційним системам. Відкритий код, швидкість роботи, маштабованість та надійність є основними перевагами даної БД.

Дана система керування базами даних з відкритим кодом була створена як альтернатива комерційним системам. MySQL з самого початку була дуже схожою на mSQL, проте з часом вона все розширювалася і зараз MySQL — одна з найпоширеніших систем керування базами даних. Вона використовується, в першу чергу, для створення динамічних веб-сторінок, оскільки має чудову підтримку з боку різноманітних мов програмування.

MySQL є компонентом стеку веб-служб LAMP (скорочення від Linux, Apache, MySQL, Perl / PHP / Python). Він використовується у багатьох веб-програмах, керованих базами даних, таких як Drupal, Joomla, WordPress тощо. Багато популярних веб-сайтів, включаючи Google, Facebook, Twitter, також використовують MySQL. Він підтримується багатьма відомими мовами програмування, такими як PHP, Java, C ++, PERL тощо. Він підтримує стандартний SQL (мова структурованих запитів). Ліцензія MySQL з відкритим кодом може бути налаштована. Отже, розробник може змінити його відповідно до вимог програми.

Також, MySQL надає корисний та зручний інструмент MySQL Workbench:

- Розробка SQL: дозволяє створювати та керувати підключеннями до серверів баз даних. На додаток до того, що дозволяє користувачеві налаштовувати параметри підключення, MySQL Workbench надає можливість запускати запити SQL на підключеннях до бази даних за допомогою вбудованого редактора SQL.
- Моделювання даних (дизайн): дозволяє графічно моделювати схему бази даних, здійснювати зворотне та пряме проектування між схемою та активною базою даних та редагувати всі аспекти бази даних за допомогою всеосяжного редактора таблиць. Редактор таблиць надає зручні засоби для редагування

таблиць, стовпців, індексів, тригерів, розділів, параметрів, вставок та привілеїв, процедур та подань.

- Адміністрування сервера: Дозволяє керувати екземплярами сервера MySQL, керуючи користувачами, виконуючи резервне копіювання та відновлення, перевіряючи дані аудиту, переглядаючи стан бази даних та контролюючи продуктивність сервера MySQL.
- Міграція даних: дозволяє переходити з Microsoft SQL Server, Microsoft Access, Sybase ASE, SQLite, SQL Anywhere, PostreSQL та інших таблиць, об'єктів та даних RDBMS до MySQL. Міграція також підтримує перехід зі старих версій MySQL на останні версії.

Підтримка MySQL Enterprise: підтримка корпоративних продуктів, таких як MySQL Enterprise Backup, MySQL Firewall та MySQL Audit.

Для некомерційного використання MySQL є безкоштовним. Можливості MySQL: простота у встановленні та використанні; підтримується необмежена кількість користувачів, що одночасно працюють із БД; кількість рядків у таблицях може досягати 50 млн.; висока швидкість виконання команд; наявність простої і ефективної системи безпеки.

Для даного проєкту, було обрано саме SQLite.

Такий вибір можна аргументувати наступними чинниками:

- Простота використання та взаємодії з базою даних
- Швидкість роботи
- Надійність
- Масштабованість
- Наявність інструменту **SQLite**, що дозволяє максимально зпростити процес проєктування бази даних
- Вийняткова безпечність

1.7. Вибір мови програмування та середовища розробки

Існує багато мов програмування, які можуть бути використані для роботи над Email-crawler.

Під час вибору мови програмування, було розглянуто наступні варіанти:

- Python – це неймовірно потужна, гнучка і проста у вивченні мова програмування, яка добре підходить для широкого спектру завдань, включаючи розробку десктопних додатків. Вона має велику, активну спільноту користувачів і розробників, а це означає, що для Pythonрозробників доступна велика кількість ресурсів і підтримки. Також вона має широкий вибір різноманітних корисних бібліотек, що значно полегшує процес розробки.
- Go (Golang): Go - мова програмування, розроблена компанією Google. Відома своєю простотою, масштабованістю і продуктивністю. Golang має схожість з C. Go була створена для заміни C ++, оскільки процес вивчення C ++ є занадто довгим і складним. Go дозволяє швидко та легко реалізувати проєкт і може використовуватись для написання мікросервісів. Це перевага для розробки спеціалізованих продуктів з дуже вузьким функціоналом. Програми, написані на Golang, зазвичай працюють швидше, ніж програми, написані іншими мовами програмування. Зараз GlobalLogic активно збільшує кількість проєктів з Go як основною мовою програмування. Ми також бачимо тенденцію великих компаній “переписувати проєкт з будь-якої іншої мови програмування на Go”. Тенденція полягає в тому, що зараз Go використовується для розробки речей, які зазвичай написані на C ++: таких як продуктивний бекенд, обробка даних та ключова нетривіальна бізнес-логіка. Загалом, те, що раніше використовувалося для розробки на C ++, .Net або Java, зараз розробляється за допомогою Golang.
- C++: C++ - універсальна мова програмування, яка широко використовується в різних сферах. Вона відома своєю продуктивністю і може бути використана для розробки високопродуктивних додатків.
- C#: C# - мова програмування, розроблена компанією Microsoft, широко використовується для побудови розподілених систем.

- JavaScript: JavaScript - популярна мова програмування, яка широко використовується для веб-розробки. На JavaScript вплинули багато мов, при розробці була мета зробити мову схожою на Java, але при цьому легкою для використання непрограмістами.+JavaScript має низку властивостей об'єктно-орієнтованої мови, але реалізоване в мові прототипування обумовлює відмінності в роботі з об'єктами в порівнянні з традиційними об'єктно-орієнтованими мовами. Крім того, JavaScript має ряд властивостей, властивих функціональним мовам, - функції як об'єкти першого класу, об'єкти як списки, каррінг, анонімні функції, замикання - що додає мові додаткову гнучкість.

В результаті, було обрано саме Python, адже, здебільшого, Email-crawler є саме консольним додатком.

Серверну частину також буде створено на Python, адже це значно спростить подальшу підтримку, розширення, обслуговування проєкту, і так далі.

1.8. Проєктування розгортання системи

Розробка програмного забезпечення для управління проєктами є доволі складним та тривалим процесом, адже, перш за все, він доволі сильно залежить від вимог та структури конкретної організації, компанії, команди чи бізнесу.

Нижче наведено перелік кроків, які варто виконати для успішного, ефективного та оптимального розгортання Email-crawler:

- Визначте вимоги: першим кроком розробки будь-якого програмного забезпечення завжди має бути саме визначення вимог, адже кожен випадок, в певній мірі, є індивідуальним. Наведіть приблизне очікуване навантаження на базу даних, кількість необхідних команд, проєктів, задач та співробітників, що взаємодіють із системою.
- Виберіть платформу для бекенду: Виходячи з вимог, вам потрібно буде вибрати серверну платформу, яка відповідає вашим потребам. Опції включають Linux, Windows та інші операційні системи.
- Виберіть апаратне забезпечення. Вибране вами обладнання залежатиме від очікуваного навантаження та вимог ваших програм. Вам потрібно буде вибрати сервери, сховище та мережеве обладнання, яке зможе обробляти очікуваний трафік і забезпечувати необхідну продуктивність.
- Налаштуйте базу даних: після успішної ініціалізації середовища виконання програмного забезпечення, варто ініціалізувати базу данних, використовуючи файли міграцій, що присутні у вкладеннях до додатку
- Налаштуйте бекенд: після того, як ви вибрали апаратне забезпечення та встановили операційну систему, вам потрібно буде налаштувати сервер відповідно до ваших вимог. Це може включати встановлення програмного забезпечення, налаштування заходів безпеки та налаштування сервера для обробки очікуваного навантаження.

- Створити інструкції користування: надати чіткі та детальні інструкції користування для кожного сценарію використання програмного забезпечення користувачем. Ці інструкції повинні включати всі необхідні кроки для досягнення цілей користувача.

2 РЕАЛІЗАЦІЯ КОМПОНЕНТІВ СИСТЕМИ

2.1. Структура бази даних

Структура бази даних визначає, яким чином будуть зберігатися дані та яким чином будуть організовані зв'язки між ними.

Для розробки project Email-crawler, потрібно буде зберігати такі дані як:

id:

Тип: INTEGER

Опис: Унікальний ідентифікатор для кожного запису в таблиці.

Особливості: Виступає як первинний ключ (PRIMARY KEY), що означає, що кожне значення в цьому полі є унікальним і не може бути NULL.

timestamp:

Тип: TEXT

Опис: Часова мітка, коли був зроблений запит.

Особливості: Зберігає дату та час у форматі тексту. Використовується метод datetime.now().isoformat() для отримання поточної дати та часу у форматі ISO 8601.

filename:

Тип: TEXT

Опис: Назва файлу, який був запитаний або оброблений.

Особливості: Текстове поле, яке не може бути NULL.

keywords:

Тип: TEXT

Опис: Ключові слова, пов'язані з запитом.

Особливості: Текстове поле, яке зберігає ключові слова, розділені комами. Також не може бути NULL.

2.2. Архітектура системи

2.2.1. Специфікація системи

Структура системи баз даних

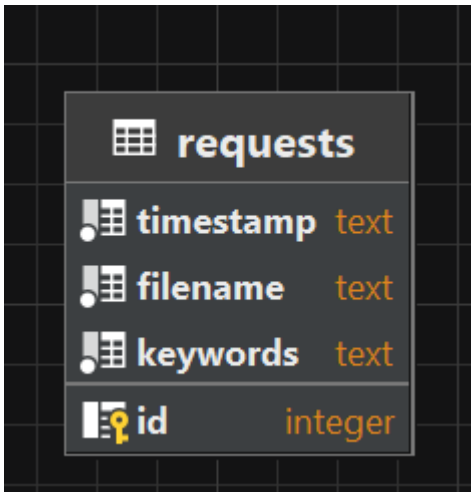


Рис. 2.1 - Структура системи баз даних SQL

На рисунку 2.1 наведено таблицю

- request

Оскільки на даному етапі розвитку проєкту сутностей не так багато, таблиць у базі даних всього одна, але протягом подальших обновлень та реалізації нового функціоналу, база даних буде розширюватись разом із кількістю таблиць в ній, а її структура – ускладнюватись.

Кожна з сутностей додатково має властивості, необхідні для реалізації шаблонів проєктування програмного забезпечення.

2.2.2. Вибір та обґрунтування патернів реалізації

Під час реалізації проєкту, було використано наступні патерни реалізації (шаблони проєктування):

- Client-server

Цей паттерн використовується, коли сервер і клієнти підключаються через Інтернет. Тут сервер є постачальником послуг. А клієнт – це споживач послуг. Зазвичай сервер знаходиться в локальній мережі або в Інтернеті. Якщо сервер розташований у локальній мережі та немає додаткових налаштувань DNS, сторонні особи не можуть отримати доступ до ресурсу.

У підході «клієнт - сервер» компоненти і сполучні елементи мають певну поведінку. Компоненти, звані «клієнтами», відправляють запити компоненту, званому «сервер», і чекають відповіді. Компонент «сервер» отримує запит від клієнта і відправляє йому відповідь.

- Команда

Це поведінковий патерн проектування, який перетворює запити на об’єкти, дозволяючи передавати їх як аргументи під час виклику методів, ставити запити в чергу, логувати їх, а також підтримувати скасування операцій.

Використовується для вирішення завдань, пов'язаних із надсиланням директорії на сервер і відображенням логів. Кожна команда ізолює логіку виконання від класу, що викликає цю команду, що сприяє розширюваності та підтримці нових операцій без зміни коду викликаючого класу.

- Bridge

Породжувальний патерн проектування, який визначає загальний інтерфейс для створення об’єктів у суперкласі, дозволяючи підкласам змінювати тип створюваних об’єктів. Патерн Міст пропонує замінити спадкування на делегування. Для цього

потрібно виділити одну з таких «площин» в окрему ієрархію і посилатися на об’єкт цієї ієрархії, замість зберігання його стану та поведінки всередині одного класу.

На рис. 2.2 можна побачити ілюстрацію патерну Bridge:

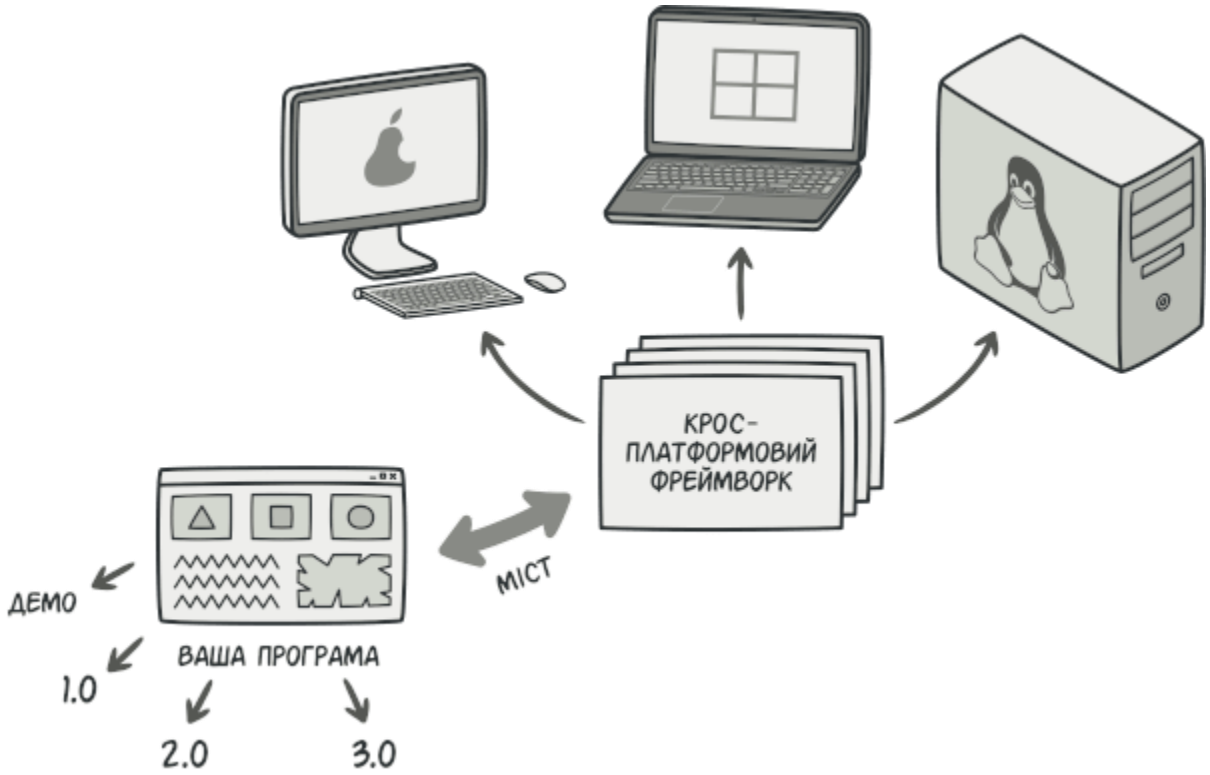


Рис. 2.2 - ілюстрація патерну Bridge

- Singleton

Патерн реалізації Singleton має за ціль забезпечити, що лише один екземпляр даного класу буде створений у програмі. Це може бути корисно у випадку, коли необхідно забезпечити лише одну точку доступу до ресурсів, таких як база даних чи файли, щоб уникнути конфліктів та помилок. Всі реалізації Singleton’а зводяться до того, аби приховати типовий конструктор та створити публічний статичний метод, який і контролюватиме життєвий цикл об’єкта-одинака. Якщо у вас є доступ до класу Singleton, отже, буде й доступ до цього статичного методу. З якої точки коду ви б його не викликали, він завжди віддаватиме один і той самий об’єкт.

- Замісник

Це структурний патерн проектування, що дає змогу підставляти замість реальних об’єктів спеціальні об’єкти-замінники. Ці об’єкти перехоплюють виклики до оригінального об’єкта, дозволяючи зробити щось *до* чи *після* передачі виклику оригіналові.

- Декоратор

Це структурний патерн проектування, що дає змогу динамічно додавати об’єктам нову функціональність, загортаючи їх у корисні «обгортки».

- Легковаговик

Це структурний патерн проектування, що дає змогу вмістити більшу кількість об’єктів у відведеній оперативній пам’яті. Легковаговик заощаджує пам’ять, розподіляючи спільний стан об’єктів між собою, замість зберігання однакових даних у кожному об’єкті.

2.3. Інструкція користувача

Для запуску проєкту, необхідно виконати етапи:

1. **Встановлення залежностей:** Переконайтеся, що встановлені всі необхідні бібліотеки та пакети. Використовуйте **pip** для встановлення залежностей. Введіть команду в терміналі:

   pip install fastapi aiofiles python-magic uvicorn python-docx pandas pyfiglet

2. **Запуск FastAPI сервера:** Запустіть FastAPI сервер. Відкрийте термінал та перейдіть в каталог, де збережений файл **server.py**. Використовуйте команду:

   uvicorn server:app --reload

Сервер буде запущено на **http://localhost:8009/**.

3. **Запуск основного скрипта:** Відкрийте новий термінал і перейдіть в каталог, де збережений основний скрипт **client.py**. Запустіть його:

   python **client**.py

Після запуску основного скрипта, програма буде очікувати введення команд користувача (1 - Обробити директорію, 2 - Показати лог, 3 - Вихід).

4. **Введення команд:**
   - Якщо ви виберете опцію "1" (Обробити директорію), вам буде запропоновано ввести шлях до директорії та файлу з ключовими словами.
   - Якщо ви виберете опцію "2" (Показати лог), програма виведе вміст лог-файлу **app.log**.
5. **Завершення роботи:**
   - Щоб завершити роботу програми, введіть "3" для виходу.

За допомогою цих кроків ви зможете ініціалізувати базу даних, запустити сервер, розгорнути фронтенд та використовувати командний інтерфейс для взаємодії з програмою "Email-crawler".

ВИСНОВКИ

Отже, розробка Email-crawler – це складний процес, що вимагає глибокого розуміння, перш за все, поставленої задачі, вимог, а також, володіння технологіями створення БД, серверної та клієнтської частини застосунків.

Шаблони проєктування — це комплект випробуваних та протестованих рішень найпоширеніших проблем у сфері проєктування ПЗ.

Використання шаблонів проектування може значно покращити проектування, організацію та обслуговування застосунків. Впроваджуючи загальні рішення повторюваних проблем у послідовний і модульний спосіб, шаблони проектування дозволяють розробникам створювати нові рішення подібних ПЗ, які є масштабованими, надійними та легкими для модифікації та розширення.

Існує багато різних шаблонів проектування, які можна застосувати до даної проблемної галузі, починаючи від творчих шаблонів, які обробляють створення об’єктів, до шаблонів поведінки, які визначають, як об’єкти взаємодіють, до структурних шаблонів, які описують, як об’єкти складаються. Вивчаючи та застосовуючи ці шаблони, розробники можуть значно покращити якість та ефективність своїх застосунків. Крім того, розуміючи основні принципи та мотивацію, що лежать в основі різних шаблонів проектування, розробники можуть отримати глибше розуміння архітектури додатку та стати більш кваліфікованими спеціалістами.

Використання шаблонів проєктування у проєктах має наступні переваги:

- Підвищена ефективність. У вирішенні проблем розробникам не доводиться винаходити велосипед. Вони застосовують перевірені, надійні рішення. Завдяки застосуванню шаблонів їхня робота стає набагато ефективнішою.
- Повторне використання. Шаблони проєктування можна змінювати для вирішення різноманітних проблем. Вони не пов’язані з окремою конкретною проблемою.
- Відсутність потреби рефакторинга коду. Оскільки шаблон проєктування вже є найкращим вирішенням проблеми, можна обійтися без рефакторингу.

- Найменший розмір бази коду. Кожен шаблон допомагає розробникам програмного забезпечення змінювати роботу системи без повного перепроєктування. Крім того, як оптимальне рішення, шаблон вимагає менше коду.
- Спільна мова. Шаблони проєктування визначають спільну мову, якою ви разом зі своєю командою можете спілкуватися ефективніше. Не потрібно пояснювати, що таке одинак (або синглет), якщо ви знаєте шаблон та його назву.
- Швидка підготовка нових розробників. Шаблони проєктування розроблялися протягом довгого часу, і вони є найкращими рішеннями певних проблем, із якими стикаються розробники. Вивчення шаблонів допоможе недосвідченим розробникам опанувати проєктування програмного забезпечення швидко й легко.
- Підвищення кваліфікації. Знання шаблонів проєктування допомагає вам знаходити подібні фрагменти в коді.

Загалом, використання шаблонів проектування при розробці Email-crawler є потужним інструментом для побудови високоякісних систем і значно спрощує задачу.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

[1] What is an email crawler? - Quora. Доступно: https://www.quora.com/What-is-an-email-crawler#:~:text=An%20email%20crawler%20is%20a,various%20sources%20on%20the%20internet.

[2] Email-crawler Comparison. Доступно:

Hunter.io https://hunter.io/

Clearbit https://clearbit.com/

Snovio https://snov.io/ua/

[3] Паттерны/шаблоны проектирования [Наукова стаття].

Доступно https://refactoring.guru/ru/design-patterns

[4] Laukik P. Raut, Overview on Kanban Methodology and its Implementation [Наукова стаття]. Доступно: https://www.researchgate.net/publication/280830456_Overview_on_Kanban_Methodology_and_its_Implementation

[5] Особливості функціональних вимог та не функціональних вимог [Електронний ресурс]. Доступно: https://uk.myservername.com/features-functional-requirements

ДОДАТКИ

Ключовий програмний код :

Client.py

```
import logging
from abc import ABC, abstractmethod
from datetime import datetime
from functools import wraps
import requests
import os
import zipfile
from generatelog import generate_html_log

logging.basicConfig(filename='app.log', level=logging.INFO, format='%(asctime)s -
%(levelname)s - %(message)s')

def log_to_file_decorator(func):
    @wraps(func)
    def wrapper(*args, **kwargs):
        directory_path = args[1]
        keywords_file = args[2]
        current_time = datetime.now()
        logging.info(
            f"Початок виконання  для директорії: {directory_path}, файл ключових слів:
{keywords_file} о {current_time}")
        result = func(*args, **kwargs)
        return result
    return wrapper

class Command(ABC):
    @abstractmethod
    def execute(self):
        pass

class ShowLogCommand(Command):
    def __init__(self, log_file_path):
        self.log_file_path = log_file_path
    def execute(self):
        try:
            with open(self.log_file_path, 'r', encoding='windows-1251',
errors='ignore') as log_file:
                log_content = log_file.read()
                print("\033[93m" + log_content + "\033[0m")
        except FileNotFoundError:
            print(f"Помилка: Лог-файл '{self.log_file_path}' не знайдено.")
        except UnicodeDecodeError:
            print(f"Помилка: Неможливо декодувати вміст лог-файлу як Windows-1251.")

class SendDirectoryCommand(Command):
    def __init__(self, sender_proxy, directory_path, keywords_file):
        self.sender_proxy = sender_proxy
        self.directory_path = directory_path
        self.keywords_file = keywords_file
    def execute(self):
        self.sender_proxy.send_directory(self.directory_path, self.keywords_file)


class DirectorySenderProxy:
    def __init__(self, url):
        self.url = url

    @log_to_file_decorator
    def send_directory(self, directory_path, keywords_file):
        if not os.path.exists(directory_path):
```

```
            logging.error(f"\033[91mПомилка: Зазначена директорія не існує:
{directory_path}\033[0m\n")
            print('\n\033[91mПомилка! Перевірте файл логів\033[0m')
            return

        if not os.path.isdir(directory_path):
            logging.error(f"\n\033[91mПомилка: Зазначений шлях не є директорією:
{directory_path}\033[0m")
            print('\n\033[91mПомилка! Перевірте файл журналу\033[0m')
            return

        if not os.listdir(directory_path):
            logging.error(f"\n\033[91mПомилка: Директорія порожня
{directory_path}\033[0m")
            print('\n\033[91mПомилка! Перевірте файл журналу\033[0m')
            return

        if not os.path.exists(keywords_file):
            logging.error(f"\n\033[91mПомилка: Файл ключових слів не знайдено
{directory_path}\033[0m")
            print('\n\033[91mПомилка! Перевірте файл журналу\033[0m')
            return

        print('\033[92mПеревірка пройшла успішно. Відправлення даних на
сервер...\033[0m')
        send_directory_to_server(self.url, directory_path, keywords_file)

def read_keywords(file_path):
    with open(file_path, 'r', encoding='utf-8') as file:
        return file.read().strip().split(',')

def create_zip_archive(directory_path, output_path):
    with zipfile.ZipFile(output_path, 'w', zipfile.ZIP_DEFLATED) as zipf:
        for root, dirs, files in os.walk(directory_path):
            for file in files:
                zipf.write(os.path.join(root, file),
os.path.relpath(os.path.join(root, file), os.path.join(directory_path,

'../../..')))

def send_directory_to_server(url, directory_path, keywords_file):
    zip_path = 'temp_archive.zip'
    create_zip_archive(directory_path, zip_path)

    keywords = ','.join(read_keywords(keywords_file))
    try:
        with open(zip_path, 'rb') as zip_file:
            files = {'archive': zip_file}
            data = {'keywords': keywords}
            response = requests.post(url, files=files, data=data)
            response_data = response.json()
            print_formatted_log(response_data)
    except requests.exceptions.JSONDecodeError:
        print("Помилка: Неможливо отримати JSON-відповідь від сервера.")
    finally:
        os.remove(zip_path)


def print_formatted_log(response_data):
    if 'log' in response_data:
        log_entries = response_data['log'].split('\n')
        for entry in log_entries:
            if entry.strip():
                print("\033[92m" + entry + "\033[0m")

    if 'errors' in response_data and response_data['errors'].strip():
        print("\n\033[91m" + response_data['errors'] + "\033[0m")
```

```
    html_log = generate_html_log(response_data)
    with open('log_output.html', 'w', encoding='utf-8') as file:
        file.write(html_log)


def main():
    url = 'http://localhost:8009/process-directory/'
    sender_proxy = DirectorySenderProxy(url)

    while True:
        print("Введіть команду:")
        print("1. Обробити директорію")
        print("2. Показати лог")
        print("3. Вихід")
        choice = input("Виберіть опцію: ")

        if choice == '1':
            directory_path = input("Введіть шлях до директорії з файлами: ")
            keywords_file = input("Введіть шлях до файлу з ключовими словами
(наприклад, dict.txt): ")
            command = SendDirectoryCommand(sender_proxy, directory_path,
keywords_file)
            command.execute()
        elif choice == '2':
            log_command = ShowLogCommand('app.log')
            log_command.execute()
        elif choice == '3':
            print("Вихід з програми.")
            break
        else:
            print("Невірна команда, спробуйте ще раз.")


if __name__ == "__main__":
    main()
```

Sever.py

```
from fastapi import FastAPI, UploadFile, File, Form
import aiofiles
import os
import shutil
import zipfile
from core2 import search_keywords_in_emails
from abc import ABC, abstractmethod
from typing import List
from database import DatabaseManager

app = FastAPI()
db_manager = DatabaseManager('requests.db')


class ArchiveProcessor(ABC):
    @abstractmethod
    def extract(self, archive_path: str, destination: str) -> None:
        pass

class ZipArchiveProcessor(ArchiveProcessor):
    def extract(self, archive_path: str, destination: str) -> None:
        with zipfile.ZipFile(archive_path, 'r') as zip_ref:
            zip_ref.extractall(destination)

class DirectoryProcessor(ABC):
    @abstractmethod
    async def process_directory(
        self, directory_path: str, log_file: str, error_file: str, keywords:
```

```
List[str], output_folder: str, flag: bool
    ) -> None:
        pass

class EmailProcessor(DirectoryProcessor):
    async def process_directory(
        self, directory_path: str, log_file: str, error_file: str, keywords:
List[str], output_folder: str, flag: bool
    ) -> None:
        await search_keywords_in_emails(directory_path, log_file, error_file,
keywords, output_folder, flag)

class ArchiveProcessorBridge:
    def __init__(self, archive_processor: ArchiveProcessor, directory_processor:
DirectoryProcessor):
        self.archive_processor = archive_processor
        self.directory_processor = directory_processor

    async def process_archive(
        self, archive_path: str, log_file: str, error_file: str, keywords: List[str],
output_folder: str, flag: bool
    ) -> None:
        temp_directory = "temp_directory"
        self.archive_processor.extract(archive_path, temp_directory)
        await self.directory_processor.process_directory(temp_directory, log_file,
error_file, keywords, output_folder, flag)



@app.post("/process-directory/")
async def process_directory(archive: UploadFile = File(...), keywords: str =
Form(...)):
    archive_path = f"temp_{archive.filename}"
    log_file = "log.txt"
    error_file = "errors.txt"
    output_folder = "output_folder"
    await db_manager.log_request(archive.filename, keywords.split(','))
    open(log_file, 'w').close()
    open(error_file, 'w').close()

    async with aiofiles.open(archive_path, "wb") as out_file:
        content = await archive.read()
        await out_file.write(content)

    archive_processor = ZipArchiveProcessor()
    directory_processor = EmailProcessor()

    bridge = ArchiveProcessorBridge(archive_processor, directory_processor)
    await bridge.process_archive(archive_path, log_file, error_file,
keywords.split(','), output_folder, False)

    log_content = ""
    if os.path.exists(log_file):
        async with aiofiles.open(log_file, "r", encoding="utf-8") as log:
            log_content = await log.read()

    errors_content = ""
    if os.path.exists(error_file):
        async with aiofiles.open(error_file, "r", encoding="utf-8") as errors:
            errors_content = await errors.read()

    os.remove(archive_path)
    shutil.rmtree("temp_directory")

    return {"message": "Directory processed", "log": log_content, "errors":
errors_content}

if __name__ == "__main__":
```

```
    import uvicorn
    uvicorn.run(app, host="0.0.0.0", port=8009)
```

Core2.py

```
import argparse
import io
import mimetypes
import os
import email
import re
import pyfiglet
import asyncio
import aiofiles
from email.header import decode_header
import docx
import pandas as pd
import xml.etree.ElementTree as ET
import time
import fitz
import subprocess
from pandas.io.sas.sas_constants import magic


class EmailProcessor:
    def __init__(self, file_path, log_file, error_file, keywords, output_folder):
        self.file_path = file_path
        self.log_file = log_file
        self.error_file = error_file
        self.keywords = keywords
        self.output_folder = output_folder

    def sanitize_filename(self, filename):
        valid_filename = re.sub(r'[\/:*?"<>|]', '_', filename)
        return valid_filename

    def decode_subject(self, subject):
        decoded = decode_header(subject)
        return decoded[0][0].decode(decoded[0][1]) if decoded[0][1] else decoded[0][0]

    def get_file_extension(self, filename, content):
        _, extension = os.path.splitext(filename)
        if extension:
            return extension.lstrip('.').lower()

        mime_type, _ = mimetypes.guess_type(filename)
        if mime_type:
            extension = mimetypes.guess_extension(mime_type)
            if extension:
                return extension.lstrip('.').lower()

        try:
            if content:
                mime_type = magic.Magic(mime=True).from_buffer(content)
                extension = mimetypes.guess_extension(mime_type)
                return extension.lstrip('.').lower() if extension else None
        except Exception:
            return None

    def decode_content(self, part):
        content_type = part.get_content_type()
        if "text/plain" in content_type:
            try:
                charset = part.get_content_charset()
                content = part.get_payload(decode=True)
                if charset:
                    content = content.decode(charset)
```

```
                return content
            except TypeError:
                return "Failed to decode content"
        elif "text/html" in content_type:
            return "HTML content:\n" + part.get_payload()
        else:
            return "Non-text content. Content type: " + content_type

    async def process_part(self, part, folder_path):
        try:
            content = self.decode_content(part)
            found_keywords = []
            for keyword in self.keywords:
                if keyword.lower() in content.lower():
                    found_keywords.append(keyword)
            if found_keywords:
                filename = self.file_path
                for keyword in found_keywords:
                    await self.log_found_keyword(filename, keyword)
        except Exception as e:
            await self.log_error(f"Помилка з файлом: {self.file_path}\nПомилка:
{str(e)}")

    async def save_attachments(self, part, folder_path):
        try:
            if part.get_content_maintype() == "multipart":
                return
            filename = part.get_filename()
            if not filename:
                return

            decoded_filename, charset = decode_header(filename)[0]
            if charset:
                decoded_filename = decoded_filename.decode(charset)

            extension = self.get_file_extension(decoded_filename,
part.get_payload(decode=True))
            if extension:
                sanitized_filename = self.sanitize_filename(decoded_filename)
                attachments_dir = os.path.join(folder_path, "attachments", extension)
                os.makedirs(attachments_dir, exist_ok=True)

                payload = part.get_payload(decode=True)

                content = None
                if extension == "txt":
                    content = payload.decode("utf-8")
                elif extension == "docx":
                    doc = docx.Document(io.BytesIO(payload))
                    content = "\n".join([paragraph.text for paragraph in
doc.paragraphs])
                elif extension == "pdf":
                    pdf_document = fitz.open(stream=io.BytesIO(payload))
                    content = ""
                    for page_num in range(len(pdf_document)):
                        page = pdf_document.load_page(page_num)
                        content += page.get_text()
                # elif extension == "xlsx":
                #     df = pd.read_excel(io.BytesIO(payload))
                #     content = df.to_string(index=False)
                elif extension == "xml":
                    tree = ET.ElementTree(ET.fromstring(payload))
                    root = tree.getroot()
                    content = ET.tostring(root, encoding="utf-8").decode("utf-8")
                elif extension == "csv":
                    df = pd.read_csv(io.BytesIO(payload))
                    content = df.to_string(index=False)
```

```
                elif extension in ["js", "css", "html", "json", "tsv"]:
                    content = payload.decode("utf-8")

                if content:
                    found_keywords = []
                    for keyword in self.keywords:
                        if keyword.lower() in content.lower():
                            found_keywords.append(keyword)

                    if found_keywords:
                        filepath = os.path.join(attachments_dir, sanitized_filename)
                        async with aiofiles.open(filepath, "wb") as attachment_file:
                            await attachment_file.write(payload)

                        for keyword in found_keywords:
                            await self.log_found_keyword(filepath, keyword)

        except Exception as e:
            await self.log_error(f"Помилка при збереженні вкладення: {str(e)}\n")
    async def log_found_keyword(self, filename, keyword):
        log_file_path = self.log_file
        drive, path = os.path.splitdrive(filename)
        dirs, filename = os.path.split(path)
        _, dirs = os.path.split(dirs)
        new_filename = os.path.join(os.path.sep, dirs, filename)

        async with aiofiles.open(log_file_path, "a", encoding="utf-8") as log:
            await log.write(f"Файл: {new_filename}, Ключевое слово: {keyword}\n")

    async def log_error(self, error_message):
        error_file_path = self.error_file
        async with aiofiles.open(error_file_path, "a", encoding="utf-8") as err_log:
            await err_log.write(error_message)

    async def process_email(self, folder_path, save_attachments=False):
        try:
            async with aiofiles.open(self.file_path, "rb") as file:
                data = await file.read()
                msg = email.message_from_bytes(data)

            tasks = []
            for part in msg.walk():
                if part.get_content_maintype() == "multipart":
                    continue
                task = self.process_part(part, folder_path)
                tasks.append(task)

                if save_attachments:
                    attachment_task = self.save_attachments(part, folder_path)
                    tasks.append(attachment_task)

            await asyncio.gather(*tasks)

        except Exception as e:
            await self.log_error(f"Помилка обробки електронної пошти в файлі:
{self.file_path}\n")
            await self.log_error(f"Помилка: {str(e)}\n")

    async def process_file(self, file_path):
        try:
            extension = self.get_file_extension(file_path, None)
            content = None

            if extension == "txt":
                async with aiofiles.open(file_path, "r", encoding="utf-8") as
text_file:
                    content = await text_file.read()
            elif extension == "docx":
```

```
                doc = docx.Document(file_path)
                content = "\n".join([paragraph.text for paragraph in doc.paragraphs])
            elif extension == "pdf":
                pdf_document = fitz.open(file_path)
                num_pages = pdf_document.page_count
                content = ""

                for page_num in range(num_pages):
                    page = pdf_document.load_page(page_num)
                    content += page.get_text()

            elif extension == "csv":
                df = pd.read_csv(file_path)
                content = df.to_string(index=False)
            elif extension == "xlsx":
                df = pd.read_excel(file_path)
                content = df.to_string(index=False)

            elif extension == "xml":
                tree = ET.parse(file_path)
                root = tree.getroot()
                content = ET.tostring(root, encoding="utf-8").decode("utf-8")

            if content:
                found_keywords = []
                for keyword in self.keywords:
                    if keyword.lower() in content.lower():
                        found_keywords.append(keyword)

                if found_keywords:
                    for keyword in found_keywords:
                        await self.log_found_keyword(file_path, keyword)

        except Exception as e:
            await self.log_error(f"Помилка з файлом: {file_path}\nПомилка: {str(e)}")


async def search_keywords_in_emails(folder_path, log_file, error_file, keywords,
output_folder, save_attachments=False):
    try:
        processed_files = set()
        os.makedirs(output_folder, exist_ok=True)
        tasks = []
        for root, dirs, files in os.walk(folder_path):
            for file_name in files:
                file_path = os.path.join(root, file_name)
                if file_path not in processed_files:
                    if file_name.endswith(".eml"):
                        email_processor = EmailProcessor(file_path, log_file,
error_file, keywords, output_folder)
                        task =
asyncio.create_task(email_processor.process_email(output_folder, save_attachments))
                        tasks.append(task)
                    else:
                        email_processor = EmailProcessor(file_path, log_file,
error_file, keywords, output_folder)
                        task =
asyncio.create_task(email_processor.process_file(file_path))
                        tasks.append(task)
                    processed_files.add(file_path)

            await asyncio.gather(*tasks)

        print(f"Оброблено файлів: {len(processed_files)} в папці: {folder_path}")
    except Exception as e:
        print(f"Помилка в search_keywords_in_emails: {str(e)}")
```

```
if __name__ == '__main__':
    parser = argparse.ArgumentParser(description="пошук приколів")
    parser.add_argument("-f", "--folder", type=str, help="Шлях до листів")
    parser.add_argument("-l", "--log", type=str, help="Шлях для збереження лог-файлу")
    parser.add_argument("-e", "--error", type=str, help="Шлях для збереження лог-файлу
з помилками")
    parser.add_argument("-k", "--keywords-file", type=str, help="Шлях до словника")
    parser.add_argument("-o", "--output-folder", type=str, help="Шлях до теки для
збереження вкладень")
    parser.add_argument("-a", "--attachments", action='store_true',
                        help="Завантажити вкладення (за замовчуванням: False)")
    parser.add_argument("-d", "--extract", action='store_true', help="Запустити
extract.py")

    args = parser.parse_args()
    folder_path = args.folder if args.folder else r"./"
    log_file = args.log if args.log else "log.txt"
    error_file = args.error if args.error else "errors.txt"
    output_folder = args.output_folder if args.output_folder else "attachments"
    save_attachments = args.attachments

    if args.extract and args.folder:
        extract_command = f"python extract.py {args.folder}"
        subprocess.Popen(extract_command, shell=True).wait()

    if args.keywords_file:
        with open(args.keywords_file, "r", encoding="utf-8") as keywords_file:
            keywords = [keyword.strip() for keyword in
keywords_file.read().split(",")]

    start_time = time.time()
    os.makedirs(output_folder, exist_ok=True)
    text_to_display = "Email Parser"
    ascii_art = pyfiglet.figlet_format(text_to_display)
    print(ascii_art)

    asyncio.run(search_keywords_in_emails(folder_path, log_file, error_file, keywords,
output_folder, save_attachments))

    end_time = time.time()
    elapsed_time = end_time - start_time
    print(f"Час роботи скрипта: {elapsed_time} секунд.")
```

database.py

```
import sqlite3
from datetime import datetime
class DatabaseManager:
    def __init__(self, db_path):
        self.db_path = db_path
        self.init_db()

    def init_db(self):
        conn = sqlite3.connect(self.db_path)
        cursor = conn.cursor()
        cursor.execute('''
            CREATE TABLE IF NOT EXISTS requests (
                id INTEGER PRIMARY KEY,
                timestamp TEXT NOT NULL,
                filename TEXT NOT NULL,
                keywords TEXT NOT NULL
            )
        ''')
        conn.commit()
        conn.close()

    async def log_request(self, filename, keywords):
```

```
        conn = sqlite3.connect(self.db_path)
        cursor = conn.cursor()
        cursor.execute('''
            INSERT INTO requests (timestamp, filename, keywords)
            VALUES (?, ?, ?)
        ''', (datetime.now().isoformat(), filename, ','.join(keywords)))
        conn.commit()
        conn.close()
```

ganerate.log

```
def generate_html_log(response_data):
    styles = """
    <style>
        body {
            font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
            margin: 0;
            padding: 0;
            background-color: #f4f4f4;
            color: #333;
        }
        .container {
            width: 80%;
            margin: auto;
            overflow: hidden;
        }
        header {
            background: #50b3a2;
            color: #fff;
            padding-top: 30px;
            min-height: 70px;
            border-bottom: #e8491d 3px solid;
        }
        header a {
            color: #ffffff;
            text-decoration: none;
            text-transform: uppercase;
            font-size: 16px;
        }
        header ul {
            padding: 0;
            margin: 0;
            list-style: none;
            overflow: hidden;
        }
        header li {
            float: left;
            display: inline;
            padding: 0 20px 0 20px;
        }
        header #branding {
            float: left;
        }
        header #branding h1 {
            margin: 0;
        }
        header nav {
            float: right;
            margin-top: 10px;
        }
        header .highlight, header .current a {
            color: #e8491d;
            font-weight: bold;
        }
        header a:hover {
```

```
            color: #ffffff;
            font-weight: bold;
        }
        .log-entry {
            background-color: #e7ffe7;
            padding: 10px;
            margin-bottom: 10px;
            border-left: 6px solid #2ecc71;
        }
        .error-entry {
            background-color: #ffebeb;
            padding: 10px;
            margin-bottom: 10px;
            border-left: 6px solid #e74c3c;
        }
    </style>
    """

    html_log = f"<html><head><title>Log Output</title>{styles}</head><body>"
    html_log += "<header><div class='container'><h1>Log Output</h1></div></header>"
    html_log += "<div class='container'>"

    if 'log' in response_data:
        log_entries = response_data['log'].split('\n')
        for entry in log_entries:
            if entry.strip():
                html_log += f"<div class='log-entry'>{entry}</div>"

    if 'errors' in response_data and response_data['errors'].strip():
        html_log += f"<div class='error-entry'>{response_data['errors']}</div>"

    html_log += "</div></body></html>"
    return html_log
```