НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
(повна назва інституту/факультету)

Кафедра інформатики та програмної інженерії
(повна назва кафедри)

«До захисту допущено»
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2025 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інженерія програмного забезпечення інформаційних систем»

спеціальності «121 Інженерія програмного забезпечення»

на тему: Вебзастосунок для пошуку потенційних B2B клієнтів шляхом вебскрапінгу мережі інтернет

Виконав студент IV курсу, групи ІП-12
(шифр групи)
Йолкін Даніїл Сергійович _____
(прізвище, ім'я, по батькові) (підпис)

Керівник доцент, к.т.н., доц., Ліхоузова Т. А. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант ст. викл., д-р філософії, Головченко М. М. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент доцент, к.т.н., доц., Солдатова М.О. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.

Студент _____
(підпис)

Київ – 2025

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 Інженерія програмного забезпечення

Освітньо-професійна програма – Інженерія програмного забезпечення інформаційних систем

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2025 р.

ЗАВДАННЯ
на дипломний проєкт студенту

Йолкін Даніїл Сергійович
(прізвище, ім'я, по батькові)

1. Тема проєкту Вебзастосунок для пошуку B2B клієнтів шляхом вебскрапінгу мережі інтернет

керівник проєкту Ліхоузова Тетяна Анатоліївна, к.т.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «23» травня 2025 р. №1705-с

2. Термін подання студентом проєкту «16» червня 2025 року

3. Вихідні дані до проєкту: технічне завдання

4. Зміст пояснювальної записки
1) Обстеження предметної області: аналіз предметної області, аналіз існуючих рішень, моделювання бізнес-процесів.
2) Розроблення вимог для програмного забезпечення: опис варіантів використання, розроблення вимог, аналіз економічних показників.
3) Конструювання та розроблення програмного забезпечення: архітектура програмного забезпечення, конструювання програмного забезпечення, аналіз безпеки даних.
4) Аналіз якості та тестування програмного забезпечення: аналіз якості, опис процесів тестування, опис контрольного прикладу.
5) Розгортання та супровід програмного забезпечення: розгортання програмного забезпечення, супровід програмного забезпечення.

5. Перелік графічного матеріалу

1) Схема структурна варіантів використань

2) Схема структурна компонентів програмного забезпечення

3) Схема бази даних

4) Креслення вигляду екранних форм

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання «15» березня 2025 року

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Вивчення рекомендованої літератури	15.03.2025	
2	Аналіз існуючих методів розв’язання задачі	19.04.2025	
3	Постановка та формалізація задачі	21.04.2025	
4	Розробка інформаційного забезпечення	26.04.2025	
5	Алгоритмізація задачі	30.04.2025	
6	Обґрунтування вибору використаних технічних засобів	05.05.2025	
7	Розробка програмного забезпечення	10.05.2025	
8	Налагодження програми	14.05.2025	
9	Виконання графічних документів	20.05.2025	
10	Оформлення пояснювальної записки	29.05.2025	
11	Подання ДП на попередній захист	02.06.2025	
12	Подання ДП рецензенту	10.06.2025	
13	Подання ДП на основний захист	16.06.2025	

Студент _____ Даніїл ЙОЛКІН
(підпис) (ініціали, прізвище)

Керівник _____ Тетяна ЛІХОУЗОВА
(підпис) (ініціали, прізвище)

АНОТАЦІЯ

Пояснювальна записка дипломного проєкту складається з п’яти розділів, містить 36 таблиць, 15 рисунків та 14 джерел – загалом 78 сторінок.

Дипломний проєкт присвячений розробці та обґрунтуванню інформаційної системи Sieve, призначеної для автоматизованого пошуку, збору та первинного аналізу даних про потенційних B2B-клієнтів з використанням технологій вебскрапінгу.

Мета даної роботи полягала у створенні гнучкого, масштабованого та безпечного вебзастосунку, що дозволяє суттєво підвищити ефективність процесу генерації потенційних клієнтів та бізнес-партнерів на міжнародних ринках, особливо у нішових або малодосліджених галузях.

У розділі 1 розглянуто теоретичні та практичні передумови створення системи. Проведено детальне передпроєктне обстеження предметної області, проаналізовано існуючі проблеми B2B-лідогенерації, а також оглянуто наявні на ринку програмні продукти та технічні рішення, що обґрунтовує актуальність та доцільність розробки нового інструменту.

Розділ 2 присвячений розробці вимог до програмного забезпечення. На основі визначених варіантів використання системи було сформульовано детальні функціональні та нефункціональні вимоги, проведено попередній аналіз економічних аспектів та сформовано фінальну постановку завдання на розробку програмного продукту Sieve.

У розділі 3 описано процес конструювання та безпосередньої розробки програмного забезпечення. Детально розглянуто спроектовану архітектуру системи, обґрунтовано вибір технологічного стеку, що включає Python, Flask, PostgreSQL та SQLAlchemy. Реалізовано ключові компоненти системи: механізми автентифікації, управління запитами, асинхронну обробку даних, а також розроблено аналітичне ядро – алгоритм Sieve. Додатково проведено аналіз аспектів безпеки даних.

Розділ 4 присвячений аналізу якості та тестуванню розробленого програмного продукту. Було проведено оцінку відповідності системи заявленим нефункціональним вимогам за допомогою обґрунтованих метрик, описано

стратегію та процеси тестування, розроблено набір тест-кейсів та представлено контрольний приклад, що наочно демонструє роботу ключового функціоналу системи.

У розділі 5 розглянуто практичні аспекти життєвого циклу програмного забезпечення. Описано сучасний підхід до розгортання системи з використанням технології контейнеризації Docker та Docker Compose, а також запропоновано комплексну стратегію подальшого супроводу, включаючи моніторинг, управління оновленнями через CI/CD конвеєр та регулярне оновлення безпеки.

Програмне забезпечення впроваджено у вигляді функціонального прототипу. Його працездатність та практична цінність були підтверджені в рамках контрольного тестування та консультаційної співпраці.

КЛЮЧОВІ СЛОВА: ВЕБЗАСТОСУНОК, FLASK, POSTGRESQL, ВЕБСКРАПІНГ, B2B, ГЕНЕРАЦІЯ ЛІДІВ, АСИНХРОННА ОБРОБКА, DOCKER, АНАЛІЗ ДАНИХ.

ABSTRACT

The explanatory note of the diploma project consists of five sections, contains 36 tables, 15 figures and 14 sources – in total 78 pages.

The purpose of the diploma project is to the develop and substantiate the Sieve information system, which is designed for the automated search, collection, and primary analysis of data on potential B2B clients using web scraping technologies.

The goal of this work was to create a flexible, scalable, and secure web application that significantly increases the efficiency of generating potential clients and business partners in international markets, especially in niche or under-researched industries.

Chapter 1 reviews the theoretical and practical prerequisites for the system's creation. A detailed preliminary investigation of the subject area was conducted, existing problems in B2B lead generation were analyzed, and available software products and technical solutions on the market were reviewed, which substantiates the relevance and feasibility of developing a new tool.

Chapter 2 is dedicated to the development of software requirements. Based on defined use cases for the system, detailed functional and non-functional requirements were formulated, a preliminary analysis of economic aspects was performed, and a final task statement for the development of the Sieve software product was formed.

Chapter 3 describes the process of software design and direct development. The system's architecture was detailed, the choice of the technology stack—including Python, Flask, PostgreSQL, and SQLAlchemy—was justified. Key system components were implemented, including authentication mechanisms, query management, asynchronous data processing, and the analytical core—the Sieve algorithm. Additionally, a data security analysis was conducted.

Chapter 4 is dedicated to the quality analysis and testing of the developed software product. The system's compliance with the stated non-functional requirements was assessed using justified metrics, a testing strategy and processes were described, a set of test cases was developed, and a control example was presented to clearly demonstrate the system's key functionality.

Chapter 5 examines the practical aspects of the software life cycle. A modern approach to system deployment using Docker and Docker Compose containerization technology is described, and a comprehensive strategy for ongoing maintenance is proposed, including monitoring, update management via a CI/CD pipeline, and regular security updates.

The software has been implemented in the form of a functional prototype. Its workability and practical value have been confirmed through control testing and consulting collaboration.

KEYWORDS: WEB APPLICATION, FLASK, POSTGRESQL, WEB SCRAPING, B2B, LEAD GENERATION, ASYNCHRONOUS PROCESSING, DOCKER, DATA ANALYSIS.

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	КПІ.ІП-1212.045440.01.91	Технічне завдання	14	
3	А4	КПІ.ІП-1212.045440.02.81	Пояснювальна записка	78	
4	А4	КПІ.ІП-1212.045440.03.12	Текст програми	42	
5	А4	КПІ.ІП-1212.045440.04.51	Програма та методика тестування	6	
6	А4	КПІ.ІП-1212.045440.05.34	Керівництво користувача	9	
7	А3	КПІ.ІП-1212.045440.06.99	Графічний матеріал	5	

Змін.	Арк.	№ докум.	Підп.	Дата	КПІ.ІП-1212.045440.00.90		
Розроб.		Йолкін Д. С.			Відомість дипломного проєкту	Літ.	Аркуш 1 / Аркушів 1
Перевір.		Ліхоузова Т. А.				КПІ ім. Ігоря Сікорського ФІОТ каф. ІПІ гр. ІП-12	
Н.контр.		Головченко М.М.					
Затв.		Жаріков Е.В.					

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”

Завідувач кафедри

_____ Едуард ЖАРІКОВ

“___” _____ 2025 р.

ВЕБЗАСТОСУНОК ДЛЯ ПОШУКУ B2B КЛІЄНТІВ ШЛЯХОМ ВЕБСКРАПІНГУ МЕРЕЖІ ІНТЕРНЕТ

Технічне завдання

КПІ.ІП-1212.045440.01.91

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Тетяна ЛІХОУЗОВА

Нормоконтроль:

_____ Максим ГОЛОВЧЕНКО

Виконавець:

_____ Даніїл ЙОЛКІН

Київ – 2025

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: Вебзастосунок для пошуку B2B клієнтів шляхом вебскрапінгу мережі інтернет.

Галузь застосування:

Наведене технічне завдання поширюється на розробку вебзастосунку для пошуку B2B клієнтів шляхом вебскрапінгу мережі інтернет [045440], котра використовується для автоматизації процесу генерації B2B-лідів шляхом збору, агрегації та первинного аналізу даних про компанії з відкритих вебджерел за критеріями, визначеними користувачем, та призначена для використання у відділах продажів, маркетингу та бізнес-девелопменту компаній різного масштабу, зокрема стартапів та підприємств, що виходять на нові або нішові міжнародні ринки.

2 ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки вебзастосунку для пошуку B2B клієнтів шляхом вебскрапінгу мережі інтернет є завдання на дипломне проєктування, затверджене кафедрою інформатики та програмної інженерії Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

3 ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для автоматизованого пошуку, збору та первинної аналітичної обробки даних про потенційних B2B-клієнтів з відкритих джерел мережі Інтернет. В експлуатаційному плані програмний виріб є вебзастосунком, що надає кінцевим користувачам (співробітникам відділів продажів, маркетингу, бізнес-девелопменту) зручний графічний інтерфейс для формування та управління пошуковими запитами, моніторингу їх виконання в асинхронному режимі, а також отримання та завантаження структурованих результатів у вигляді файлів для подальшого використання.

Метою даного дипломного проєкту є підвищити ефективність пошуку потенційних B2B клієнтів.

Мета досягається шляхом розробки вебзастосунку, який дозволяє автоматизовано формувати базу потенційних B2B клієнтів шляхом збору, фільтрації та обробки інформації з відкритих джерел мережі Інтернет. Програмне забезпечення має забезпечити користувачу інструмент для пошуку компаній за ключовими словами, аналізу отриманих результатів та експорту релевантних даних у зручному форматі.

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Вимоги до функціональних характеристик

Програмне забезпечення повинно забезпечувати виконання наступних основних функцій

4.1.1 Користувацького інтерфейсу

– Навігація по сторінках вебзастосунку, включаючи перехід на головну сторінку, сторінки реєстрації, автентифікації та управління запитами.

– Управління доступом користувача, що включає реєстрацію, вхід та вихід з системи.

– Формування пошукових запитів, що дозволяє користувачеві вводити критерії пошуку, такі як ключові слова,країна та специфічні слова-фільтри.

– Моніторинг завдань, що надає можливість переглядати історію власних запитів та відстежувати їхні поточні статуси виконання.

– Отримання результатів, що забезпечує завантаження фінального звіту у форматі XLSX для завершених запитів.

Прототипи екранних форм зображені на рисунку (рисунок 4.1):

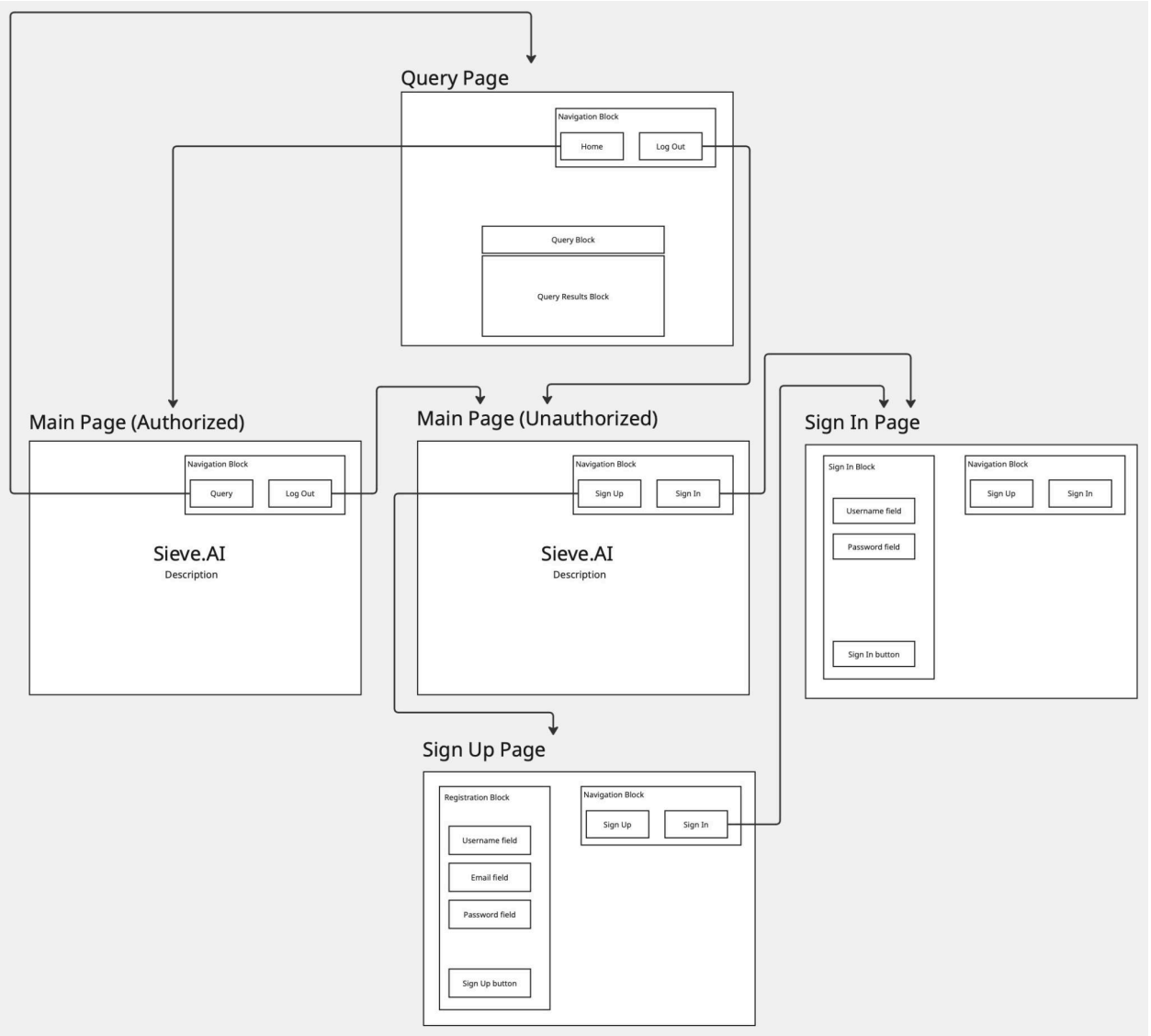


Рисунок 4.1 – Прототипи екранних форм

4.1.2 Облік користувачів

Для ролі "Авторизований користувач":

– Реєстрація в системі за допомогою унікального імені користувача, адреси електронної пошти та пароля.

– Автентифікація в системі з використанням імені користувача та пароля.

– Вихід із системи (завершення сесії).

– Створення нового пошукового запиту шляхом заповнення форми з полями "Ключові слова", "Країна" та "Whitelist Words".

– Перегляд історії власних пошукових запитів зі всіма їхніми параметрами.

– Перегляд поточного статусу кожного власного запиту ("submitted", "processing", "completed", "failed").

– Завантаження файлу результатів у форматі XLSX для запитів, що мають статус "completed".

4.2 Вимоги до надійності

В системі передбачено контроль введення інформації та захист від некоректних дій користувача. Це реалізується на рівні серверної логіки за допомогою валідаторів для всіх полів форм, що приймають дані від користувача. Система перевіряє дані на обов'язковість заповнення, відповідність формату (наприклад, для email), довжину (для паролів та імен користувачів) та інші критерії. У разі введення некоректних даних система не переходить у непередбачуваний стан, а інформує користувача про помилку, запобігаючи запису невалідних даних у базу. Забезпечується цілісність інформації в базі даних. Це досягається за допомогою механізмів реляційної бази даних PostgreSQL, зокрема через використання зовнішніх ключів (Foreign Key constraints), що підтримують зв'язки між таблицями user, search_query та search_result. Крім того, всі операції запису до бази даних виконуються в рамках транзакцій, що гарантує їх атомарність. У коді додатку це реалізується через використання сесій SQLAlchemy, де успішні операції підтверджуються (db.session.commit()), а у разі виникнення помилки всі зміни відкочуються (db.session.rollback()), запобігаючи збереженню часткових або некоректних даних.

4.3 Умови експлуатації

Умови експлуатації згідно СанПін 2.2.2.542 – 96.

4.3.1 Вид обслуговування

Вимоги до виду обслуговування не висуваються

4.3.2 Обслуговуючий персонал

Вимоги до обслуговуючого персоналу не висуваються

4.4 Вимоги до складу і параметрів технічних засобів

Програмне забезпечення повинно функціонувати на IBM-сумісних персональних комп‘ютерах.

Мінімальна конфігурація технічних засобів:

– тип процесору: Intel Core i5;
– об‘єм ОЗП: 4 Гб;
– підключення до мережі Інтернет зі швидкістю від 20 мегабіт;

Рекомендована конфігурація технічних засобів:

– тип процесору: Apple M1;
– об‘єм ОЗП: 8 Гб;
– підключення до мережі Інтернет зі швидкістю від 1000 мегабіт;

4.5 Вимоги до інформаційної та програмної сумісності

Програмне забезпечення повинно працювати під управлінням операційних систем сімейства WIN32 (Windows XP, Windows NT і т.д.) або Unix.

4.5.1 Вимоги до вхідних даних

Вхідні дані повинні бути представлені в наступному форматі: інформація, що вводиться користувачем інтерактивно через графічний вебінтерфейс застосунку.

Це включає:

– Дані для реєстрації та автентифікації: текстові рядки для імені користувача, адреси електронної пошти та пароля.

– Параметри пошукового запиту: текстовий рядок з ключовими словами для пошуку, значення країни, обране з визначеного списку, та текстовий рядок зі списком "білих слів", розділених комою.

Система повинна забезпечувати валідацію цих даних на відповідність очікуваному формату та повноті перед обробкою.

4.5.2 Вимоги до вихідних даних

Результати повинні бути представлені в наступному форматі:

– Інформація в інтерфейсі користувача: система повинна відображати актуальні статуси обробки для кожного пошукового запиту ("submitted", "processing", "completed", "failed"), а також надавати контекстні повідомлення про успішне виконання операцій або помилки.

– Файл з результатами аналізу: основний результат роботи алгоритму повинен бути представлений у вигляді структурованого файлу формату Microsoft Excel (XLSX), доступного для завантаження користувачем. Цей файл має містити таблицю з переліком знайдених та проаналізованих потенційних B2B-клієнтів та пов'язану з ними інформацію.

4.5.3 Вимоги до мови розробки

Розробку виконати на мові програмування Python, версії 3.9 або вище. Вибір обґрунтований широкою підтримкою необхідних бібліотек для веброзробки, аналізу даних та автоматизації.

4.5.4 Вимоги до середовища розробки

Розробку виконати на платформі (за допомогою середовища), що включає наступні основні компоненти:

– Інтегроване середовище розробки (IDE): наприклад, JetBrains PyCharm або Visual Studio Code з відповідними розширеннями для Python.

– Основний фреймворк: вебфреймворк Flask з використанням розширень Flask-SQLAlchemy, Flask-Login, Flask-WTF.

– Система управління базами даних: PostgreSQL.
– Система контролю версій: Git.

4.5.5 Вимоги до представленню вихідних кодів

Вихідний код програми має бути представлений у вигляді структурованої сукупності файлів та директорій, що відповідають архітектурі Flask-додатку та відокремленого модуля алгоритму algorithm_app. Проєкт має бути організований у вигляді репозиторію системи контролю версій Git, що забезпечує відстеження змін.

4.6 Вимоги до маркування та пакування

Вимоги до маркування та пакування не висуваються.

4.7 Вимоги до транспортування та зберігання

Вимоги до транспортування та зберігання не висуваються.

4.8 Спеціальні вимоги

Спеціальні вимоги не висуваються.

5 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

5.1 Попередній склад програмної документації

У склад супроводжувальної документації повинні входити наступні документи на аркушах формату А4:

- пояснювальна записка;
- технічне завдання;
- текст програми;
- програма та методика тестування;
- керівництво користувача.

Графічна частина повинна бути виконана на аркушах формату А3 та містити наступні документи:

- Схема структурна варіантів використань.
- Схема структурна компонентів програмного забезпечення.
- Схема бази даних.
- Креслення вигляду екранних форм.

5.2 Спеціальні вимоги до програмної документації

Програмні модулі, котрі розробляються, повинні бути задокументовані, тобто тексти програм повинні містити всі необхідні коментарі.

6 СТАДІЇ І ЕТАПИ РОЗРОБКИ

№	Назва етапу	Строк	Звітність
1.	Вивчення рекомендованої літератури	15.03.2025	
2.	Аналіз існуючих методів розв’язання задачі	19.04.2025	Порівняльна таблиця існуючих методів
3.	Постановка та формалізація задачі	21.04.2025	Технічне завдання
4.	Розробка інформаційного забезпечення	26.04.2025	Схема структурна програмного забезпечення та специфікація компонентів
5.	Алгоритмізація задачі	30.04.2025	Схема алгоритму
6.	Обґрунтування вибору використаних технічних засобів	05.05.2025	Технічна документація
7.	Розробка програмного забезпечення	10.05.2025	Програмне забезпечення
8.	Налагодження програми	14.05.2025	Тести, результати тестування
9.	Виконання графічних документів	20.05.2025	Графічні матеріали проєкту
10.	Оформлення пояснювальної записки	29.05.2025	Пояснювальна записка
11.	Подання ДП на попередній захист	02.06.2025	
12.	Подання ДП рецензенту	10.06.2025	

7 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

Тестування розробленого програмного продукту виконується відповідно до “Програми та методики тестування”.

Пояснювальна записка
до дипломного проєкту

на тему: **Вебзастосунок для пошуку B2B клієнтів шляхом вебскрапінгу мережі інтернет**

КПІ.ІП-1212.045440.02.81

Київ – 2025

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

B2B	–	Business-to-Business
B2C	–	Business-to-Consumer
WS	–	Web Scraping
CSV	–	Comma-Separated Value (файловий формат)

ВСТУП

Проблема ефективного виходу на міжнародні ринки та пошуку нових бізнес-партнерів (B2B-лідів) є однією з ключових для сучасних компаній, що прагнуть масштабування та глобальної присутності. Особливої гостроти це питання набуває в умовах динамічної зміни ринкової кон'юнктури та обмежених ресурсів, що характерно для малого та середнього бізнесу, а також для компаній, що оперують у нішових або експериментальних галузях. Поштовхом до даного дослідження та розробки стала практична потреба, виявлена під час консультаційної співпраці з одним із відділів провідної української продуктової IT-компанії у липні 2024 року. Ключовим завданням, що постало перед компанією, була розробка ефективної стратегії пошуку потенційних клієнтів та бізнес-партнерів у специфічній сфері діяльності на закордонному ринку.

Традиційні підходи до генерації B2B-лідів, такі як участь у міжнародних конференціях, використання баз даних компаній (наприклад, Crunchbase, Apollo.io), що агрегують інформацію з відкритих реєстрів та даних фондових ринків, виявилися недостатньо ефективними для вирішення поставленої задачі. Основними обмежуючими факторами стали нішовий характер цільової сфери, що зумовлювало обмеженість даних у традиційних базах, експериментальний та недостатньо досліджений стан ринку, а також низька медійна присутність більшості його гравців, які переважно є локальними компаніями. Ці обставини вимагали застосування новаторських підходів до збору та аналізу ринкової інформації.

В результаті аналізу альтернативних методів, увагу було зосереджено на потенціалі технологій вебскрапінгу та методів інтелектуального аналізу даних. Попередній аналіз ринку існуючих рішень продемонстрував, що поєднання вебскрапінгу з можливостями сучасних технологій обробки даних (з перспективою інтеграції алгоритмів штучного інтелекту) є не просто тимчасовою тенденцією, а потужним інструментом, здатним забезпечити якісно новий рівень пошуку B2B-клієнтів. Особливої значущості набув підхід, що дозволяє ідентифікувати компанії, які не представлені у традиційних базах

даних, але активно функціонують у своїх нішах та мають цифрову присутність в мережі Інтернет – у вигляді власних вебсайтів, публікацій у локальних бізнес-каталогах або згадок у відкритих джерелах.

Актуальність теми даної дипломної роботи зумовлена зростаючою потребою бізнесу в ефективних, гнучких та економічно виправданих інструментах для пошуку партнерів на міжнародних ринках. Традиційні методи часто не встигають адаптуватися до швидких змін цифрового бізнес-середовища та специфіки нішових ринків. У цьому контексті, застосування автоматизованого збору даних з вебресурсів у поєднанні з інтелектуальними алгоритмами для їх подальшої обробки, класифікації, фільтрації та ранжування видається надзвичайно перспективним напрямом.

Провідні дослідницькі установи та комерційні компанії останніми роками активно вивчають та впроваджують рішення в сфері інтелектуального збору даних з відкритих джерел. Наприклад, Stanford University та MIT досліджують нові підходи до розуміння змісту вебсторінок, а стартапи на кшталт Diffbot чи Clearbit створюють комерційні API, які дозволяють витягувати структуровані дані про компанії з неструктурованого вебконтенту.

У рамках цієї дипломної роботи було розроблено вебзастосунок, призначений для пошуку B2B клієнтів шляхом автоматизованого вебскрапінгу інтернету. Основна мета – підвищити ефективність пошуку потенційних партнерів шляхом створення динамічного та масштабованого інструменту, здатного працювати з широким діапазоном галузей і регіонів. Вебзастосунок дозволяє здійснювати:

– збір та агрегацію інформації про компанії з відкритих вебджерел;

– фільтрацію та класифікацію знайдених компаній відповідно до заданих бізнес-критеріїв;

– експорт результатів у форматі CSV.

Таким чином, практичне значення роботи полягає у створенні інструменту, який дозволяє компаніям автоматизувати процес генерації лідів без необхідності у дорогих підписках чи вручну зібраних базах даних. Отримане рішення може

бути використане стартапами, маркетинговими агенціями, sales-відділами та консультантами з бізнес-розвитку.

Результати роботи можуть знайти своє застосування у таких сферах, як B2B продажі, міжнародний бізнес-девелопмент, маркетинг-аналітика, а також у дослідженнях бізнес-ландшафтів для венчурного інвестування та стратегічного планування.

1 ПЕРЕДПРОЄКТНЕ ОБСТЕЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Постановка завдання дипломного проєктування

У рамках дипломного проєктування необхідно реалізувати повний цикл розробки вебзастосунку для пошуку B2B клієнтів шляхом вебскрапінгу мережі Інтернет. Для досягнення поставленої мети слід виконати такі основні завдання:

– Провести аналіз предметної області, виявити основні проблеми та обмеження традиційних рішень для пошуку B2B клієнтів.

– Дослідити сучасні методи та інструменти вебскрапінгу, зокрема підходи до пошуку компаній у відкритих джерелах, обробки HTML-структури та захисту від блокування.

– Вивчити існуючі рішення на ринку, визначити їх переваги та недоліки, а також виявити можливості для інноваційного підходу.

– Спроєктувати архітектуру вебзастосунку, включаючи компоненти збору даних, обробки, фільтрації, класифікації та представлення результатів користувачу.

– Розробити систему автоматизованого збору даних про компанії, яка б підтримувала масштабованість, розподілену обробку та обхід обмежень сайтів.

– Реалізувати алгоритми фільтрації та класифікації даних, що дозволяють визначати релевантність знайдених компаній згідно заданих параметрів (географія, сфера діяльності, тип компанії тощо).

– Розробити інтерфейс користувача вебзастосунку, який забезпечує введення критеріїв пошуку, перегляд результатів та їх експорт у зручному форматі.

– Реалізувати можливість експорту даних у XLSX форматі.

– Забезпечити базову авторизацію користувачів для захисту доступу до функціоналу застосунку.

– Провести тестування розробленої системи на реальних прикладах пошуку потенційних клієнтів у вибраних галузях та регіонах.

– Оцінити ефективність реалізованого рішення, порівнявши його з альтернативними підходами.

– Оформити технічну документацію та пояснювальну записку до дипломного проєкту відповідно до встановлених вимог.

1.2 Аналіз предметної області

У сучасному цифровому світі B2B сектор відіграє ключову роль у глобальній економіці. B2B-взаємодія охоплює процеси співпраці між компаніями, зокрема у сферах постачання, логістики, консалтингу, IT, маркетингу тощо [1]. На відміну від B2C, де основним споживачем є кінцевий клієнт, B2B-модель вимагає глибшого аналізу потреб партнерів, налагодження тривалих ділових стосунків та часто – індивідуального підходу до кожного клієнта.

Одним з головних завдань компаній, що працюють у B2B-сфері, є ефективний пошук нових клієнтів і партнерів. Традиційно для цього використовуються такі інструменти як:

– CRM-системи (наприклад, Salesforce, Zoho CRM);

– бази даних компаній (як-от Crunchbase, Apollo.io, ZoomInfo) [2];

– професійні соціальні мережі (LinkedIn) [3];

– бізнес-конференції та галузеві заходи.

Існуючі IT-рішення для пошуку B2B клієнтів базуються переважно на фіксованих джерелах даних: готових базах, ручному зборі контактів з конференцій або імпорту даних з онлайн-реєстрів. Подібні системи зазвичай вимагають підписки або ліцензії, що робить їх недоступними для малого та середнього бізнесу.

Більшість таких сервісів обмежена попередньо визначеним переліком компаній та їхніх атрибутів. Через це вони не враховують динамічність ринку – появу нових гравців, стартапів, нішових компаній, або тих, що активно розвивають лише онлайн-присутність без реєстрації у традиційних базах [4].

Іншим підходом до вирішення проблеми пошуку нових клієнтів є використання вебскрапінгу – автоматизованого процесу збору інформації з вебсайтів. У поєднанні з алгоритмами фільтрації та штучного інтелекту, вебскрапінг дозволяє виявляти компанії, які відповідають заданим критеріям, навіть якщо вони не представлені у загальнодоступних базах. Популярні фреймворки для реалізації вебскрапінгу – Scrapy, BeautifulSoup, Playwright, Selenium – дозволяють автоматизувати збір контенту навіть зі складних, динамічно згенерованих вебсторінок [5].

Незважаючи на технічний прогрес, існує кілька проблем у B2B-пошуку:

– неефективне охоплення нішевих або локальних компаній, які не представлені у відкритих реєстрах;

– висока вартість доступу до якісних баз даних;

– неможливість гнучкого налаштування критеріїв пошуку, особливо коли йдеться про специфічні сфери діяльності;

– обмежена інтеграція з системами аналітики та CRM;

– фіксований обсяг оновлень та відсутність динамічного моніторингу ринку.

З огляду на вищевказані проблеми, активно розвиваються нові підходи:

– індивідуальний вебскрапінг як альтернатива фіксованим базам даних;

– інтеграція з LLM/AI для аналізу змісту вебсайтів і визначення їх належності до певної сфери;

– інфраструктурна автоматизація процесу збору даних за допомогою контейнеризації (Docker) і розподілених систем обробки;

– інтеграція з BI/CRM-платформами для автоматичного поповнення баз даних потенційних клієнтів [6].

У рамках даного дипломного проєкту обрано інноваційний підхід, який полягає у поєднанні:

– сучасних методів вебскрапінгу для збору інформації про компанії з відкритих джерел;

– алгоритмів класифікації компаній за змістом їхніх сайтів;

– побудови вебзастосунку, який дозволяє кінцевому користувачеві задавати параметри пошуку, отримувати релевантні результати, та експортувати їх.

Такий підхід дозволяє зробити пошук B2B клієнтів більш гнучким, доступним і точним, особливо у випадках роботи з нішевими ринками та регіональними напрямами діяльності.

1.3 Аналіз існуючих рішень

Проаналізуємо відоме на сьогодні алгоритмічне забезпечення у даній області та технічні рішення, що допоможуть у реалізації вебзастосунку для пошуку B2B клієнтів. Далі будуть розглянуті готові програмні рішення, допоміжні програмні засоби та засоби розробки.

1.3.1 Аналіз відомих програмних продуктів

У межах предметної області – автоматизованого пошуку B2B-клієнтів — існує ряд вже реалізованих програмних рішень. Серед найбільш популярних продуктів – Crunchbase, Similarweb, Apollo.io, Salesforce та LinkedIn. Усі ці сервіси мають власні підходи до збирання, зберігання та візуалізації бізнес-даних.

Crunchbase – це база даних про компанії, стартапи, інвесторів та угоди між ними [7]. Основним призначенням сервісу є пошук потенційних клієнтів, конкурентів або партнерів.

До основних функцій продукту належать:

– пошук компаній за галузями, локацією, інвестиційною активністю тощо;

– експорт результатів у CSV;

– інтеграція з CRM;

– детальні профілі компаній (керівництво, контакти, дата створення, інвестиції тощо).

На рисунку 1.1 зображено інтерфейс Crunchbase вебсайту.

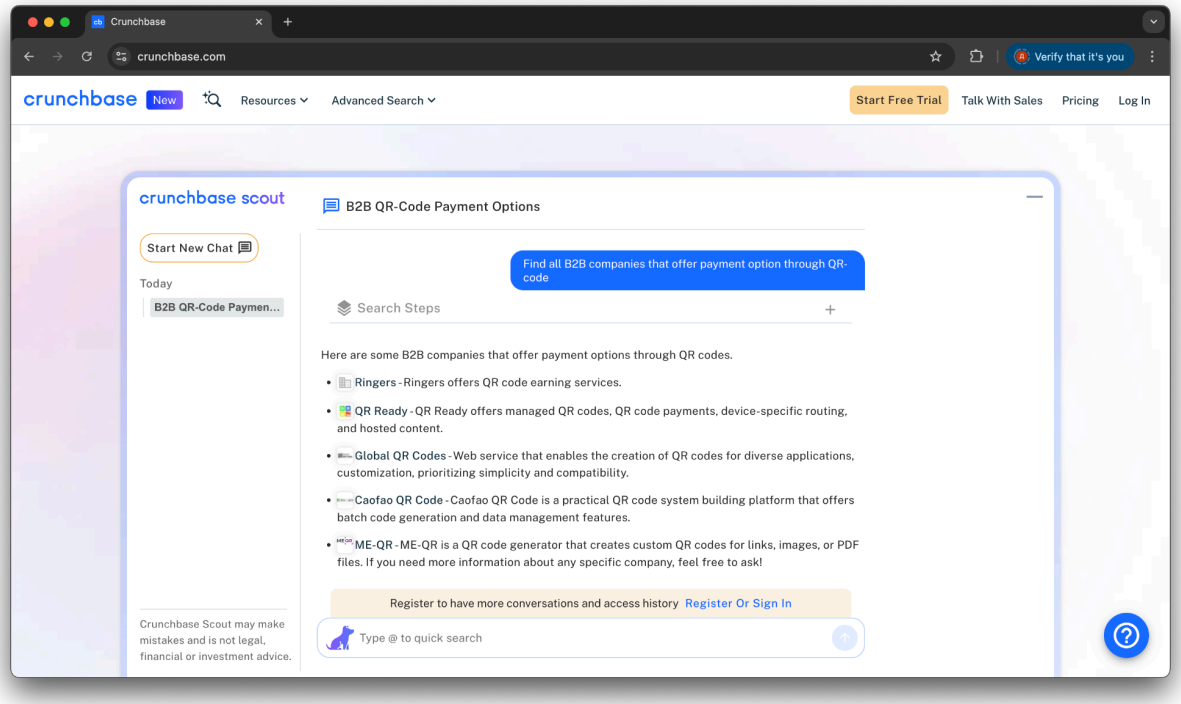


Рисунок 1.1 – Інтерфейс Crunchbase

Similarweb – сервіс для аналізу трафіку вебсайтів, який дозволяє отримувати детальну інформацію про джерела трафіку, аудиторію та конкурентів [8].

До основних функцій продукту належать:

– аналіз вебтрафіку компаній;
– визначення схожих сайтів;
– географічна аналітика користувачів;
– оцінка ключових джерел відвідуваності.

На рисунку 1.2 зображено інтерфейс Similarweb вебсайту.

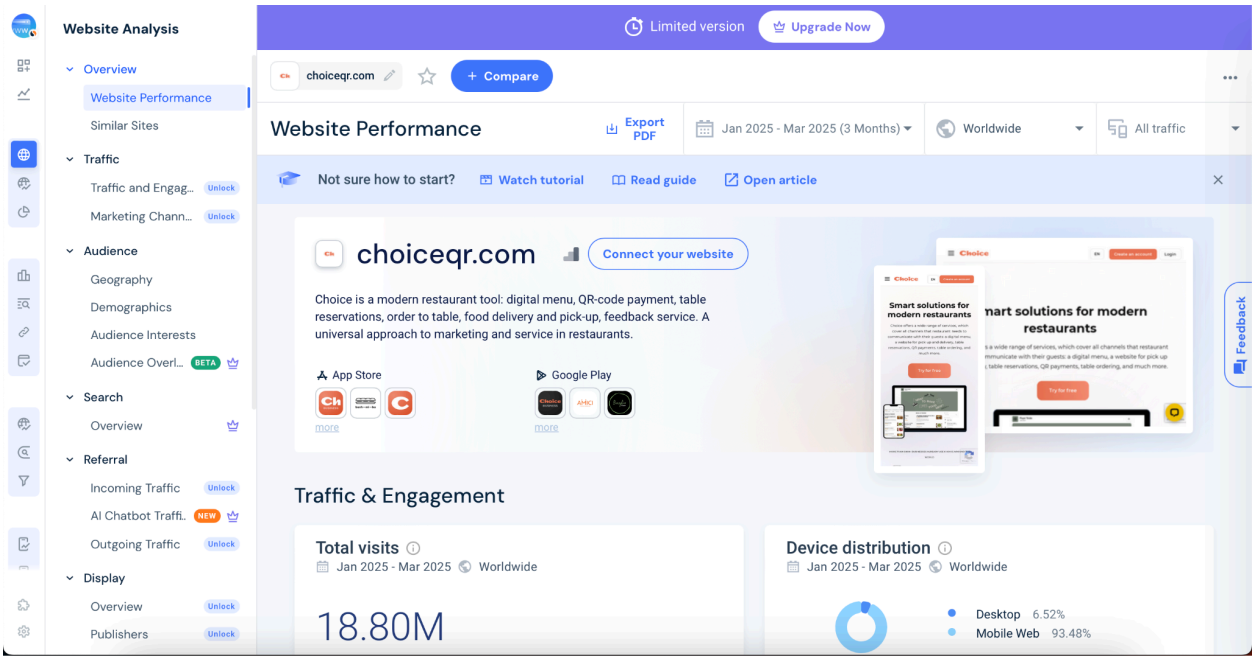


Рисунок 1.2 – Інтерфейс Similarweb

Apollo.io – це повноцінна платформа для генерації B2B-лідів з базою понад 250 млн контактів [9].

До основних функцій продукту належать:

– фільтрація за ролями, індустріями, локаціями;

– доступ до корпоративних email-адрес;

– вбудоване надсилання e-mail кампаній;

– інтеграція з Salesforce, HubSpot, Outreach.

На рисунку 1.3 зображено інтерфейс Apollo.io вебсайту.

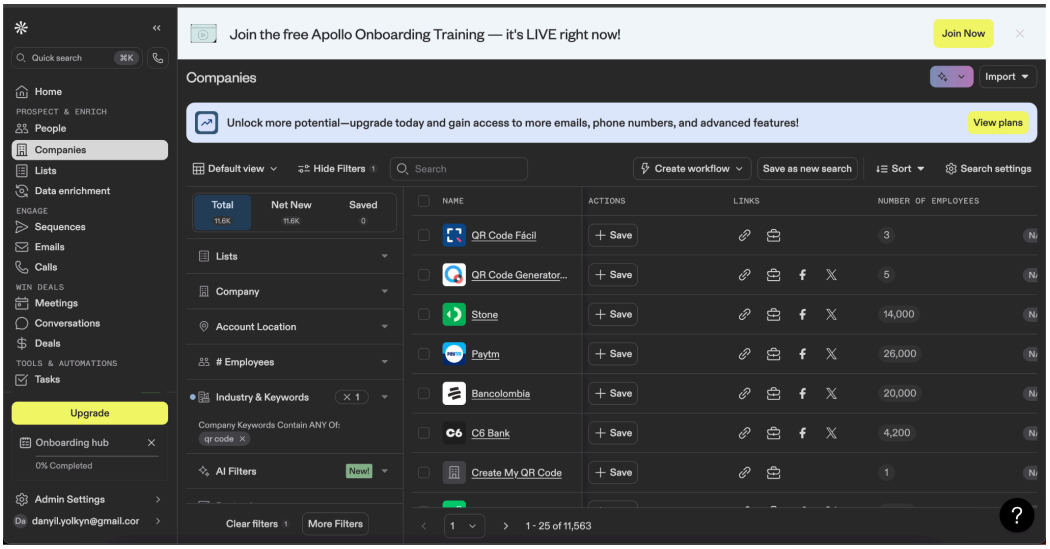


Рисунок 1.3 – Інтерфейс Apollo.io

Salesforce – одна з найпопулярніших CRM-систем у світі. Хоча вона не надає відкритого доступу до баз компаній, вона дозволяє управляти потенційними клієнтами та процесом продажів [10].

До основних функцій продукту належать:

– управління лідами, контактами, можливостями;
– побудова воронок продажів;
– інтеграції з зовнішніми сервісами;
– аналітика та звітність.

На рисунку 1.4 зображено інтерфейс Salesforce вебсайту.

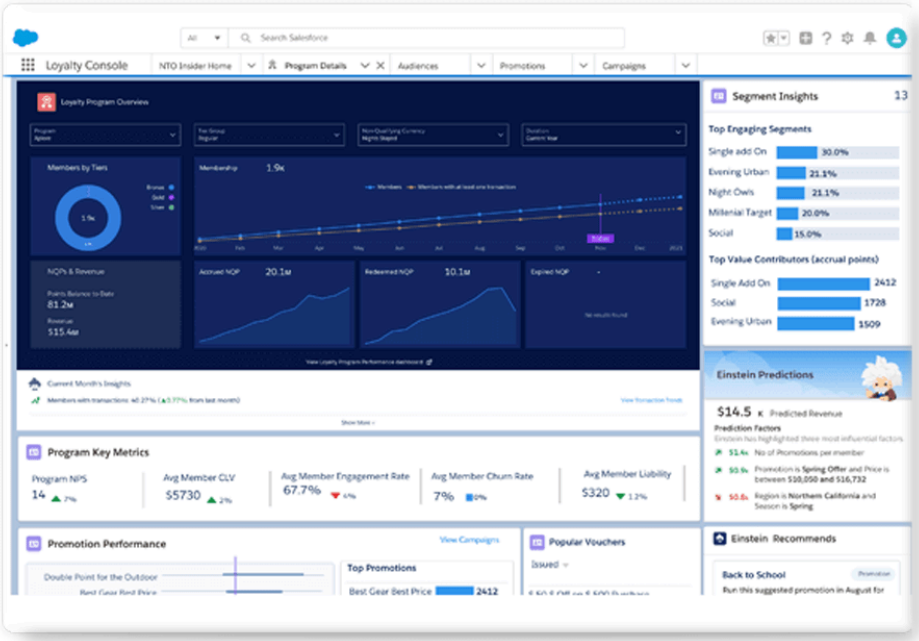


Рисунок 1.4 – Інтерфейс Salesforce

LinkedIn – професійна соціальна мережа, яка активно використовується для пошуку B2B-контактів [11].

До основних функцій продукту належать:

– пошук компаній і контактів за різними фільтрами;
– доступ до профілів ключових осіб компанії;
– можливість надсилати запити та вести комунікацію;
– платформа Sales Navigator для B2B-продажів.

На рисунку 1.5 зображено інтерфейс LinkedIn вебсайту.

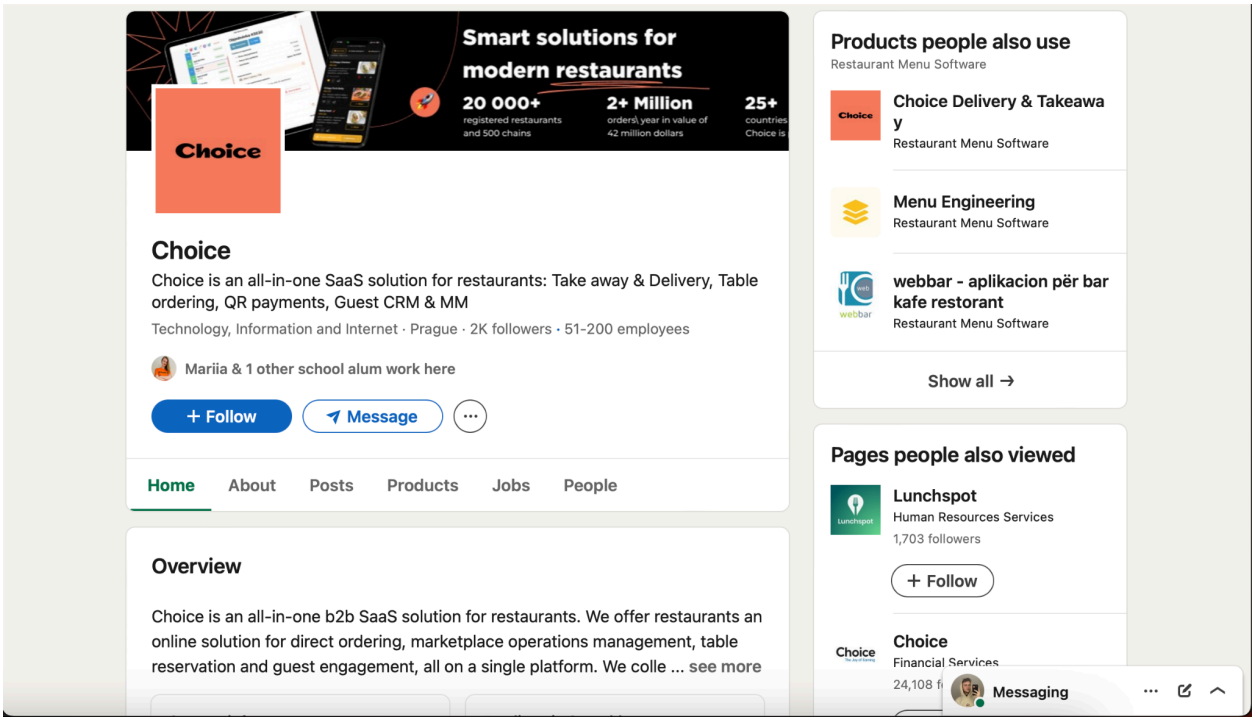


Рисунок 1.5 – Інтерфейс LinkedIn

Для порівняння проєкту з аналогом можна скористатись таблицею 1.1.

Таблиця 1.1 – Порівняння з аналогами

Функціонал	Sieve (ДП)	Crunch base	Similar Web	Apollo. io	Sales force	Linked In	Пояснення
Пошук компаній за ключовими словами	+	+	-	+	-	+	Можливість знаходити компанії за описом, послугами, напрямками
Фільтрація за локацією	+	+	+	+	+	+	Пошук компаній за містом, країною, регіоном
Пошук за галузями	+	+	-	+	+	+	Відбір компаній за галуззю (наприклад, фінанси, IT, медицина)

Продовження таблиці 1.1

Функціонал	Sieve (ДП)	Crunch base	Similar Web	Apollo. io	Sales force	Linked In	Пояснення
Аналіз вебсайту компанії	+	-	+	-	-	-	Аналіз структури, контенту та SEO-даних сайту
Отримання email-адрес	+	+	-	+	+	+	Збір контактної інформації для зв'язку
Експорт результатів	+	+	-	+	+	+	Можливість завантажити список знайдених компаній у CSV/XLS
Актуальність даних	+	-	+	+	+	+	Оновлення даних про компанії в реальному часі

1.3.2 Аналіз відомих алгоритмічних та технічних рішень

У рамках дипломного проєкту, який передбачає автоматизований пошук потенційних B2B-клієнтів через вебсканування та обробку інформації з відкритих джерел, виникає потреба у вирішенні низки задач. Серед них – генерація пошукових запитів, парсинг HTML-контенту, витягування контактних даних, визначення релевантності сторінок та оцінка активності компанії. Для реалізації цих задач доцільно використати комбінацію відомих алгоритмічних підходів та технічних рішень, які забезпечать ефективність, масштабованість та гнучкість системи.

Для формування пошукових запитів було розглянуто підходи на основі вагових моделей, зокрема TF-IDF (Term Frequency-Inverse Document Frequency) та BM25. Ці методи дозволяють формувати запити з урахуванням релевантності

ключових слів, однак не враховують семантичні зв’язки та синонімію. У зв’язку з цим, у проєкті буде застосовано комбінацію шаблонізованих запитів із вручну підібраними варіаціями ключових фраз, що забезпечить кращу якість результатів при використанні пошукових API.

Для обробки HTML-контенту обрано бібліотеку BeautifulSoup, яка забезпечує зручну навігацію та вибір елементів DOM-дерева за допомогою CSS-селекторів. Це рішення є легким, стабільним і ефективним для більшості статичних сайтів. У випадках, коли сторінки завантажуються динамічно через JavaScript, можливе підключення Selenium або Puppeteer, однак такі рішення є значно повільнішими, що не відповідає вимогам до продуктивності при обробці великого обсягу сайтів.

Задача витягування контактної інформації (електронної пошти, номерів телефонів, посилань на соцмережі) вирішується за допомогою регулярних виразів. Цей підхід є простим у реалізації, проте потребує додаткових етапів фільтрації шуму, нормалізації та валідації.

Оцінка релевантності вебсайтів до заданих критеріїв виконується на основі Cosine Similarity між вектором ключових слів користувача та вектором вмісту сторінки. Такий підхід дозволяє гнучко визначати ступінь відповідності знайденого ресурсу запиту, уникаючи жорстких фільтрів на основі лише ключових слів. У разі потреби передбачено побудову евристичного фільтра, який поєднує кілька факторів – наявність контактів, галузеву належність, ключові слова, тип домену тощо.

На рівні технічної реалізації проєкт базується на мікросервісній архітектурі з використанням контейнеризації через Docker. Такий підхід забезпечує гнучкість, масштабованість та простоту розгортання окремих компонентів системи. Обробка вебзапитів здійснюється через асинхронну бібліотеку aiohttp у поєднанні з requests та BeautifulSoup.

Щодо інтеграції із зовнішніми пошуковими системами, у рамках проєкту було проаналізовано можливості Google Custom Search API та Bing Search API. Оскільки Google API має значні обмеження (CAPTCHA, складність

налаштування, обмеження на кількість запитів), було прийнято рішення використовувати Bing Search API як основне джерело видачі, що є більш стабільним та гнучким у роботі.

Таким чином, на основі проведеного аналізу було сформовано набір як алгоритмічних, так і технічних рішень, що найкраще відповідають вимогам проєкту. У разі виявлення обмежень у продуктивності чи точності деяких підходів передбачається розробка власних евристичних або гібридних рішень, які об’єднуватимуть переваги кількох моделей.

1.4 Аналіз та моделювання бізнес-процесів

Розробка програмного забезпечення для автоматичного пошуку B2B-клієнтів у рамках дипломного проєкту потребує моделювання низки бізнес-процесів, які охоплюють весь ланцюг взаємодії користувача з системою – від створення запиту до отримання та обробки результатів.

Серед усіх процесів, які бере на себе система, було виокремлено такі ключові бізнес-процеси, що є критично важливими для коректної роботи та задоволення очікувань кінцевого користувача:

– Формування пошукового запиту, де користувач надає ключові слова або фрази, які описують цільовий ринок, галузь або тип клієнтів. Система формує відповідні шаблонізовані запити до пошукового API.

– Обробка результатів пошуку, при якій отримані з пошукового API посилання зберігаються, проходять фільтрацію за заданими критеріями (наприклад, виключення з blacklist) і передаються на етап парсингу.

– Парсинг та аналіз вебсайтів, під час яких система завантажує вміст сторінок, здійснює витягування ключових елементів (назва компанії, контакти, галузь діяльності), класифікує компанії за тематиками.

– Формування списку потенційних клієнтів, при яких на основі зібраної інформації будується фінальний список компаній із контактними даними. Результати виводяться у вигляді таблиці та можуть бути експортовані у форматі XLSX.

– Зворотний зв’язок від користувача, при якому користувач може позначити релевантні та нерелевантні компанії. Ці дані можуть бути використані для покращення точності майбутніх пошуків (наприклад, через донавчання моделей або оновлення евристик).

Для наочного представлення вищезгаданих процесів обрано нотацію BPMN (Business Process Model and Notation), яка є сучасним стандартом для графічного моделювання бізнес-процесів. Цей підхід дозволяє чітко відобразити послідовність дій, точки прийняття рішень, взаємодію між компонентами системи та зовнішніми службами.

Отже, нижче наведена BPMN-діаграма "Пошук компаній за ключовими словами", яка охоплює етапи від введення запиту до отримання попереднього списку компаній (рисунок 1.6).

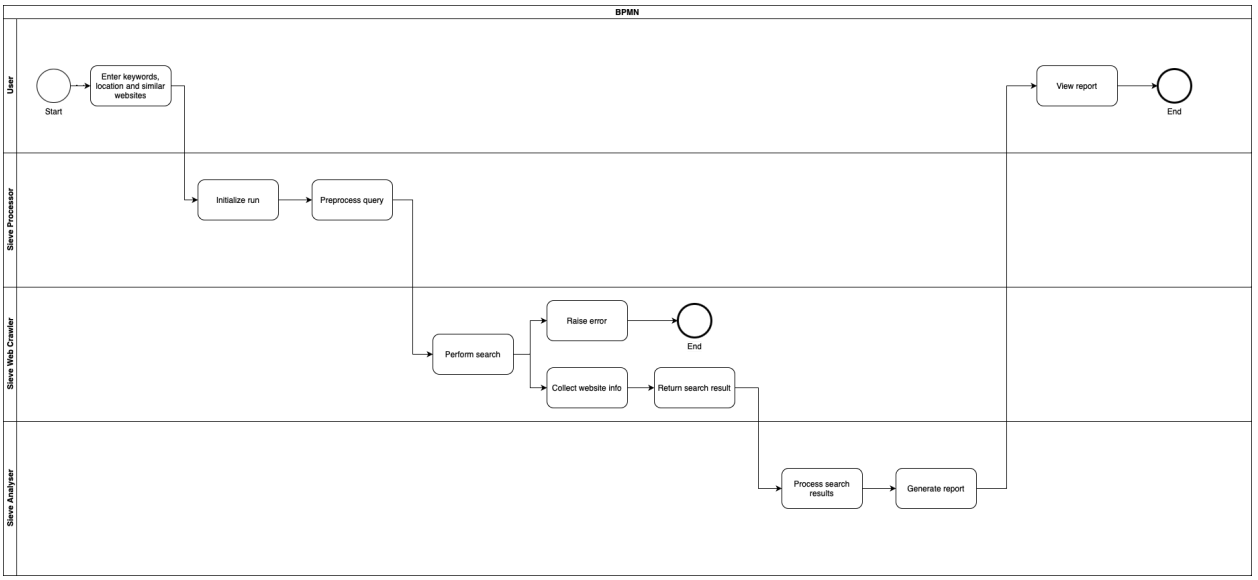


Рисунок 1.6 – BPMN-діаграма "Пошук компаній за ключовими словами"

Опис моделі бізнес-процесу пошуку компаній за ключовими словами:

– користувач вводить ключові слова, локації для пошуку та схожі вебсайти;

– Sieve Processor (SP) ініціалізує виконання пошуку та попередньо оброблює запит для оптимального пошуку;

– Sieve Web Crawler (SWC) виконує пошук;

– якщо пошук успішний, SWC збирає результати по вебсайтах, інакше повертається помилка пошуку;

– Sieve Analyser (SA) обробляє результати по вебсайтах та генерує репорт для користувача;

– користувач має можливість проглянути репорт.

Висновки до розділу

У цьому розділі було проведено аналіз предметної області, визначено актуальність та практичну цінність теми дипломного проєкту. У ході дослідження було з’ясовано, що в умовах цифрової трансформації та високої конкуренції на ринку B2B-сектору, ефективний пошук потенційних клієнтів є одним із ключових чинників успішного ведення бізнесу.

Було проаналізовано сучасний стан процесу генерації B2B клієнтів і виявлено, що традиційні підходи втрачають свою ефективність через високу трудомісткість і обмежену масштабованість. Водночас розвиток технологій вебскрапінгу та обробки великих обсягів відкритих даних дає змогу створювати автоматизовані рішення, здатні значно підвищити ефективність пошуку потенційних партнерів.

В рамках обґрунтування вибору теми, було зазначено, що створення вебзастосунка для автоматичної генерації B2B клієнтів на основі вебскрапінгу дозволяє реалізувати повний цикл роботи з даними: від збору, обробки, класифікації та зберігання до візуалізації та експорту результатів. Окрім того, дана тема поєднує в собі теоретичну цінність із практичною користю, що підтверджується успішною апробацією створеного інструменту у реальному середовищі.

2 РОЗРОБЛЕННЯ ВИМОГ ДЛЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Варіанти використання програмного забезпечення

Діаграма варіантів використання з ключовими акторами та основними функціями наведена в графічних матеріалах (аркуш 1).

В таблицях 2.1-2.7 наведено опис варіантів використання програмного забезпечення.

Таблиця 2.1 – Варіант використання UC-01

Use case name	Реєстрація користувача
Use case ID	UC-01
Goals	Реєстрація нового користувача в системі
Actors	Незареєстрований користувач
Trigger	Користувач бажає зареєструватися
Pre-conditions	-
Flow of Events	Користувач переходить на сторінку реєстрації. Вводить електронну пошту, пароль, підтверджує пароль і погоджується з умовами. Натискає кнопку "Зареєструватися". Отримує повідомлення про успішну реєстрацію та переходить на сторінку входу
Extension	Некоректно введені дані підсвічуються червоним; кнопка неактивна, доки всі поля не заповнені правильно
Post-Condition	Створено обліковий запис користувача, користувач перенаправлений на сторінку входу

Таблиця 2.2 – Варіант використання UC-02

Use case name	Авторизація користувача
Use case ID	UC-02
Goals	Вхід користувача в систему
Actors	Зареєстрований користувач
Trigger	Користувач бажає увійти до свого облікового запису
Pre-conditions	Користувач має зареєстрований обліковий запис

Продовження таблиці 2.2

Flow of Events	Користувач відкриває сторінку входу, вводить електронну пошту та пароль, натискає кнопку "Увійти". Успішна авторизація перенаправляє на головну сторінку
Extension	Якщо дані некоректні, з'являється повідомлення про помилку
Post-Condition	Користувач увійшов у свій акаунт, доступ до основного функціоналу відкрито

Таблиця 2.3 – Варіант використання UC-03

Use case name	Перегляд сторінки привітання
Use case ID	UC-03
Goals	Показати користувачеві привітальну сторінку після авторизації
Actors	Незареєстрований користувач
Trigger	Успішна авторизація
Pre-conditions	Користувач пройшов авторизацію
Flow of Events	Система перенаправляє користувача на сторінку привітання з коротким описом функціоналу
Extension	-
Post-Condition	Користувач бачить головне меню / можливості сервісу

Таблиця 2.4 – Варіант використання UC-04

Use case name	Визначення релевантності запитів
Use case ID	UC-04
Goals	Визначити відповідність ключових слів до B2B-запитів
Actors	Авторизований користувач
Trigger	Користувач вводить ключові слова
Pre-conditions	Користувач має активну сесію

Продовження таблиці 2.4

Flow of Events	Користувач вводить запит, система виконує фільтрацію релевантних сайтів або компаній
Extension	-
Post-Condition	Визначено список релевантних компаній

Таблиця 2.5 – Варіант використання UC-05

Use case name	Генерація репорту
Use case ID	UC-05
Goals	Створити звіт за знайденими даними
Actors	Авторизований користувач
Trigger	Користувач бажає зберегти результати аналізу
Pre-conditions	Дані були отримані після аналізу
Flow of Events	Користувач натискає кнопку генерації. Система створює структуру репорту з аналітикою по знайденим компаніям
Extension	Якщо дані не знайдені – виводиться повідомлення про помилку
Post-Condition	Звіт сформовано

Таблиця 2.6 – Варіант використання UC-06

Use case name	Експорт результатів
Use case ID	UC-06
Goals	Надати користувачу можливість завантажити результати
Actors	Авторизований користувач
Trigger	Користувач натискає "Експортувати"
Pre-conditions	Є готовий репорт
Flow of Events	Система формує файл (CSV) і пропонує його для завантаження
Extension	Якщо файл не згенеровано – повідомлення про помилку
Post-Condition	Користувач завантажив файл з результатами

Таблиця 2.7 – Варіант використання UC-07

Use case name	Вихід із акаунту
Use case ID	UC-07
Goals	Завершення сесії користувача
Actors	Авторизований користувач
Trigger	Користувач натискає “Вийти”
Pre-conditions	Користувач авторизований
Flow of Events	Користувач натискає кнопку виходу. Сесія завершується. Перенаправлення на сторінку входу
Extension	-
Post-Condition	Сесія завершена, користувач вийшов із системи

2.2 Розроблення функціональних вимог

Програмне забезпечення розділене на модулі. Кожен модуль має свій певний набір функцій. В таблиці 2.8 наведено загальну модель вимог, а в таблиці 2.9 наведений опис функціональних вимог до програмного забезпечення.

Таблиця 2.8 – Загальна модель вимог

№	Назва	ID Вимоги	Пріоритети	Ризики
1	Перегляд вітальної сторінки	FR-1	Низький	Низький
2	Система авторизації користувача	FR-2	Високий	Високий
2.1	Реєстрація нового користувача	FR-3	Високий	Високий
2.2	Авторизація користувача	FR-4	Високий	Високий

Продовження таблиці 2.8

№	Назва	ID Вимоги	Пріоритети	Ризики
2.3	Вихід із акаунту	FR-5	Середній	Низький
3	Система Sieve Processor	FR-6	Високий	Високий
3.1	Введення пошукового запиту для генерації потенційних клієнтів	FR-7	Високий	Високий
4	Система Sieve Web Crawler	FR-8	Високий	Високий
4.1	Скрапінг (збір) відкритих даних з Інтернету	FR-9	Високий	Високий
5	Система Sieve Analyser	FR-10	Високий	Високий
5.1	Алгоритмічна фільтрація та класифікація компаній	FR-11	Високий	Середній
5.2	Формування звіту за результатами пошуку	FR-12	Високий	Високий
5.3	Експорт результатів у файл	FR-13	Високий	Високий
6	Визначення релевантності сайтів до запиту користувача	FR-14	Низький	Низький
7	Відображення статусу обробки пошуку	FR-15	Низький	Низький
8	Повідомлення про помилки / неправильні дані	FR-16	Середній	Середній

Таблиця 2.9 – Перелік функціональних вимог

Назва	Опис
FR-1	Перегляд сторінки привітання. Неавторизований користувач бачить вітальну вебсторінку, яка містить загальну інформацію про застосунок та можливості реєстрації або входу до системи.
FR-2	Система авторизації користувача. Система забезпечує загальну логіку автентифікації, включно з реєстрацією, входом та виходом користувача.
FR-3	Реєстрація нового користувача. Користувач може створити новий обліковий запис, вказавши email, пароль та, за потреби, інші ідентифікаційні дані.
FR-4	Авторизація користувача. Користувач може увійти до системи за допомогою раніше зареєстрованих облікових даних.
FR-5	Вихід із акаунту. Користувач має змогу завершити сесію, після чого повертається на вітальну сторінку.
FR-6	Система Sieve Processor. Ініціалізує обробку пошукового запиту користувача, здійснює базову валідацію та підготовку запиту до запуску вебскрапінгу.
FR-7	Введення пошукового запиту для генерації клієнтів. Користувач вводить ключові слова, регіон або інші параметри пошуку для генерації списку потенційних B2B клієнтів.
FR-8	Система Sieve Web Crawler. Відповідає за запуск вебскрапінгу — здійснює автоматичний обхід інтернет-ресурсів за заданими параметрами.
FR-9	Скрапінг відкритих даних з Інтернету. Вебкраулер здійснює збір доступних публічних даних з релевантних сайтів у відповідь на запит користувача.

Продовження таблиці 2.9

FR-10	Система Sieve Analyser. Після завершення збору даних система аналізує інформацію та готує її до виводу у структурованому вигляді.
FR-11	Алгоритмічна фільтрація та класифікація компаній. Зібрані дані проходять фільтрацію за критеріями релевантності, B2B спрямуванням, галуззю діяльності тощо.
FR-12	Формування звіту за результатами пошуку. Генерується звіт з підсумками пошуку, що включає список знайдених компаній та ключову інформацію про них.
FR-13	Експорт результатів у файл. Користувач має можливість експортувати звіт у форматі CSV для подальшого використання.
FR-14	Визначення релевантності сайтів до запиту користувача. Система використовує критерії релевантності, щоб фільтрувати непридатні сайти та зосередитися на потенційно цінних.
FR-15	Відображення статусу обробки пошуку. Користувач бачить прогрес обробки запиту (наприклад, через прогрес-бар, повідомлення про стан виконання).
FR-16	Повідомлення про помилки / неправильні дані. У випадку помилки (наприклад, поганий запит, відсутність результатів, проблеми з мережею) користувач отримує відповідне повідомлення.

Далі наведемо матрицю трасування вимог, яка узгоджує функціональні вимоги з варіантами використання (рисунок 2.1).

	FR-1	FR-2	FR-3	FR-4	FR-5	FR-6	FR-7	FR-8	FR-9	FR-10	FR-11	FR-12	FR-13	FR-14	FR-15	FR-16
UC-1	☐	☑	☑	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
UC-2	☐	☑	☐	☑	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
UC-3	☑	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
UC-4	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☑	☐	☐
UC-5	☐	☐	☐	☐	☑	☑	☑	☑	☑	☑	☑	☐	☐	☐	☑	☐
UC-6	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☑	☑	☐	☐	☑
UC-7	☐	☐	☐	☑	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐

Рисунок 2.1 – Матриця трасування вимог

2.3 Розроблення нефункціональних вимог

Нижче наведено перелік нефункціональних вимог, що висуваються до вебзастосунку для генерації B2B клієнтів шляхом вебскрапінгу. Ці вимоги визначають якісні характеристики системи та забезпечують її стабільну, безпечну та зручну експлуатацію.

– Вимоги до продуктивності. Система повинна забезпечувати час відповіді не більше 3 годин для типового запиту (до 20 ключових слів). Це дозволить користувачу оперативно отримувати релевантні та актуальні результати на запити.

– Вимоги до масштабованості. Архітектура системи має передбачати можливість обробки одночасних запитів від щонайменше 10 активних користувачів без суттєвого зниження продуктивності.

– Вимоги до доступності. Система повинна бути доступною не менше ніж 99% часу протягом календарного місяця, за винятком запланованих періодів технічного обслуговування.

– Вимоги до зручності користування. Інтерфейс користувача має бути інтуїтивно зрозумілим, що дозволяє користувачу виконувати основні дії (реєстрація, формування запиту, експорт результатів) без додаткового навчання або інструкцій.

– Вимоги до кросплатформенності. Система повинна коректно функціонувати у найбільш поширених сучасних браузерах (Google Chrome, Mozilla Firefox, Microsoft Edge, Safari) на різних операційних системах (Windows, macOS, Linux).

– Вимоги до модульності. Система має бути реалізована у вигляді окремих логічних модулів (реєстрація, авторизація, пошук, генерація звіту, експорт тощо), що дозволяє спростити супровід, тестування та розширення функціоналу.

– Вимоги до розширюваності. Архітектура системи повинна забезпечувати можливість додавання нових джерел даних для скрапінгу або нових форматів експорту результатів без істотних змін у наявному коді.

2.4 Аналіз економічних показників програмного забезпечення

Для оцінки трудомісткості, витрат і загальної складності розробки вебзастосунку для генерації B2B клієнтів застосовується метод функціональних точок (Function Point Analysis, FPA). Метод FPA дозволяє оцінити розмір програмного забезпечення, виходячи з кількості логічних функціональних блоків системи та їх складності. Це дозволяє прогнозувати зусилля на розробку, а також орієнтовну вартість проєкту.

Основні компоненти, що аналізуються за методом функціональних точок:

– ILF (Internal Logical File) внутрішні логічні файли.

– EI (External Input) зовнішні введення.

– EO (External Output) зовнішні виведення.

– EQ (External Query) зовнішні запити.

– EIF (External Interface File) зовнішні інтерфейсні файли.

Внутрішні логічні файли (ILF) – це логічно зв’язані дані, які зберігаються в межах системи. Вебзастосунок оперує такими ILF: користувачі, пошукові запити, результати сканування, чорний список доменів, історія експорту. Разом 49 функціональних точок. Внутрішні логічні файли з відповідною кількістю функціональних точок наведено в таблиці 2.10.

Таблиця 2.10 – Внутрішні логічні файли

Назва об’єкта	Опис	Складність	FP
Користувачі	Зберігає інформацію про зареєстрованих користувачів (логін, пароль, роль)	Середня	10
Пошукові запити	Запити користувача з ключовими словами	Низька	7
Результати сканування	Збереження знайдених компаній (назва, сайт, опис, країна тощо)	Висока	15
Чорний список доменів	Фільтровані/небажані домени	Низька	7
Історія експорту	Логи експорту даних	Середня	10

Зовнішні введення (EI) – це функції введення даних користувачем у систему. Наступні EI наявні в додатку: форма реєстрації і логіну, створення запиту, додавання доменів у чорний список. В сумі 10 функціональних точок. Зовнішні введення з відповідною кількістю функціональних точок наведено в таблиці 2.11.

Таблиця 2.11 – Зовнішні введення

Назва об’єкта	Опис	Складність	FP
Форма реєстрації/логіну	Введення даних користувача	Низька	3
Створення запиту	Введення ключових слів та параметрів	Середня	4
Додавання доменів у чорний список	Фільтрація небажаних результатів	Низька	3

Зовнішні виведення (EO) – це генерація вихідної інформації для користувача. В застосунку є 3 об’єкти: перегляд результатів пошуку, повідомлення про статус запиту та звіти про експорт. Зовнішні виведення з відповідною кількістю функціональних точок наведено в таблиці 2.12.

Таблиця 2.12 – Зовнішні виведення

Назва об’єкта	Опис	Складність	FP
Перегляд результатів пошуку	Виведення знайдених сайтів	Середня	5
Повідомлення про статус запиту	Інформування про хід обробки	Низька	4
Звіти про експорт	Вивід структури експорту	Середня	5

Зовнішні запити (EQ) – це запити на пошук інформації з бази з виведенням результатів без їх модифікації. Зовнішні запити з відповідною кількістю функціональних точок наведено в таблиці 2.13.

Таблиця 2.13 – Зовнішні запити

Назва об’єкта	Опис	Складність	FP
Пошук серед результатів	Фільтрація знайдених компаній	Середня	5
Перегляд історії запитів	Запити до бази без змін	Низька	3

Зовнішні інтерфейсні файли (EIF) – це зовнішні API, з якими взаємодія система. Зовнішні інтерфейсні файли з відповідною кількістю функціональних точок наведено в таблиці 2.14.

Таблиця 2.14 – Зовнішні інтерфейсні файли

Назва об’єкта	Опис	Складність	FP
API пошукової системи	Отримання результатів пошуку	Середня	7

2.5 Постановка завдання на розробку програмного забезпечення

Метою даного дипломного проєкту є підвищити ефективність пошуку потенційних B2B клієнтів.

Мета досягається шляхом розробки вебзастосунку, який дозволяє автоматизовано формувати базу потенційних B2B клієнтів шляхом збору, фільтрації та обробки інформації з відкритих джерел мережі Інтернет. Програмне забезпечення має забезпечити користувачу інструмент для пошуку компаній за ключовими словами, аналізу отриманих результатів та експорту релевантних даних у зручному форматі.

Задачами розробки є забезпечити користувача можливістю формування запитів на пошук компаній за заданими ключовими словами, реалізувати механізм збору вебсторінок компаній з використанням пошукових систем, забезпечити автоматичне вилучення релевантної інформації з вебресурсів (назва компанії, вебсайт, опис діяльності тощо), розробити модулі класифікації та

фільтрації знайдених сайтів відповідно до критеріїв якості та тематики, реалізувати механізм збереження та експорту результатів пошуку та забезпечити захищений доступ до функціоналу системи через особистий кабінет користувача.

Основними завданнями в роботі є проведення аналізу існуючих методів вебскрапінгу для збору даних про компанії, розробка архітектури вебзастосунку для автоматичного пошуку потенційних B2B клієнтів, реалізація алгоритму фільтрації та класифікації отриманих даних за релевантністю до запиту, оптимізація швидкості обробки та точності знаходження компаній, розробка функціоналу експорту даних у зручному форматі.

Висновки до розділу

В ході виконання розділу було розроблено варіанти використання мого програмного застосунку. Використовуючи приклади варіантів використання було сформульовано ключові функціональні та нефункціональні вимоги. Також було проведено оцінку трудомісткості, витрат і загальної складності розробки вебзастосунку для генерації B2B клієнтів за допомогою методу функціональних точок.

Також у розділі були сформульовані мета та завдання дипломного проєкту, які визначають напрям подальшого дослідження й реалізації. Метою є створення інноваційного вебзастосунка, що дозволяє компаніям швидко знаходити потенційних клієнтів на основі ключових слів і географії, використовуючи сучасні методи автоматизації збору даних.

Таким чином, у цьому розділі було закладено теоретичну та методологічну основу для реалізації дипломного проєкту, визначено ключові технології, напрями дослідження та підтверджено практичну цінність розробки.

За результатами розділу сформовано технічне завдання на розробку програмного забезпечення.

3 КОНСТРУЮВАННЯ ТА РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Архітектура програмного забезпечення

Для розробки вебзастосунку, призначеного для генерації B2B клієнтів шляхом вебскрапінгу, було обрано монолітну архітектуру. Такий вибір зумовлений низкою переваг, особливо актуальних на етапі створення та демонстрації функціональності в рамках дипломного проєкту.

Перш за все, це простота розробки та розгортання. На початковому етапі, коли ключовим завданням є реалізація основного функціоналу – збору даних про компанії з відкритих вебджерел, їх подальшої фільтрації, класифікації та забезпечення можливості експорту результатів, – монолітна структура пропонує значно меншу початкову складність порівняно з альтернативними підходами, наприклад, мікросервісною архітектурою. Усі функціональні компоненти, такі як модуль вебскрапінгу, система аналізу та обробки зібраної інформації, а також користувацький інтерфейс, інтегровані в єдину кодову базу. Це спрощує взаємодію між ними та загальне розуміння системи. Крім того, розгортання монолітного застосунку є менш трудомістким процесом, оскільки передбачає роботу з єдиним програмним пакетом, що є важливим для ефективної демонстрації результатів дипломного проєктування.

Другою важливою складовою є продуктивність для специфічних завдань обробки даних. У контексті розроблюваного вебзастосунку, де передбачається інтенсивна обробка та аналіз зібраних даних (наприклад, фільтрація списку компаній за заданими критеріями, класифікація вебсайтів), монолітна архітектура може забезпечити вищу продуктивність. Взаємодія між компонентами системи відбувається через прямі виклики функцій або методів в межах одного процесу. Це усуває накладні витрати на мережеву комунікацію, які були б неминучими при використанні розподілених архітектур для подібних операцій, що особливо важливо для ефективного аналізу потенційних B2B клієнтів.

Також треба врахувати переваги монолітної архітектури через спрощення тестування наскрізних сценаріїв. Тестування ключових сценаріїв роботи вебзастосунку, таких як повний цикл від введення користувачем пошукового запиту до отримання фінального звіту з переліком B2B клієнтів та можливістю його експорту у форматі CSV, є простішим в умовах монолітної архітектури. Наскрізне (end-to-end) тестування не вимагає комплексної координації та налаштування взаємодії між численними окремими сервісами, що дозволяє ефективніше перевірити коректність роботи всього функціоналу системи.

Обрана монолітна архітектура дозволяє зосередитись на реалізації основних завдань дипломного проєкту, забезпечуючи при цьому достатню продуктивність та керованість на поточному етапі розробки вебзастосунку для генерації B2B клієнтів.

Пропоную детальніше розглянути архітектуру за допомогою діаграм 1-го та 2-го рівня C4 Model представлення.

Діаграми C4 Model 1-го та 2-го рівнів надають значні переваги для візуалізації та комунікації архітектури програмного забезпечення. Діаграма контексту системи (рівень 1) є відправною точкою, що показує, як програмна система взаємодіє зі своїми користувачами та іншими системами в її оточенні; це допомагає технічним і нетехнічним зацікавленим сторонам зрозуміти межі системи та її роль у ширшій екосистемі. Діаграма контейнерів (рівень 2) деталізує внутрішню структуру системи, розбиваючи її на основні виконувані або розгортані одиниці (наприклад, вебсервери, бази даних, мобільні застосунки) та показуючи їх взаємозв'язки й технологічний вибір. Разом ці два рівні забезпечують чітке, високорівневе уявлення про архітектуру, яке легко зрозуміти різним аудиторіям, сприяючи кращій комунікації, ефективному онбордингу нових членів команди та полегшуючи обговорення архітектурних рішень на ранніх етапах проєктування.

Далі наведемо C4 Model діаграму 1-го рівня (рисунок 3.1).

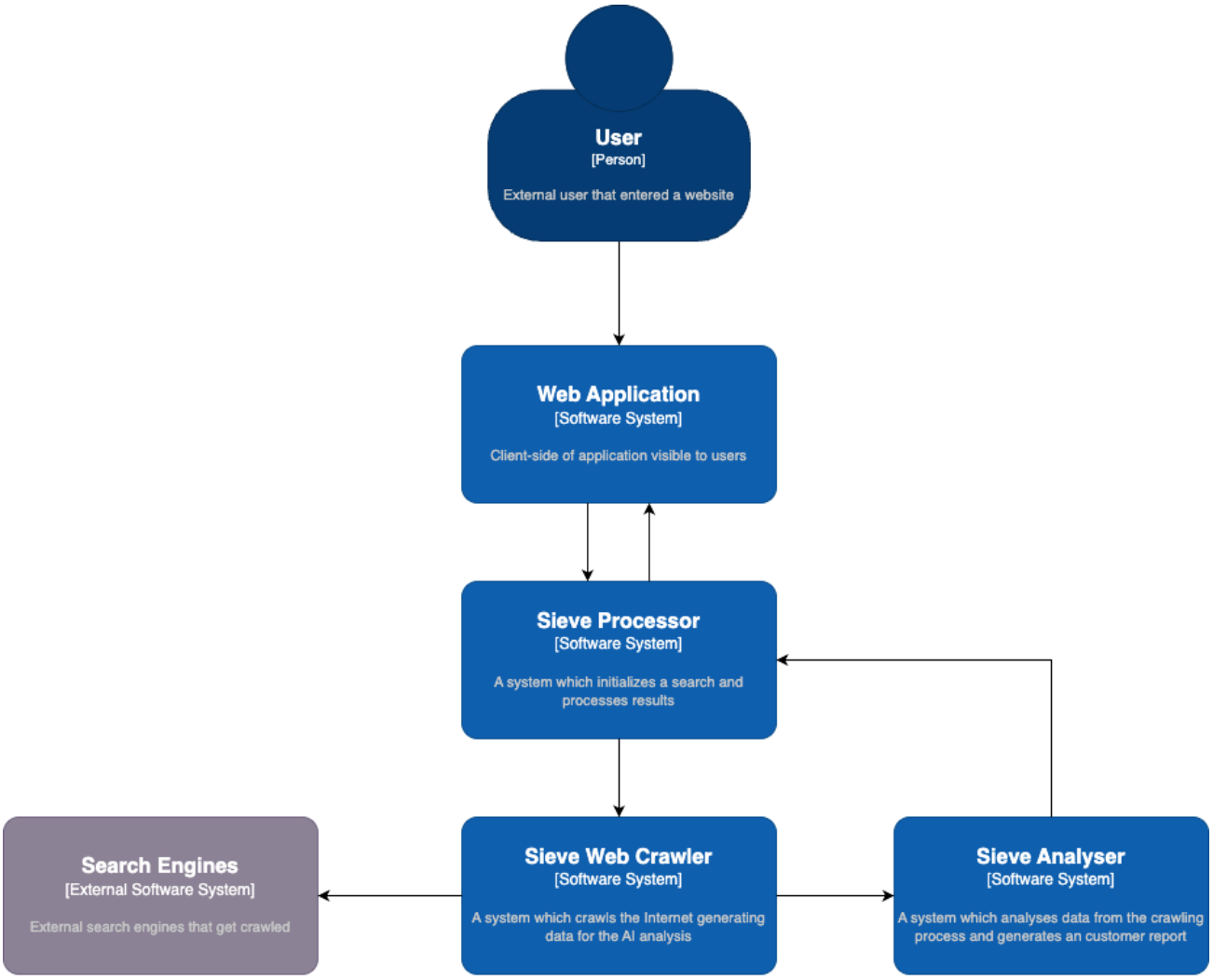


Рисунок 3.1 – C4 Model діаграма 1-го рівня

На діаграмі контекстів представлені наступні елементи:

– User (Person): Зовнішній актор, що представляє кінцевого користувача, який здійснює вхід на вебсайт системи та ініціює її функції.

– Web Application (Software System): Програмна система, що є клієнтською частиною розроблюваного застосунку, через яку користувач взаємодіє із системою.

– Sieve Processor (Software System): Внутрішня програмна система, відповідальна за ініціалізацію процесу пошуку та обробку отриманих результатів.

– Sieve Web Crawler (Software System): Внутрішня програмна система, призначена для автоматичного сканування ресурсів мережі Інтернет з метою

збору даних для подальшого аналізу, зокрема з використанням алгоритмів штучного інтелекту.

– Search Engines (External Software System): Зовнішні програмні системи (пошукові машини), які використовуються компонентом Sieve Web Crawler як джерело інформації для вебскрапінгу.

– Sieve Analyser (Software System): Внутрішня програмна система, що здійснює аналіз даних, отриманих у процесі вебскрапінгу, та генерує підсумковий звіт для користувача.

Користувач (User) взаємодіє з клієнтською частиною (Web Application), передаючи запит на генерацію B2B клієнтів. Web Application передає цей запит до Sieve Processor, який координує подальшу роботу. Sieve Processor активує Sieve Web Crawler для збору даних з Інтернету, використовуючи зовнішні пошукові системи (Search Engines). Зібрані дані передаються до Sieve Analyser для аналізу, класифікації та формування звіту. Сформований звіт надається користувачеві через Web Application.

На схемі структурній компонентів програмного забезпечення представлені наступні елементи:

– User (Person): Зовнішній користувач, який увійшов на вебсайт.

– Client-side application (Container: Flask): Доставляє статичний контент користувачеві.

– API Application (Container: Python): Надає функціональність застосунку Sieve через JSON/HTTPS API.

– Database (Container: Postgres): Зберігає інформацію про реєстрацію користувачів, пошукові запити за ключовими словами, результати сканування тощо.

– Search Engines (External Software System): Зовнішні пошукові системи, які скануються.

Схема структурна компонентів програмного забезпечення наведена в графічних матеріалах (аркуш 2).

Користувач (User) ініціює взаємодію через Client-side application. Client-side application надсилає запити до API Application для виконання операцій, таких як пошук та генерація звітів. API Application обробляє ці запити, взаємодіючи з Database для збереження та отримання даних, а також із зовнішніми Search Engines для збору інформації. Результати обробки повертаються через API Application до Client-side application для відображення користувачеві.

3.2 Архітектурні рішення та обґрунтування вибору засобів розробки

У цьому розділі представлено опис архітектури розробленої інформаційної системи. Детально обґрунтовується вибір програмних засобів, технологій та архітектурних рішень, що були застосовані для реалізації поставлених завдань.

Архітектура системи реалізована за принципом монолітного вебдодатка з логічно відокремленим модулем для виконання ресурсомістких обчислювальних задач. Далі розберемо основні компоненти системи.

Першим компонентом пропоную розглянути вебсервер, розроблений на базі мікрофреймворку Flask мовою програмування Python. Цей компонент відповідає за обробку HTTP-запитів від клієнтської частини, реалізацію бізнес-логіки користувацьких сценаріїв (реєстрація, автентифікація, управління запитами), взаємодію з базою даних та ініціалізацію роботи модуля алгоритму.

Другим важливим компонентом є база даних, для якої використовується реляційна система управління базами даних PostgreSQL. Вона призначена для персистентного зберігання даних користувачів, їхніх пошукових запитів, параметрів цих запитів, статусів обробки та шляхів до файлів з результатами.

Наступним компонентом є модуль алгоритму представлений ізольованим програмним модулем, написаний мовою програмування Python, що виконує основні завдання з пошуку та обробки інформації. Він включає логіку вебскрейпінгу, первинного аналізу зібраних даних відповідно до заданих користувачем даних (зокрема, "whitelist words") та генерацію підсумкових файлів у форматі XLSX. Його робота ініціюється вебсервером асинхронно.

Завершальним основним компонентом є клієнтський інтерфейс, реалізований за допомогою серверного рендерингу HTML-сторінок з використанням шаблонізатора Jinja2. Він забезпечує взаємодію користувача з системою через браузер.

При розробці системи було використано Model-View-Template (MVT) як архітектурний патерн вебзастосунка, що є характерним патерном для фреймворку Flask [12]. Модель представлена SQLAlchemy-об’єктами, що описують структуру даних та взаємодію з базою даних. Шаблонами є HTML-файли, оброблювані шаблонізатором Jinja2. Вони відповідають за відображення даних користувачеві. Представленнями є функції-обробники маршрутів у Flask, що приймають запити, взаємодіють з моделями та обирають відповідні шаблони для відображення.

Вибір технологічного стека ґрунтувався на вимогах до функціональності системи, необхідності швидкої розробки прототипу та доступності інструментів.

Python було обрано як основну мову програмування завдяки насиченості стандартної бібліотеки та екосистеми сторонніх пакетів. В рамках дипломної роботи було використано його фреймворки для веброзробки (Flask, SQLAlchemy), вебскрапінгу (Selenium, Requests, BeautifulSoup) та модулі для роботи з файлами. Також завдяки активній спільноті, мова насичена великою кількістю якісної документації, навчальних матеріалів та форумів для розв'язання проблем.

Для реалізації серверної частини було обрано мікрофреймворк Flask. Flask надає базовий набір інструментів, дозволяючи розробнику самостійно обирати та інтегрувати необхідні компоненти та розширення, що забезпечує високий рівень контролю над структурою додатка [13]. Також існує велика кількість офіційних та сторонніх розширень (Flask-SQLAlchemy, Flask-Login, Flask-WTF, Flask-Migrate тощо), які покривають типові потреби вебдодатків в авторизації, безпеки даних та взаємодії з базою даних.

Вибір PostgreSQL як системи управління базами даних обґрунтований такими факторами як відповідність ACID, що гарантує цілісність даних та

надійність транзакцій, підтримкою широкого спектра SQL-стандартів, розширюваністю та можливістю роботи зі складними типами даних [14]. Також СУБД має хороші показники продуктивності та можливості для масштабування в майбутньому.

3.3 Конструювання програмного забезпечення

На цьому етапі розробки дипломного проєкту здійснено реалізацію основних функціональних компонентів програмного забезпечення. Конструювання включає як побудову логіки вебскрапінгу та обробки результатів, так і розробку структури бази даних, яка забезпечує зберігання та подальшу роботу з отриманими даними.

У межах цього підрозділу розглядаються:

– Розробка алгоритму Sieve, що відповідає за фільтрацію, нормалізацію та попередню обробку зібраних результатів, відсіюючи нерелевантні або низькоякісні сайти.

– Опис структури бази даних, що детально висвітлює моделювання основних сутностей, їх зв’язків та забезпечення цілісності даних у системі.

Ці компоненти є критично важливими для забезпечення коректної та ефективної роботи вебзастосунку, оскільки поєднують алгоритмічну частину пошуку з надійним механізмом зберігання даних.

3.3.1 Розробка алгоритму Sieve

Алгоритм Sieve є алгоритмом багатоетапного аналізу списку вебпосилань з метою їх фільтрації, збагачення даними та структурування. В основі алгоритму лежить концепція конвеєрної обробки, реалізована за допомогою класів SieveAnalyser та Pipeline.

Клас SieveAnalyser виступає в ролі оркестратора процесу аналізу. При ініціалізації приймає шлях до вхідного файлу з вебпосиланнями (у форматі NDJSON), шлях для збереження результатів аналізу та список whitelist words для

фільтрації за метаданими. Основним методом класу є run(), який визначає послідовність фільтрів та запускає їх виконання через об'єкт класу Pipeline.

Клас Pipeline реалізує механізм гнучкого конвеєра для послідовного застосування набору фільтрів до вхідних даних. При ініціалізації об'єкт Pipeline отримує на вхід два параметри – filters та base_link_objects.

Перший є списком, що описує конфігурацію конвеєра. Кожен елемент цього списку містить наступні поля:

– Порядковий індекс фільтра в конвеєрі.

– Список індексів фільтрів даних (filter_receive_indexes). Значення 0 вказує на використання початкового набору даних (base_link_objects). Інші значення відповідають порядковим індексам попередньо виконаних фільтрів, результати роботи яких (result_link_objects, що зберігаються в екземплярі відповідного фільтра) будуть передані на вхід поточному фільтру. Це дозволяє реалізовувати складні залежності між етапами обробки, де один фільтр може використовувати результати декількох попередніх.

– Екземпляр класу конкретного фільтра (нащадка BasicFilter), що реалізує певну логіку обробки даних.

Другий є початковим списком об'єктів (в даному випадку, словників, що представляють вебпосилання), які підлягають обробці.

Основний метод run() класу Pipeline ітеративно проходить по сконфігурованій послідовності фільтрів. Для кожного фільтра він динамічно формує список вхідних параметрів (params), збираючи результати роботи попередніх фільтрів (або початкові дані) згідно зі списком filter_receive_indexes. Після цього поточний фільтр виконується, і його вихідні дані (link_objects) передаються для подальшої обробки або стають фінальним результатом роботи конвеєра.

У методі run() класу SieveAnalyser визначається наступна послідовність фільтрів (етапів обробки), що застосовуються до списку вебпосилань:

– Нормалізація URL (RegularizeLinksFilter), що відповідає за приведення URL-адрес до канонічного вигляду (наприклад, видалення префіксу 'www.').

– Підрахунок входжень URL (OccurrencesCountFilter), що визначає частоту кожного URL у списку.

– Групування за геолокацією (LocationGroupingFilter), що відповідає за групування посилань на основі їх ймовірного географічного походження.

– Фільтрація за чорним списком (BlacklistFilter), що видаляє посилання, що відповідають критеріям чорного списку.

– Дедуплікація (DeduplicationFilter), що видаляє дублікати посилань.

– Порівняння з підрахованою кількістю входжень (MatchOccurrencesCountFilter), що фільтрує та збагачує дані на основі результатів роботи фільтрів OccurrencesCountFilter та DeduplicationFilter.

– Вилучення структурованих даних з вебсайтів (WebsiteDataExtractionFilter), що відповідає за перехід за посиланнями та парсить вебсторінки для вилучення корисної інформації.

– Отримання контактної інформації (ExtractContactInformationFilter), що виконує пошук та вилучення контактних даних (телефони, email тощо) з раніше отриманої інформації.

– Переклад даних (TranslationFilter), що здійснює переклад вилученого текстового контенту на англійську мову.

– Перевірка метаданих (CheckMetadataFilter), що аналізує метадані вебсторінок на відповідність списку "білих слів", переданому при ініціалізації SieveAnalyser.

Результат роботи всього конвеєра (оброблений список об'єктів посилань) зберігається у вихідний файл у форматі NDJSON за допомогою функції append_to_json_file.

На рисунку нижче наведена концептуальне представлення роботи алгоритму (рисунок 3.2).

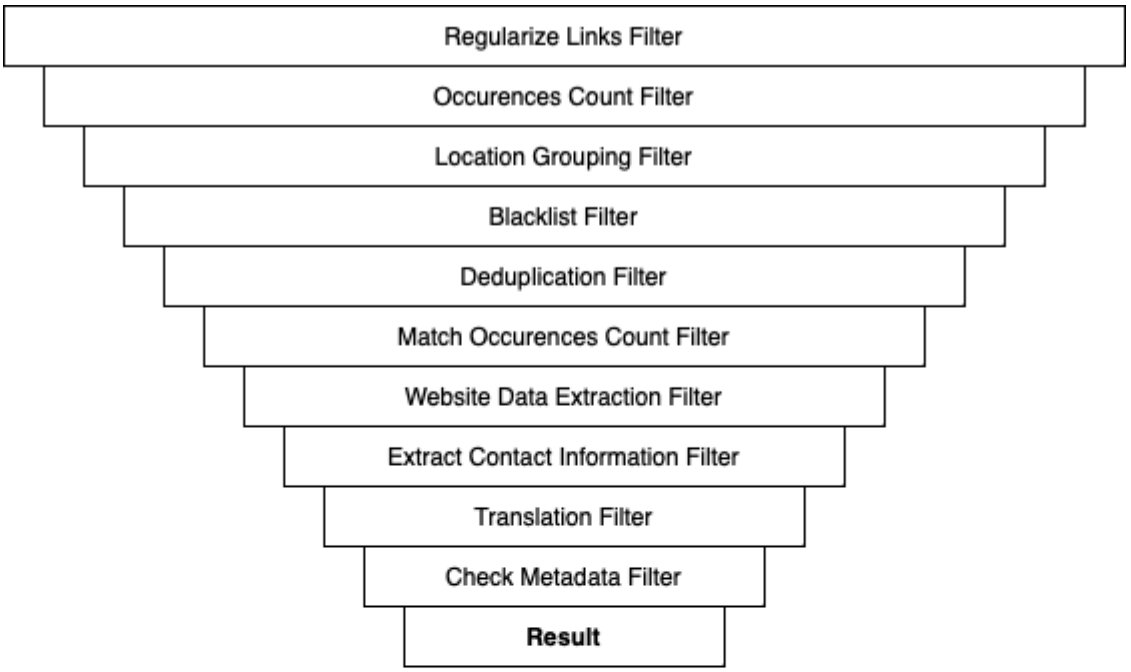


Рисунок 3.2 – Концептуальна діаграма алгоритму Sieve

3.3.2 Опис структури бази даних

В якості системи управління базами даних використовується PostgreSQL. База даних сервера призначена для зберігання операційної інформації необхідної для діяльності вебзастосунка До такої інформації відносяться сутності користувача (дані для авторизації користувача), пошукового запиту (дані про пошукові запити користувача), результату запиту (дані про результати пошукового запиту), а також даних про заблоковані домени та історію експортів. Опис таблиць бази даних наведено у таблицях 3.1-3.5.

Таблиця 3.1 – Опис таблиці user

Назва поля	Тип даних	Опис
id	Integer PK	Унікальний ідентифікатор користувача
username	Varchar	Логін користувача
email	Varchar	Електронна пошта
password_hash	Varchar	Хеш пароля
role	Varchar	Роль користувача
created_at	Timestamp	Дата реєстрації

Таблиця 3.2 – Опис таблиці search_query

Назва поля	Тип даних	Опис
id	Integer PK	Унікальний ідентифікатор запиту
user_id	Integer FK	Унікальний ідентифікатор користувача
keywords	Varchar	Ключови слова для пошуку
country	Varchar	Вибрана країна для фільтрації результатів
status	Varchar	Статус запиту (new, processing, done)
created_at	Timestamp	Дата створення запиту

Таблиця 3.3 – Опис таблиці search_result

Назва поля	Тип даних	Опис
id	Integer PK	Унікальний ідентифікатор результату
query_id	Integer FK	Унікальний ідентифікатор запит
website_url	Varchar	URL сайту
description	Varchar	Короткий опис
country	Varchar	Країна компанії
is_exported	Boolean	Чи був результат експортований
scraped_at	Timestamp	Дата та час отримання результату

Таблиця 3.4 – Опис таблиці blacklist_domains

Назва поля	Тип даних	Опис
id	Integer PK	Унікальний ідентифікатор запису
domain	Varchar	Назва домену
reason	Varchar	Причина блокування
created_at	Timestamp	Дата додавання

Таблиця 3.5 – Опис таблиці export_logs

Назва поля	Тип даних	Опис
id	Integer PK	Унікальний ідентифікатор запису
query_id	Integer FK	Унікальний ідентифікатор запиту
export_format	Varchar	Формат файлу
export_time	Timestamp	Час створення експорту
file_path	Varchar	Шлях до файлу

Схема бази даних наведена в графічних матеріалах (аркуш 3).

Опис утиліт, бібліотек та іншого стороннього програмного забезпечення, що використовується у розробці наведено в таблиці 3.6.

Таблиця 3.6 – Опис утиліт

№ п/п	Назва утиліти	Опис застосування
1	PyCharm	Головне середовище розробки програмного забезпечення серверної частини курсової роботи.
2	DataGrip Workbench	Програмне забезпечення яке надає легкий графічний інтерфейс для доступу до бази даних.

Тексти програмного коду наведені в окремому документі «Текст програми».

3.4 Аналіз безпеки даних

Забезпечення належного рівня безпеки даних є критично важливим аспектом функціонування інформаційної системи розробленої в рамках дипломного проєкту, оскільки вона обробляє як персональні дані користувачів, так і інформацію, що стосується їхніх пошукових запитів та результатів роботи алгоритму, яка може мати комерційну цінність. Даний аналіз розглядає основні заходи безпеки, впроваджені в системі

На рівні логіки вебзастосунку реалізовані наступні заходи безпеки.

Система автентифікації, побудована з використанням розширення Flask-Login, відповідає за перевірку ідентичності користувачів. Об'єкт login_manager належним чином ініціалізований, із зазначенням маршруту для входу (login_view = 'signin'). Авторизація доступу до окремих функцій та даних реалізується через декоратор @login_required, який ефективно захищає маршрути /query, /logout та /download_result від неавторизованого доступу. Маршрут /download_result додатково перевіряє, чи поточний користувач є власником запиту (search_query_item.user_id != current_user.id).

Управління сесіями користувачів здійснюється за допомогою бібліотеки Flask-Login, який використовує підписані сесійні cookie, захищені за допомогою app.secret_key.

Для протидії поширеним вебвразливостям вжито відповідних заходів. Ризик SQL-ін'єкцій значно знижується завдяки використанню ORM SQLAlchemy, яка параметризує запити до бази даних. Захист від міжсайтового скриптингу (XSS) частково забезпечується автоматичним екрануванням змінних у HTML-шаблонах шаблонізатором Jinja2, що використовується Flask за замовчуванням. Класи форм, успадковані від класу FlaskForm (з бібліотеки Flask-WTF), автоматично включають CSRF-токен, що забезпечує захист від цього типу атак для всіх POST-запитів, оброблених через ці форми. Валідація вхідних даних, що надходять від користувача через форми, також здійснюється за допомогою валідаторів бібліотеки wtforms (DataRequired, Email, Length, EqualTo), що запобігає введенню некоректних або потенційно шкідливих даних.

Обробка помилок у додатку використовує механізм flash для інформування користувача. У маршрутах завантаження файлів та обробки запитів присутнє логування помилок за допомогою app.logger.error.

Висновки до розділу

У ході виконання даного розділу було здійснено комплекс робіт з проектування та розробки ключових аспектів програмного забезпечення інформаційної системи Sieve. Було детально опрацьовано архітектуру програмного продукту, що включає вебдодаток на основі Flask, модуль асинхронної обробки даних algorithm_app та реляційну базу даних PostgreSQL. На основі аналізу вимог та специфіки проекту було обґрунтовано вибір технологічного стеку, зокрема мови програмування Python, мікрофреймворку Flask та системи управління базами даних PostgreSQL.

Центральною частиною роботи стало конструювання програмного забезпечення, під час якого було розроблено основні функціональні модулі системи. Це включає механізми реєстрації та автентифікації користувачів, інтерфейс для створення та управління пошуковими запитами, логіку асинхронного запуску алгоритму обробки, а також функціонал для відображення статусів запитів та завантаження файлів результатів. Особливу увагу було приділено розробці алгоритму Sieve, що реалізує конвеєрну обробку вебпосилань за допомогою послідовності спеціалізованих фільтрів, та його інтеграції з основним вебдодатком.

Також у розділі було визначено та реалізовано структуру бази даних за допомогою моделей SQLAlchemy, що забезпечує ефективне зберігання та управління даними користувачів, пошукових запитів та результатів аналізу. Завершальним етапом став аналіз безпеки даних розробленого вебзастосунку, в ході якого було розглянуто основні вектори загроз та впроваджені базові механізми захисту на рівні передачі даних, їх зберігання та логіки додатку.

Таким чином, у цьому розділі було закладено інженерно-технічну та практичну основу для функціонування інформаційної системи Sieve. Розроблені

архітектурні рішення та програмні компоненти підтверджують життєздатність концепції та створюють фундамент для подальшого тестування, вдосконалення та потенційного впровадження системи. Результатом роботи над розділом є функціональний прототип вебзастосунку, що реалізує ключовий заявлений функціонал.

4 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Аналіз якості ПЗ

Оцінка якості розробленого програмного забезпечення є важливим етапом, що дозволяє визначити ступінь відповідності системи заявленим характеристикам та очікуванням. Цей аналіз проводиться на основі визначених нефункціональних вимог та обґрунтованих метрик якості, що дозволяють кількісно та якісно оцінити різні аспекти функціонування програмного продукту.

Аналіз якості буде здійснено шляхом послідовного розгляду кожної нефункціональної вимоги. Для кожної вимоги буде проаналізовано, наскільки архітектура та реалізація системи, відображені у наданій кодовій базі, сприяють її виконанню. Обрані метрики якості включають як кількісні показники, так і якісні оцінки, що базуються на аналізі дизайну та структури застосунку.

Продуктивність системи є однією з ключових вимог, що передбачає час відповіді не більше трьох годин для типового пошукового запиту, який включає до двадцяти ключових слів. Враховуючи, що обробка запиту включає потенційно тривалі операції вебскрапінгу та аналізу даних такий часовий ліміт виглядає досяжним для пакетної обробки. Для підтвердження цієї вимоги необхідно провести навантажувальне тестування, імітуючи типові запити та вимірюючи середній та максимальний час їх повного виконання. Метриками тут слугуватимуть середній час обробки запиту, а також 95-й перцентиль часу відповіді.

Масштабованість архітектури передбачає можливість обробки одночасних запитів від щонайменше 5 активних користувачів без суттєвого зниження продуктивності. Для перевірки цієї вимоги слід провести тестування масштабованості, поступово збільшуючи кількість одночасних користувачів, що ініціюють обробку запитів, та моніторити час відповіді системи, рівень завантаження процесора, використання пам'яті та кількість активних потоків. Якщо десять користувачів одночасно ініціюють виконання алгоритму, поточна

конфігурація з двома робочими потоками призведе до утворення черги. Для досягнення заявленої масштабованості може знадобитися збільшення max_workers або перехід до більш масштабованих рішень для фонових задач (наприклад, Celery).

Доступність системи на рівні не менше 99% протягом календарного місяця є стандартною вимогою для вебдодатків. Її досягнення залежить від стабільності серверного оточення, бази даних PostgreSQL та самого Flask-додатку. Впроваджені механізми обробки помилок (try-except блоки, db.session.rollback()) та логування (app.logger.error) сприяють підвищенню надійності. Розрахунок фактичної доступності проводиться за стандартною формулою, враховуючи заплановані періоди технічного обслуговування.

Зручність користування оцінюється через інтуїтивну зрозумілість інтерфейсу для виконання основних операцій без потреби у додатковому навчанні. Метриками будуть слугувати час виконання типових завдань (реєстрація, створення запиту, завантаження результату), кількість помилок користувача та суб'єктивна оцінка задоволеності.

Кросплатформенність та кросбраузерність передбачають коректну роботу у поширених сучасних браузерах та операційних системах. Оскільки фронтенд генерується на сервері за допомогою Jinja2 та стандартних HTML/CSS, очікується високий рівень сумісності. Однак, для підтвердження необхідно провести тестування ключового функціоналу на заявлених платформах (Google Chrome, Mozilla Firefox).

Модульність системи є важливою якісною характеристикою, що впливає на супровід, тестування та розширення функціоналу. Вимога щодо реалізації системи у вигляді окремих логічних модулів частково підтверджується структурою коду: відокремлення algorithm_app, використання класів для моделей даних (User, SearchQuery, SearchResult), форм (FlaskForm) та логіки маршруті. Для більш глибокої оцінки модульності застосовуються метрики складності коду. Однією з таких метрик є цикломатична складність, яка вимірює кількість лінійно незалежних шляхів через вихідний код програми. Нижчі

значення цикломатичної складності для окремих функцій та методів свідчать про кращу структурованість, легкість розуміння та тестування коду, що опосередковано вказує на вищу якість модульного поділу. Аналіз коду на предмет зв'язності всередині модулів та зчеплення між модулями також є важливим; бажано досягати високої зв'язності та низького зчеплення.

Розширюваність архітектури має забезпечувати можливість додавання нових джерел даних для скрапінгу або нових форматів експорту без істотних змін у наявному коді. Архітектура алгоритму, зокрема використання класу Pipeline, сприяє розширюваності. На рівні Flask-додатку, додавання нових форматів експорту може бути реалізовано через створення нових маршрутів та функцій обробки. Оцінка цієї вимоги значною мірою базується на аналізі дизайну та гнучкості ключових компонентів, зокрема модулю алгоритму та механізмів взаємодії з ним.

4.2 Опис процесів тестування

Тестування виконується згідно документу «Програма та методика тестування».

Було виконане мануальне тестування програмного забезпечення, опис відповідних тестів наведено у таблицях 4.1 – 4.15.

Таблиця 4.1 – Тест 1.1 Реєстрація нового користувача з валідними даними

Початковий стан системи	Користувач не автентифікований. Відсутній користувач з тестовими даними в системі.

Продовження таблиці 4.1

Вхідні дані	Ім'я користувача: 123 Пароль: 123 Підтвердження пароля: 123
Опис проведення тесту	Відкрити головну сторінку. Натиснути на посилання "Sign Up". Заповнити поля форми реєстрації валідними даними. Натиснути кнопку "Sign Up".
Очікуваний результат	Система успішно реєструє нового користувача. Користувача перенаправлено на головну сторінку, з повідомленням про успішну реєстрацію. У базі даних створено запис для нового користувача з гешованим паролем.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.2 – Тест 1.2 Спроба реєстрації з існуючим ім'ям користувача

Початковий стан системи	Користувач не автентифікований. У системі існує користувач з ім'ям 123.
Вхідні дані	Ім'я користувача: 123 Пароль: 123 Підтвердження пароля: 123
Опис проведення тесту	Відкрити головну сторінку. Натиснути на посилання "Sign Up". Заповнити поля форми реєстрації, вказавши ім'я користувача, яке вже існує. Натиснути кнопку "Sign Up".

Продовження таблиці 4.2

Очікуваний результат	Система не реєструє нового користувача. На сторінці реєстрації відображається повідомлення про помилку
Фактичний результат	Збігається з очікуваним.

Таблиця 4.3 – Тест 1.3 Автентифікація користувача з валідними даними

Початковий стан системи	Користувач не автентифікований. У системі існує зареєстрований користувач з ім'ям 123 та паролем 123.
Вхідні дані	Ім'я користувача: 123 Пароль: 123
Опис проведення тесту	Відкрити головну сторінку. Натиснути на посилання "Sign In". Заповнити поля форми входу валідними даними. Натиснути кнопку "Sign In".
Очікуваний результат	Система успішно автентифікує користувача. Користувача перенаправлено на головну сторінку. У навігаційній панелі замість посилань "Sign Up"/"Sign In" відображається ім'я користувача та посилання "Log Out" та “Query”. Відображається повідомлення про успішний вхід.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.4 – Тест 1.4 Спроба автентифікації з невірним паролем

Початковий стан системи	Користувач не автентифікований. У системі існує зареєстрований користувач з ім'ям 123.
Вхідні дані	Ім'я користувача: 123 Пароль: WrongPassword
Опис проведення тесту	Відкрити головну сторінку. Натиснути на посилання "Sign In". Заповнити поля форми входу, вказавши правильне ім'я користувача та невірний пароль. Натиснути кнопку "Sign In".
Очікуваний результат	Система не автентифікує користувача. На сторінці входу відображається повідомлення про помилку.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.5 – Тест 1.5 Створення нового пошукового запиту (успішне)

Початковий стан системи	Користувач автентифікований в системі.
Вхідні дані	Ключові слова: QR menu, QR payment, restaurants, provider, subscription, café, bar, create menu, easy, online, customer, benefits, POS system, scan, order, pay, smart, free, dine in, e-menu, serve, reduce, increase, tips, sale, digital Країна: United Kingdom Whitelist Words: qr code, qr-code, qr ordering, qr-ordering, digital menu, qr order, qr-order, qr menu, qr-menu, digital payment, digital ordering, online order, contactless menu, contactless ordering

Продовження таблиці 4.5

Опис проведення тесту	Перейти на сторінку "Query". Заповнити всі поля форми створення запиту валідними даними. Натиснути кнопку "Find".
Очікуваний результат	Система приймає запит. Відображається повідомлення про успішне збереження запиту та початок обробки. Новий запит з'являється у списку "Your Submitted Queries" зі статусом "submitted" або "processing". У базі даних створено відповідний запис в таблиці search_query. Асинхронний процес обробки алгоритму запущено.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.6 – Тест 1.6 Спроба створення пошукового запиту з незаповненими

Початковий стан системи	Користувач автентифікований в системі.
Вхідні дані	Ключові слова: (пусто) Країна: Portugal Whitelist Words: tourism tech
Опис проведення тесту	Перейти на сторінку "Query". Залишити поле "Keywords" порожнім, заповнити інші поля. Натиснути кнопку "Find".
Очікуваний результат	Система не приймає запит. Під незаповненим обов'язковим полем (або загалом для форми) відображається повідомлення про помилку валідації. Запис у базі даних не створюється.

Продовження таблиці 4.6

Фактичний результат	Збігається з очікуваним.

Таблиця 4.7 – Тест 1.7 Перевірка оновлення статусу запиту після завершення роботи алгоритму

Початковий стан системи	Користувач автентифікований. Існує пошуковий запит (ID: Y), для якого алгоритм успішно завершив роботу.
Вхідні дані	Відсутні (перевірка стану існуючого об'єкта).
Опис проведення тесту	Перейти на сторінку "Query". Знайти у списку "Your Submitted Queries" запит з ID: Y.
Очікуваний результат	Статус запиту з ID: Y відображається як "completed". Поруч із запитом з'являється кнопка/посилання "Download Results".
Фактичний результат	Збігається з очікуваним.

Таблиця 4.8 – Тест 1.8 Завантаження файлу результатів для завершеного запиту

Початковий стан системи	Користувач автентифікований. Існує пошуковий запит (ID: Z) зі статусом "completed". Відповідний XLSX файл існує на сервері, шлях до нього коректно збережений у search_result.website_url.
Вхідні дані	Відсутні (взаємодія з існуючим елементом UI).
Опис проведення тесту	Перейти на сторінку "Query". Знайти запит з ID: Z. Натиснути на кнопку/посилання "Download Results" для цього запиту.

Продовження таблиці 4.8

Очікуваний результат	Браузер ініціює завантаження XLSX файлу. Назва файлу відповідає очікуваній (analysed_links.xlsx).
Фактичний результат	Збігається з очікуваним.

Таблиця 4.9 – Тест 1.9 Спроба завантаження результатів для запиту зі статусом "processing"

Початковий стан системи	Користувач автентифікований. Існує пошуковий запит (ID: P) зі статусом "processing".
Вхідні дані	Відсутні.
Опис проведення тесту	Перейти на сторінку "Query". Знайти запит з ID: P.
Очікуваний результат	Кнопка/посилання "Download Results" для запиту з ID: P відсутня.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.10 – Тест 1.10 Вихід користувача з системи (Logout)

Початковий стан системи	Користувач автентифікований в системі.
Вхідні дані	Відсутні.
Опис проведення тесту	Натиснути на посилання "Log Out" у навігаційній панелі.
Очікуваний результат	Користувача виведено з системи. Користувача перенаправлено на головну сторінку (або сторінку входу). У навігаційній панелі відображаються посилання "Sign Up" та "Sign In".

Продовження таблиці 4.10

Фактичний результат	Збігається з очікуваним.

Таблиця 4.11 – Тест 1.11 Перевірка валідації довжини пароля при реєстрації (мінімальна довжина)

Початковий стан системи	Користувач не автентифікований.
Вхідні дані	Ім'я користувача: shortpassuser Email: shortpass@example.com Пароль: 12345 (менше 6 символів) Підтвердження пароля: 12345
Опис проведення тесту	Відкрити сторінку "Sign Up". Заповнити форму, вказавши пароль коротший за мінімально дозволений. Натиснути кнопку "Sign Up".
Очікуваний результат	Система не реєструє користувача. Під полем "Password" відображається повідомлення про помилку валідації.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.12 – Тест 1.12 Перевірка валідації невідповідності пароля та підтвердження пароля

Початковий стан системи	Користувач не автентифікований.
Вхідні дані	Ім'я користувача: mismatchuser Email: mismatch@example.com Пароль: Password123 Підтвердження пароля: Password456

Продовження таблиці 4.12

Опис проведення тесту	Відкрити сторінку "Sign Up". Заповнити форму, вказавши різні значення для пароля та його підтвердження. Натиснути кнопку "Sign Up".
Очікуваний результат	Система не реєструє користувача. Під полем "Confirm Password" відображається повідомлення про помилку валідації.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.13 – Тест 1.13 Спроба доступу до сторінки "Query" неавторизованим користувачем

Початковий стан системи	Користувач не автентифікований.
Вхідні дані	URL: /query.
Опис проведення тесту	Спробувати перейти безпосередньо за URL-адресою /query.
Очікуваний результат	Системою не надано доступ до сторінки /query.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.14 – Тест 1.14 Спроба завантаження результату іншого користувача

Початковий стан системи	Користувач А (userA) автентифікований. Існує завершений запит (ID: X), що належить користувачу Б (userB).

Продовження таблиці 4.14

Вхідні дані	URL: /download_result/X (де X – ID запиту користувача Б).
Опис проведення тесту	Користувач А намагається перейти за прямою URL-адресою для завантаження результату запиту X, який належить користувачу Б.
Очікуваний результат	Система не дозволяє завантаження файлу.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.15 – Тест 1.15 Перевірка роботи кнопки "Home" з різних сторінок

Початковий стан системи	Користувач знаходиться на сторінці "Query" (або "Sign In", "Sign Up").
Вхідні дані	Відсутні.
Опис проведення тесту	Натиснути на посилання/кнопку "Home" у верхньому лівому куті сторінки.
Очікуваний результат	Користувача перенаправлено на головну сторінку вебзастосунку (маршрут /).
Фактичний результат	Збігається з очікуваним.

4.3 Опис контрольного прикладу

Для демонстрації ключового функціоналу розробленого вебзастосунку Sieve та ілюстрації основних сценаріїв взаємодії користувача з системою, розглянемо контрольний приклад. Цей приклад охоплює повний цикл роботи:

від реєстрації нового користувача до створення пошукового запиту, моніторингу його обробки та отримання фінальних результатів.

– Етап 1: Реєстрація нового користувача в системі. Першим кроком для нового користувача є створення облікового запису. Користувач відкриває головну сторінку вебзастосунку.

У навігаційній панелі він обирає опцію "Sign Up" (Зареєструватися), після чого система перенаправляє його на сторінку реєстрації.

На цій сторінці користувач заповнює обов'язкові поля: унікальне ім'я користувача, дійсну адресу електронної пошти, та пароль, який відповідає вимогам безпеки, а також підтвердження пароля. Після введення всіх даних користувач натискає кнопку "Sign Up". Система валідує введені дані. У випадку успішної валідації та відсутності конфліктів (наприклад, вже існуючого користувача з таким ім'ям або email), створюється новий обліковий запис, пароль зберігається у гешованому вигляді.

Користувач отримує повідомлення про успішну реєстрацію (наприклад, "Account created successfully! You can now sign in.") та, як правило, перенаправляється на сторінку входу для подальшої автентифікації. У разі некоректного введення даних (наприклад, невідповідність паролів, невалідний формат email, занадто коротке ім'я користувача) система відображає відповідні повідомлення про помилки безпосередньо у формі реєстрації, дозволяючи користувачеві виправити введені дані.

– Етап 2: Автентифікація зареєстрованого користувача. Після успішної реєстрації (або якщо користувач вже має обліковий запис), він здійснює вхід до системи. Для цього користувач переходить на сторінку "Sign In" (Увійти).

Тут він вводить своє ім'я користувача та пароль, вказані під час реєстрації. Після натискання кнопки "Sign In", система перевіряє відповідність наданих облікових даних тим, що зберігаються в базі. У разі успішної автентифікації користувач перенаправляється на основну робочу сторінку застосунку. Важливою ознакою успішного входу є зміна навігаційної панелі: замість опцій "Sign Up" та "Sign In" з'являється ім'я поточного користувача та опції "Log Out"

(Вийти) та “Query” (Сторінка запитів). Якщо ж введені дані невірні, система відображає повідомлення про помилку автентифікації.

– Етап 3: Створення та відправлення пошукового запиту. Автентифікований користувач переходить на сторінку "Query" (Запити). На цій сторінці розташована форма для створення нового пошукового запиту.

Користувач заповнює поля форми Keywords (Ключові слова), Country (Країна), Whitelist Words (Білий список слів).

Після заповнення всіх необхідних полів користувач натискає кнопку "Find" (Знайти). Система проводить валідацію введених даних. У разі успіху, параметри запиту зберігаються у базі даних (таблиця search_query) з початковим статусом "submitted" (Відправлено). Користувач бачить повідомлення про успішне створення запиту та початок його обробки, наприклад, "Query saved (ID: 123) and processing started!".

Новий запит негайно з'являється у списку "Your Submitted Queries" (Ваші відправлені запити) на цій же сторінці, відображаючи його ID, параметри та поточний статус.

Одночасно, у фоновому режимі, система ініціює асинхронний запуск модуля algorithm_app для обробки цього запиту. Після відправлення запиту користувач може відслідковувати процес його виконання на сторінці "Query". Статус запиту у списку при оновленні сторінки змінюється, відображаючи етапи роботи алгоритму. Спочатку статус може змінитися на "processing" (Обробляється), вказуючи на те, що модуль algorithm_app активно виконує вебскрейпінг та аналіз даних. Тривалість цього етапу залежить від складності запиту, кількості ключових слів та обсягу даних, що аналізуються, і може становити від декількох хвилин до декількох годин, відповідно до нефункціональних вимог. Після завершення всіх етапів обробки алгоритмом, статус запиту оновлюється на "completed" (Завершено) у випадку успішного виконання, або "failed" (Помилка) у випадку виникнення проблем під час обробки.

– Етап 4: Отримання та завантаження результатів. Коли статус запиту користувача змінюється на "completed", поруч із ним у списку запитів з'являється активне посилання або кнопка "Download Results" (Завантажити результати).

Користувач натискає на це посилання. Система, перевіривши права доступу користувача до даного запиту та наявність файлу, ініціює завантаження згенерованого XLSX-файлу на локальний комп'ютер користувача. Цей файл містить структуровану інформацію про знайдених потенційних B2B-клієнтів, що відповідають критеріям запиту. Якщо файл з якихось причин відсутній на сервері або запит ще не завершено, система виводить відповідне інформаційне повідомлення.

– Етап 5: Навігація та вихід із системи

Протягом роботи з системою користувач може використовувати навігаційні елементи. Наприклад, натиснувши на посилання "Home" у верхньому лівому куті, користувач повертається на головну сторінку застосунку. По завершенню роботи, для виходу зі свого облікового запису, користувач натискає на посилання "Log Out" у навігаційній панелі. Система завершує поточну сесію користувача та перенаправляє його на головну сторінку Навігаційна панель повертається до стану для неавторизованого користувача, відображаючи опції "Sign Up" та "Sign In".

Усі креслення екранних форм наведені в графічних матеріалах (аркуш 4).

Висновки до розділу

У даному розділі дипломної роботи було проведено всебічне дослідження аспектів якості та процедур тестування розробленого програмного забезпечення Sieve. Основною метою цього етапу було забезпечення відповідності створеного вебзастосунку встановленим функціональним і нефункціональним вимогам, а також підтвердження його працездатності та надійності перед потенційним впровадженням.

Першим ключовим компонентом розділу став детальний аналіз якості програмного забезпечення. Цей аналіз базувався на попередньо сформульованих нефункціональних вимогах, що охоплювали такі критичні характеристики системи, як продуктивність, масштабованість, доступність, зручність користування, кросплатформенність, модульність та розширюваність. Для оцінки ступеня відповідності цим вимогам було розглянуто архітектурні рішення та особливості реалізації системи. Результати аналізу якості дозволили зробити попередні висновки про сильні сторони архітектури, такі як використання асинхронної обробки для ресурсомістких завдань та модульний підхід.

Другий підрозділ був присвячений опису процесів тестування програмного забезпечення Sieve. Було окреслено загальну стратегію тестування, яка передбачає комплексний підхід, що включає функціональне тестування для перевірки коректності реалізації всіх користувацьких сценаріїв та бізнес-логіки, нефункціональне тестування для оцінки відповідності вимогам до продуктивності, надійності та зручності використання, а також тестування користувацького інтерфейсу для забезпечення його інтуїтивності та кросбраузерної сумісності. В рамках підготовки до тестування було розроблено набір деталізованих тест-кейсів, що покривають основний функціонал системи. Ці тест-кейси слугують основою для систематичної перевірки працездатності програмного продукту.

Третім важливим елементом розділу став опис контрольного прикладу роботи системи. Цей приклад детально ілюструє наскрізний сценарій взаємодії типового користувача з вебзастосунком Sieve, починаючи від процесу реєстрації і до успішного завантаження згенерованого файлу результатів. Опис контрольного прикладу супроводжувався посиланнями на ілюстрації ключових екранів та інтерфейсів, що дозволило наочно продемонструвати основні функціональні можливості системи та підтвердити логічність і послідовність користувацьких сценаріїв.

Таким чином, робота, проведена в рамках цього розділу, забезпечила глибоку оцінку якості розробленого програмного забезпечення Sieve та створила методологічну основу для його всебічного тестування. Проведений аналіз нефункціональних вимог, описані процеси тестування разом із розробленими тест-кейсами та демонстрація ключового функціоналу на контрольному прикладі підтверджують відповідність системи основним проектним цілям на даному етапі. Результати цього розділу є підґрунтям для проведення фінального етапу тестування, виявлення та усунення можливих недоліків, та, в кінцевому підсумку, для підготовки програмного продукту до практичного застосування.

5 РОЗГОРТАННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

5.1 Розгортання програмного забезпечення

У даному підрозділі детально розглядається процес розгортання розробленого вебзастосунку Sieve на сервері для забезпечення його доступності кінцевим користувачам. Описуються обрані засоби розгортання, наводиться концептуальна діаграма та покрокова інструкція для налаштування виробничого середовища.

Для розгортання програмного застосунку використовуються технології контейнеризації Docker та інструменту оркестрації Docker Compose. Таке рішення обрано для забезпечення ізоляції компонентів, консистентності оточень на різних етапах життєвого циклу програмного продукту (розробка, тестування, продакшн) та спрощення процесів розгортання та масштабування.

Архітектура розгортання "Sieve.ai" передбачає використання трьох основних взаємопов'язаних Docker-контейнерів:

– Контейнер бази даних PostgreSQL (postgres_db): Базується на офіційному образі postgres з Docker Hub. Відповідає за зберігання всіх персистентних даних системи (інформація про користувачів, пошукові запити, результати). Для забезпечення збереження даних при перезапусках контейнера використовується Docker volume, що монтується до директорії зберігання даних PostgreSQL всередині контейнера. Конфігурація (ім'я бази даних, користувач, пароль) задається через змінні середовища.

– Контейнер вебдодатку Flask (flask_app): Створюється на основі кастомного Dockerfile з базовим образом Python. Включає код основного Flask-додатку (app.py, шаблони, статичні файли), встановлює необхідні залежності з requirements.txt. Конфігураційні параметри, такі як SECRET_KEY та SQLALCHEMY_DATABASE_URI (що вказує на сервіс postgres_db у внутрішній Docker-мережі), передаються через змінні середовища. Цей контейнер також відповідає за ініціалізацію асинхронних завдань для обробки алгоритмом.

– Контейнер модуля алгоритму Sieve (algorithm_service): Також базується на образі Python та містить код модуля algorithm_app з його специфічними залежностями. Цей контейнер реалізує внутрішній API-ендпоінт, який приймає запити на запуск алгоритму обробки. Він взаємодіє з контейнером postgres_db для отримання деталей запиту (включаючи whitelist-слова) та для оновлення статусу запиту й збереження шляхів до файлів результатів. Згенеровані XLSX-файли зберігаються у спільному Docker volume, доступному також для контейнера flask_app (для реалізації функції завантаження).

Для наглядніщого розуміння процесу розгортання нижче наведено діаграму розгортання (рисунок 5.1).

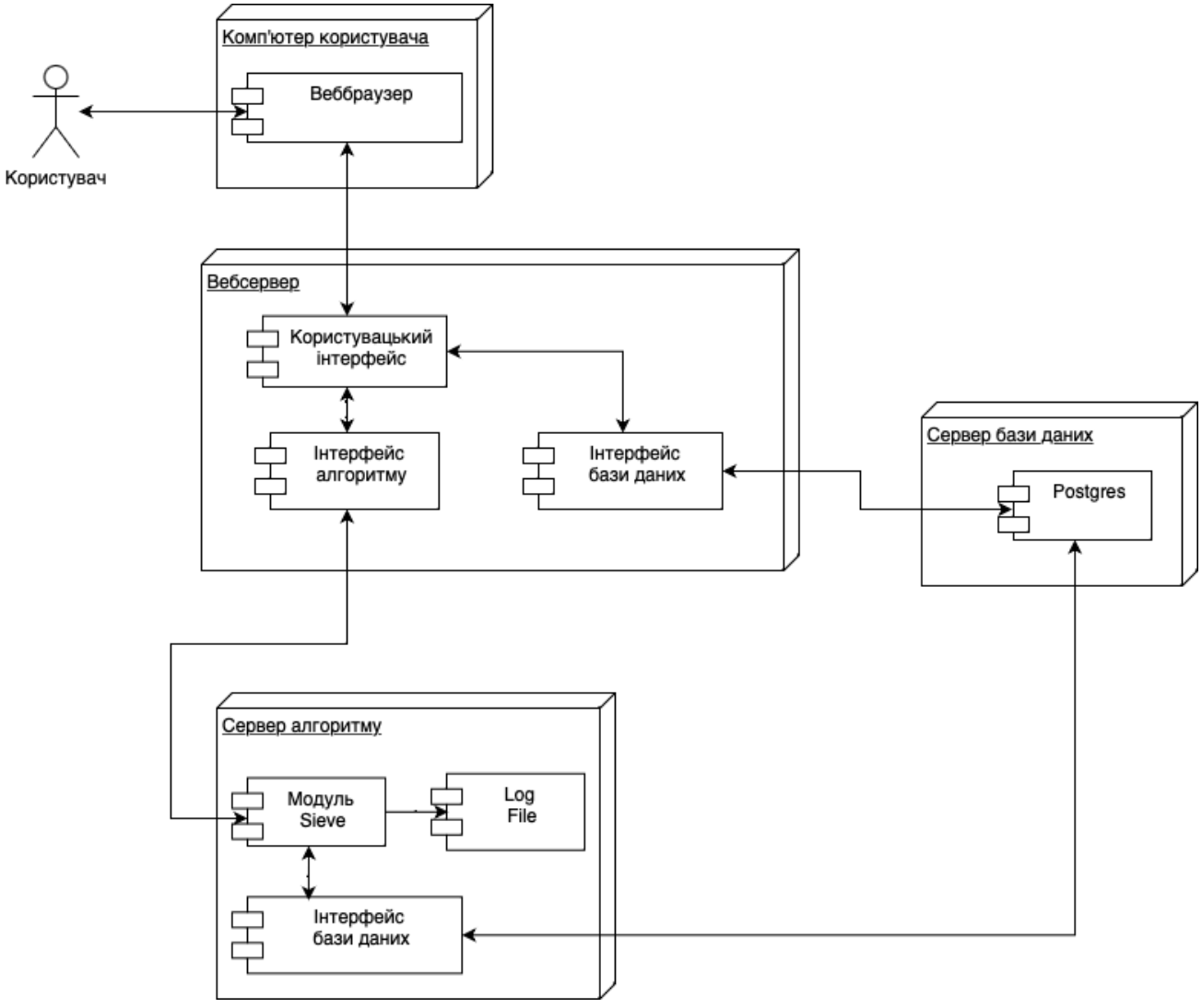


Рисунок 5.1 – Діаграма розгортання Sieve

Розгортання вебзастосунку за допомогою Docker та Docker Compose включає наступні етапи:

– На цільовому сервері встановлюються останні стабільні версії Docker Engine та Docker Compose. Створюється структура директорій для проекту, якщо код буде монтуватися в контейнери, або для зберігання docker-compose.yml та Dockerfile-ів.

– Створення Dockerfile для вебдодатку Flask. Цей файл описує процес побудови образу для Flask-додатку. Він включає такі етапи як: вибір базового образу Python, встановлення робочої директорії всередині контейнера, копіювання файлу requirements.txt та встановлення залежностей за допомогою pip install --no-cache-dir -r requirements.txt, копіювання всього коду Flask-додатку (включаючи app.py, директорії templates, static).

– Створення Dockerfile для модуля алгоритму Sieve. Аналогічно до Flask-додатку, створюється Dockerfile для сервісу алгоритму, що включає такі етапи як: вибір базового образу Python, встановлення робочої директорії, копіювання файлу залежностей для algorithm_app та їх встановлення та копіювання коду algorithm_app та конфігураційного файлу config.yaml.

– Створення файлу Docker Compose (docker-compose.yml). Цей файл є центральним для оркестрації всіх контейнерів. В ньому вказується версія синтаксису Docker Compose, визначаються всі необхідні сервіси (postgres_db, flask_app, algorithm_service) та їх детальні налаштування.

5.2 Супровід програмного забезпечення

Завершення розробки та розгортання інформаційної системи "Sieve.ai" є початком її життєвого циклу, який потребує належного супроводу для забезпечення стабільної, безпечної та ефективної експлуатації. Процес супроводу програмного забезпечення (ПЗ) охоплює моніторинг функціонування системи, управління виправленнями та оновленнями, обслуговування бази даних, підтримку користувачів, управління конфігурацією та регулярне оновлення безпеки.

Для забезпечення безперебійної роботи системи "Sieve.ai" та своєчасного реагування на можливі проблеми впроваджується система моніторингу. На рівні

додатку здійснюється збір та аналіз логів роботи Flask-сервера та модуля алгоритму. Розглядається можливість інтеграції зі спеціалізованими системами трекінгу помилок (наприклад, Sentry) для автоматизованого збору та агрегації інформації про виняткові ситуації. Моніторинг серверної інфраструктури включає відстеження ключових показників, таких як завантаження центрального процесора, використання оперативної пам'яті, заповненість дискового простору та мережевий трафік. Окрему увагу приділено моніторингу стану та продуктивності бази даних PostgreSQL, включаючи регулярне створення резервних копій. Для контролю доступності вебзастосунку використовуються зовнішні сервіси перевірки аптайму.

Процес внесення виправлень, вдосконалень та нового функціоналу до системи Sieve базується на принципах версіонування коду та автоматизації процесів збірки та розгортання.

Весь вихідний код проекту зберігається у системі контролю версій Git (на платформі GitHub), що дозволяє відстежувати зміни, працювати з різними гілками розробки (наприклад, main для стабільних версій, develop для поточної розробки, окремі гілки для нових функцій та виправлень) та повертатися до попередніх версій у разі потреби. Для управління завданнями, повідомленнями про помилки та запитами на нові функції використовується система трекінгу задач (GitHub Issues). Для автоматизації процесу доставки оновлень та нових версій застосунку у виробниче середовище, яке використовує Docker-контейнери, налаштовується конвеєр безперервної інтеграції та безперервного розгортання (CI/CD) за допомогою сервісу GitHub Actions.

Підтримка бази даних PostgreSQL включає регулярне автоматизоване створення резервних копій для запобігання втраті даних. Також здійснюється моніторинг продуктивності запитів до бази даних та використання дискового простору. Застосування змін до схеми бази даних у процесі оновлення програмного забезпечення відбувається контрольовано за допомогою системи міграцій (Flask-Migrate), що дозволяє версіонувати зміни та безпечно їх застосовувати або відкочувати.

Висновки до розділу

У даному розділі дипломного проєкту було детально розглянуто практичні аспекти завершальної фази життєвого циклу програмного забезпечення Sieve, а саме процеси його розгортання у потенційному виробничому середовищі та стратегії подальшого супроводу. Метою цього етапу було визначення оптимальних підходів для забезпечення доступності, стабільності, безпеки та довготривалої життєздатності розробленої інформаційної системи.

Ключовим рішенням для розгортання вебзастосунку Sieve було обрано сучасну технологію контейнеризації з використанням інструментів Docker та Docker Compose. Такий підхід дозволяє інкапсулювати окремі компоненти системи – вебдодаток на базі Flask, модуль алгоритму Sieve та систему управління базами даних PostgreSQL – у вигляді ізольованих, портативних контейнерів. У підрозділі було детально описано покроковий процес розгортання, що включає підготовку Docker-оточення на сервері, створення спеціалізованих Dockerfile для вебдодатку та модуля алгоритму, а також розробку файлу docker-compose.yml для оркестрації взаємодії цих контейнерів. Обрана стратегія розгортання сприяє відтворюваності середовища, спрощує оновлення та масштабування системи в майбутньому.

У контексті супроводу програмного забезпечення було визначено комплекс заходів, спрямованих на підтримку його надійної та безпечної роботи після введення в експлуатацію. Розглянуто процедури моніторингу ключових показників системи, включаючи стан сервера, бази даних та самого додатку, а також збір та аналіз логів для своєчасного виявлення та діагностики проблем. Особливу увагу приділено процесу управління виправленнями та оновленнями, де центральну роль відіграє система контролю версій Git та автоматизований конвеєр безперервної інтеграції та безперервного розгортання (CI/CD) на базі GitHub Actions.

Таким чином, цей розділ демонструє не лише теоретичну готовність розробленого програмного продукту до впровадження, але й закладає інженерні та організаційні підвалини для його довгострокової та стабільної експлуатації.

Представлені стратегії розгортання з використанням контейнеризації та комплексний план супроводу підтверджують практичну спрямованість виконаної роботи та орієнтацію на створення надійного, безпечного та підтримуваного програмного рішення, здатного адаптуватися до майбутніх викликів та потреб користувачів.

ВИСНОВКИ

У ході виконання даного дипломного проєкту було успішно досягнуто поставленої мети – підвищити ефективність пошуку потенційних партнерів шляхом створення динамічного та масштабованого інструменту, здатного працювати з широким діапазоном галузей і регіонів. Результатом роботи є вебзастосунок, що надає користувачам інструментарій для ефективної генерації лідів на міжнародних ринках шляхом застосування сучасних методів вебскрапінгу та обробки даних.

В рамках дипломного проєкту було розроблено повнофункціональний вебзастосунок Sieve, реалізований на мові програмування Python з використанням мікрофреймворку Flask та включає наступні ключові компоненти та функціональні можливості: захищений механізм реєстрації та автентифікації користувачів, інтерфейс для формування комплексних пошукових запитів із зазначенням ключових слів, цільової країни та специфічних "білих списків" слів, систему асинхронного запуску та виконання основного аналітичного модуля, який відповідає за вебскрапінг, обробку та фільтрацію даних, механізми для зберігання параметрів запитів та їх результатів у реляційній базі даних PostgreSQL, інтерфейс для відстеження статусів виконання запитів ("submitted", "processing", "completed", "failed"), а також функціонал для завантаження оброблених результатів у форматі XLSX. Розгортання системи передбачено з використанням технології контейнеризації Docker, що забезпечує портативність та масштабованість рішення.

Розроблений програмний продукт є практичним інструментом, що дозволяє автоматизувати значну частину рутинних операцій з пошуку лідів, особливо у нішових або малодосліджених ринкових сегментах, де традиційні методи виявляються недостатньо ефективними.

Усі завдання, поставлені в рамках дипломного проєктування, були послідовно та повною мірою вирішені.

На першому етапі було проведено детальне передпроєктне обстеження предметної області, що включало аналіз існуючих проблем у сфері

B2B-лідогенерації, огляд наявних програмних продуктів, алгоритмічних та технічних рішень, а також моделювання бізнес-процесів, що обґрунтувало необхідність розробки запропонованої системи.

На другому етапі було сформульовано вичерпні вимоги до програмного забезпечення: розроблено варіанти використання, визначено функціональні та нефункціональні вимоги, проведено попередній аналіз економічних аспектів та сформовано постановку завдання на розробку.

Третій розділ був присвячений безпосередньому конструюванню та розробленню програмного забезпечення: детально спроектовано архітектуру системи, обґрунтовано вибір технологічного стеку, реалізовано ключові функціональні модулі, включаючи серверну логіку на Flask, модуль алгоритму Sieve з його конвеєрною обробкою даних, та структуру бази даних за допомогою SQLAlchemy. Також було проведено аналіз аспектів безпеки даних.

Четвертий розділ охопив питання аналізу якості та тестування розробленого програмного забезпечення; було оцінено відповідність системи нефункціональним вимогам за допомогою обраних метрик, описано процеси тестування та розроблено набір тест-кейсів, а також представлено контрольний приклад, що демонструє наскрізну роботу ключового функціоналу.

Нарешті, у п'ятому розділі було запропоновано сучасну стратегію розгортання програмного забезпечення з використанням Docker-контейнерів та окреслено комплекс заходів для його подальшого супроводу, включаючи моніторинг, управління оновленнями через CI/CD та забезпечення безпеки.

Результати, отримані в ході виконання дипломного проєкту, підтверджують високу практичну цінність розробленого вебзастосунку та доцільність подальшого розвитку в даній предметній області. Зростаюча потреба бізнесу в ефективних інструментах для міжнародної експансії та пошуку клієнтів, особливо в умовах цифрової трансформації, створює сприятливі умови для вдосконалення та розширення функціоналу системи.

Перспективними напрямками подальшої роботи є інтеграція більш складних алгоритмів машинного навчання та штучного інтелекту для глибшого

аналізу даних та скорингу потенційних лідів, розширення переліку джерел даних для вебскрапінгу, розробка більш гнучких аналітичних інструментів та звітів у самому додатку, вдосконалення користувацького інтерфейсу на основі зворотного зв'язку, а також перехід на більш потужні системи управління фоновими завданнями (наприклад, Celery) для забезпечення вищої масштабованості та надійності при обробці великої кількості запитів. Таким чином, продовження розробки та впровадження системи Sieve є обґрунтованим та має значний потенціал для створення комерційно успішного продукту.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1) B2B взаємодія: переваги та типи роботи. *allretail.ua* URL: https://allretail.ua/news/77631-b2b-vzayemodiya-perevagi-ta-tipi-roboti (дата звернення: 10.05.2025).

2) Apollo.io vs. zoominfo: where to get quality leads?. *phantombuster.com*. URL: https://phantombuster.com/blog/automation/apollo-vs-zoominfo-6p7a8SA2kRCDn1W7GnRPyC (дата звернення: 15.05.2025).

3) Шевченко К. Що таке LinkedIn та як з ним розвивати кар'єру. Кейси CTO та рекрутера. *journal.gen.tech*. URL: https://journal.gen.tech/post/sho-take-linkedin-ta-yak-nim-koristuvatisya (дата звернення: 17.05.2025).

4) Оберемок З. 14 найкращих інструментів для веб-скрапінгу. *claspo.io*. URL: https://claspo.io/ua/blog/14-best-web-scraping-tools-for-data-extraction-in-2023/ (дата звернення: 20.05.2025).

5) Антонюк Д. Бути єдинорогом тепер недостатньо. Інвесторам цікаві стартапи вищої ліги – декакорни. Хто вони такі. *forbes.ua*. URL: https://forbes.ua/innovations/byt-edinorogom-teper-nedostatochno-investoram-interesny-startapy-vysshey-ligi-dekakorny-kto-oni-takie-23112021-2820 (дата звернення: 18.05.2025).

6) Chitsaz S. CRM-enriched search filters. *crunchbase.com*. URL: https://support.crunchbase.com/hc/en-us/articles/6789628443667-CRM-enriched-search-filters (дата звернення: 23.05.2025).

7) Crunchbase About Us. *crunchbase.com*. URL: https://about.crunchbase.com/ (дата звернення: 11.06.2025).

8) Similarweb About Us. *similarweb.com*. URL: https://www.similarweb.com/corp/about/ (дата звернення: 11.06.2025).

9) AI Sales Platform. *apollo.io*. URL: https://www.apollo.io (дата звернення: 11.06.2025).

10) What is Salesforce. *salesforce.com*. URL: https://www.salesforce.com/products/what-is-salesforce/ (дата звернення: 11.06.2025).

11) LinkedIn Україна. *linkedin.com*. URL: https://ua.linkedin.com/ (дата звернення: 11.06.2025).

12) Django Project MVT Structure. *geeksforgeeks.org*. URL: https://www.geeksforgeeks.org/django-project-mvt-structure/ (дата звернення: 11.06.2025).

13) Welcome to Flask. *flask.palletsprojects.com*. URL: https://flask.palletsprojects.com/en/stable/ (дата звернення: 11.06.2025).

14) PostgreSQL. *postgresql.org*. URL: https://www.postgresql.org (дата звернення: 11.06.2025).

ДОДАТОК А ЗВІТ ПОДІБНОСТІ

StrikePlagiarism.com

Дата звіту 6/1/2025
Дата редагування 6/1/2025

Документ прийнятий

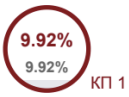
Звіт подібності

метадані

Назва організації
National Technical University of Ukraine Igor Sikorskyi Kyiv Politech Institute
Заголовок
ІП-12_Йолкін_ПЗ
Автор Науковий керівник / Експерт
ЙолкінЛіхоузова Т.А.
підрозділ
ФІОТ, К-а інформатики та програмної інженерії

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.



10	12167	96190
Довжина фрази для коефіцієнта подібності 2	Кількість слів	Кількість символів

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2025 р.

ВЕБЗАСТОСУНОК ДЛЯ ПОШУКУ B2B КЛІЄНТІВ ШЛЯХОМ ВЕБСКРАПІНГУ МЕРЕЖІ ІНТЕРНЕТ

Текст програми

КПІ.ІП-1212.045440.03.12

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Тетяна ЛІХОУЗОВА

Нормоконтроль:
_____ Максим ГОЛОВЧЕНКО

Виконавець:
_____ Даніїл ЙОЛКІН

Київ – 2025

Посилання на репозиторій з повним текстом програмного коду
https://github.com/DaniyilYolkin/sieveV2

Файл app.py

Реалізація функціональних задач формування запитів на пошук компаній за ключовими словами та захищеного доступ до функціоналу системи через кабінет користувача

```
from flask import Flask, render_template, request, redirect, url_for, flash, session, jsonify, send_file
from flask_sqlalchemy import SQLAlchemy
from flask_wtf import FlaskForm
from wtforms import StringField, PasswordField, SubmitField, SelectField
from wtforms.validators import DataRequired, Email, EqualTo, Length
from werkzeug.security import generate_password_hash, check_password_hash
from datetime import datetime
from flask_login import LoginManager, UserMixin, login_user, logout_user, login_required, current_user

from concurrent.futures import ThreadPoolExecutor
import atexit

import sys
import os

PROJECT_ROOT = os.path.abspath(os.path.join(os.path.dirname(__file__), '..'))
if PROJECT_ROOT not in sys.path:
  sys.path.insert(0, PROJECT_ROOT)

from algorithm_app.main import main_flask_async as process_query_algorithm

app = Flask(__name__)
app.secret_key = 'supersecretkey'
app.config['SQLALCHEMY_DATABASE_URI'] = 'postgresql://myuser:mysecretpassword@localhost:5432/mydb'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
db = SQLAlchemy(app)

login_manager = LoginManager(app)
login_manager.login_view = 'signin'
login_manager.login_message_category = 'info'

executor = ThreadPoolExecutor(max_workers=2)


# --- Models ---
```

```
class User(UserMixin, db.Model):
    __tablename__ = 'user'
    __table_args__ = {'schema': 'public'}

    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(80), unique=True, nullable=False)
    email = db.Column(db.String(120), unique=True, nullable=False)
    password_hash = db.Column(db.String(128))
    role = db.Column(db.String(50), default='user')
    created_at = db.Column(db.DateTime, default=datetime.utcnow)

    def set_password(self, password):
        self.password_hash = generate_password_hash(password)

    def check_password(self, password):
        return check_password_hash(self.password_hash, password)

    def __repr__(self):
        return f'<User {self.username}>'


class SearchQuery(db.Model):
    __tablename__ = 'search_query'
    __table_args__ = {'schema': 'public'}

    id = db.Column(db.Integer, primary_key=True)
    user_id = db.Column(db.Integer, db.ForeignKey('public.user.id'), nullable=False)
    keywords = db.Column(db.String, nullable=False)
    country = db.Column(db.String)
    status = db.Column(db.String, default='submitted')
    created_at = db.Column(db.DateTime, default=datetime.utcnow)
    whitelist_words = db.Column(db.Text, nullable=True)

    def __repr__(self):
        return f'<SearchQuery {self.id} by User {self.user_id}>'


class SearchResult(db.Model):
    __tablename__ = 'search_result'
    __table_args__ = {'schema': 'public'}

    id = db.Column(db.Integer, primary_key=True, autoincrement=True)
        query_id = db.Column(db.Integer, db.ForeignKey('public.search_query.id'),
nullable=False)
    website_url = db.Column(db.String, nullable=False)
    description = db.Column(db.String, nullable=True)
    country = db.Column(db.String, nullable=True)
    is_exported = db.Column(db.Boolean, nullable=True, default=False)
    scraped_at = db.Column(db.DateTime, nullable=True, default=datetime.utcnow)

    def __repr__(self):
        return f'<SearchResult {self.id} for Query {self.query_id}>'
```

```
@login_manager.user_loader
def load_user(user_id):
  return User.query.get(int(user_id))


# --- Forms ---

class SignUpForm(FlaskForm):
      username = StringField('Username', validators=[DataRequired(), Length(min=3, max=80)])
  email = StringField('Email', validators=[DataRequired(), Email()])
  password = PasswordField('Password', validators=[DataRequired(), Length(min=6)])
   confirm_password = PasswordField('Confirm Password', validators=[DataRequired(), EqualTo('password')])
  submit = SubmitField('Sign Up')


class SignInForm(FlaskForm):
  username = StringField('Username', validators=[DataRequired()])
  password = PasswordField('Password', validators=[DataRequired()])
  submit = SubmitField('Sign In')


class QueryForm(FlaskForm):
  keywords = StringField('Keywords', validators=[DataRequired()])
  country = SelectField('Country', choices=[
    ('GB', 'United Kingdom'),
    ('PT', 'Portugal'),
    ('ES', 'Spain')
  ], validators=[DataRequired()])
      whitelist_words = StringField('Whitelist Words (comma-separated)', validators=[DataRequired()])
  submit = SubmitField('Find')


# --- Helpers for background task ---

def run_algorithm_task(query_id):
  """
  Wrapper to call the algorithm and handle application context for database access
  if the algorithm needs it directly (though it's better if it doesn't).
  """
  with app.app_context():
    if not hasattr(db, 'session'):
      print(
        f"CRITICAL DEBUG [run_algorithm_task]: Global 'db' object in app.py does NOT have a 'session' attribute before calling algorithm!")

    try:
      process_query_algorithm(
        query_id=query_id,
```

```
                db_instance=db,
                search_result_model=SearchResult,
                search_query_model=SearchQuery
            )
        except Exception as e:
            app.logger.error(f"Error in background algorithm task for query_id {query_id}: {e}", exc_info=True)


    def shutdown_executor():
      print("Shutting down ThreadPoolExecutor...")
      executor.shutdown(wait=True)
      print("ThreadPoolExecutor shut down.")


    atexit.register(shutdown_executor)


    # --- Routes (home, signup, signin, logout - as defined before) ---

    @app.route('/signup', methods=['GET', 'POST'])
    def signup():
      """Handles user sign-up."""
      if current_user.is_authenticated:
        return redirect(url_for('home'))
      form = SignUpForm()
      if form.validate_on_submit():
            existing_user = User.query.filter((User.username == form.username.data) | (User.email == form.email.data)).first()
        if existing_user:
          flash('Username or email already exists. Please choose different ones.', 'danger')
          return render_template('signup.html', form=form)

        new_user = User(username=form.username.data, email=form.email.data)
        new_user.set_password(form.password.data)
        db.session.add(new_user)
        try:
          db.session.commit()
          flash('Account created successfully! You can now sign in.', 'success')
          return redirect(url_for('signin'))
        except Exception as e:
          db.session.rollback()
          flash(f'An error occurred: {e}', 'danger')

      return render_template('signup.html', form=form)


    @app.route('/signin', methods=['GET', 'POST'])
    def signin():
      """Handles user sign-in."""
      if current_user.is_authenticated:
        return redirect(url_for('home'))
      form = SignInForm()
```

```
    if form.validate_on_submit():
        user = User.query.filter_by(username=form.username.data).first()
        if user and user.check_password(form.password.data):
            login_user(user)
            flash('Logged in successfully!', 'success')
            return redirect(url_for('home'))
        else:
            flash('Invalid username or password.', 'danger')
    return render_template('signin.html', form=form)


@app.route('/logout')
@login_required
def logout():
    """Logs the user out."""
    logout_user()
    flash('You have been logged out.', 'info')
    return redirect(url_for('home'))


@app.route('/download_result/<int:query_id>')
@login_required
def download_result(query_id):
    search_query_item = db.session.get(SearchQuery, query_id)

    if not search_query_item:
        flash('Search query not found.', 'danger')
        return redirect(url_for('query'))

    if search_query_item.user_id != current_user.id:
        flash('You are not authorized to download this file.', 'danger')
        return redirect(url_for('query'))

    if search_query_item.status != 'completed':
        flash('The analysis for this query is not yet completed or has failed.', 'warning')
        return redirect(url_for('query'))

    search_result_item = SearchResult.query.filter_by(query_id=query_id).order_by(SearchResult.id.desc()).first()

    if not search_result_item or not search_result_item.website_url:
        flash('Result file path not found for this query.', 'danger')
        return redirect(url_for('query'))

    file_path = search_result_item.website_url

    if not os.path.exists(file_path):
        flash(f'Error: File not found on server at path: {file_path}. Please contact support.', 'danger')
        app.logger.error(f"File not found for download. Query ID: {query_id}, Path: {file_path}")
        return redirect(url_for('query'))
```

```
        try:
            return send_file(
                file_path,
                as_attachment=True,
                download_name=os.path.basename(file_path)
            )
        except Exception as e:
            app.logger.error(f"Error sending file for query_id {query_id}: {e}", exc_info=True)
            flash('An error occurred while trying to download the file.', 'danger')
            return redirect(url_for('query'))


    @app.route('/query', methods=['GET', 'POST'])
    @login_required
    def query():
        form = QueryForm()
        if form.validate_on_submit():
            new_query = SearchQuery(
                user_id=current_user.id,
                keywords=form.keywords.data,
                country=form.country.data,
                whitelist_words=form.whitelist_words.data,
                status='submitted'
            )
            db.session.add(new_query)
            try:
                db.session.commit()
                query_id_for_algorithm = new_query.id
                executor.submit(run_algorithm_task, query_id_for_algorithm)
                    flash(f'Query saved (ID: {query_id_for_algorithm}) and processing started!',
'success')
            except Exception as e:
                db.session.rollback()
                flash(f'An error occurred while saving query: {e}', 'danger')
                app.logger.error(f"Error during query submission: {e}", exc_info=True)
            return redirect(url_for('query'))

                                                                queries_list =
SearchQuery.query.filter_by(user_id=current_user.id).order_by(SearchQuery.created_at.desc()).a
ll()
        return render_template('query.html', form=form, queries=queries_list)


    @app.route('/')
    def home():
        """Renders the home page."""
        return render_template('index.html')


    if __name__ == '__main__':
        with app.app_context():
            db.create_all()
        app.run(debug=True)
```

Файл index.html

Реалізація функціональної задачі формування запитів на пошук компаній за ключовими словами

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Sieve.ai - Find Your Next Business Partners</title>
  <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
</head>
<body>
  <nav class="top-nav">
  {% if current_user.is_authenticated %}
     <a href="{{ url_for('query') }}">Query</a>
     <a href="{{ url_for('logout') }}">Log Out</a>
     <a href="#">{{ current_user.username }}</a>
  {% else %}
     <a href="{{ url_for('signup') }}">Sign Up</a>
     <a href="{{ url_for('signin') }}">Sign In</a>
  {% endif %}
</nav>
  <div class="center-container">
     <h1>Sieve.ai</h1>
     <p>Leverage AI to find your next business partners</p>
  </div>
</body>
</html>
```

Файл query.html

Реалізація функціональної задачі формування запитів на пошук компаній за ключовими словами

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Submit Query - Sieve.ai</title>
  <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
</head>
<body>
  <nav class="top-nav">
     <a href="{{ url_for('home') }}" class="home-link-top-left">Home</a>
     {% if current_user.is_authenticated %}
        <a href="{{ url_for('query') }}">Query</a>
        <a href="{{ url_for('logout') }}">Log Out</a>
```

```
            <a href="#">{{ current_user.username }}</a>
          {% else %}
            <a href="{{ url_for('signup') }}">Sign Up</a>
            <a href="{{ url_for('signin') }}">Sign In</a>
          {% endif %}
        </nav>

        <div class="page-container">
          <h2>Create a New Query</h2>
          {% with messages = get_flashed_messages(with_categories=true) %}
            {% if messages %}
              <ul class="flashes">
              {% for category, message in messages %}
                <li class="{{ category }}">{{ message }}</li>
              {% endfor %}
              </ul>
            {% endif %}
          {% endwith %}

          <form method="POST" action="{{ url_for('query') }}" class="query-form">
            {{ form.csrf_token }}
            <div class="form-group">
              {{ form.keywords.label(class="form-label") }}
              {{ form.keywords(class="form-control", placeholder="Enter keywords...") }}
              {% for error in form.keywords.errors %}
                <span class="error">{{ error }}</span>
              {% endfor %}
            </div>
            <div class="form-group">
              {{ form.country.label(class="form-label") }}
              {{ form.country(class="form-control") }}
              {% for error in form.country.errors %}
                <span class="error">{{ error }}</span>
              {% endfor %}
            </div>
            <div class="form-group">
              {{ form.whitelist_words.label(class="form-label") }}
                    {{ form.whitelist_words(class="form-control", placeholder="e.g., word1,
phrase two, word3") }}
              {% for error in form.whitelist_words.errors %}
                <span class="error">{{ error }}</span>
              {% endfor %}
            </div>
            {{ form.submit(class="btn-submit") }}
          </form>

          <hr>
          <h2>Your Submitted Queries</h2>
          <div class="queries-list">
            {% if queries %}
              <ul>
              {% for query_item in queries %}
                <li>
```

```
                        <strong>Keywords:</strong> {{ query_item.keywords }}<br>
                        <strong>Country:</strong> {{ query_item.country }}<br>
                        {% if query_item.whitelist_words %}
                            <strong>Whitelist:</strong> <small>{{ query_item.whitelist_words |
truncate(50) }}</small><br>
                        {% endif %}
                         <strong>Status:</strong> <span class="status-{{ query_item.status | lower
}}">{{ query_item.status }}</span><br>
                                                                <strong>Submitted:</strong>  {{
query_item.created_at.strftime('%Y-%m-%d %H:%M') }}

                        {% if query_item.status == 'completed' %}
                            <a href="{{ url_for('download_result', query_id=query_item.id) }}"
class="btn-download">Download Results</a>
                        {% endif %}
                    </li>
                {% endfor %}
                </ul>
            {% else %}
                <p>You haven't submitted any queries yet.</p>
            {% endif %}
        </div>
    </div>
</body>
</html>
```

Файл signin.html

Реалізація функціональної задачі формування запитів на пошук компаній за ключовими словами

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Sign In - Sieve.ai</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
</head>
<body>
    <nav class="top-nav">
        <a href="{{ url_for('home') }}" class="home-link-top-left">Home</a>
        {% if current_user.is_authenticated %}
            <a href="#">{{ current_user.username }}</a>
            <a href="{{ url_for('logout') }}">Log Out</a>
        {% else %}
            <a href="{{ url_for('signup') }}">Sign Up</a>
            <a href="{{ url_for('signin') }}">Sign In</a>
        {% endif %}
    </nav>
    <div class="center-container form-container">
        <h2>Sign In to Sieve.ai</h2>
        {% with messages = get_flashed_messages(with_categories=true) %}
```

```
            {% if messages %}
              <ul class="flashes">
              {% for category, message in messages %}
                <li class="{{ category }}">{{ message }}</li>
              {% endfor %}
              </ul>
            {% endif %}
          {% endwith %}
          <form method="POST" action="{{ url_for('signin') }}">
            {{ form.csrf_token }}
            <div class="form-group">
              {{ form.username.label }}
              {{ form.username(class="form-control") }}
              {% for error in form.username.errors %}
                <span class="error">{{ error }}</span>
              {% endfor %}
            </div>
            <div class="form-group">
              {{ form.password.label }}
              {{ form.password(class="form-control") }}
              {% for error in form.password.errors %}
                <span class="error">{{ error }}</span>
              {% endfor %}
            </div>
            {{ form.submit(class="btn-submit") }}
          </form>
        </div>
      </body>
      </html>
```

Файл signup.html

Реалізація функціональної задачі формування запитів на пошук компаній за ключовими словами

```
      <!DOCTYPE html>
      <html lang="en">
      <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>Sign Up - Sieve.ai</title>
        <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
      </head>
      <body>
        <nav class="top-nav">
          <a href="{{ url_for('home') }}" class="home-link-top-left">Home</a>
          <a href="/signup">Sign Up</a>
          <a href="/signin">Sign In</a>
        </nav>
        <div class="center-container form-container">
          <h2>Create Your Sieve.ai Account</h2>
          {% with messages = get_flashed_messages(with_categories=true) %}
```

```
            {% if messages %}
              <ul class="flashes">
              {% for category, message in messages %}
                <li class="{{ category }}">{{ message }}</li>
              {% endfor %}
              </ul>
            {% endif %}
          {% endwith %}
          <form method="POST" action="{{ url_for('signup') }}">
            {{ form.csrf_token }}
            <div class="form-group">
              {{ form.username.label }}
              {{ form.username(class="form-control") }}
              {% for error in form.username.errors %}
                <span class="error">{{ error }}</span>
              {% endfor %}
            </div>
            <div class="form-group">
              {{ form.email.label }}
              {{ form.email(class="form-control") }}
              {% for error in form.email.errors %}
                <span class="error">{{ error }}</span>
              {% endfor %}
            </div>
            <div class="form-group">
              {{ form.password.label }}
              {{ form.password(class="form-control") }}
              {% for error in form.password.errors %}
                <span class="error">{{ error }}</span>
              {% endfor %}
            </div>
            <div class="form-group">
              {{ form.confirm_password.label }}
              {{ form.confirm_password(class="form-control") }}
              {% for error in form.confirm_password.errors %}
                <span class="error">{{ error }}</span>
              {% endfor %}
            </div>
            {{ form.submit(class="btn-submit") }}
          </form>
        </div>
      </body>
      </html>
```

Файл style.css

Реалізація функціональної задачі формування запитів на пошук компаній за ключовими словами

```
      html, body {
        height: 100%;
        margin: 0;
        font-family: sans-serif;
```

```
  background-color: #f4f4f4;
}

.top-nav {
  position: absolute;
  top: 20px;
  right: 20px;
}

.top-nav a {
  text-decoration: none;
  color: #333;
  margin-left: 15px;
  font-size: 1.1em;
}

.top-nav a:first-child:not([href*="signup"]):not([href*="signin"]) {
  font-weight: bold;
  color: #007bff;
}

.top-nav a:hover {
  text-decoration: underline;
}

.center-container {
  display: flex;
  flex-direction: column;
  justify-content: center;
  align-items: center;
  height: 100%;
  text-align: center;
}

h1 {
  font-size: 3em;
  margin-bottom: 0.5em;
  color: #333;
}

p {
  font-size: 1.2em;
  color: #555;
}

.form-container {
  background-color: #fff;
  padding: 30px;
  border-radius: 8px;
  box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);
  width: 300px;
  text-align: left;
}
```

```
.form-container h2 {
  text-align: center;
  margin-bottom: 20px;
  color: #333;
}

.form-group {
  margin-bottom: 15px;
}

.form-group label {
  display: block;
  margin-bottom: 5px;
  color: #555;
}

.form-control {
  width: 100%;
  padding: 10px;
  border: 1px solid #ccc;
  border-radius: 4px;
  box-sizing: border-box;
}

.btn-submit {
  width: 100%;
  padding: 10px;
  background-color: #5cb85c;
  color: white;
  border: none;
  border-radius: 4px;
  cursor: pointer;
  font-size: 1.1em;
}

.btn-submit:hover {
  background-color: #4cae4c;
}

.error {
  color: red;
  font-size: 0.9em;
}

.flashes {
  list-style: none;
  padding: 0;
  margin-bottom: 15px;
}

.flashes li {
  padding: 10px;
```

```
  margin-bottom: 10px;
  border-radius: 4px;
}

.flashes .success {
  background-color: #d4edda;
  color: #155724;
  border: 1px solid #c3e6cb;
}

.flashes .danger {
  background-color: #f8d7da;
  color: #721c24;
  border: 1px solid #f5c6cb;
}

.page-container {
  max-width: 800px;
  margin: 80px auto 20px auto;
  background-color: #fff;
  padding: 30px;
  border-radius: 8px;
  box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);
}

.page-container h2 {
  text-align: center;
  margin-bottom: 25px;
  color: #333;
}

.query-form {
  margin-bottom: 40px;
}

.queries-list hr {
  margin: 30px 0;
  border: 0;
  border-top: 1px solid #eee;
}

.queries-list ul {
  list-style: none;
  padding: 0;
}

.queries-list li {
  background-color: #f9f9f9;
  border: 1px solid #eee;
  padding: 15px;
  margin-bottom: 10px;
  border-radius: 4px;
}
```

```
.queries-list li strong {
  color: #333;
}
```

Файл webcrawler.py

Реалізація функціональної задачі створення механізму збору вебсторінок компаній з пошукових систем

```
import os

from .utils import append_to_json_file, annotate
from typing import Any
from selenium import webdriver
from selenium.common import TimeoutException, ElementClickInterceptedException
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.chrome.options import Options
from selenium.webdriver.support.ui import WebDriverWait
from webdriver_manager.chrome import ChromeDriverManager
from selenium.webdriver.support import expected_conditions as EC

import yaml
import logging
import time
import urllib.parse
import base64

APP_DIR = os.path.dirname(os.path.abspath(__file__))
CONFIG_FILE_PATH = os.path.join(APP_DIR, 'config.yaml')

with open(CONFIG_FILE_PATH, 'r') as file:
  config: dict = yaml.safe_load(file)

DEFAULT_LANGUAGE: str = config['search']['default_language']
SEARCH_PAGES: int = config['search']['search_pages']
WEBSITE: str = config['search']['website']
HEADLESS_MODE: bool = config['search']['headless_mode']


class SieveWebCrawler:

  def __init__(self, run_dir, res_dir, driver_file=None):
    self.driver = None
    self.driver_file = driver_file
    self.get_driver()
    self.run_dir = run_dir
    self.res_dir = res_dir
```

```
        @annotate
        def install_driver(self) -> None:
            ChromeDriverManager().install()

os.chmod("/root/.wdm/drivers/chromedriver/linux64/127.0.6533.72/chromedriver-linux64/chrom
edriver", 0o755)

        @annotate
        def _get_driver_path(self) -> str | None:
            """
            Installs (if necessary) and returns the path to the ChromeDriver.
            Optionally attempts to set execute permissions on non-Windows OS.
            """
            try:
                driver_path = ChromeDriverManager().install()
                if os.name != 'nt':
                    try:
                        os.chmod(driver_path, 0o755)
                                logging.info(f"Ensured execute permissions for ChromeDriver at
{driver_path}")
                    except OSError as e:
                        logging.warning(f"Could not set execute permissions for {driver_path}: {e}.
"
                                        "This might be okay if permissions are already correct.")
                return driver_path
            except Exception as e:
                logging.error(f"ChromeDriverManager().install() failed: {e}")
                return None

        @annotate
        def get_options(self) -> Options:
            options = None
            try:
                options = Options()
                options.add_argument('--disable-gpu')
                options.add_argument('--window-size=1920,1200')
                options.add_argument('--ignore-certificate-errors')
                options.add_argument('--disable-extensions')
                options.add_argument('--no-sandbox')
                options.add_argument('--disable-dev-shm-usage')
                if HEADLESS_MODE:
                    options.add_argument('--headless')
            except Exception as e:
                logging.warning(f"Exception occurred in get_options: {e}")
                print(f"Exception occurred in get_options: {e}")
            finally:
                return options

        @annotate
        def get_service(self) -> Service:
            service = None
            try:
```

```
                                                                    service =
Service(executable_path="/root/.wdm/drivers/chromedriver/linux64/127.0.6533.72/chromedriver
-linux64/chromedriver")
            except Exception as e:
                logging.warning(f"Exception occurred in get_service: {e}")
                print(f"Exception occurred in get_service: {e}")
            finally:
                return service

        @annotate
        def get_driver(self):
            """Initializes and configures the Chrome WebDriver."""
            driver_path = None
            try:
                options = self.get_options()
                if not options:
                    raise RuntimeError("Failed to retrieve Chrome options. WebDriver cannot be
initialized.")

                driver_path = self._get_driver_path()
                if not driver_path:
                    raise RuntimeError("Failed to get ChromeDriver path. WebDriver cannot be
initialized.")

                service = Service(executable_path=driver_path)

                self.driver = webdriver.Chrome(options=options, service=service)

                print('Init - Redirecting to main website...')
                self.redirect_to_website(WEBSITE)
                print('Init - Redirecting to main website complete')
                print('Init - Switching off redirection...')
                self.switch_off_redirection()
                print('Init - Switching off redirection complete')
            except Exception as e:
                logging.warning(f'{e}')
                print(f'{e}')

        def _decode_bing_url(self, bing_url: str) -> str:
            """Decodes a Bing tracking URL to get the actual destination URL."""
            try:
                if bing_url and "bing.com/ck/a" in bing_url:
                    parsed_url = urllib.parse.urlparse(bing_url)
                    query_params = urllib.parse.parse_qs(parsed_url.query)

                    u_param_list = query_params.get('u')
                    if not u_param_list:
                        return bing_url

                    u_param = u_param_list[0]

                    if u_param and u_param.startswith('a1'):
                        encoded_url_part = u_param[2:]
```

```
                padding_needed = len(encoded_url_part) % 4
                if padding_needed:
                    encoded_url_part += '=' * (4 - padding_needed)

                decoded_bytes = base64.b64decode(encoded_url_part)
                return decoded_bytes.decode('utf-8')
            return bing_url
        except Exception as e:
            logging.warning(f"Failed to decode Bing URL '{bing_url}': {e}")
            return bing_url

    def switch_off_redirection(self):
        self.open_location_settings()
        checkbox = WebDriverWait(self.driver, 10).until(
            EC.presence_of_element_located((By.XPATH, "//input[@id='newwnd']"))
        )
        if checkbox.is_selected():
            print('Redirect checkbox is checked! Unchecking...')
            checkbox.click()

    def redirect_to_website(self, website: str = "https://www.bing.com", wait_time: int = 1)
-> Any:
        self.driver.get(website)
        time.sleep(wait_time)

    def click_element(self, by, value, retry_attempt: int = 0, wait_time: int = 2) -> Any:
        try:
            element = WebDriverWait(self.driver, 10).until(
                EC.element_to_be_clickable((by, value))
            )
            self.driver.execute_script("arguments[0].scrollIntoView(true);", element)
            WebDriverWait(self.driver, 10).until(
                EC.visibility_of_element_located((by, value))
            )
            element.click()
            time.sleep(wait_time)
        except ElementClickInterceptedException as e:
            if retry_attempt < 3:
                logging.warning(f"Element click intercepted, retrying... Attempt: {retry_attempt
+ 1}")
                time.sleep(wait_time)
                self.take_screenshot(f"click_element_error_attempt_{retry_attempt}")
                try:
                    self.decline_cookies()
                except e:
                    logging.info(f'Reject cookies button is not found')
                self.click_element(by, value, retry_attempt + 1)
            else:
                logging.error(f"Max retry attempts reached for clicking element: {e}")
                self.take_screenshot("click_element_error")
                raise
        except Exception as e:
```

```
            logging.error(f"Error clicking element: {e}")
            self.take_screenshot("click_element_error")
            raise

        def generate_new_url(self, url: str, current_position: int, increment: int) -> str:
            if not url:
                url = '&'.join(
                    [f'first={current_position + increment}' if element.__contains__('first=') else
element for element in
                     url.split('&')])
            else:
                next_button = WebDriverWait(self.driver, 10).until(
                    EC.element_to_be_clickable((By.XPATH, f"//a[contains(@title, 'Next page')]"))
                )
                url = '&'.join(
                    [f'first={current_position + increment}' if element.__contains__('first=') else
element for element in
                     next_button.get_attribute('href').split('&')])
            return url

        def switch_language(self, language: str) -> Any:
            try:
                print(language)
                print('Redirecting to the website')
                self.redirect_to_website()
                print('Clicking ID')
                self.click_element(By.ID, "id_sc")
                print('Clicking selector')
                self.click_element(By.CSS_SELECTOR, "div#hbsettings")
                print('Clicking XPATH 1')
                                                self.click_element(By.XPATH,     "//a[contains(@href,
'region_section#region-section')]")
                print('Clicking XPATH 2')
                self.click_element(By.XPATH, f"//li[.//div[text()='{language}']]//a")
            except Exception as e:
                logging.error(f'Error occurred: {e}')
                print(e)

        def enter_search_query(self, search_query: str, wait_time: int = 1) -> Any:
            search_box = WebDriverWait(self.driver, 10).until(
                EC.presence_of_element_located((By.NAME, "q"))
            )
            search_box.click()

            time.sleep(wait_time)

            search_box.send_keys(search_query)
            search_box.send_keys(Keys.RETURN)

            time.sleep(wait_time)

        def open_location_settings(self) -> Any:
            self.click_element(By.ID, "id_sc")
```

```
            self.click_element(By.CSS_SELECTOR, "div#hbsettings")
                                                self.click_element(By.XPATH,       "//a[contains(@href,
'region_section#region-section')]")

        def switch_search_city(self, city: str, wait_time: int = 2) -> Any:
            self.redirect_to_website()
            self.open_location_settings()
            city_search_box = WebDriverWait(self.driver, 10).until(
                        EC.presence_of_element_located((By.XPATH,  "//input[@type='text'  and
(@placeholder='Enter city, address or postcode' or @placeholder='Enter city, address or zip
code')]"))
            )
            city_search_box.clear()
            city_search_box.click()
            time.sleep(wait_time)
            city_search_box.send_keys(city)
            time.sleep(wait_time)
            city_search_box.send_keys(Keys.RETURN)
            time.sleep(wait_time)
            self.click_element(By.XPATH, "//input[@type='submit' and @id='sv_btn']")

        def switch_search_result_language(self, languages: list[str]) -> Any:
            self.redirect_to_website()
            self.open_location_settings()
            self.click_element(By.XPATH, "//input[@id='preferuilang']")
            self.click_element(By.XPATH, "//input[@type='submit' and @id='sv_btn']")
            self.open_location_settings()
            is_language_changed = False
            for index, language in enumerate(languages):
                if language != 'English':
                    if index == 0:
                        self.click_element(By.XPATH, "//input[@id='langlimit']")
                                                                self.click_element(By.XPATH,
"//div[@id='search-languages-seemore-btn']//a[contains(text(), 'See more languages')]")
                    label_xpath = f"//label[contains(text(), '{language}')]"
                    label_element = WebDriverWait(self.driver, 10).until(
                        EC.presence_of_element_located((By.XPATH, label_xpath))
                    )
                                        self.click_element(By.XPATH,  f"//input[@type='checkbox'  and
@id='{label_element.get_attribute('for')}']")
                    is_language_changed = True
            if is_language_changed:
                self.click_element(By.XPATH, "//input[@type='submit' and @id='sv_btn']")

        def switch_search_country(self, country: str, wait_time: int = 1) -> Any:
            self.redirect_to_website()
            self.open_location_settings()
            time.sleep(wait_time)
            self.click_element(By.XPATH, "//a[contains(text(), 'See more countries/regions')]")
            time.sleep(wait_time)
            self.click_element(By.XPATH, f"//a[text()='{country}']")

        def switch_search_location(self, location_object: dict) -> Any:
```

```
            try:
                city, languages, country = location_object['city'], location_object['language'],
location_object['country']
                print(f'Switching search country to {country}')
                self.switch_search_country(country)
                print(f'Switching search languages to {languages}')
                self.switch_search_result_language(languages)
                print(f'Switching search city to {city}')
                self.switch_search_city(city)
            except Exception as e:
                raise Exception(f"Error during switching location: {e}")

        def decline_cookies(self, cookie_reject_button_class: str = "bnp_btn_reject") -> Any:
            try:
                reject_button = WebDriverWait(self.driver, 5).until(
                                                EC.presence_of_element_located((By.CLASS_NAME,
cookie_reject_button_class))
                )
                reject_button.click()
            except TimeoutException:
                logging.info("Reject button doesn't exist!")

        def take_screenshot(self, name: str) -> Any:
            self.driver.save_screenshot(f"{self.run_dir}/{name}.png")

        def scroll_to_bottom(self, wait_time: int = 2) -> Any:
            self.driver.execute_script("window.scrollTo(0, document.body.scrollHeight);")
            time.sleep(wait_time)

        def search(self, location: dict, search_query: str) -> list[dict]:
            self.switch_search_location(location)
            self.redirect_to_website(WEBSITE)
            time.sleep(1)
            self.enter_search_query(search_query)
            time.sleep(1)
            self.decline_cookies()
            time.sleep(1)
            link_objects: list[dict] = []
            urls_processed: list = []
            current_search_page = 1
            while current_search_page <= SEARCH_PAGES:
                time.sleep(1)
                results = self.driver.find_elements(By.CSS_SELECTOR, "li.b_algo h2 a")
                logging.info([result.get_attribute("href") for result in results])
                print(f"Current search page: {current_search_page}")
                for result in results:
                    link = self._decode_bing_url(result.get_attribute("href"))
                    if link:
                        domain_part = link.split("/")[2]
                        if not urls_processed.__contains__(domain_part):
                            link_objects.append({'url': link, 'num_occurrences': 1})
                            urls_processed.append(domain_part)
                        else:
```

```
                    for link_object in link_objects:
                        if link_object['url'].__contains__(domain_part):
                            link_object['num_occurrences'] += 1
                            break
                print([link_object['url'] for link_object in link_objects])

                try:
                    self.scroll_to_bottom()
                    self.take_screenshot("next_button_click_before")
                            self.click_element(By.XPATH, f"//a[contains(@title, 'Next page')]",
wait_time=1)
                    current_search_page += 1
                    self.take_screenshot("next_button_click_after")
                except TimeoutException as e:
                    self.take_screenshot("next_button_error")
                    logging.error(f'Error occurred: {e}')
                    break
            append_to_json_file([{
                "loc": location['country'],
                "url": link_object['url'],
                "num_occurrences": link_object['num_occurrences']
            } for link_object in link_objects], f"{self.res_dir}/crawled_links.json")
            return link_objects

        def quit(self) -> Any:
            self.driver.quit()
```

Файл utils.py

Реалізація функціональної задачі створення модулів класифікації та фільтрації сайтів відповідно до якості та тематики

```
import re
from typing import Any

import os
import time
import json
import logging
import pandas as pd


def annotate(func) -> Any:
    def wrapper(*args, **kwargs):
        start_time = time.time()
        try:
            result = func(*args, **kwargs)
        except Exception as e:
            logging.error(f"Error in {func.__name__}: {e}")
            result = None
        end_time = time.time()
        print(f'Total time taken for {func.__name__} - {end_time-start_time} seconds')
```

```
            logging.info(f'Total time taken for {func.__name__} - {end_time-start_time} seconds')
        return result
    return wrapper


@annotate
def initialize_run(run_dir: str, webcrawler_dir: str, analyser_dir: str, log_file: str) -> Any:
    os.makedirs(run_dir, exist_ok=True)
    os.makedirs(webcrawler_dir, exist_ok=True)
    os.makedirs(analyser_dir, exist_ok=True)
    logging.basicConfig(filename=f'{run_dir}/{log_file}', level=logging.INFO,
                        format='%(asctime)s:%(levelname)s:%(message)s')


@annotate
def append_to_json_file(data: list[dict], filepath: str) -> Any:
    try:
        with open(filepath, 'a') as file:
            for row in data:
                json.dump(row, file)
                file.write('\n')
    except Exception as e:
        print(f"Error writing to the JSON file: {e}")


@annotate
def read_from_ndjson_file(filepath: str) -> list[dict]:
    data = []
    try:
        with open(filepath, 'r') as f:
            for line in f:
                data.append(json.loads(line))
    except Exception as e:
        print(f"Error reading the JSON file: {e}")
    finally:
        return data


@annotate
def read_from_json_file(filepath: str) -> dict:
    data = ''
    try:
        with open(filepath, 'r', encoding='utf-8') as file:
            data = json.load(file)
    except Exception as e:
        print(f"Error reading the JSON file: {e}")
    finally:
        return data


def clean_text(text):
    return re.sub(r'[^\x20-\x7E]', '', text)
```

```
def clean_dataframe(df):
  for col in df.select_dtypes(include=['object']).columns:
     df[col] = df[col].apply(lambda x: clean_text(x) if isinstance(x, str) else x)
  return df


@annotate
def save_json_to_excel(json_filepath: str, excel_filepath: str, json_format: str) -> Any:
  if json_format == 'json':
     data = read_from_json_file(json_filepath)
     df = pd.DataFrame(data)
     df = clean_dataframe(df)
     df.to_excel(excel_filepath, index=False)
  elif json_format == 'ndjson':
     data = read_from_ndjson_file(json_filepath)
     df = pd.DataFrame(data)
     df = clean_dataframe(df)
     df.to_excel(excel_filepath, index=False)
  else:
        logging.warning('Error during saving json file to excel: JSON format is not supported')
```

Файл pipeline.py

Реалізація функціональної задачі створення модулів класифікації та фільтрації сайтів відповідно до якості та тематики

```
from .filters import BasicFilter
from .utils import annotate, append_to_json_file


class Pipeline:
     def __init__(self, filters: list[list[int, list[int], BasicFilter]], base_link_objects: list[dict]):
        self.filters = filters
        self.base_link_objects = base_link_objects

  @annotate
  def execute_filter(self, pipeline_filter: BasicFilter, params: list[dict]) -> list[dict]:
     return pipeline_filter.run(*params)

  def run(self) -> list[dict]:
     link_objects = self.base_link_objects
     for filter_index, filter_receive_indexes, pipeline_filter in self.filters:
                params = [self.filters[index-1][2].result_link_objects if index else self.base_link_objects for index in filter_receive_indexes]
        link_objects = self.execute_filter(pipeline_filter, params)
     return link_objects
```

Файл main.py

Реалізація функціональної задачі створення механізму збору вебсторінок компаній з пошукових систем

```
from .utils import annotate, initialize_run, save_json_to_excel, read_from_json_file
from .webcrawler import SieveWebCrawler
from .analyser import SieveAnalyser
from datetime import datetime
from typing import Any, TYPE_CHECKING

import yaml
import os
import time

if TYPE_CHECKING:
  from flask_app.app import db
  from flask_sqlalchemy import SQLAlchemy
    from flask_app.app import SearchResult as SearchResultModel, SearchQuery as SearchQueryModel

APP_DIR = os.path.dirname(os.path.abspath(__file__))
CONFIG_FILE_PATH = os.path.join(APP_DIR, 'config.yaml')

with open(CONFIG_FILE_PATH, 'r') as file:
  config: dict = yaml.safe_load(file)

TIMESTAMP = datetime.now().strftime(config['timestamp_format'])
RUN_DIR: str = config['directories']['run_dir'].replace("{timestamp}", TIMESTAMP)
CRAWLER_DIR = f"{RUN_DIR}/{config['directories']['webcrawler_dir_name']}"
ANALYSER_DIR = f"{RUN_DIR}/{config['directories']['analyser_dir_name']}"

LOG_FILE: str = config['files']['log_file']
DRIVER_FILE: str = f"{config['directories']['driver_dir']}/{config['files']['driver_file']}"
CRAWLER_RESULTS_FILE_JSON = f"{CRAWLER_DIR}/crawled_links.json"
ANALYSER_RESULTS_FILE_JSON = f"{ANALYSER_DIR}/analysed_links.json"
CRAWLER_RESULTS_FILE_XLSX = f"{CRAWLER_DIR}/crawled_links.xlsx"
ANALYSER_RESULTS_FILE_XLSX = f"{ANALYSER_DIR}/analysed_links.xlsx"

DEFAULT_LANGUAGE: str = config['search']['default_language']

COUNTRY_MAPPING = {
  'GB': {
    'country': 'United Kingdom',
    "city": 'London',
    "language": ["English"],
  },
  'PT': {
    'country': 'Portugal',
    'city': 'Lisbon',
    'language': ['Portuguese (Brazil)'],
  },
  'ES': {
```

```
            'country': 'Spain',
            'city': 'Madrid',
            'language': ['Spanish']
        }
    }


    @annotate
    def algorithm_crawl(location_query_mappings: dict) -> Any:
        sieve_webcrawler = SieveWebCrawler(RUN_DIR, CRAWLER_DIR, DRIVER_FILE)
        print(sieve_webcrawler.driver)
        print('Switching language to default...')
        sieve_webcrawler.switch_language(language=DEFAULT_LANGUAGE)
        print('Switching language to default done.')
        for country, location_object in location_query_mappings.items():
            for query in location_object['query']:
                location = {
                    'country': country,
                    'language': location_object['language'],
                    'city': location_object['city']
                }
                retry = 0
                serp_links = sieve_webcrawler.search(location=location, search_query=query)
                while not serp_links and retry != 3:
                    serp_links = sieve_webcrawler.search(location=location, search_query=query)
                    retry += 1
                if retry == 3:
                    raise Exception(f'Error searching for SERPs in {location["country"]}')
        sieve_webcrawler.quit()


    @annotate
    def algorithm_analyse(query_whitelist: list[str]) -> Any:
                    sieve_analyser = SieveAnalyser(CRAWLER_RESULTS_FILE_JSON,
ANALYSER_RESULTS_FILE_JSON, query_whitelist)
        sieve_analyser.run()


    def execute_algorithm(query_id: int, db_instance: 'SQLAlchemy',
                    search_result_model: type['SearchResultModel'],
                    search_query_model: type['SearchQueryModel']
                    ) -> Any:
        query_country, query_keywords, query_whitelist = None, None, None

        try:
            retrieved_search_query = db_instance.session.get(search_query_model, query_id)
            if retrieved_search_query:
                    print(f"[{os.getpid()}] Successfully fetched SearchQuery object with ID
{query_id}.")
                print(f"[{os.getpid()}] Keywords: {retrieved_search_query.keywords}")
                print(f"[{os.getpid()}] Country: {retrieved_search_query.country}")
                query_country = retrieved_search_query.country
                query_keywords = retrieved_search_query.keywords
```

```
                query_whitelist = [element.strip() for element in retrieved_search_query.whitelist_words.split(',')]
            else:
                print(f"[{os.getpid()}] No SearchQuery object found with ID {query_id}.")
        except Exception as e:
            print(f"[{os.getpid()}-{time.strftime('%H:%M:%S')}] ERROR fetching SearchQuery object with ID {query_id}: {e}")

        location_query_mappings = {
            COUNTRY_MAPPING[query_country]['country']: {
                'city': COUNTRY_MAPPING[query_country]['city'],
                'language': COUNTRY_MAPPING[query_country]['language'],
                'query': [query_keywords]
            }
        }

        print(location_query_mappings)
        print(query_whitelist)

        json_format = 'ndjson'
        initialize_run(RUN_DIR, CRAWLER_DIR, ANALYSER_DIR, LOG_FILE)
        algorithm_crawl(location_query_mappings)
        algorithm_analyse(query_whitelist)
                save_json_to_excel(CRAWLER_RESULTS_FILE_JSON, CRAWLER_RESULTS_FILE_XLSX, json_format)
            save_json_to_excel(ANALYSER_RESULTS_FILE_JSON, ANALYSER_RESULTS_FILE_XLSX, json_format)

        if os.path.exists(ANALYSER_RESULTS_FILE_XLSX):
            print(f"[{os.getpid()}] Saving result path to DB for query_id: {query_id}. Path: {ANALYSER_RESULTS_FILE_XLSX}")
            try:
                new_result_entry = search_result_model(
                    query_id=query_id,
                    website_url=ANALYSER_RESULTS_FILE_XLSX,
                    description="Algorithm analysis result file",
                    country=None
                )
                db_instance.session.add(new_result_entry)
                db_instance.session.commit()
                print(f"[{os.getpid()}] Successfully saved path to search_result table for query_id: {query_id}")
            except Exception as e:
                db_instance.session.rollback()
                    print(f"[{os.getpid()}-{time.strftime('%H:%M:%S')}] ERROR saving to search_result for query_id {query_id}: {e}")
                raise


    def main_flask_async(query_id: int, db_instance: 'SQLAlchemy', search_result_model: type['SearchResultModel'], search_query_model: type['SearchQueryModel']) -> Any:
        """
        Main function for the algorithm processing.
```

```
        This function will be called asynchronously.
        """
        print(f"[{os.getpid()}-{time.strftime('%H:%M:%S')}] Algorithm processing started for query_id: {query_id}")
        query_to_update = None
        try:
            query_to_update = db_instance.session.get(search_query_model, query_id)
            if not query_to_update:
                print(
                    f"[{os.getpid()}-{time.strftime('%H:%M:%S')}] ERROR: SearchQuery with id {query_id} not found. Cannot update status.")
                return

            query_to_update.status = 'processing'
            db_instance.session.commit()

                                execute_algorithm(query_id, db_instance, search_result_model, search_query_model)

            query_to_update.status = 'completed'
                    print(f"[{os.getpid()}-{time.strftime('%H:%M:%S')}] Algorithm processing COMPLETED for query_id: {query_id}")

        except Exception as e:
                print(f"[{os.getpid()}-{time.strftime('%H:%M:%S')}] ERROR during algorithm processing for query_id {query_id}: {e}")
            if query_to_update:
                query_to_update.status = 'failed'

        finally:
            if db_instance.session.is_active:
                try:
                    db_instance.session.commit()
                except Exception as commit_exc:
                    print(f"[{os.getpid()}-{time.strftime('%H:%M:%S')}] ERROR committing final status for query_id {query_id}: {commit_exc}")
                    db_instance.session.rollback()
            else:
                print(f"[{os.getpid()}-{time.strftime('%H:%M:%S')}] WARNING: DB session not active, cannot commit final status for query_id {query_id}")
```

Файл analyser.py

Реалізація функціональної задачі створення механізму збереження та експорту результатів пошуку

```
from .utils import annotate, append_to_json_file, read_from_ndjson_file
from .pipeline import Pipeline
from .filters import *

import os
import yaml
```

```
APP_DIR = os.path.dirname(os.path.abspath(__file__))
CONFIG_FILE_PATH = os.path.join(APP_DIR, 'config.yaml')

with open(CONFIG_FILE_PATH, 'r') as file:
  config: dict = yaml.safe_load(file)

URL_COLUMN_NAME = config['analyser']['url_column_name']
NUM_OCCURRENCES_COLUMN_NAME = config['analyser']['num_occurrences_column_name']
LOCATION_COLUMN_NAME = config['analyser']['location_column_name']


class SieveAnalyser:
  def __init__(self, links_filepath, analyser_results_filepath, whitelist_words):
    self.links_filepath = links_filepath
    self.whitelist_words = whitelist_words
    self.links_objects = read_from_ndjson_file(links_filepath)
    self.analyser_results_filepath = analyser_results_filepath

  @annotate
  def run(self):
    filters = [
      [1, [0], RegularizeLinksFilter(URL_COLUMN_NAME, 'www.')],
      [2, [1], OccurrencesCountFilter(URL_COLUMN_NAME, NUM_OCCURRENCES_COLUMN_NAME)],
      [3, [1], LocationGroupingFilter(URL_COLUMN_NAME, LOCATION_COLUMN_NAME)],
      [4, [3], BlacklistFilter(URL_COLUMN_NAME)],
      [5, [4], DeduplicationFilter(URL_COLUMN_NAME)],
      [6, [5, 2], MatchOccurrencesCountFilter(URL_COLUMN_NAME, NUM_OCCURRENCES_COLUMN_NAME, LOCATION_COLUMN_NAME)],
      [7, [6], WebsiteDataExtractionFilter(URL_COLUMN_NAME, NUM_OCCURRENCES_COLUMN_NAME, LOCATION_COLUMN_NAME)],
      [8, [7], ExtractContactInformationFilter()],
      [9, [8], TranslationFilter(URL_COLUMN_NAME)],
      [10, [9], CheckMetadataFilter(self.whitelist_words)],
    ]
    pipeline = Pipeline(filters, self.links_objects)
    append_to_json_file(pipeline.run(), self.analyser_results_filepath)
```

Файл __init__.py

Реалізація функціональної задачі автоматичного вилучення релевантної інформації з вебресурсів (назва компанії, вебсайт, опис діяльності тощо)

```
from .basic_filter import BasicFilter
from .blacklist_filter import BlacklistFilter
from .regularize_links_filter import RegularizeLinksFilter
from .occurrences_count_filter import OccurrencesCountFilter
from .match_occurences_count_filter import MatchOccurrencesCountFilter
from .deduplication_filter import DeduplicationFilter
from .location_grouping_filter import LocationGroupingFilter
```

```
from .website_data_extraction_filter import WebsiteDataExtractionFilter
from .request_adapter import RequestAdapter
from .translation_filter import TranslationFilter
from .check_metadata_filter import CheckMetadataFilter
from .extract_contact_information_filter import ExtractContactInformationFilter

__all__ = ["BasicFilter", "BlacklistFilter", "RegularizeLinksFilter", "OccurrencesCountFilter",
           "MatchOccurrencesCountFilter", "DeduplicationFilter", "LocationGroupingFilter",
           "WebsiteDataExtractionFilter", "RequestAdapter", "TranslationFilter", "CheckMetadataFilter", "ExtractContactInformationFilter"]
```

Файл basic_filter.py

Реалізація функціональної задачі автоматичного вилучення релевантної інформації з вебресурсів (назва компанії, вебсайт, опис діяльності тощо)

```
class BasicFilter:
  def __init__(self, result_link_objects: list[dict] = None):
    self.result_link_objects = result_link_objects

  def run(self, link_objects: list[dict]) -> list[dict]:
    self.result_link_objects = link_objects
    return self.result_link_objects
```

Файл blacklist_filter.py

Реалізація функціональної задачі автоматичного вилучення релевантної інформації з вебресурсів (назва компанії, вебсайт, опис діяльності тощо)

```
from copy import deepcopy
from .basic_filter import BasicFilter

import os
import yaml

APP_DIR = os.path.dirname(os.path.abspath(__file__))
CONFIG_FILE_PATH = os.path.join(APP_DIR, '..', 'config.yaml')

with open(CONFIG_FILE_PATH, 'r') as file:
  config: dict = yaml.safe_load(file)

BLACKLIST_WORDS = config['blacklist_filter']['link_words']


class BlacklistFilter(BasicFilter):

  def __init__(self, url_column_name: str):
    super().__init__()
    self.__url_column_name = url_column_name

  def __is_containing_blacklist_words(self, link: str) -> bool:
```

```
        return any(blacklist_word in link for blacklist_word in BLACKLIST_WORDS)

    def run(self, link_objects: list[dict]) -> list[dict]:
        updated_link_objects = deepcopy(link_objects)
            return super().run([link_object for link_object in updated_link_objects if not self.__is_containing_blacklist_words(link_object[self.__url_column_name])])
```

Файл check_metadata_filter.py

Реалізація функціональної задачі автоматичного вилучення релевантної інформації з вебресурсів (назва компанії, вебсайт, опис діяльності тощо)

```
from copy import deepcopy
from .basic_filter import BasicFilter


class CheckMetadataFilter(BasicFilter):

    def __init__(self, metadata_words_whitelist: list[str]) -> None:
        super().__init__()
        self.__metadata_words_whitelist = metadata_words_whitelist

    def __is_text_containing_whitelist_words(self, text: str) -> bool:
                return any(whitelist_word in text for whitelist_word in self.__metadata_words_whitelist)

    def run(self, link_objects: list[dict]) -> list[dict]:
        updated_link_objects = deepcopy(link_objects)
        for link_object in updated_link_objects:
                    link_object["metadata_contains_key_words"] = "True" if self.__is_text_containing_whitelist_words(link_object['title'].lower()) or self.__is_text_containing_whitelist_words(link_object['description'].lower()) else "False"
        return super().run(updated_link_objects)
```

Файл deduplication_filter.py

Реалізація функціональної задачі автоматичного вилучення релевантної інформації з вебресурсів (назва компанії, вебсайт, опис діяльності тощо)

```
from copy import deepcopy
from .basic_filter import BasicFilter


class DeduplicationFilter(BasicFilter):
    def __init__(self, url_column_name: str):
        super().__init__()
        self.__url_column_name = url_column_name

    def run(self, link_objects: list[dict]) -> list[dict]:
        deduplicated_link_objects = []
        checked_domains = []
        updated_link_objects = deepcopy(link_objects)
```

```
        for link_object in updated_link_objects:
            link_domain_name = link_object[self.__url_column_name].split("/")[2]
            if not checked_domains.__contains__(link_domain_name):
                deduplicated_link_objects.append(link_object)
                checked_domains.append(link_domain_name)
        return super().run(deduplicated_link_objects)
```

Файл extract_contact_information_filter.py

Реалізація функціональної задачі автоматичного вилучення релевантної інформації з вебресурсів (назва компанії, вебсайт, опис діяльності тощо)

```
import re
from copy import deepcopy
from .basic_filter import BasicFilter


class ExtractContactInformationFilter(BasicFilter):

    def __init__(self) -> None:
        super().__init__()

    def __find_phone_numbers_by_regex(self, text: str) -> list[str]:
        phone_pattern = r'\+?\d[\d -]{8,12}\d'
        found_phone_numbers = re.findall(phone_pattern, text)
        return found_phone_numbers

    def __find_emails_by_regex(self, text: str) -> list[str]:
        email_pattern = r'[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}'
        found_website_emails = re.findall(email_pattern, text)
        return found_website_emails

    def run(self, link_objects: list[dict]) -> list[dict]:
        updated_link_objects = deepcopy(link_objects)
        for link_object in updated_link_objects:
                                                    link_object['phone_numbers']      =
self.__find_phone_numbers_by_regex(link_object['text'])
            link_object['corporate_emails'] = self.__find_emails_by_regex(link_object['text'])
        return super().run(updated_link_objects)
```

Файл location_grouping_filter.py

Реалізація функціональної задачі автоматичного вилучення релевантної інформації з вебресурсів (назва компанії, вебсайт, опис діяльності тощо)

```
from copy import deepcopy
from .basic_filter import BasicFilter


class LocationGroupingFilter(BasicFilter):
    def __init__(self, url_column_name: str, location_column_name: str):
```

```
        super().__init__()
        self.__url_column_name = url_column_name
        self.__location_column_name = location_column_name

    def run(self, link_objects: list[dict]) -> list[dict]:
        filtered_objects = deepcopy(link_objects)
        for index, link_object in enumerate(filtered_objects):
            link_url = link_object[self.__url_column_name]
            link_domain_name = link_url.split("/")[2]
                                    filtered_objects[index][self.__location_column_name] =
[link_object[self.__location_column_name]]
            for self_link_object in link_objects:
                self_link_url = self_link_object[self.__url_column_name]
                self_link_domain_name = self_link_url.split("/")[2]
                if link_domain_name == self_link_domain_name \
                        and link_url != self_link_url\
                                                                                and not
link_object[self.__location_column_name].__contains__(self_link_object[self.__location_colum
n_name]):

link_object[self.__location_column_name].append(self_link_object[self.__location_column_na
me])
        return super().run(filtered_objects)
```

Файл match_occurences_count_filter.py

Реалізація функціональної задачі автоматичного вилучення релевантної інформації з вебресурсів (назва компанії, вебсайт, опис діяльності тощо)

```
from copy import deepcopy
from .basic_filter import BasicFilter


class MatchOccurrencesCountFilter(BasicFilter):

    def __init__(self, url_column_name: str, num_occurrences_column_name: str,
location_column_name: str):
        super().__init__()
        self.__url_column_name = url_column_name
        self.__num_occurrences_column_name = num_occurrences_column_name
        self.__location_column_name = location_column_name

    def run(self, link_objects: list[dict], domain_occurrences_objects: list[dict]) ->
list[dict]:
        filtered_link_objects = []
        updated_link_objects = deepcopy(link_objects)
        for link_object in updated_link_objects:
            link_domain_name = link_object[self.__url_column_name].split("/")[2]
            for domain_occurrences_object in domain_occurrences_objects:
                if domain_occurrences_object['domain'] == link_domain_name:
                    filtered_link_objects.append(
                        {
```

```
                                                    self.__location_column_name:
link_object[self.__location_column_name],
                        self.__url_column_name: link_object[self.__url_column_name],
                                                    self.__num_occurrences_column_name:
domain_occurrences_object[self.__num_occurrences_column_name]
                    }
                )
                break
        return super().run(filtered_link_objects)
```

Файл occurences_count_filter.py

Реалізація функціональної задачі автоматичного вилучення релевантної інформації з вебресурсів (назва компанії, вебсайт, опис діяльності тощо)

```
from copy import deepcopy
from .basic_filter import BasicFilter


class OccurrencesCountFilter(BasicFilter):

  def __init__(self, url_column_name: str, num_occurrences_column_name: str):
    super().__init__()
    self.__url_column_name = url_column_name
    self.__num_occurrences_column_name = num_occurrences_column_name

  def run(self, link_objects: list[dict]) -> list[dict]:
    link_domain_objects = []
    domains_processed = []
    updated_link_objects = deepcopy(link_objects)
    for link_object in updated_link_objects:
      domain_name = link_object[self.__url_column_name].split("/")[2]
      count = link_object[self.__num_occurrences_column_name]
      if not domains_processed.__contains__(domain_name):
        domains_processed.append(domain_name)
                                    link_domain_objects.append({'domain':   domain_name,
self.__num_occurrences_column_name: count})
      else:
        for link_domain_object in link_domain_objects:
          if link_domain_object['domain'] == domain_name:
            link_domain_object[self.__num_occurrences_column_name] += count
    return super().run(link_domain_objects)
```

Файл regularize_links_filter.py

Реалізація функціональної задачі автоматичного вилучення релевантної інформації з вебресурсів (назва компанії, вебсайт, опис діяльності тощо)

```
from copy import deepcopy
from .basic_filter import BasicFilter
```

```
class RegularizeLinksFilter(BasicFilter):

  def __init__(self, url_column_name: str, replace_string: str):
    super().__init__()
    self.__url_column_name = url_column_name
    self.__replace_string = replace_string

  def run(self, link_objects: list[dict]) -> list[dict]:
    filtered_link_objects = []
    updated_link_objects = deepcopy(link_objects)
    for link_object in updated_link_objects:
        link_object[self.__url_column_name ] = link_object[self.__url_column_name
].replace(self.__replace_string, '')
      filtered_link_objects.append(link_object)
    return super().run(filtered_link_objects)
```

Файл request_adapter.py

Реалізація функціональної задачі автоматичного вилучення релевантної інформації з вебресурсів (назва компанії, вебсайт, опис діяльності тощо)

```
import ssl
import certifi
import asyncio
import aiohttp
import re
import os
import yaml
import logging

from bs4 import BeautifulSoup
from copy import deepcopy

APP_DIR = os.path.dirname(os.path.abspath(__file__))
CONFIG_FILE_PATH = os.path.join(APP_DIR, '..', 'config.yaml')

with open(CONFIG_FILE_PATH, 'r') as file:
  config: dict = yaml.safe_load(file)

HEADERS = config['search']['headers']

ssl._create_default_https_context = ssl._create_unverified_context
ssl._create_default_https_context = ssl.create_default_context(cafile=certifi.where())


class RequestAdapter:

  def __init__(self, links):
    self.links = links
    self.client_errors = 0
    self.timeout_errors = 0
```

```
        self.other_errors = 0

    def __get_website_text(self, soup: BeautifulSoup) -> dict:
        text = ' '.join([tag.text for tag in soup.find_all(['p', 'h1', 'h2', 'h3', 'h4', 'h5', 'h6'])])
        stripped_text = re.sub(r'\s+', ' ', text).strip() if text else ''
        return {
            "text_found_status": True if stripped_text else False,
            "text": stripped_text
        }

    async def __fetch_website_data(self, session: aiohttp.ClientSession, url: str) -> dict:
        title = ""
        description = ""
        text = ""
        try:
            ssl_context = ssl.create_default_context(cafile=certifi.where())
            async with session.get(url, ssl=ssl_context, timeout=300) as response:
                html_response = await response.text()
                soup = BeautifulSoup(html_response, 'html.parser')
                website_text = self.__get_website_text(soup)['text']
                meta_description = soup.find('meta', attrs={'name': 'description'})
                title = soup.title.string if soup.title else "Title cannot be extracted"
                    description = meta_description['content'] if meta_description else "Description
cannot be extracted"
                    text = website_text if website_text else "Text cannot be extracted"
        except aiohttp.ClientError as e:
            logging.info(f"Error fetching metadata for {url}: {e}")
            title = "Metadata cannot be extracted"
            description = "Metadata cannot be extracted"
            text = "Text cannot be extracted"
            self.client_errors += 1
        except (TimeoutError, aiohttp.ClientTimeout) as e:
            logging.info(f"Timeout error fetching metadata for {url}: {e}")
            title = "Metadata cannot be extracted"
            description = "Metadata cannot be extracted"
            text = "Text cannot be extracted"
            self.timeout_errors += 1
        except Exception as e:
            logging.info(f"Unexpected error fetching metadata for {url}: {e}")
            title = "Metadata cannot be extracted"
            description = "Metadata cannot be extracted"
            text = "Text cannot be extracted"
            self.other_errors += 1
        finally:
            return {
                "url": url if url else "URL is missing",
                "title": title if title else "Title is missing",
                "description": description if description else "Description is missing",
                "text": text if text else "Text is missing"
            }

    async def __extract_website_data_async(self) -> tuple:
        async with aiohttp.ClientSession(headers=HEADERS) as session:
```

```
            tasks = [self.__fetch_website_data(session, link) for link in self.links]
            results = await asyncio.gather(*tasks)
        return results

    def run(self) -> list[dict]:
        websites_data = asyncio.run(self.__extract_website_data_async())
        print(f"Request Adapter run finished!")
                print(f"Total errors occurred: {self.client_errors + self.timeout_errors +
self.other_errors}")
        print(f"Total client occurred: {self.client_errors}")
        print(f"Total timeout occurred: {self.timeout_errors}")
        print(f"Total other errors occurred: {self.other_errors}")
        return websites_data
```

Файл translation_filter.py

Реалізація функціональної задачі автоматичного вилучення релевантної інформації з вебресурсів (назва компанії, вебсайт, опис діяльності тощо)

```
import asyncio
from copy import deepcopy
import logging
import string
from .basic_filter import BasicFilter


class TranslationFilter(BasicFilter):

    def __init__(self, url_column_name: str, concurrency_limit: int = 50) -> None:
        super().__init__()
        self.__url_column_name = url_column_name
        from googletrans import Translator
        self._translator = Translator(service_urls=['translate.googleapis.com'])
        self._semaphore = asyncio.Semaphore(concurrency_limit)
                    logging.info(f"TranslationFilter initialized with concurrency limit:
{concurrency_limit}")

    def __remove_punctuation(self, text: str) -> str:
        return "".join([char for char in text if char not in string.punctuation])

    async def __async_translate_text(self, text_object: dict) -> dict:
        async with self._semaphore:
            updated_object = deepcopy(text_object)
            original_text = updated_object.get('text')
            if not original_text or not isinstance(original_text, str) or not original_text.strip():
                updated_object['text'] = 'Original text was empty or invalid'
                return updated_object
            logging.debug(f"Semaphore acquired. Translating: '{original_text[:50]}...'")
            try:
                translated_obj = await self._translator.translate(original_text, dest='en')
                translated_text_content = translated_obj.text
                updated_object[
```

```
                    'text'] = translated_text_content if translated_text_content else 'Translation
resulted in empty text'
                except Exception as e:
                    logging.warning(f"Error translating text (first 50 chars: '{original_text[:50]}...')
occurred: {e}")
                    updated_object['text'] = "Translation failed"
                return updated_object

            async def __async_batch_translate_texts(self, text_objects: list[dict]) -> list[dict]:
                if not text_objects:
                    return []
                tasks = [self.__async_translate_text(text_object) for text_object in text_objects]
                results = await asyncio.gather(*tasks, return_exceptions=False)
                return results

            async def _orchestrate_all_translations(self, titles_data: list[dict], descriptions_data:
list[dict]) -> tuple[
                list[dict], list[dict]]:
                """Orchestrates title and description translations within a single async context."""
                translated_titles_list = []
                translated_descriptions_list = []

                if titles_data:
                    logging.info(f"Orchestrating translation for {len(titles_data)} titles.")
                    translated_titles_list = await self.__async_batch_translate_texts(titles_data)

                if descriptions_data:
                    logging.info(f"Orchestrating translation for {len(descriptions_data)}
descriptions.")
                    translated_descriptions_list = await
self.__async_batch_translate_texts(descriptions_data)

                return translated_titles_list, translated_descriptions_list

            def run(self, link_objects: list[dict]) -> list[dict]:
                if not link_objects:
                    logging.info("No link objects to process.")
                    return []
                updated_link_objects = deepcopy(link_objects)

                titles_data = [
                    {
                        self.__url_column_name: link_object[self.__url_column_name],
                        'text': self.__remove_punctuation(link_object['title'])
                    } for link_object in updated_link_objects if
                    link_object.get('title') and link_object.get(self.__url_column_name)
                ]
                descriptions_data = [
                    {
                        self.__url_column_name: link_object[self.__url_column_name],
                        'text': self.__remove_punctuation(link_object['description'])
                    } for link_object in updated_link_objects if
                    link_object.get('description') and link_object.get(self.__url_column_name)
```

```
        ]

        translated_titles_list = []
        translated_descriptions_list = []

        try:
            current_loop = asyncio.get_running_loop()
            if current_loop.is_running():
                logging.error("TranslationFilter.run() was called from an already running event
loop. "
                              "This method should be made 'async def' or called from a synchronous
context.")

        except RuntimeError:
            pass

        if titles_data or descriptions_data:
            try:
                logging.info("Starting orchestrated translations.")
                translated_titles_list, translated_descriptions_list = asyncio.run(
                    self._orchestrate_all_translations(titles_data, descriptions_data)
                )
                logging.info("Finished orchestrated translations.")
            except Exception as e:
                logging.error(f'Error occurred during orchestrated translations: {e}')
        else:
            logging.info("No titles or descriptions to translate.")

        translated_titles_map = {
            item[self.__url_column_name]: item['text']
            for item in translated_titles_list if
            isinstance(item, dict) and self.__url_column_name in item and 'text' in item
        }
        translated_descriptions_map = {
            item[self.__url_column_name]: item['text']
            for item in translated_descriptions_list if
            isinstance(item, dict) and self.__url_column_name in item and 'text' in item
        }
        for link_object in updated_link_objects:
            url = link_object.get(self.__url_column_name)
            if not url: continue
            if url in translated_titles_map: link_object['title'] = translated_titles_map[url]
                            if url in translated_descriptions_map: link_object['description'] =
translated_descriptions_map[url]

        return super().run(updated_link_objects)
```

Файл website_data_extraction_filter.py

Реалізація функціональної задачі автоматичного вилучення релевантної інформації з вебресурсів (назва компанії, вебсайт, опис діяльності тощо)

```
from copy import deepcopy
from .basic_filter import BasicFilter
from .request_adapter import RequestAdapter

import re


class WebsiteDataExtractionFilter(BasicFilter):

    def __init__(self, url_column_name: str, num_occurrences_column_name: str, location_column_name: str):
        super().__init__()
        self.__url_column_name = url_column_name
        self.__num_occurrences_column_name = num_occurrences_column_name
        self.__location_column_name = location_column_name

    def __clean_text(self, text):
        return re.sub(r'[^\x20-\x7E]', '', text)

    def run(self, link_objects: list[dict]) -> list[dict]:
        updated_link_objects = deepcopy(link_objects)
        urls = [link_object[self.__url_column_name] for link_object in updated_link_objects]
        request_adapter = RequestAdapter(urls)
        website_data_results = request_adapter.run()
        for website_data in website_data_results:
            url = website_data[self.__url_column_name]
            title = self.__clean_text(website_data['title'])
            description = self.__clean_text(website_data['description'])
            text = self.__clean_text(website_data['text'])
            for link_object in updated_link_objects:
                if link_object[self.__url_column_name] == url:
                    link_object['title'] = title
                    link_object['description'] = description
                    link_object['text'] = text # f"{text[:500]}..." if len(text) > 500 else text

        return super().run(updated_link_objects)
```

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2025 р.

ВЕБЗАСТОСУНОК ДЛЯ ПОШУКУ B2B КЛІЄНТІВ ШЛЯХОМ ВЕБСКРАПІНГУ МЕРЕЖІ ІНТЕРНЕТ

Програма та методика тестування

КПІ.ІП-1212.045440.04.51

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Тетяна ЛІХОУЗОВА

Нормоконтроль:
_____ Максим ГОЛОВЧЕНКО

Виконавець:
_____ Даніїл ЙОЛКІН

Київ – 2025

ЗМІСТ

1 ОБ’ЄКТ ВИПРОБУВАНЬ

Об’єктом випробування є розроблений в рамках дипломного проєкту вебзастосунок Sieve. Тестуванню підлягає система як єдине ціле, включаючи її ключові компоненти: серверну частину, реалізовану на фреймворку Flask, клієнтський вебінтерфейс для взаємодії з користувачем, модуль асинхронної обробки даних algorithm_app та його інтеграцію з основним додатком, а також коректність взаємодії з базою даних PostgreSQL.

Перевірка працездатності програмного забезпечення проводиться у найбільш поширених сучасних веббраузерах, таких як Google Chrome та Safari, на різних настільних операційних системах, зокрема Windows та macOS.

2 МЕТА ТЕСТУВАННЯ

Метою тестування є наступне:

– Перевірка правильності роботи програмного забезпечення відповідно до функціональних вимог.

– Перевірка надійності та цілісності збереження даних у базі даних.

– Перевірка сумісності вебзастосунку з останніми версіями сучасних браузерів.

– Перевірка сумісності вебзастосунку з різними операційними системами.

– Знаходження та ідентифікація проблем, помилок і недоліків з метою їх подальшого усунення.

– Перевірка зручності та інтуїтивної зрозумілості графічного інтерфейсу користувача.

– Оцінка відповідності системи нефункціональним вимогам, зокрема продуктивності та безпеки.

3 МЕТОДИ ТЕСТУВАННЯ

Для забезпечення якості та надійності розробленого програмного забезпечення необхідно застосувати структуровану стратегію тестування. Ця стратегія базується на моделі піраміди тестування та поєднує різні типи та методи тестування для комплексного покриття функціональності та перевірки відповідності нефункціональним вимогам.

Для тестування програмного забезпечення використовуються такі методи:

– Мануальне тестування, яке використовується для виконання розроблених тест-кейсів вручну, а також для проведення дослідницького тестування та оцінки зручності користування, що є важливим для виявлення неочевидних помилок та покращення користувацького досвіду.

– Функціональне тестування є ключовим для перевірки відповідності реалізованої поведінки програмного забезпечення очікуваній поведінці, що описана у функціональних вимогах. Воно застосовується на всіх рівнях, від перевірки окремої функції до комплексних наскрізних сценаріїв.

– Тестування «сірої скриньки» (gray-box testing). Об’єктом тестування тут є деякі особливості внутрішньої поведінки програми. Перевіряється коректність вихідних даних при заданих вхідних, застосовується для тестування окремих функцій.

– Тестування «чорної скриньки» (black-box testing). Об’єктом тестування тут є функції присутні у програмі. Перевіряється коректність вихідних даних при заданих вхідних.

4 ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Під час проведення тестування будуть використовуватись наступні допоміжні засоби:

– Браузер та вбудовані інструменти розробника – для аналізу роботи клієнтської частини, діагностики помилок JavaScript та моніторингу мережевих запитів.

– Клієнти для роботи з PostgreSQL – для перевірки коректності операцій з базою даних та цілісності збережених даних.

Порядок проведення тестування буде наступним:

– Юніт-тестування окремих компонентів системи (функцій, методів класів) для підтвердження коректності їхньої індивідуальної логіки.

– Тестування взаємодії між окремими модулями системи, зокрема перевірка коректності обміну даними між вебдодатком Flask, базою даних PostgreSQL та модулем algorithm_app.

– Комплексна перевірка всієї системи як єдиного цілого на відповідність функціональним вимогам, що включає виконання розроблених тест-кейсів для всіх основних користувацьких сценаріїв.

– Мануальна оцінка інтуїтивності інтерфейсу, логічності навігації та загального користувацького досвіду для виявлення потенційних незручностей.

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2025 р.

ВЕБЗАСТОСУНОК ДЛЯ ПОШУКУ B2B КЛІЄНТІВ ШЛЯХОМ ВЕБСКРАПІНГУ МЕРЕЖІ ІНТЕРНЕТ

Керівництво користувача

КПІ.ІП-1212.045440.05.34

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Тетяна ЛІХОУЗОВА

Нормоконтроль:
_____ Максим ГОЛОВЧЕНКО

Виконавець:
_____ Даніїл ЙОЛКІН

Київ – 2025

ЗМІСТ

1 ПРИЗНАЧЕННЯ ПРОГРАМИ

Програмне забезпечення Sieve є вебзастосунком, розробленим для автоматизації процесу пошуку та первинного аналізу даних про потенційних B2B-клієнтів. Система надає зареєстрованим користувачам можливість формувати складні пошукові запити, що включають ключові слова, цільову країну та специфічні слова-фільтри. На основі цих критеріїв система в асинхронному режимі виконує вебскрапінг та аналіз даних з відкритих джерел мережі Інтернет. Користувач може відстежувати статус виконання своїх запитів та, після їх успішного завершення, завантажувати структуровані результати у вигляді файлу формату XLSX для подальшого використання у своїй комерційній діяльності.

Інсталяційний пакет представлений у вигляді структурованого репозиторію системи контролю версій Git та включає:

– Вихідний код Flask-додатку та модуля algorithm_app.

– Файл залежностей requirements.txt для встановлення необхідних Python-пакетів.

– Шаблон конфігураційного файлу config.yaml для налаштування параметрів алгоритму.

– Документацію з інструкціями по розгортанню.

Репозиторій програмного забезпечення має наступну логічну структуру для забезпечення модульності та зручності супроводу:

– flask_app/: Директорія, що містить основний код Flask-додатку. Код складається з головного файл додатку, що містить визначення маршрутів, моделей даних та логіку взаємодії; директорію з HTML-шаблонами для рендерингу вебсторінок; директорію зі статичними файлами (CSS-стилі, JavaScript-скрипти, зображення).

– algorithm_app/: Директорія, що містить код аналітичного модуля Sieve, включаючи логіку скрапінгу, аналізу даних, файли конфігурації (config.yaml) та допоміжні утиліти.

– requirements.txt: Файл із переліком всіх зовнішніх Python-бібліотек, необхідних для роботи проєкту.

2 ПІДГОТОВКА ДО РОБОТИ З ПРОГРАМНИМ ЗАБЕЗПЕЧЕННЯМ

2.1 Системні вимоги для коректної роботи

Для успішної роботи даного застосунку необхідне виконання наступних вимог:

– Тип процесору: Intel Core i5 (Apple M1 – рекомендовано).

– Об‘єм ОЗП: 4 Гб (8 Гб – рекомендовано).

– Підключення до мережі Інтернет зі швидкістю від 20 мегабіт (1000 мегабіт – рекомендовано).

2.2 Завантаження застосунку

Перш за все, необхідно підготувати сервер (рекомендовано Linux-дистрибутив, наприклад, Ubuntu Server). Це включає оновлення системних пакетів до актуальних версій, а також встановлення ключових компонентів таких як Python (версії 3.9 або вище) та менеджера пакетів pip, та системи управління базами даних PostgreSQL.

Доступ до програмних файлів забезпечується через репозиторій системи контролю версій Git. Вихідний код застосунку завантажується на сервер за допомогою команди git clone. Потім у директорії проекту створюється ізольоване віртуальне оточення Python для уникнення конфліктів залежностей: python3 -m venv venv. Перед встановленням пакетів віртуальне оточення активується командою source venv/bin/activate. Далі, усі необхідні Python-бібліотеки встановлюються в активне оточення з файлу requirements.txt за допомогою команди pip install -r requirements.txt.

Після виконання попередніх кроків необхідно налаштувати конфігурацію застосунку:

– Чутливі дані, такі як SECRET_KEY та SQLALCHEMY_DATABASE_URI (з обліковими даними доступу до PostgreSQL), задаються як змінні середовища на сервері. Також встановлюється FLASK_ENV=production.

– Виконується первинне створення таблиць у базі даних. Якщо використовується Flask-Migrate, застосовуються міграції. В іншому випадку, виконується спеціальний скрипт або команда Flask CLI, що викликає db.create_all().

2.3 Перевірка коректної роботи

По завершенню процесу розгортання програмного забезпечення, вебзастосунок Sieve повинен стати доступним за своєю мережевою адресою (URL). У разі, якщо при спробі відкрити цю адресу у веббраузері виникає помилка з'єднання або помилка сервера (наприклад, "502 Bad Gateway"), процес розгортання вважається неуспішним. Інакше, користувач має змогу отримати доступ до системи. Після успішного переходу за адресою на екрані повинна коректно відобразитись головна сторінка застосунку з назвою "Sieve.ai" та основними елементами навігації.

3 ВИКОНАННЯ ПРОГРАМИ

Першим кроком для нового користувача є створення облікового запису. Користувач відкриває головну сторінку вебзастосунку.

У навігаційній панелі він обирає опцію "Sign Up" (Зареєструватися), після чого система перенаправляє його на сторінку реєстрації.

На цій сторінці користувач заповнює обов'язкові поля: унікальне ім'я користувача, дійсну адресу електронної пошти, та пароль, який відповідає вимогам безпеки, а також підтвердження пароля. Після введення всіх даних користувач натискає кнопку "Sign Up". Система валідує введені дані. У випадку успішної валідації та відсутності конфліктів (наприклад, вже існуючого користувача з таким ім'ям або email), створюється новий обліковий запис, пароль зберігається у гешованому вигляді. Користувач отримує повідомлення про успішну реєстрацію (наприклад, "Account created successfully! You can now sign in.") та, як правило, перенаправляється на сторінку входу для подальшої автентифікації. У разі некоректного введення даних (наприклад, невідповідність паролів, невалідний формат email, занадто коротке ім'я користувача) система відображає відповідні повідомлення про помилки безпосередньо у формі реєстрації, дозволяючи користувачеві виправити введені дані.

Після успішної реєстрації (або якщо користувач вже має обліковий запис), він здійснює вхід до системи. Для цього користувач переходить на сторінку "Sign In" (Увійти).

Тут він вводить своє ім'я користувача та пароль, вказані під час реєстрації. Після натискання кнопки "Sign In", система перевіряє відповідність наданих облікових даних тим, що зберігаються в базі. У разі успішної автентифікації користувач перенаправляється на основну робочу сторінку застосунку. Важливою ознакою успішного входу є зміна навігаційної панелі: замість опцій "Sign Up" та "Sign In" з'являється ім'я поточного користувача та опції "Log Out" (Вийти) та “Query” (Сторінка запитів).

Якщо ж введені дані невірні, система відображає повідомлення про помилку автентифікації.

Автентифікований користувач переходить на сторінку "Query" (Запити). На цій сторінці розташована форма для створення нового пошукового запиту.

Користувач заповнює поля форми Keywords (Ключові слова), Country (Країна), Whitelist Words (Білий список слів).

Після заповнення всіх необхідних полів користувач натискає кнопку "Find" (Знайти). Система проводить валідацію введених даних. У разі успіху, параметри запиту зберігаються у базі даних (таблиця search_query) з початковим статусом "submitted" (Відправлено). Користувач бачить повідомлення про успішне створення запиту та початок його обробки, наприклад, "Query saved (ID: 123) and processing started!". Новий запит негайно з'являється у списку "Your Submitted Queries" (Ваші відправлені запити) на цій же сторінці, відображаючи його ID, параметри та поточний статус.

Одночасно, у фоновому режимі, система ініціює асинхронний запуск модуля algorithm_app для обробки цього запиту. Після відправлення запиту користувач може відслідковувати процес його виконання на сторінці "Query". Статус запиту у списку при оновленні сторінки змінюється, відображаючи етапи роботи алгоритму. Спочатку статус може змінитися на "processing" (Обробляється), вказуючи на те, що модуль algorithm_app активно виконує вебскрейпінг та аналіз даних. Тривалість цього етапу залежить від складності запиту, кількості ключових слів та обсягу даних, що аналізуються, і може становити від декількох хвилин до декількох годин, відповідно до нефункціональних вимог. Після завершення всіх етапів обробки алгоритмом, статус запиту оновлюється на "completed" (Завершено) у випадку успішного виконання, або "failed" (Помилка) у випадку виникнення проблем під час обробки.

Коли статус запиту користувача змінюється на "completed", поруч із ним у списку запитів з'являється активне посилання або кнопка "Download Results" (Завантажити результати).

Користувач натискає на це посилання. Система, перевіривши права доступу користувача до даного запиту та наявність файлу, ініціює завантаження згенерованого XLSX-файлу на локальний комп'ютер користувача. Цей файл містить структуровану інформацію про знайдених потенційних B2B-клієнтів, що відповідають критеріям запиту. Якщо файл з якихось причин відсутній на сервері або запит ще не завершено, система виводить відповідне інформаційне повідомлення.

Протягом роботи з системою користувач може використовувати навігаційні елементи. Наприклад, натиснувши на посилання "Home" у верхньому лівому куті, користувач повертається на головну сторінку застосунку. По завершенню роботи, для виходу зі свого облікового запису, користувач натискає на посилання "Log Out" у навігаційній панелі. Система завершує поточну сесію користувача та перенаправляє його на головну сторінку Навігаційна панель повертається до стану для неавторизованого користувача, відображаючи опції "Sign Up" та "Sign In".

Усі креслення екранних форм наведені в графічних матеріалах (аркуш 4).

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2025 р.

ВЕБЗАСТОСУНОК ДЛЯ ПОШУКУ B2B КЛІЄНТІВ ШЛЯХОМ ВЕБСКРАПІНГУ МЕРЕЖІ ІНТЕРНЕТ

Графічний матеріал

КПІ.ІП-1212.045440.06.99

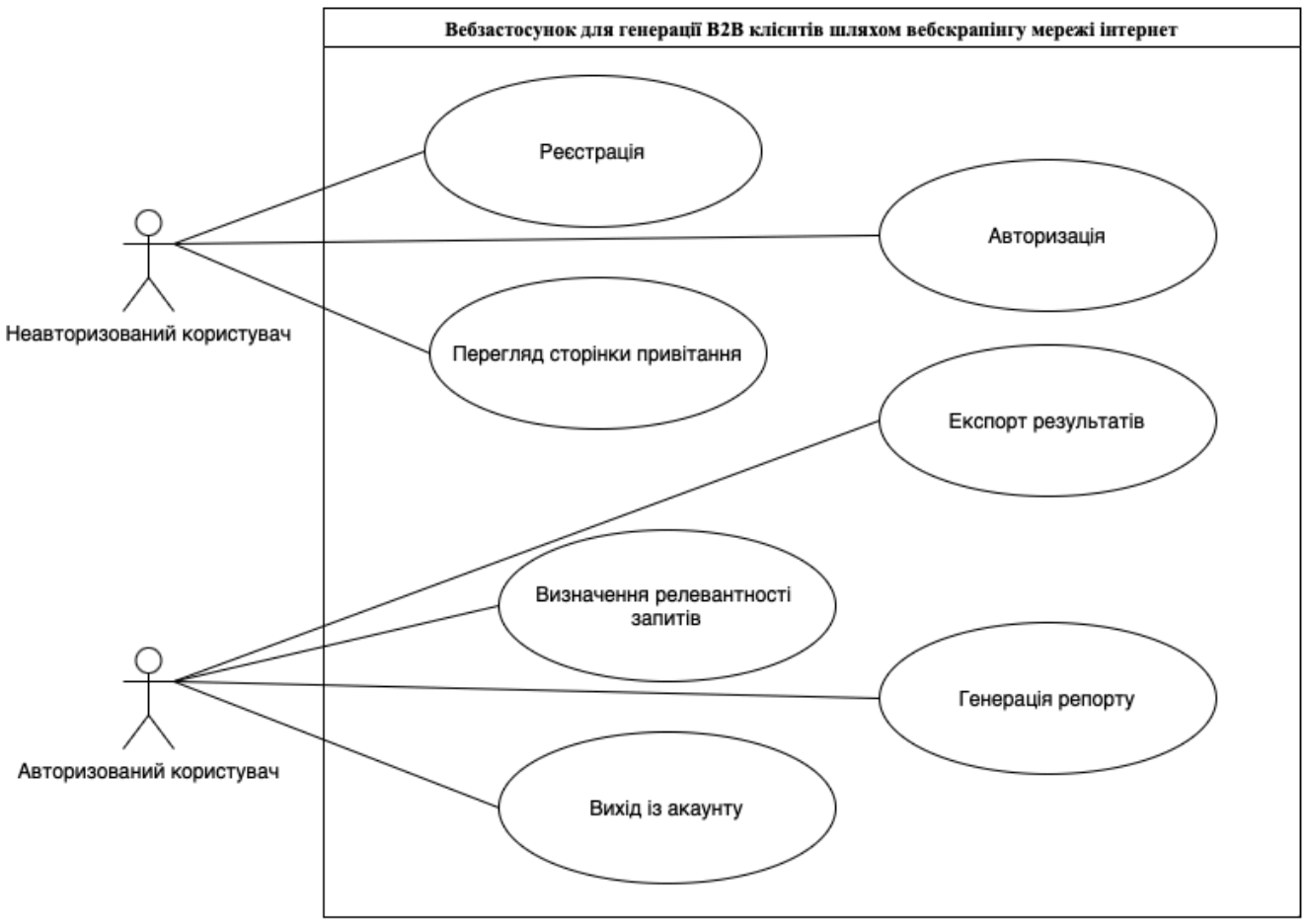
“ПОГОДЖЕНО”
Керівник проєкту:
_____ Тетяна ЛІХОУЗОВА

Нормоконтроль:
_____ Максим ГОЛОВЧЕНКО

Виконавець:
_____ Даніїл ЙОЛКІН

Київ – 2025

КПІ.ІП-1212.045440.06.99.ССВ



					КПІ.ІП-1212.045440.06.99.ССВ	
Зм.	Арк.	№ докум.	Підп.	Дата	Схема структурна варіантів використань	Лит. / Маса / Масштаб
Розробив		Йолкін Д.С.				
Перевірив		Ліхоузова Т.А.				Аркуш 1 / Аркушів 4
Т. контр.						
Н. контр.		Головченко М.М			Вебзастосунок для пошуку B2B клієнтів шляхом вебскрапінгу мережі інтернет	КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІП-12
Затвердив		Жаріков Е.В.				

Зм.	Арк.	№ докум.	Підп.	Дата
Розробив		Волкін Д.С.		
Перевірив		Лихогзова Т.А.		
Т. контр.				
Н. контр.		Голованенко М.М		
Затвердив		Жарікова Е.В.		

КПІ.ІП-1212.045440.06.99.ССМ

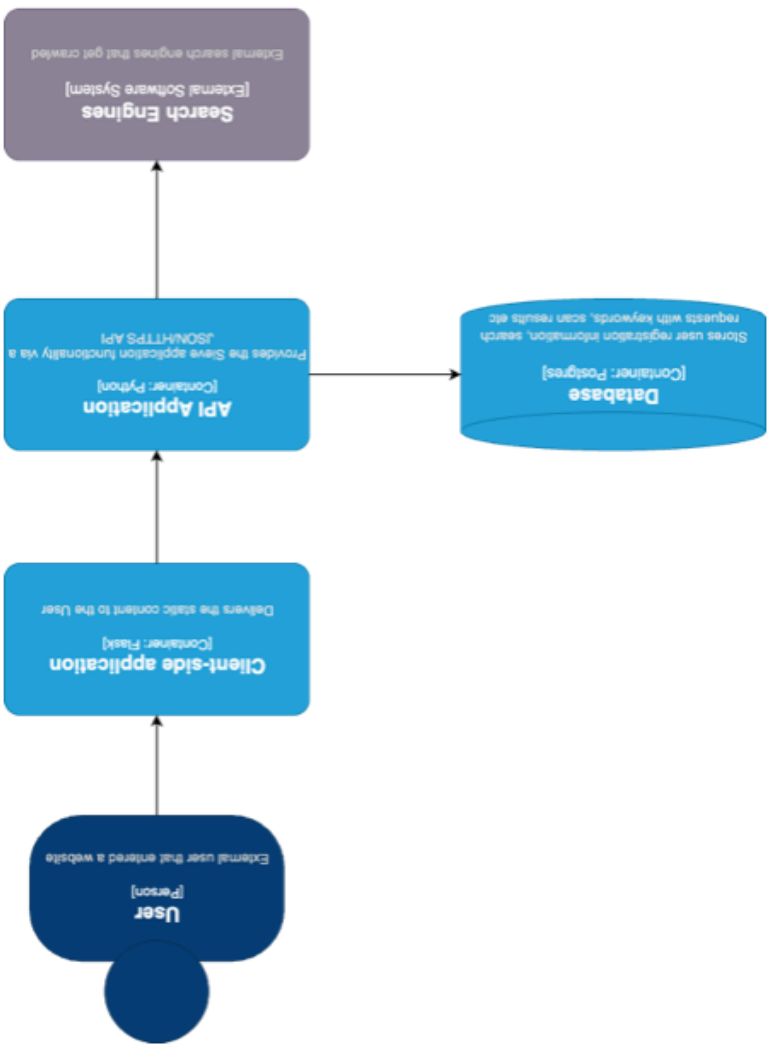
Літ.	Маса	Масштаб

Схема структурна компонентів програмного забезпечення

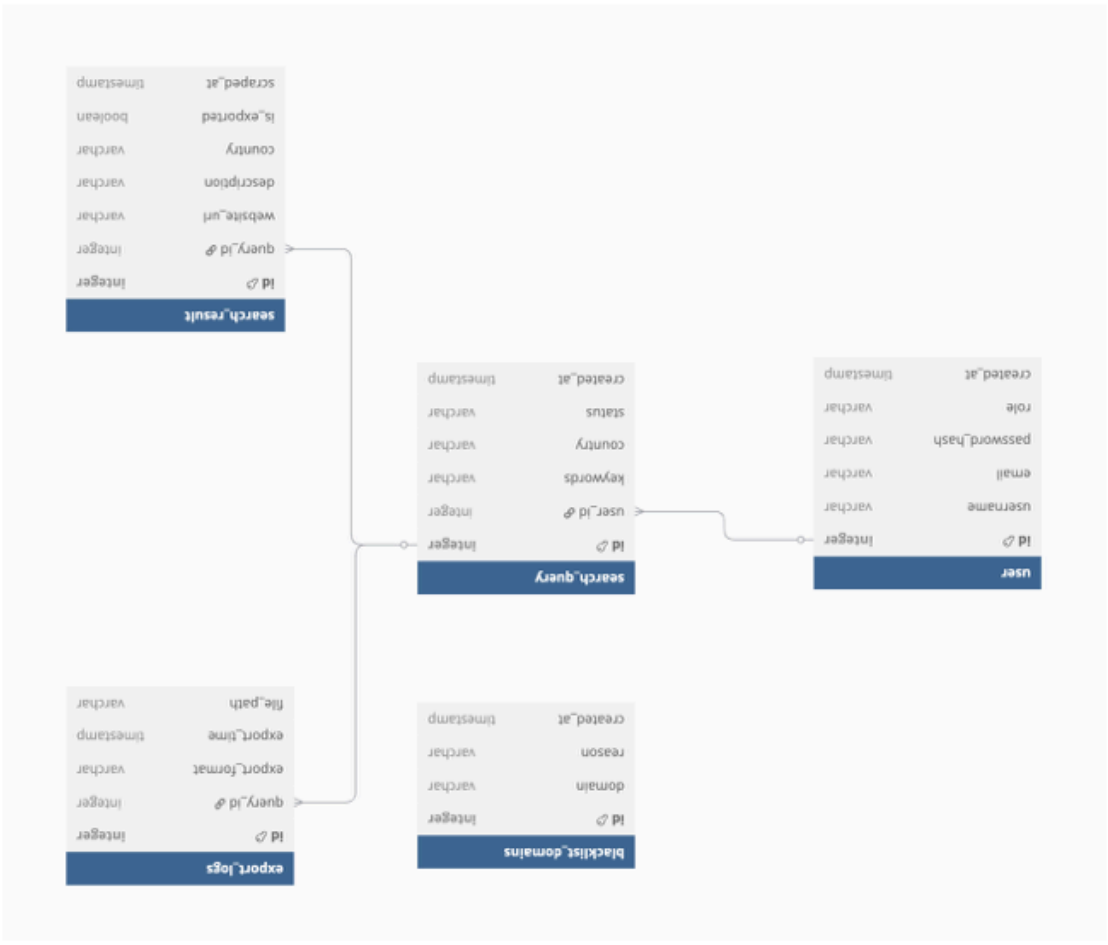
Аркуш 2	Аркушів 4

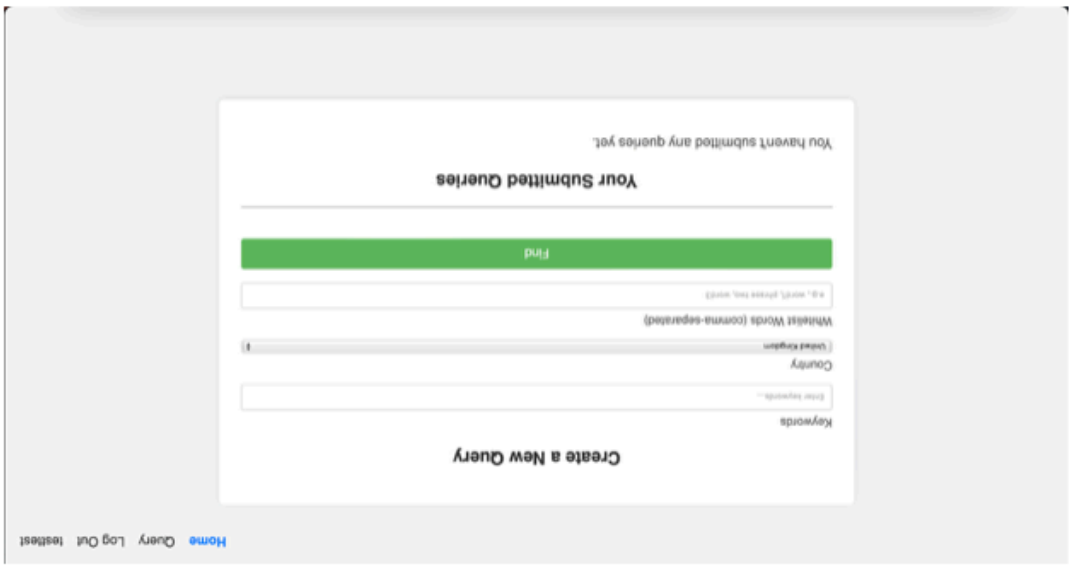
Вебзастосунок для пошуку B2B клієнтів шляхом вебскрапінгу мережі інтернет

КПІ ім.Ігоря Сікорського
Кафедра ІПІ
гр. ІП-12

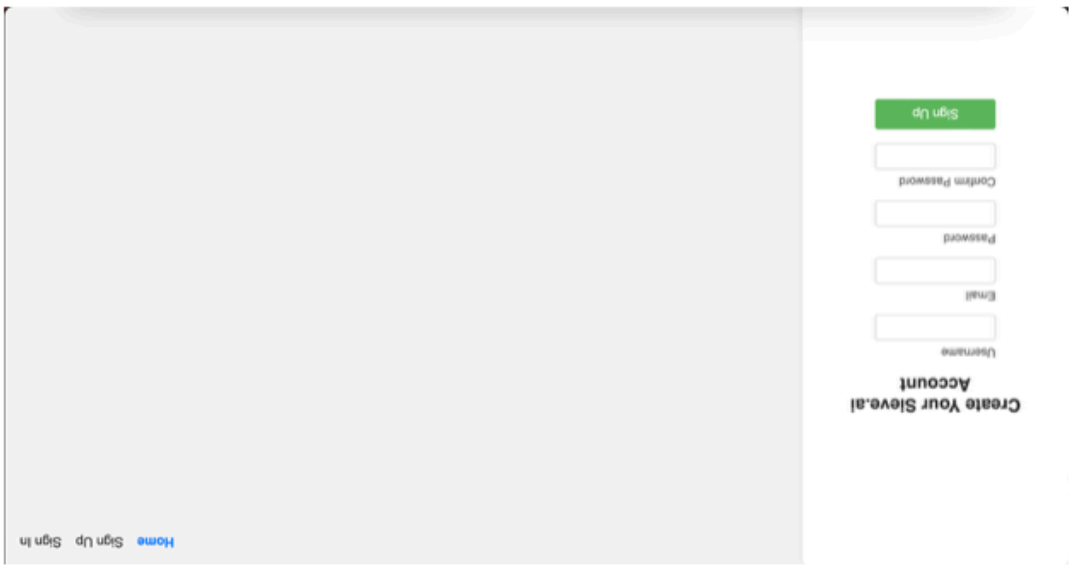


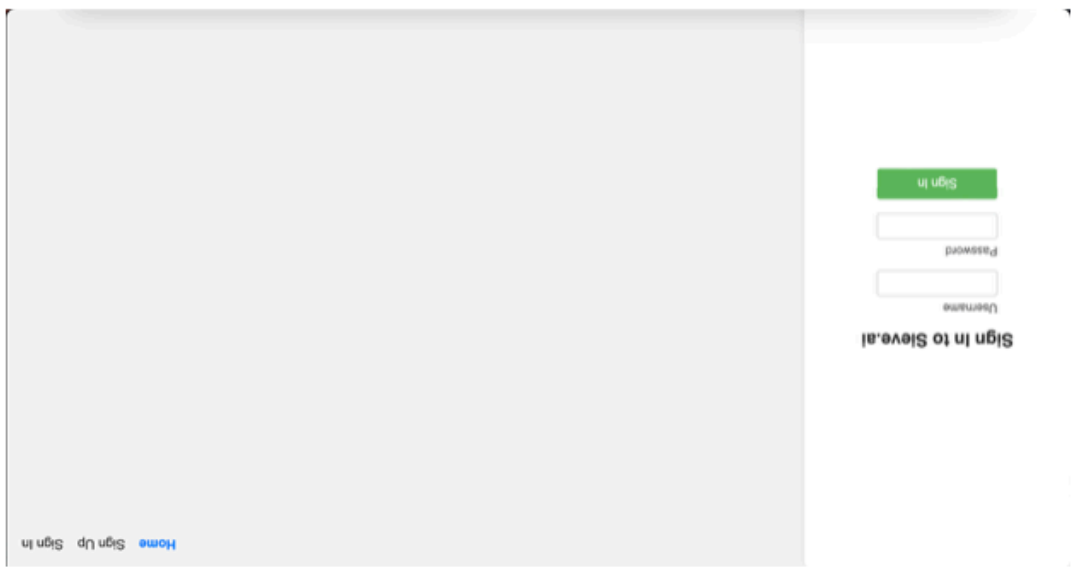
Зм.	Арк.	№ докум.	Підп.	Дата	КПІ.ІП-1212.045440.06.99.СБД		
Розробив		Волкін Д.С.					
Перевірив		Лихогзова Т.А.					
Т. контр.					Схема бази даних	Літ.	Маса	Масштаб
Н. контр.		Голованенко М.М				Аркуш 3	Аркушів 4
Затвердив		Жарікова Е.В.			Вебзастосунок для пошуку B2B клієнтів шляхом вебскрапінгу мережі інтернет	КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІП-12	

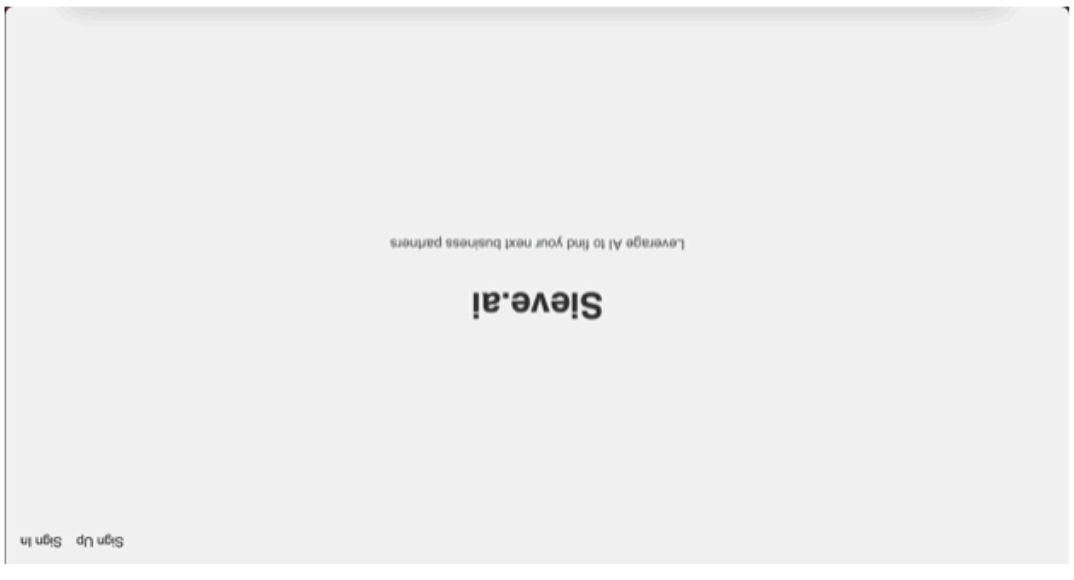




Зм.	Арк.	№ докум.	Підп.	Дата	КПІ.ІП-1212.045440.06.99.КЕ		
Розробив		Йолкін Д.С.			Кресленяя екранних форм	Літ.	Маса	Масштаб
Перевірив		Лихоузова Т.А.						
Т. контр.						Аркуш 4		Аркушів 4
Н. контр.		Голованенко М.М			Вебзастосунок для пошуку B2B клієнтів шляхом вебскрапінгу мережі інтернет	КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІП-12		
Затвердив		Жарікова Е.В.						







КПІ.ІП-1212.045440.06.99.КЕ