Київський національний торговельно-економічний університет

Кафедра комп'ютерних наук та інформаційних систем

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
на тему:

«Reverse Engineering пошукових систем»

Студента 2м курсу, 3 групи,
спеціальності
122 «Комп'ютерні науки» _____ Колесникова Микити Миколайовича
підпис студента

Науковий керівник
Доктор технічних наук, професор _____ Краскевич Валерій Євгенович
підпис керівника

Гарант освітньої програми
кандидат технічних наук, доцент _____ Пурський Олег Іванович
підпис керівника

Київ 2021

Київський національний торговельно-економічний університет

Факультет інформаційних технологій
Кафедра комп'ютерних наук та інформаційних систем
Спеціальність 122 «Комп'ютерні науки»

Затверджую
Зав. кафедри _____Пурський О.І.
«20» грудня 2020р.

Завдання
на випускну кваліфікаційну роботу студенту

Колесникову Микиті Миколайовичу
(прізвище, ім'я, по батькові)

1. Тема випускної кваліфікаційної роботи
«Reverse Engineering пошукових систем»
Затверджена наказом ректора від *«05» листопада 2020 р. № 3311*
2. Строк здачі студентом закінченої роботи *26 листопада 2021 року*
3. Цільова установка та вихідні дані до роботи
Мета роботи: практичне а також теоретичне дослідження інструментів та методології аналізу веб-додатків Reverse Engineering.
Об'єкт дослідження: способи аналізу застосування механізмів роботи сторонніх сервісів.
Предмет дослідження: теоретико-методологічні та практичні аспекти дослідження роботи web-додатків а також браузерні інструменти розробника (ChromeDevTools).
4. Перелік графічного матеріалу_____
_____
_____

5. Консультанти по роботі із зазначенням розділів, за якими здійснюється консультування:

Розділ	Консультант	Підпис, дата

	(прізвище, ініціали)	Завдання видав	Завдання прийняв
1	Краскевич В.Є.	15.12.2020р.	15.12.2020р.
2	Краскевич В.Є.	15.12.2020р.	15.12.2020р.
3	Краскевич В.Є.	15.12.2020р.	15.12.2020р.

6. Зміст випускної кваліфікаційної роботи (перелік питань за кожним розділом)

ВСТУП

РОЗДІЛ 1. Теоретичні аспекти reverse engineering та web-scraping у контексті веб-розробки

1.1. Поняття та можливості reverse engineering

1.2. Сутність Web-scraping

1.3. Явище асинхронності у веб-розробці

РОЗДІЛ 2. Огляд методик reverse engineering та розробки веб-скраперів

2.1. Інструментарій для аналізу роботи веб-сайтів та додатків – Chrome DevTools

2.2. Reverse engineering як засіб виявлення проблем з додатком – debug tools

2.3. Reverse engineering як методологія дослідження механізмів роботи додатків

2.4. Асинхронність на прикладах Python

РОЗДІЛ 3. Reverse engineering пошукових систем та розробка веб-скраперів

3.1. Планування алгоритму роботи

3.2. Моделювання процесу розробки

3.3. Reverse engineering пошукових систем та створення веб-скраперів

ВИСНОВКИ

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

7. Календарний план виконання роботи

№ Пор.	Назва етапів випускної кваліфікаційної роботи	Строк виконання етапів роботи	
		За планом	фактично
1	2	3	4
1	*Вибір теми випускної кваліфікаційної*	*01.11.2020*	*01.11.2020*

	роботи		
2	*Розробка та затвердження завдання на випускну кваліфікаційну роботу*	*05.12.2020*	*05.12.2020*
3	*Вступ*	*01.06.2021*	
4	*РОЗДІЛ 1. Теоретичні аспекти reverse engineering та web-scraping у контексті веб-розробки*	*25.06.2021*	
5	*РОЗДІЛ 2. Огляд методик reverse engineering та розробки веб-скраперів*	*02.09.2021*	
6	*РОЗДІЛ 3. Reverse engineering пошукових систем та розробка веб-скраперів*	*21.10.2021*	
7	*Висновки*	*02.11.2021*	
8	*Підготовка статті у збірник наукових статей магістрів*	*18.06.2021*	
9	*Здача випускної кваліфікаційної роботи на кафедру науковому керівнику*	*05.11.2021*	
10	*Попередній захист випускної кваліфікаційної роботи*	*01.12.2021*	
11	*Виправлення зауважень, зовнішнє рецензування випускної кваліфікаційної роботи*	*03.12.2021*	
12	*Представлення готової зшитої випускної кваліфікаційної роботи на кафедру*	*06.12.2021*	
13	*Публічний захист випускної кваліфікаційної роботи*	*За розкладом роботи ЕК*	

8. Дата видачі завдання «15» грудня 2020 р.

Керівник випускної кваліфікаційної роботи

Краскевич В.Є.
(прізвище, ініціали, підпис)

Гарант освітньої програми Пурський О.І.
(прізвище, ініціали, підпис)

Завдання прийняв до виконання студент-дипломник

Колесников М. М.
(прізвище, ініціали, підпис)

9. Відгук керівника випускної кваліфікаційної роботи

____
____
____
____
____
____
____
____
____
____
____
____
____
____
____
____
____
____
____
____
____
____
____

Керівник випускної кваліфікаційної роботи
________________ 20__р.
(підпис, дата)

10. Висновок про випускну кваліфікаційну роботу

Випускна кваліфікаційна робота студента ____________________
(прізвище, ініціали)
може бути допущена до захисту в екзаменаційній комісії.

Гарант освітньої програми_________________________Пурський О.І.____
(підпис, прізвище, ініціали)

Завідувач кафедри ______________________________Пурський О.І.____
(підпис, прізвище, ініціали)
«_____»__________________2021 р.

Анотація

У даній випускній кваліфікаційній роботі проаналізовано особливості використання, задачі та сфери впливу емпіричного методу аналізу роботи додатків та сторонніх мікро-сервісів за допомогою reverse engineering. Наведено практичні приклад застосування даного методу дослідження, розглянуто особливості web-scraping та створено додаток, що використовує веб-скрапери Google Search та Bing Search.

Ключові слова: API, додаток, Reverse Engineering, зворотня інженерія, Web-scraping, веб-скрапінг, Debug, Debugger, Network, ChromeDevTools, Python, JavaScript, XHR, Asyncio, асинхронність, Event loop, url-параметри, HTTP, HTTP-headers, HTTP-заголовки, HTML

Annotation

This final qualification project analyzes the features of use, tasks and influence scope of the empirical method of applications analysis and third-party micro-services analysis using reverse engineering. Provides a practical example of this research method, considers the features of web-scraping and describes creation of an application that using Google Search and Bing Search web-scrappers.

Keywords: API, application, Reverse Engineering, Web-scraping, Debug, Debugger, Network, ChromeDevTools, Python, JavaScript, XHR, Asyncio, asynchronous, Event loop, url-parameters, HTTP, HTTP-headers, HTML

ЗМІСТ

ВСТУП **8**
РОЗДІЛ 1.ТЕОРЕТИЧНІ АСПЕКТИ REVERSE ENGINEERING ТА WEB-SCRAPING У КОНТЕКСТІ ВЕБ-РОЗРОБКИ **10**
1.1. Поняття та можливості reverse engineering 10
1.2. Сутність Web-scraping 12
1.3. Явище асинхронності у веб-розробці 14
РОЗДІЛ 2. ОГЛЯД МЕТОДИК REVERSE ENGINEERING ТА РОЗРОБКИ ВЕБ-СКРАПЕРІВ **18**
2.1. Інструментарій для аналізу роботи веб-сайтів та додатків – Chrome DevTools 18
2.2. Reverse engineering як засіб виявлення проблем з додатком – debug tools 22
2.3. Reverse engineering як методологія дослідження механізмів роботи додатків 27
2.4. Асинхронність на прикладах Python 32
РОЗДІЛ 3. REVERSE ENGINEERING ПОШУКОВИХ СИСТЕМ ТА РОЗРОБКА ВЕБ-СКРАПЕРІВ **39**
3.1. Планування алгоритму роботи 39
3.2. Моделювання процесу розробки 41
3.3. Reverse engineering пошукових систем та створення веб-скраперів 43
РЕЗУЛЬТАТИ ТА ВИСНОВКИ ..... Ошибка! Закладка не определена.
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ **71**

ВСТУП

Обрана тема є актуальною здебільшого для розробників серверної частини додатків (надалі Back-End розробників), оскільки у роботі розглядаються практичні моменти дослідження роботи веб-орієнтованих мікросервісів. На момент 21-го сторіччя припадає надзвичайно стрімке зростання можливостей IT-технологій, імплементацію роботи яких є частою задачею будь-якого сучасного розробника програмного забезпечення. Методологія Reverse Engineering дозволяє дослідити як «під капотом» влаштований незнайомий механізм, з метою його подальшого використання. Однак переймання хороших практик Reverse Engineering може у вкрай позитивному ключі відобразитися й на будь-яких інших сферах життєдіяльності людини при зустрічі з невідомим явищем.

Практична значущість та актуальність виконаної роботи може бути підтверджена численними прецедентами використання результатів у комерційних цілях: веб-скрапінг передбачає собою добування даних, необхідних для подальшої її обробки та аналізу, це може бути автоматизація прийняття рішень стосовно торгівлі певною валютою (в тому числі крипто-валютою); збирання великої кількості конкурентної інформації як наприклад сайт hotline.ua, що використовує численну кількість даних з інших джерел для відображення поточних цін на певний товар; десятки різноманітних пошукових систем можуть застосовуватися у комбінації у якості джерела даних, як це робить Rinalogy Search, й взаємодіяти з нейронною мережею інтерактивного навчання, щоб отримувати персоналізовані результати на основі відгуків користувачів у реальному часі та таким чином значно оптимізувати процес пошуку; результати пошукових запитів можуть використовуватися для SEO-аналізу (SEO – Search Engine Optimization, оптимізація для пошукового движку), що призначений дати інформацію щоб визначити дії для досягнення розміщення сайту у топ-10 результатах пошуку (на першій сторінці) та ефективної індексації у пошукових системах.

Метою роботи є практичне а також теоретичне дослідження інструментів та методології аналізу веб-додатків Reverse Engineering.

Завдання, що забезпечать реалізацію мети:

1. Теоретичний огляд інструментів Reverse Engineering;
2. Аналіз роботи та захисту веб-додатків у контексті Web-Scraping;
3. Дослідження теоретичної бази щодо створення додатку, що використовує веб-скрапери;
4. Створення веб-скраперів на основі інформації, добутої у ході Reverse Engineering.

Об’єктом дослідження є аналіз механізмів роботи сторонніх сервісів.

Предметом дослідження є теоретико-методологічні та практичні аспекти дослідження роботи web-додатків а також браузерні інструменти розробника (ChromeDevTools).

Методи дослідження ґрунтуються на принципах інформаційного моделювання предметної області, загальнонаукового аналітичного методу а також на основі емпіричного аналізу

Інформаційна база дослідження: наукові публікації вітчизняних та зарубіжних фахівців, електронні ресурси Інтернет, навчальні посібники.

Наукова новизна одержаних результатів полягає в удосконаленні механізму аналізу веб-орієнтованих додатків а також у ґрунтовному дослідженні методів автоматичного добування інформації з веб-ресурсів.

Результати дослідження опубліковано у збірнику наукових статей студентів, які здобувають освітній ступінь магістра за спеціалізацією «Комп’ютерні науки» КНТЕУ на тему: «Особливості Reverse Engineering як інструмента аналізу та імплементації сторонніх мікросервісів», 2021 р.

Практична значущість роботи полягає в отриманні навичок аналізу веб-додатків, створення веб-скраперів а також в обізнаності щодо методів захисту від веб-скрапінгу та їх обходження. Результати роботи можуть бути використані у якості основи для сервісу веб-скрапінгу.

РОЗДІЛ 1.
ТЕОРЕТИЧНІ АСПЕКТИ REVERSE ENGINEERING ТА WEB-SCRAPING У КОНТЕКСТІ ВЕБ-РОЗРОБКИ

1.1. Поняття та можливості reverse engineering

Зворотна інженерія (також відома як Reverse Engineering) - це процес або метод, при застосуванні яких намагаються зрозуміти за допомогою дедуктивних міркувань як пристрій, процес, система або частина програмного забезпечення виконує завдання з дуже слабким розумінням (якщо воно взагалі є) того, як саме вона виконує завдання. В контексті IT, даний метод представляє собою певний паттерн, сукупність дій та інформації на яку слід звернути увагу, що підказує як саме можна досягти необхідного результату.

Зворотне проектування має широкий спектр задач в різних галузях. Reverse engineering бере свій початок з аналізу обладнання суперників для досягнення комерційної або військової переваги. Однак процес зворотного проектування, як такий, не стосується створення копії. Це лише аналіз для виведення конструктивних особливостей продуктів[1]. Зворотна інженерія застосовується в галузях обчислювальної техніки, машинобудування, проектування, електронної техніки, програмної інженерії, хімічної інженерії та біології.

Зворотна інженерія цінна для з’ясування причини поломки, причини чому щось не вдалося. Якщо літак виходить з ладу, механікам, можливо, доведеться його розібрати або вивчити конструкції, щоб з’ясувати причину. Дослідження машини за допомогою зворотної інженерії може привести доказ пошкодження деталей або недоліків конструкції, які раніше не були помічені. Процес, коли беруть одне ціле і зводять його до взаємопов’язаних складових частин – можна порівняти з детективним розслідуванням.

Найбільш поширеним використанням зворотної інженерії є заміна застарілих деталей, що передбачає вивчення та відтворення певних деталей

великих машин, щоб вони працювали. Якщо у компанії є стара, велика конвеєрна система, яка постійно руйнується, а виробник не працює, компанія може зробити одне з двох: по-перше, інвестувати в нову конвеєрну систему та зупинити виробництво на тижні під час монтажу; або, по-друге, вони можуть здійснити зворотне проектування процесу та замінити деталі або деталі, які неодноразово руйнуються, заощаджуючи купу грошей і часу. Даний прецедент використання в контексті розробки програмного забезпечення працює за ідентичним сценарієм, тільки по відношенню до програмного коду [2].

Диференціювання задач процесу, що є ключовим у даній роботі, обмальовує спрощену загальну картину того, чим є дане явище у найрізноманітніших сферах життєдіяльності людини. Однак в контексті інформаційних технологій дана дедуктивна методологія має певні особливості. Загалом, вирізняють різні 2 ситуації зворотної інженерії: коли доступ до програмного коду додатку є, і коли його немає.

Перший випадок відкриває надійний, потужний інструмент під назвою Debugging (з англ. – покрокове виконання коду з відображенням усіх локальних та глобальних змінних з паузами), яким можна користуватись у будь-якому сучасному редакторі коду, наприклад PyCharm, VSCode, IntelliJ IDEA, Eclipse, WebStorm тощо, і надає можливість напряму скласти ланцюг взаємозв’язків додатку а також власноруч крок за кроком продивитись точний процес виконання функціоналу додатку.

Другий випадок передбачає собою використання різних сервісів для аналізу запитів мережі та дещо більших дедуктивних зусиль для дослідження роботи стороннього API. Існує широкий спектр додатків для імітування або створення реальних запитів, такі як PostMan, Apache JMeter, проте для аналізу роботи веб-сервісів як правило вистачає потужного браузерного аналізатора – ChromeDevTools.

1.2. Сутність Web-scraping

Web-scraping - це процес автоматизованого збору структурованих веб - даних. Деякі з основних випадків використання web-scraping включають моніторинг та аналіз цін, моніторинг новин, генерацію потенційних клієнтів, автоматизацію бізнесу, моніторинг мінімальної рекламованої ціни і дослідження ринку серед багатьох інших. Загалом, вилучення веб -даних використовується людьми та підприємствами, які хочуть використовувати величезну кількість загальнодоступних веб -даних для прийняття ефективних рішень.

Метод роботи веб-скраперів за своєю суттю нагадує копіювання інформації з веб-сайту власноруч, тільки у надзвичайно великих масштабах. На відміну від повсякденного, гальмуючого розум процесу вилучення даних вручну, веб-скрапінг використовує інтелектуальну автоматизацію для отримання сотень, мільйонів або навіть мільярдів точок даних із, здавалося б, нескінченного кордону Інтернету.

Web-scraping є популярним, оскільки дана методологія має безцінну прерогативу у вигляді надходження структурованих даних з будь-якого загальнодоступного веб-сайту або їх сукупності. Справжня сила веб-скрапінгу даних полягає не тільки у своїй сучасній зручності, але й у його здатності створювати та використовувати найбільш революційні бізнес-додатки у світі [4].

Парсер - це спеціалізований інструмент, призначений для точного та швидкого вилучення даних із веб-сторінки. Веб-скрапери сильно відрізняються за дизайном і складністю, залежно від проекту. Важливою частиною кожного скрапера є локатори даних (або селектори), які використовуються для пошуку даних, які потрібно витягти з файлу HTML - зазвичай застосовуються спеціальні бібліотеки для парсингу даних (для різних мов програмування відповідно різні бібліотеки), регулярні вирази (найшвидший проте потребуючий певної ерудиції метод) або їх комбінація. Також, є альтернативні варіанти отримання даних з веб-ресурсів – окрім

HTML-сторінки з даними, веб-сайт може мати окремі запити до власного API, якими можна користуватися й стороннім особам при належному використанні reverse engineering. Зазвичай це XHR-запити або AJAX-запити.

Однак даний інструментарій у значній мірі може заважати роботі сайту або взагалі через використання веб-скраперів прибуткова компанія може понести збитки. Розповсюдженим прикладом є атаки ботів-скраперів, щоб збирати та використовувати контент із веб-сайту — наприклад, для повторної публікації вмісту без накладних витрат або автоматичного зниження ціни. Як інший приклад, Інтернет-магазини часто наймають професійних розробників веб-скраперів або використовують відповідні інструменти для збору конкурентної інформації для розробки майбутніх стратегій роздрібного ціноутворення та каталогів продуктів.

Підраховано, що підприємства електронної комерції втрачають 2% прибутку в Інтернеті через веб-скрейпінгу [7]. У 2019 році глобальні продажі електронної комерції становили 3,53 трильйони доларів, це становить понад 70 мільярдів доларів [8].

Тому розповсюдженою практикою також стало використання різних методів захисту від веб-скрапінгу. Загальні стратегії захисту від сканування даних включають:

- Моніторинг нових або наявних облікових записів користувачів з високим рівнем активності та без покупок;
- Виявлення аномально великої кількості переглядів товару як ознаки нелюдської діяльності;
- Відстеження активності конкурентів за ознаками відповідності ціни та каталогу продукції;
- Дотримання положень та умов сайту, які зупиняють шкідливий веб-скрапінг;

- Використання можливостей захисту від ботів із глибоким аналізом поведінки, щоб виявити поганих ботів та запобігти веб-скрапінг в Інтернеті, таких методів особливо багато.

Прикладами сервісів захисту від ботів які поєднують усі ці методи є CloudFlare, DataDome та інші. Проте якісні та складні скрапери зроблять все можливе, щоб замаскувати себе за справжніх користувачів. Хороше рішення для виявлення ботів або рішення анти-краулер зможе визначати поведінку відвідувачів, яка демонструє ознаки веб-скрапінгу, в режимі реального часу, і автоматично блокувати шкідливих ботів, перш ніж робота ботів розкриється, зберігаючи безперебійну роботу для реальних користувачів. Щоб правильно ідентифікувати шахрайський трафік та інструменти для блокування веб-скрапінгу, рішення захисту від ботів має бути здатним аналізувати як технічні, так і поведінкові дані [6]. До технічних даних можна віднести аналіз HTTP-headers, тоді тоді як до поведінкових можна віднести аналіз частоти запитів з одної IP-адреси.

Еталоном захисту від парсингу є будь-яке онлайн-казино або сайт ставок на спорт, де від ступеню захищеності напряму залежить прибуток компанії. Конкретно на прикладі сайтів ставок: при низькому рівні захисту людина, що професійно займається розробкою веб-скраперів, може автоматизувати збір та аналіз інформації про ставки з декількох таких сайтів та, наприклад, знаходити так звані «букмекерські виделки» й автоматично робити безпрограшні ставки – така поведінка є надзвичайно збитковою для сайтів з подібним родом діяльності.

1.3. Явище асинхронності у веб-розробці

Центральна частина комп’ютера, частина, яка виконує окремі кроки, що складають наші програми, називається процесором. Програми, які ми бачили досі, - це речі, які будуть тримати процесор зайнятим, поки він не завершить свою роботу. Швидкість, з якою може виконуватися щось на зразок циклу, який маніпулює числами, майже повністю залежить від

швидкості процесора.

Але багато програм взаємодіють з речами за межами процесора. Наприклад, вони можуть спілкуватися через комп’ютерну мережу або запитувати дані з жорсткого диска, що набагато повільніше, ніж отримання їх з пам’яті.

Коли таке трапляється, було б соромно залишити процесор бездіяльним — тим часом він міг би виконати якусь іншу роботу. Частково цим займається ваша операційна система, яка перемикає процесор між кількома запущеними програмами. Але це не допомагає, коли ми хочемо, щоб одна програма могла прогресувати, поки вона очікує запиту від мережі.

Процесори можуть виконувати програми асинхронно. Асинхронність, багатопоточність та паралелізм — дуже важливі аспекти програмування, без яких важко уявити сучасні комп'ютери. Для розуміння як працює асинхронність насамперед важливо розуміти **як працює процесор**.

За одиницю часу (вона називається tick, з англ. «мить». Надалі буде використовуватись у тексті як «тік») процесор може виконати лише одне завдання - наприклад, прочитати значення комірки пам'яті. Тому на те, щоб виконати наступні операції, знадобиться 4 тіки:

1. Прочитати дані із двох комірок.
2. Скласти їх.
3. Записати результат до іншої комірки.

Це стосується лише атомарних операцій, тобто найменших та неподільних. Більш складні завдання можуть складатися із кількох атомарних. Наприклад, щоб провести множення числа A на число B, потрібно буде додати до A саме себе B - 1 раз. 5 * 5 = 5 + 5 + 5 + 5 + 5.

Ще більше тіків потрібно на операцію ділення, а якщо операцію потрібно провести над числами з плаваючою комою, то їх потрібно буде навіть ще більше.

Уявляючи повсякденний контекст використання комп’ютера, у людини може бути відкрито десяток вкладок у браузері, плеєр, месенджер, редактор

коду та ще багато всього. Тому дивно, що нічого з цього насправді не працює одночасно. Незважаючи на те, що процесор виконує лише одне завдання за один раз, інженери та програмісти знайшли спосіб розподіляти час його роботи так, щоб він витрачав трохи часу на одне завдання, потім перемикався на інше і так далі. У цьому їм допомогло те, що за одну секунду процесор може виконувати величезну кількість операцій, тому людина не помічає, що вони виконуються по черзі. Кількість тіків вимірюється в герцах (Гц) - це одиниця виміру частоти перебігу періодичних процесів.

Наприклад, якщо повз будинок раз на секунду проїжджає гоночний автомобіль, то його частота дорівнюватиме 1 Гц. Якщо автомоболіль проїжджає двічі на секунду, його частота — 2 Гц; Процесор так швидко виконує процеси, що його частота вимірюється у гігагерцах. 1 ГГц = 1000000000 Гц. Частота сучасних процесорів зазвичай дорівнює 2-4 ГГц. Як процесор виконує програми?

Кожна програма складається з багатьох процесів: наприклад потрібно 500 разів провести додавання, записати дані в 2000 комірок, перемножити їх, а потім, нарешті, поділити. Якщо виконувати увесь перелік лінійно, програми стануть дуже незручними. Наприклад, при натисканні на кнопку скачування в браузері, комп'ютер буде блокуватися до тих пір, поки скачування не завершиться. Добре, якщо хоча б буде видно, скільки відсотків завантажилося. Тоді код програми виглядатиме приблизно так: поки виконується цей цикл, людина за комп’ютером не зможе зробити нічого такого, що не прописано всередині циклу.

Тому, щоб комп'ютерами було зручно користуватися, всі програми діляться на потоки. Припустимо, запущено 10 програм, а процесор працює на швидкості 100 тіків на секунду. Тоді кожен потік отримає по 10 тіків. Тобто процесор протягом 10 тіків виконуватиме інструкції від одного потоку, а потім буде переходити до інструкцій іншого — і так по колу.

Також кожен потік має пріоритет: важливіші програми отримуватимуть більше тіків. В операційній системі кожна запущена

програма виконується в окремому потоці. Це відбувається автоматично, тому що так спроектована ОС. Проте при написанні власної програми також є можливість створювати нові потоки: це дозволить робити програми зручніше для користування [9], а у випадках де є програма взаємодіє з великою кількістю даних , як у випадку з веб-скрапінгом, – швидше. Це і є поняттям асинхронного програмування — це форма паралельного програмування, яка дозволяє одиниці роботи виконуватися окремо від основного потоку програми.

Найчастіше асинхронність потрібна у програмах із графічним інтерфейсом, у програмах які роблять запити до сторонніх сервісів (таких як бази даних, веб-API та інші), та при роботі з великим об’ємом даних. У програмах із графічним інтерфейсом, наприклад, основна логіка та робота із зображенням поміщені у різні потоки. Тому, коли логіка зайнята, все ще є можливість використання програми. Якщоусе перелічене виконується в одному потоці, то додаток буде «підвисати», коли виконується складна інструкція. В ОС Windows часто можна помітити, що коли програма щось робить, при кліку на неї в заголовку вікна можна побачити словосполучення "Не відповідає". Це не завжди означає, що програма зависла. Можливо, вона просто зайнята вирішенням якогось складного завдання, яке виконується в тому ж потоці [9].

РОЗДІЛ 2.
ОГЛЯД МЕТОДИК REVERSE ENGINEERING ТА РОЗРОБКИ ВЕБ-СКРАПЕРІВ

2.1. Інструментарій для аналізу роботи веб-сайтів та додатків – Chrome DevTools

DevTools – це набір інструментів веб-розробника, які вбудовані безпосередньо в браузер Google Chrome. DevTools дозволяє переглядати та змінювати / маніпулювати DOM (Document Object Model – блочна модель HTML-документу), змінювати стилі сторінки (CSS) у середовищі попереднього перегляду та працювати з JavaScript кодом сайту, дозволяючи налагоджувати, переглядати повідомлення, запускати код JavaScript, відстежувати усі запити, файли локального сховища (localStorage, sessionStorage, cookies), статистику, метрику використання пам'яті та багато іншого. Сучасні браузери, Safari, Firefox, Microsoft Edge, Chrome, Яндекс та інші мають вбудовані інструменти розробника, що дозволяють переглянути вихідний код сайту. Окремо встановлювати їх не потрібно.

Прерогативою DevTools є набір великої кількісті інструментів, корисний для широкого спектру IT-спеціальностей:

1) Використовується веб-розробниками у процесі верстки сторінки, дозволяючи переглядати та редагувати HTML теги та атрибути сторінки а також проводити різноманітні маніпуляції із CSS-стилями сторінки. Надає можливість перевіряти зовнішній вигляд веб-сайту імітуючи розширення екранів різних девайсів, таких як смартфони, планшети, ноутбуки та інших;
2) Надає доступ до консолі JavaScript, що у свою чергу надає надзвичайно широкі можливості взаємодії із веб-сайтом, що відкриває дороги до тестування, зворотнього інжинірингу, маніпуляцій з будь-якими елементами сторінки та запитами на будь-

яке API, автоматично надає інформацію про помилки стосовно Front-end сторони коду та результатів веб-запитів, тощо;

3) Надає можливість переглядати та локально змінювати усі скомпільовані Front-end ресурси та код веб-сайту (при оновленні сторінки усі зміни пропадуть; зроблені зміни на сервер сайту ніяк не перенесуться), що відкриває великий асортимент можливостей reverse engineering а також є головним інструментом при аналізі проблем, оскільки DevTools надає також свій власний дебагер (інструмент, що дозволяє виконувати код покроково та переглядати усі глобальні та локальні змінні на момент виконання коду);
4) Вкладка Network, що є основним місцем для аналізу роботи веб-додатку, надає інформацію про мережеві запити сторінки (список web-ресурсів, що завантажуються сторінкою для відображення контенту, що демонструється). Містить інформацію про необхідні контракти з API додатку, HTTP-заголовки, метод та відповідь (результат) запиту, тощо;
5) Дозволяє маніпулювати з файлами локального сховища таких як cookies, localStorage, sessionStorage, а також дозволяє спостерігати за продуктивністю роботи сайту.

У правому куті розміщеної нижче ілюстрації (Рис. 2.1) можна побачити структуру веб-сторінки та стилі, застосовані до поточного елемента.

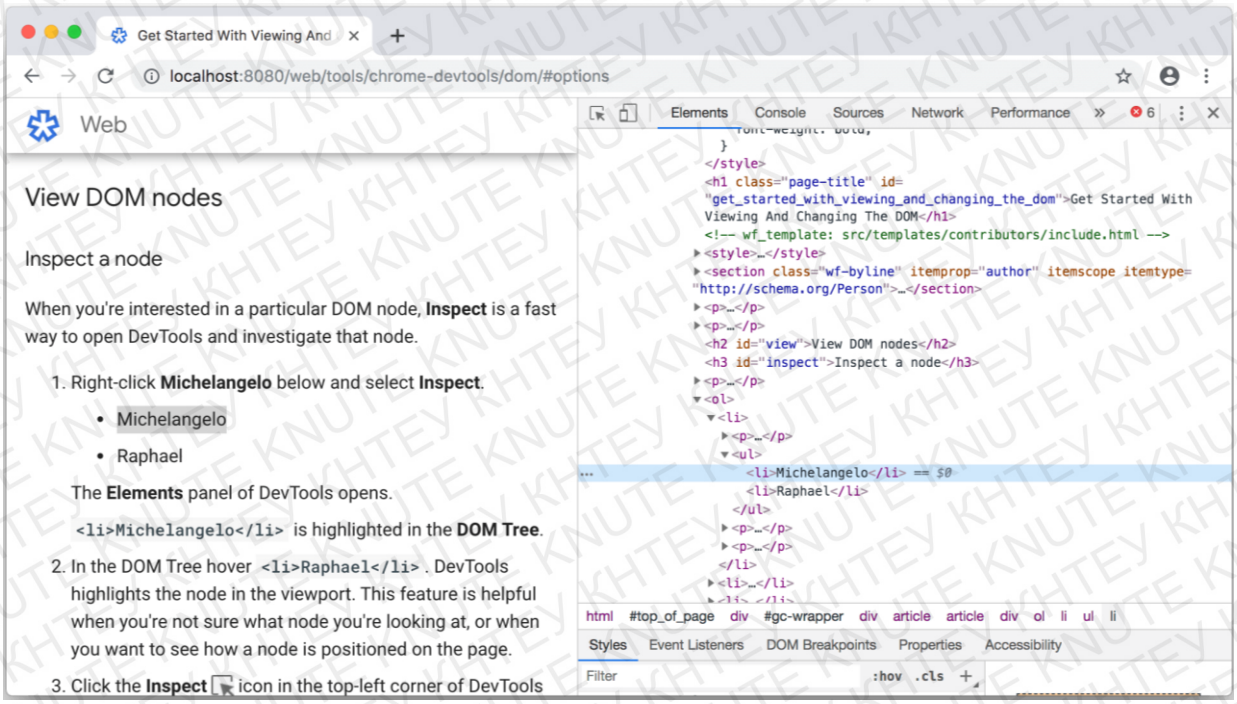


Рис. 2.1. ChromeDevTools

Типова веб-сторінка є текстовим файлом у форматі HTML, який визначає структуру та контент сторінки, а також може містити посилання на файли в інших форматах (текст, графічні зображення, відео, аудіо, бази даних та ін), а також гіперпосилання для швидкого переходу на інші веб-сторінки або доступу до посилальних файлів.

Декілька веб-сторінок, об'єднаних спільною темою та дизайном, а також пов'язаних між собою посиланнями, утворюють веб-сайт. При цьому сторінки, що створюють веб-сайт, можуть знаходитися на одному або декількох веб-серверах, які можуть розташовуватися в одному дата-центрі або віддалено один від одного, часто в різних країнах.

HTML (HyperText Markup Language) - це скелет веб-сторінки. Для того, щоб HTML-сторінка мала більше функціоналу потрібен Javascript (календарі, меню, спливаючі вікна, анімація та інше, робиться за допомогою JS). Для надання сторінці гарного та зручного для сприйняття вигляду знадобиться CSS (каскадні таблиці стилів). Даний перелік можна переглянути а також змінювати за допомогою DevTools. Це корисно у першу чергу при розробці, оскільки зміни застосовуються моментально й дає можливість наглядно

побачити внесені зміни й допомагає зрозуміти, чи є обране рішення релевантним [10]. З точки зору аналізу роботи веб-додатку, доступ до даних ресурсів допомагає дослідити елементи сторінки, у які записується інформація з серверу, наприклад які атрибути HTML-елемента потрібні для відображення даних у цьому елементі, чи коректно прийшли дані з серверу, чи правильно сформувалися динамічно сформовані гіперпосилання на сторонні ресурси чи путь до файлів медіа, тощо.

Методом аналізу та редагування отриманої сторінки веб-сайту, можна проводити різноманітні маніпуляції:

- Наприклад, можна обходити слабкі рівні захисту на стороні Front-end такі як наприклад захист від копіювання, блокатори контенту (необхідно визначити в коді блок-елемент, що закриває потрібну ділянку контенту й видалити елемент зі сторінки (Рис. 2.2)), розблокування форм введення;
- Скачувати та отримувати доступні ресурси – медіа файли, контент веб-ресурсу, є можливість скопіювати усю сторінку разом із стилями та кодом JavaScript (хоча функції що зав’язані на взаємодії з сервером працювати не будуть);

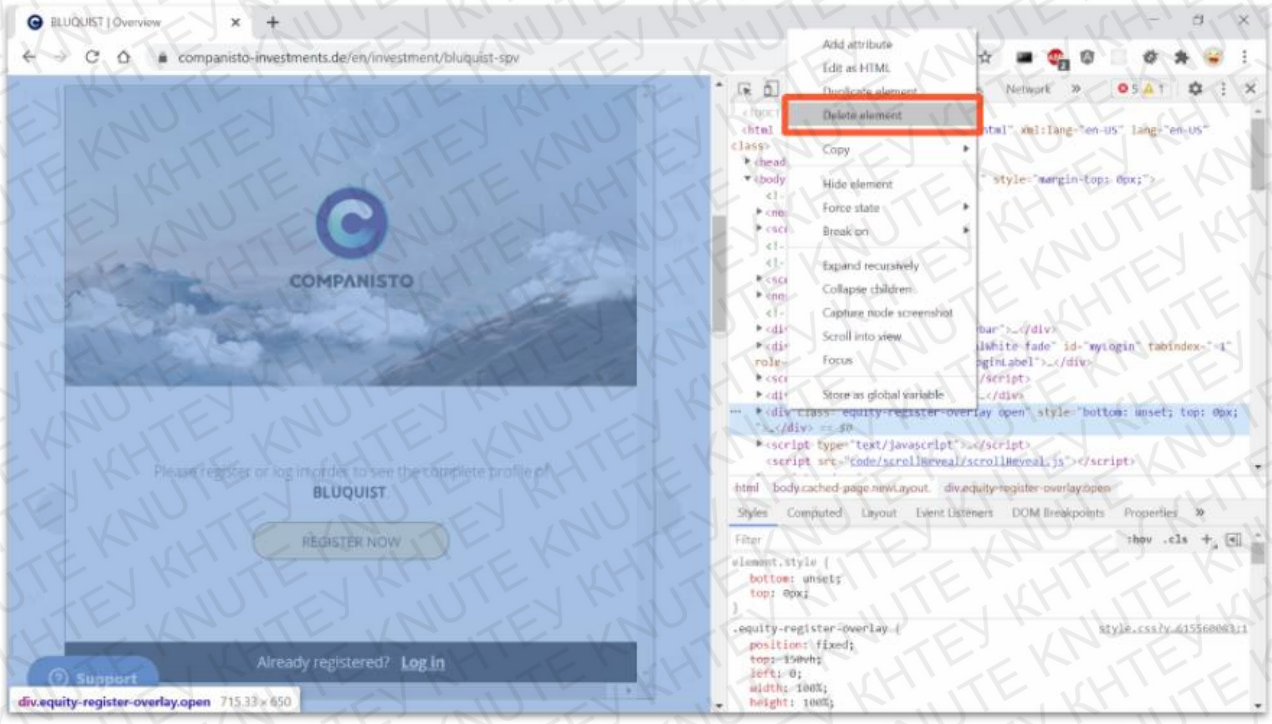


Рис. 2.2. Видалення блокатору контенту

2.2. Reverse engineering як засіб виявлення проблем з додатком – debug tools

Reverse engineering використовується не тільки як аналіз додатку з точки зору розуміння механізмів його роботи для подальшого використання, а й для налагодження виникаючих проблем. У веб-розробці стандартною практикою є праця з чужими розробками та кодом, написаним іншою компанією, тому при інтеграції власних розробок нормальним явищем є часте виникнення помилок. Саме для спрощення даного виду роботи є інструменти Dubigging . **Debug** (з англ. «налагодження») — це процес виявлення та видалення наявних і потенційних помилок (також званих «багами» – помилками) у програмному коді, які можуть призвести до несподіваної поведінки або збою. Щоб запобігти некоректній роботі програмного забезпечення або системи, дебагінг використовується для пошуку та усунення помилок або дефектів.

Дебагінг на стороні Front-end (на клієнтській частині) виконує ChromeDevTools, з можливістю ставити **breakpoints** (з англ. «точки

перелому» або «точки зупинки» – місця у яких зупиняється виконання коду для покрокового перегляду усіх локальних та глобальних змінних).

Найвідомішим типом точки зупинки є line-of-code – обрати необхідну строку коду. Але точки зупинки line-of-code можуть бути неефективними, особливо якщо точно не відомо де шукати помилку, або якщо це робота з громіздким кодом. Можна заощадити час під час налагодження, знаючи, як і коли використовувати інші типи точок зупинки [11]:

Тип брейкпоінту	*Доцільне використання, коли необхідно зробити паузу...*
Line-of-code	На точну область коду.
Conditional line-of-code	У точній області коду, але тільки тоді, коли виконується інша умова.
DOM	У коді, який змінює або видаляє певний вузол DOM або його дочірні елементи.
XHR	Якщо URL-адреса XHR містить рядковий шаблон.
Event listener	У коді, який запускається після події, наприклад кліку, запускається.
Exception	У рядку коду, який створює перехоплену або неперехоплену помилку (вийняток).
Function	Щоразу, коли викликається конкретна функція

Таблиця 2.1. Типи брейкпоінтів

Line-of-code ставиться, коли відома точна область коду яку потрібно дослідити. DevTools завжди призупиняється перед виконанням цього рядка коду.

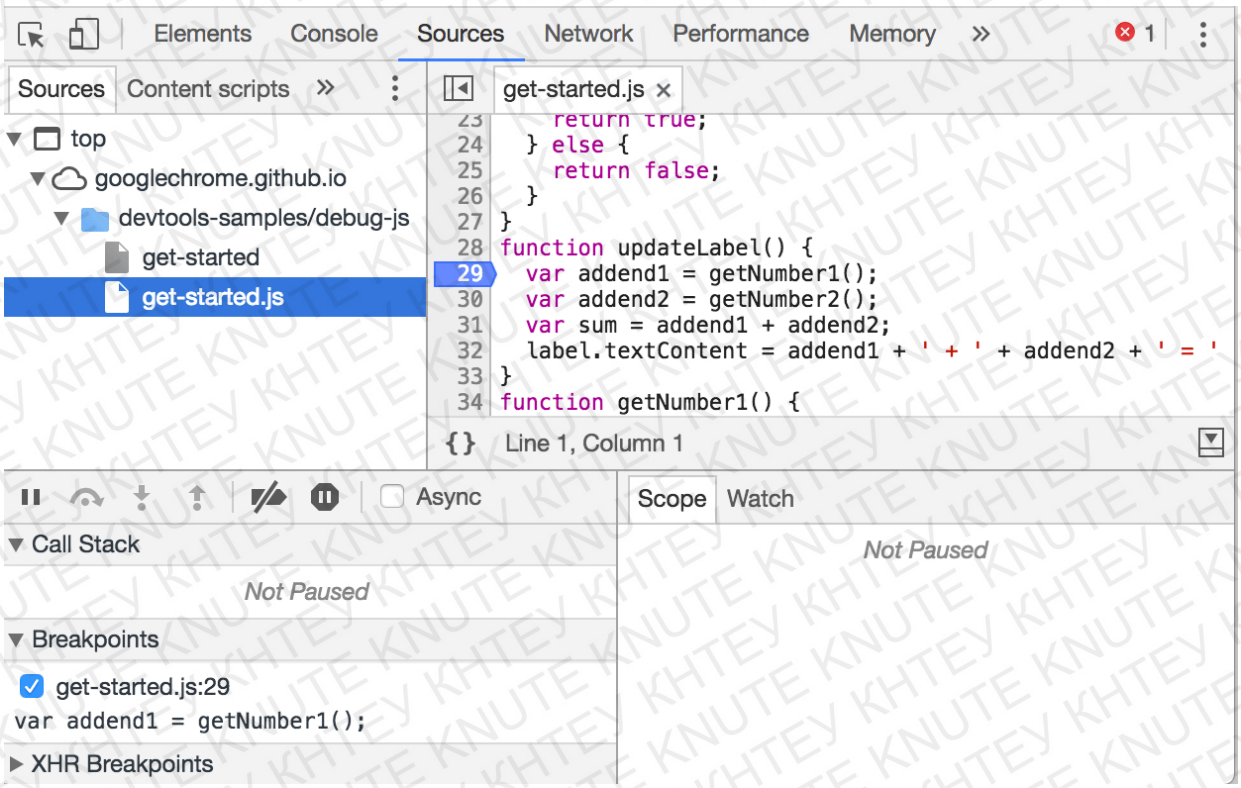


Рис. 2.3. Line-of-code breakpoint

Також, у JavaScript є можливість викликати дебагер із власного коду щоб створити breakpoint – викликати у коді debugger. Це еквівалентно точці зупинки рядка коду (Line-of-code), за винятком того що точка зупинки встановлюється у власному файлі с кодом, а не в інтерфейсі DevTools.

Conditional line-of-code (з англ. «умовна точка зупинки рядка коду») – те самое що й Line-of-code, але коли потрібно призупинити процес виконання коду лише коли виконується інша умова.

Проводячи навігацію по DOM-дереву, можна ставити breakpoints для HTML-елементів (Рис.2.4), існують наступні типи:

- Модифікації піддерева (Subtree modifications). Запускається, коли дочірній вузол вибраного в даний момент видаляється або додається, або змінюється вміст дочірнього вузла. Не запускається при змінах атрибутів дочірнього вузла або будь-яких змінах у поточному вибраному вузлі.

- Зміни атрибутів (Attributes modifications): ініціюється, коли атрибут додається або видаляється на поточному вибраному вузлі або коли змінюється значення атрибута.
- Видалення вузла (Node Removal): ініціюється, коли вибраний на даний момент вузол видаляється [11].

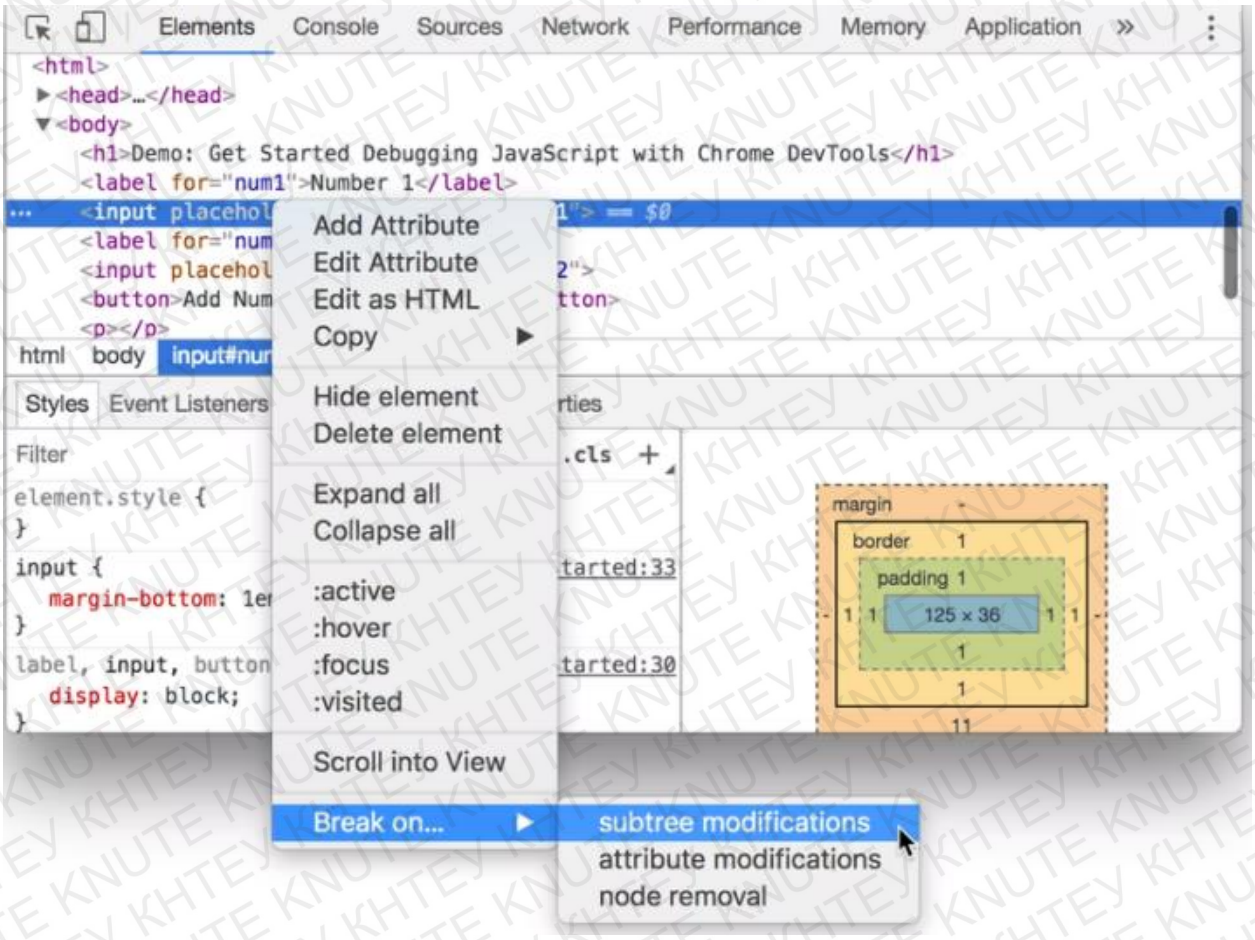


Рис.2.4. Доступні типи брейкпоінтів для DOM

Точка зупинки XHR використовується, коли URL-адреса запиту XHR містить вказаний рядок. DevTools зупиняється в рядку коду, де XHR викликає send() (Рис. 2.5). Один із прикладів того, коли це корисно, — коли видно, що сторінка запитує неправильну URL-адресу, і ви потрібно швидко знайти вихідний код AJAX- або Fetch-запиту, який спричиняє неправильний запит.

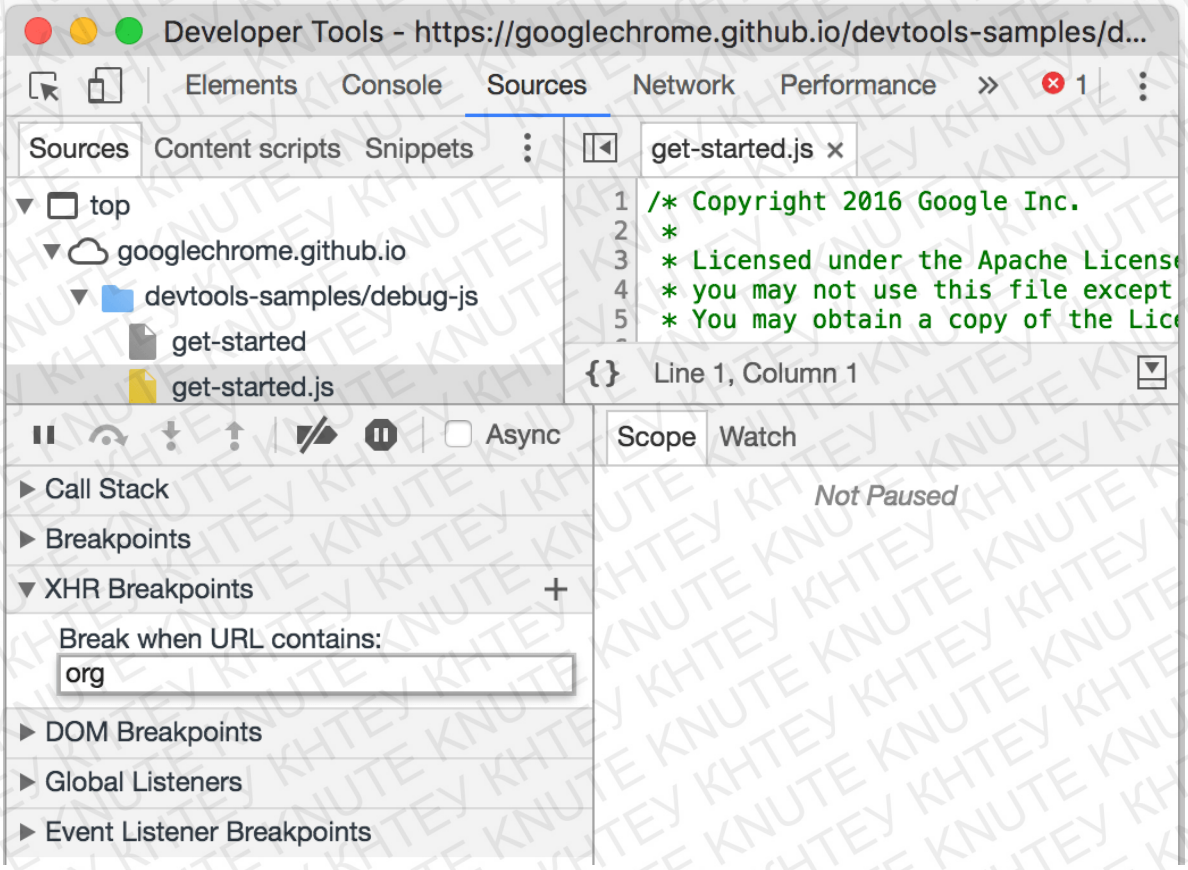


Рис. 2.5. XHR точка зупинки

На сайті може відбуватися велика кількість найрізноманітніших подій, як визначених за замовчуванням у JavaScript так і створених власноруч функцій які прив’язані до певного елементу та викликаються на певну взаємодію користувача з ним.

Точки зупинки Event listener (а англ. «прослуховувача подій») доцільно використовувати тоді, коли потрібно призупинити код прослуховування подій, який запускається після запуску події. Можна вибрати конкретні події, наприклад клік миші, або категорії подій, як наприклад усі події миші. DevTools показує список категорій подій, таких як анімація та інші (Рис. 2.6).

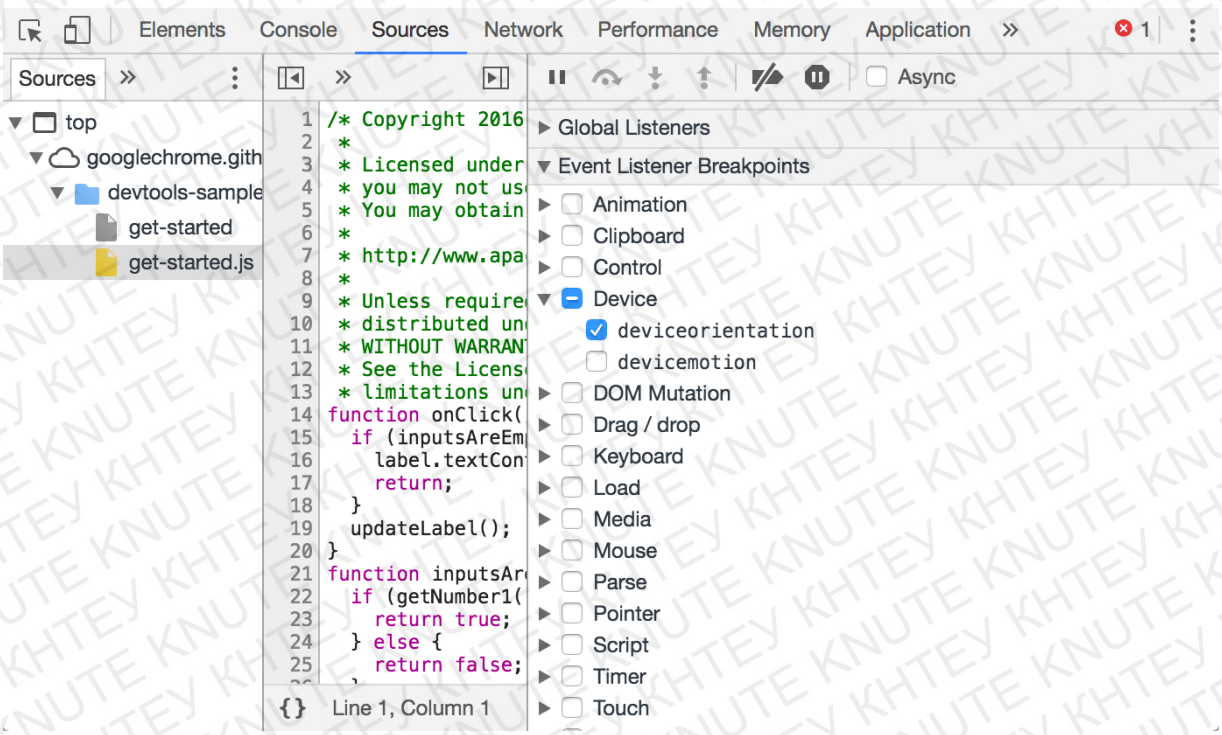


Рис. 2.6. Брейкпоінт прослуховувача подій

Для призупинення коду у разі виникненні помилок можна включити необхідний прапорець в інтерфейсі ChromeDevTools.

У JavaScript є метод debug(), у котрий можна передати в якості аргумента функцію, яку необхідно налагодити.

Увесь перелік методів дебагінгу використовується як складова reverse engineering, оскільки має значний вклад у знаходженні кореня проблем. Також використання даних інструментів надає широкі можливості для аналізу невідомого додатку – у даному випадку, використовуючи тільки Front-end інструмент налагодження.

2.3. Reverse engineering як методологія дослідження механізмів роботи додатків

Не дивлячись на прерогативи методів налагодження помилок та використання точок зупинки, найважливіший сегмент корисної для reverse engineering інформації та надзвичайно потужний інструмент для використання методології зворотної інженерії є вкладка Network у ChromeDevTools (Рис. 2.7).

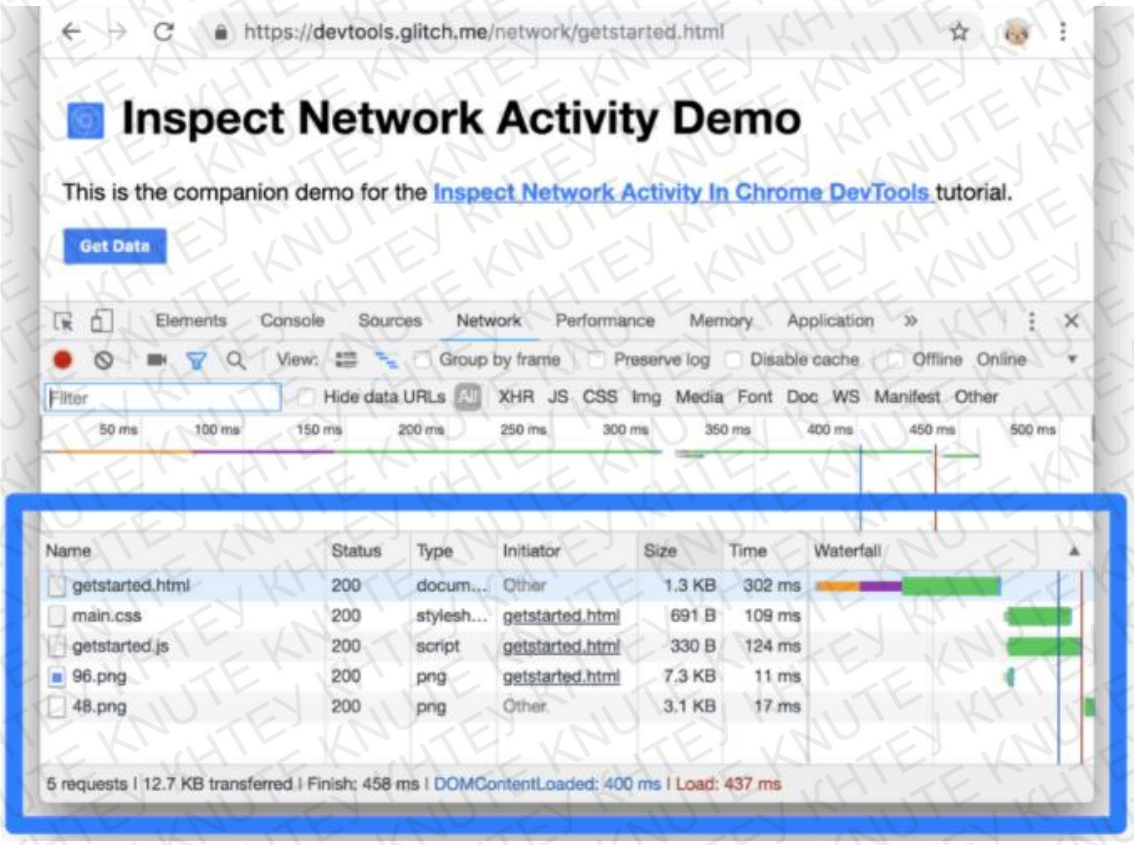


Рис.2.7. Журнал мережі

Кожен рядок Мережевого журналу представляє ресурс. За замовчуванням ресурси перелічені в хронологічному порядку. Найпершим ресурсом, як правило, є основний документ HTML. Нижній ресурс - це те, що було запитано останнім.

Кожен стовпець представляє інформацію про ресурс. На рисунку 2.8 у таблиці запитів за замовчуванням відображаються такі стовпці:

- **Ім'я**. Назва файлу або ідентифікатор ресурсу.
- **Статус**. Код стану HTTP протоколу.
- **Тип**. Тип MIME запитуваного ресурсу.
- **Ініціатор**. Наступні об’єкти або процеси можуть ініціювати запити:
    - **Парсер**. Синтаксичний аналізатор HTML Chrome.
    - **Переспрямування**. Перенаправлення HTTP.
    - **Скрипт**. Функція JavaScript.

- **Інший**. Якийсь інший процес або дія, наприклад, перехід на сторінку за посиланням або введення URL-адреси в адресний рядок.

- **Розмір**. Комбінований розмір заголовків відповідей плюс тіло відповіді, надане сервером.
- **Час**. Загальна тривалість, від початку запиту до отримання останнього байта у відповіді.
- **Хронологія життєвого циклу запиту**. Візуальна розбивка діяльності кожного запиту [3].

У якості прикладу використання Reverse Engineering для подальшої імплементації стороннього веб-сервісу буде проаналізовано використання сервісу пошуку патентів Google Patents. На рисунку 2.8 зображено знімок екрану з виконаним патентним пошуком а також відкритим ChromeDevTools.

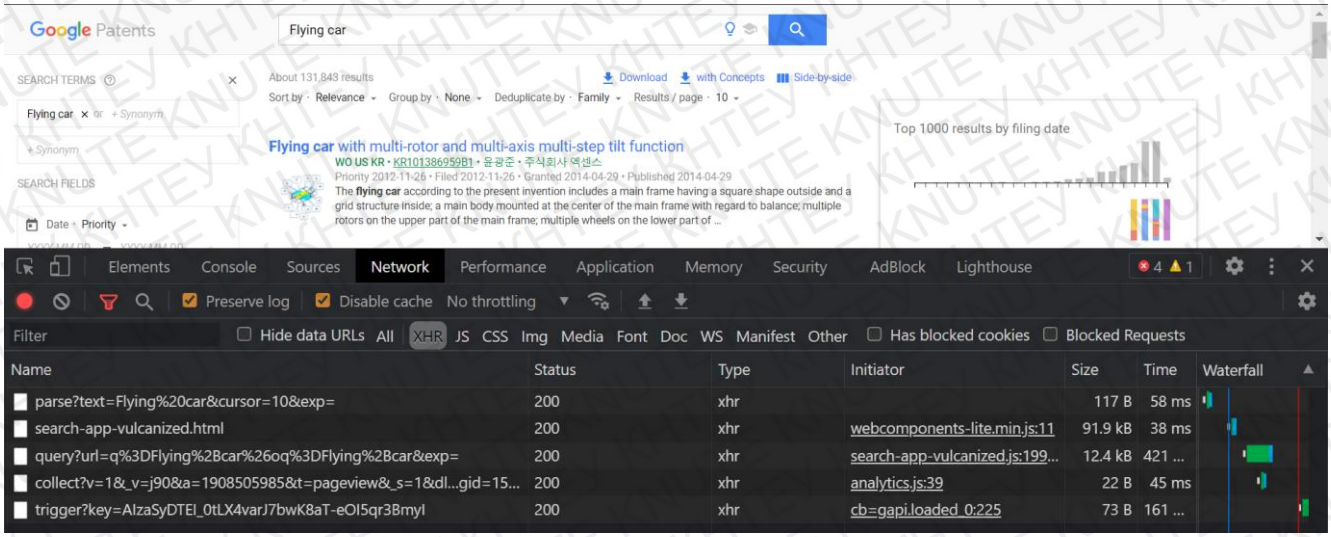


Рис.2.8. Знімок екрану під час пошуку по патентам з відкритим ChromeDevTools

Хорошою практикою є розробка певного паттерну, списку інформативних речей на які можна звернути увагу в першу чергу:

- **Network**:
  - **MIME-type:** майже у будь-якому випадку, необхідний тип запиту буде **xhr** (AJAX-запити, запити на API) або **document** (HTML-

сторінка). Саме ці два типи (у переважній більшості перший) містять серіалізовану інформацію яка не потребує обробки, або відповідно всю HTML-сторінку яку необхідно «відфільтрувати» та «дістати» з неї дані у такому вигляді щоб робота з ними була зручною.

- **Name:** при фільтруванні по xhr запитам дуже зручно придивлятися на усі імена запитів – url з наявними у ній url-параметрами – як правило імена запитів розробляють семантично вірними, тому наявна цілком велика вірогідність того що необхідний для аналізу запит буде мати інтуїтивно зрозумілу назву.
- **Подробиці конкретного запиту:**
  - **Вкладка "Preview":** Містить відповідь від сервера**.** Як правило, саме по змісту цієї вкладки й визначається, чи є даний запит тим який віддає з API інформацію в чистому вигляді. Для підтвердження, зазвичай потрібно власноруч порівняти кілька результатів що відображені у відповіді запиту і результати, що відображені на web-сторінці. При співпадінні результатів – запит, що розглядається, і є шуканим.
  - **URL-параметри**: коли шуканий запит знайдено, і він не є HTML-документом, необхідні url-параметри й можуть бути основним механізмом контролю над результатами – як правило це фільтри та \ або точна інформація, введена користувачем власноруч у форму на сайті.
  - **Request Headers:** якщо контроль над результатами не виконується за допомогою попереднього пункту, то як правило це виконується великою кількістю налаштованих параметрів запиту: API стороннього серверу може очікувати як сгенерований файл cookie (що зазвичай передбачає обов'язкову авторизацію на сайті де використовується

досліджуване API), так і динамічно сгенеровані зашифровані ключі які перевіряють використовувався браузер при запиті чи це запрограмований сервер родить автоматичні запити й блокує такі;

- **URL-рядок:** у випадках, коли усі попередні пункти не пройшли, є вихід у вигляді створення обробника веб-сторінки – парсера або веб-скрапера, що буде робити GET-запит на сторінку з результатами, що містить у собі потрібну інформацію, й витягувати її фільтруючи HTML-сторінку. У такому випадку, достатньо буде знати необхідні url-параметри щоб робити запити. Це є досить трудомістким процесом, оскільки така ситуація потребує аналізу великої кількості HTML-коду для знаходження коректного шаблону розмітки сторінки, яка містить потрібну інформацію.

Судячи по запитах у вкладці Network з фільтром по xhr-запитах, назва «query» (з англ. «запит») має велику вірогідність виявитися потрібним для отримання результатів у чистому вигляді запитом. Тому наступним кроком є аналіз відповіді даного GET-запиту (Рис. 2.9).

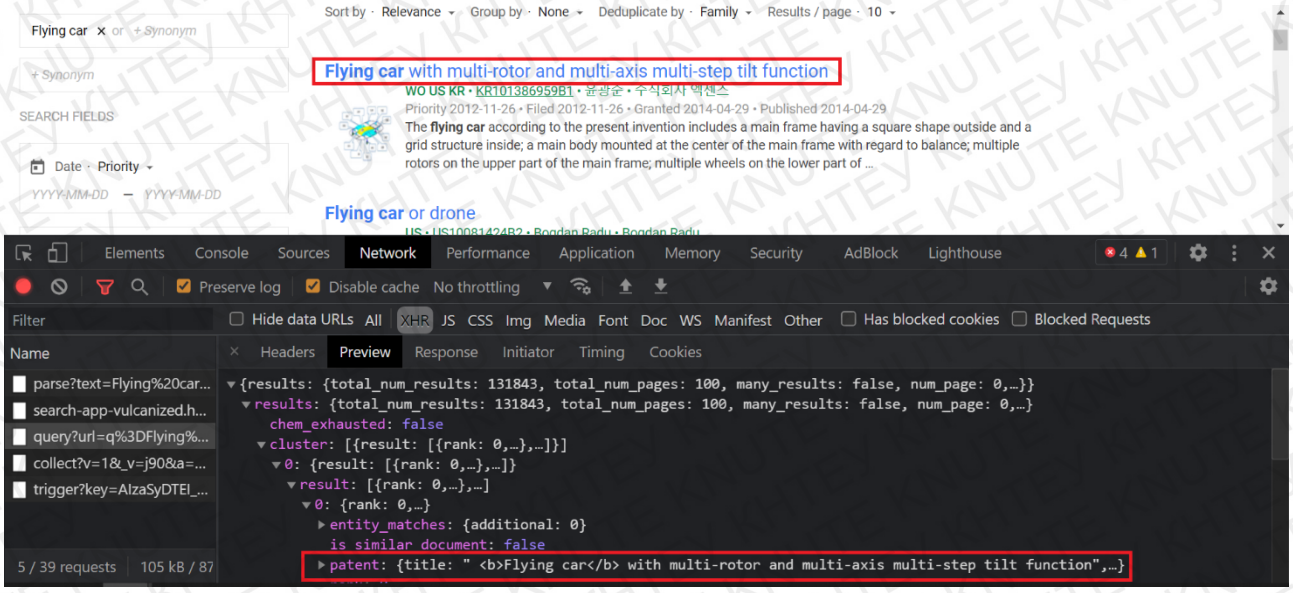


Рис.2.9. Відповідь xhr GET-запиту до API Google Patents.

Отриманий результат містить у собі достатньо великий об’єм інформації, проте при поглибленні в ієрархію об’єкта відповіді, серед

інформації виявляється також й та, що відображається на веб-сторінці. Коректна інформація знайдена.

Проте судячи з кількості об’єктів отриманих від GET-запиту досліджений запит віддає тільки 10 результатів. Для того щоб дослідити механізм отримання більшої кількості результатів, необхідно використовуючи інтерфейс веб-сайту перейти на наступну сторінку результатів. Дана маніпуляція також є сукупністю запитів, які чітко видно в журналі запитів. При порівнянні колишнього запиту з практично аналогічним що з’явився після натискання кнопки «наступна сторінка» на сайті, виявляється наступна взаємозалежність: для контролю кількістю результатів, сервіс Google Patents використовую url-параметр «page», що служить пагінацією на веб-сайті; це означає, що користуючись мовами програмування, можна створити цикл, що кожну ітерацію буде додавати до url-параметру «page» одиницю – що є інтуїтивно зрозумілим механізмом переходу на наступну сторінку з результатами. Тепер, сервіс Google Patents є мінімально досліджений і готовий для імплементації у будь-який власний додаток.

Факт того, що даний HTTP-запит є типом GET, значно облегшує використання API Google Patents, оскільки даний тип запиту не передбачає передавання серверу стороннього сервісу певних серіалізованих даних або правильно налаштованих заголовків Request Headers. Це означає, що у даному випадку достатньо буде коректно маніпулювати URL-параметрами при створенні автоматизованого запиту з серверу, у який буде імплементовано роботу Google Patents.

2.4. Асинхронність на прикладах Python

Веб-скрапінг передбачає собою використання запитів до сайту або API, що є об’єктом веб-скрапінгу. Виконання коду синхронним методом буде займати набагато більше часу, це буде особливо помітно якщо потрібно, наприклад, спарсити декілька сайтів – у такому випадку одночасно

оброблятися буде 1 сайт, і тільки після того як додаток закінчив його обробку, додаток перейде до роботи з наступним у черзі сайтом.

IT-гіганти, такі як Facebook, Twitter, фреймворк React Native, база даних RocksDB та інші, використовують асинхронність у цілях підвищення продуктивності своїх додатків.

Асинхронність у програмуванні — виконання процесу в режимі неблокування системного виклику, що дозволяє потоку програми продовжити обробку.

У мові програмування Python використовується парадигма асинхронного програмування з використанням корутин (Coroutines – з англ. «співпрограми». Слово «coroutine» складається з двох слів: «co» (кооперативний) і «routines» (підпрограми, функції))

Співпрограми, або корутини — це функції, виконання яких можна призупинити в певний момент з можливістю відновити виконання роботи з тієї ж точки пізніше, коли буде необхідно або коли Event loop не має задач на поточний момент часу [12, 13].

В основі роботи корутин лежить механізм python під назвою «генератор». Генератори дозволяють обробляти великі потоки даних. Припустимо, що є файл надзвичайно великих розмірів і є необхідність обробити та вичленувати необхідну інформацію. Малоймовірно, що локально буде достатньо місця та пам'яті на власному ПК для обробки величезного обсягу даних, якщо тільки не робити це робити це частинами. У Python обробкою великих масивів даних займаються генератори.

Генератори не обчислюють значення відразу всіх елементів, а зберігають лише останній обчислений елемент, а також умови та правило переходу до наступного. Наступне значення обчислюється лише під час виконання методу next(). Попередня інформація стирається з пам’яті. Приклад генераторного об’єкту наведено на рис. 2.10.

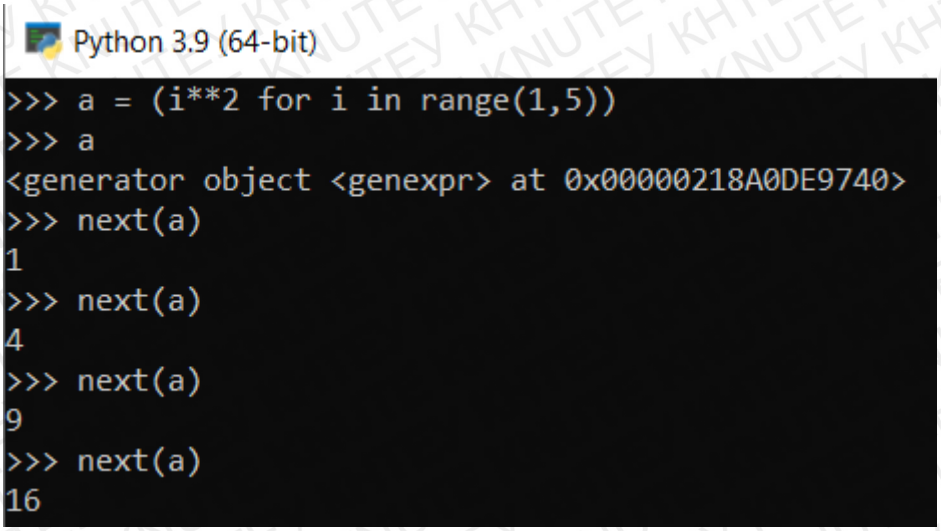


Рис. 2.10. Приклад створення generator object

При кожному виклику *next(a)* значення генератора 1, 4, 9, і 16 будуть розраховуватися по одному. Усі попередні дані будуть видалені.

Щоб не потрапити на блокування вводу-виводу при виконанні коду, слід використовувати або асинхронне програмування, або багатопоточність. Python надає можливість використання **зелених потоків** або **функцій зворотного виклику** (**callbacks**):

- **Зелені потоки** керуються програмами, а не процесорами, але мають проблеми, властиві потоковому програмуванню.
- **Зворотний виклик** використовує співпрограми, які можуть виконувати свої завдання без участі програміста, але мають труднощі налагодження та неможливість обробки винятків.

Ці проблеми здатні вирішити генератори, про які було згадано вище. Завдяки їм функції можуть повертати один елемент списку (генератору) за один раз. Виконання буде зупинятися до моменту запиту наступного елемента. У роботі генераторів теж є свої нюанси — вони залежать від тієї функції, яка їх викликає. Однак ця проблема вирішується синтаксисом *yield from*. Завдяки чому генератори здатні отримувати результати один одного, створювати винятки та підтримувати чергу викликів (callstack).

На цьому принципі була створена Asyncio — асинхронна бібліотека, де в циклі подій можуть запускатися генератори. Щоб до співпрограми був доданий генератор, необхідно додати декоратор @coroutine.

Проте, завдяки своїй універсальності та потужності бібліотека Asyncio стала основною в Python. Для універсальності, більш точного розуміння асинхронного коду та поділу методів та генераторів використовуються ключові слова *async* (показують асинхронність методу) та *await* (очікування завершення співпрограми).

Приклад (Рис. 2.11). Звернення до трьох URL-адрес одночасно з використанням бібліотеки Asyncio з використанням синтаксису async\await. Метод повертає співпрограму і вона перебуває в очікуванні.

```
import asyncio
import aiohttp


urls = ['http://www.google.com', 'http://www.yandex.ru', 'http://www.python.org']


async def call_url(url):
    print('Starting {}'.format(url))
    response = await aiohttp.get(url)
    data = await response.text()
    print('{}: {} bytes: {}'.format(url, len(data), data))
    return data

futures = [call_url(url) for url in urls]
loop = asyncio.get_event_loop()
loop.run_until_complete(asyncio.wait(futures))
```

Рис. 2.11. Приклад використання Asyncio з синтаксисом async\await

На момент написання даної роботи, асинхронні додатки Python використовують співпрограми як основний компонент. Для їхнього запуску вони використовують бібліотеку Asyncio. Але є й інші важливі елементи, які можна вважати ключовими для асинхронних додатків:

- **Цикли подій (Event Loops)**. Asyncio створює цикли подій та керує ними. Цикли запускають співпрограми ще до завершення. Для зручності відстеження процесу одночасно може виконуватись лише один цикл обробки подій.

- **Завдання (Tasks)**. При запуску співпрограми в циклі подій можна повернути об'єкт Task. Він управляє поведінкою співпрограми поза циклом подій.

Нижче наведено приклад коду парсера з циклами подій та об'єктами завдань у дії (Рис. 2.12).

```
import asyncio
from web_scraping_library import read_from_site_async

tasks = []


async def main(url_list):
    for n in url_list:
        tasks.append(asyncio.create_task(read_from_site_async(n)))
    print(tasks)
    return await asyncio.gather(*tasks)


urls = ['http://site1.com', 'http://othersite.com', 'http://newsite.com']
loop = asyncio.get_event_loop()
results = loop.run_until_complete(main(urls))
print(results)
```

Рис. 2.12. Приклад використання Asyncio Tasks та Event loop.

Метод *.get_event_loop()* дає об'єкт, що дозволяє керувати циклом подій. Усі асинхронні функції передаються через *.run_until_complete()*, який запускає поставлені завдання до тих пір, поки вони не будуть виконані.

Метод *.create_task()* віддає об'єкт *Task* для запуску та приймає функцію. Кожна URL-адреса відправляється як окремий *Task* в цикл подій. Об'єкти *Task* зберігаються у списку. Варто звернути увагу, що це можна зробити всередині асинхронної функції чи, простіше кажучи, всередині циклу подій.

Контроль над циклом подій та завданнями безпосередньо залежить від складності програми. У прикладі зі скриптом парсера сайту для пильного контролю немає потреби. Тут досить просто збирати кінцеві дані, отримані внаслідок запуску завдань [13]. *Task*, який у майбутньому повинен бути повернути якесь значення, називається *Future* (з англ. «майбутнє»), й бібліотека Asyncio має відповідний об’єкт, який успадковується від *Task*. Усі

фіксовані завдання без проблем виконуватимуться тут одночасно, а якщо точно, то до тих пір поки виконується задача парсеру й, наприклад, очікує відповіді від запиту на конкретний URL – Python у час очікування оброблює наступний Task.

Асинхронність у мові програмування Python має в основі роботу з event loop та одним потоком, прямо як, наприклад, однопоточна мова програмування JavaScript, асинхронність у якій реалізована через взаємодією циклу подій з функціями зворотнього виклику.

Підбиваючи підсумки, **асинхронний режим** у порівнянні з **багатопоточністю** має ряд переваг:

- Функції асинхронності набагато легші за потоки. Це означає, що сотні та тисячі асинхронних операцій, що виконуються одночасно, будуть витрачати набагато менше ресурсів, ніж сотні та тисячі потоків.
- Порівняно з потоками, асинхронними операціями набагато легше керувати. Наприклад, їх можна скасовувати, задіявши об'єкт *Task - asyncio.create_task()*.
- У циклі асинхронних подій завдання виконуються в одному потоці, їх простіше розуміти, контролювати та відстежувати.

Фактично асинхронність йде пліч-о-пліч з багатопроцесорністю в Python. Є можливість використовувати *asyncio.run_in_executor()* для завдань, які активно використовують центральний процесор. Також асинхронність вирішує проблеми потоків:

1. Бібліотека Asyncio здійснює перемикання контексту лише на рівні програми, а не процесора. Використовується цикл подій.
2. Відсутність стану гонки потоків. Оскільки з Asyncio перемикання контексту здійснюється в заздалегідь створених точках, код не має проблеми гонки. Запускається одна співпрограма, яка перемикається лише у вказаних точках.

3. Набагато менше ресурсне голодування. Звичайно, Asyncio має pool (сукупність) потоків, який може впливати на витрачання ресурсів (запуск великої кількості процесів). Проте запуск співпрограм в одному потоці задіює набагато менше пам'яті на виконання процесів.
4. Взаємне блокування. Відсутність гонки потоків практично зводить до нуля різноманітні блокування виконання коду.

Є й невеликі вади використання асинхронної бібліотеки Asyncio. Щоб уникнути блокування циклу подій та тимчасових втрат на виконанні асинхронних функцій, весь код також повинен бути асинхронним, хоча це відносно незначна проблема [13, 14].

РОЗДІЛ 3.
REVERSE ENGINEERING ПОШУКОВИХ СИСТЕМ ТА РОЗРОБКА ВЕБ-СКРАПЕРІВ

3.1. Планування алгоритму роботи

Для створення алгоритму роботи над розробкою додатку, доцільним буде визначення усіх необхідних складових:

- Визначити завдання;
- Визначитися з архітектуру проекту й її реалізацією;
- Провести reverse engineering кількох популярних пошукових систем а також прикладів локальних (розташованих на сайтах) пошуковиках;
- Спроектувати та створити веб-скрапери.

Завдання даної випускної кваліфікаційної роботи полягає у вивченні та обгрунтуванні теми, на прикладах розкладення на складові частини відомі пошукові системи. Буде розроблено додаток, що використовує веб-скрапери як наслідок отриманого в ході використання методик reverse engineering досвіду. На найвищому рівні проектування, розробка буде являти собою додаток, що використовує веб-скрапери як джерело даних. Практична значущість та актуальність виконаної роботи може бути підтверджена численними прецедентами використання результатів у комерційних цілях: веб-скрапінг передбачає собою добування даних, необхідних для подальшої її обробки та аналізу, це може бути автоматизація прийняття рішень стосовно торгівлі певною валютою (в тому числі крипто-валютою); збирання великої кількості конкурентної інформації як наприклад сайт hotline.ua, що використовує численну кількість даних з інших джерел для відображення поточних цін на певний товар; десятки різноманітних пошукових систем можуть застосовуватися у комбінації у якості джерела даних, як це робить Rinalogy Search, й взаємодіяти з нейронною мережею інтерактивного навчання, щоб отримувати персоналізовані результати на основі відгуків

користувачів у реальному часі та таким чином значно оптимізувати процес пошуку; результати пошукових запитів можуть використовуватися для SEO-аналізу (SEO – Search Engine Optimization, оптимізація для пошукового движку), що призначений дати інформацію щоб визначити дії для досягнення розміщення сайту у топ-10 результатах пошуку (на першій сторінці) та ефективної індексації у пошукових системах.

Python – відносно легка у вивченні мова програмування за рахунок простого синтаксису і універсальності, завдяки багатій стандартній бібліотеці (набору інструментів і готових рішень, які не вимагають додаткової установки та налаштування), тому дану мову застосовують у різних областях. Завдяки своїй різноманітності бібліотек, в якій би галузі не довелося робити проект, швидше за все для цього вже є готова бібліотека Python: обробка зображень, математика, нейронні мережі, розпізнавання мови — для всього є інструменти. Також, у комбінації з простим синтаксисом, на мові Python зручно створювати парсери, робота з якими також є у стандартній бібліотеці.

Веб-скрапінг передбачає велику кількість запитів до сторонніх сервісів та сайтів, що займає певний час, саме тому використання асинхронності у структурі додатку є квінтесенцією оптимізації продуктивності.

Незначним недоліком використання асинхронності у мові Python є те, що при побудові архітектури необхідно буде увесь код створювати асинхронним, що є невеликим ускладненням інтуїтивного розуміння роботи коду.

У проекті буде використовуватися найпопулярніша модель проектування MVC. MVC (Model-View-Controller) — це шаблон у розробці програмного забезпечення, який зазвичай використовується для реалізації інтерфейсів користувача, даних та логіки керування. Він підкреслює поділ між бізнес-логікою програмного забезпечення та її відображенням. Таке «розділення обов’язків» забезпечує кращий розподіл роботи та оптимізоване обслуговування. Три частини шаблону програмного забезпечення MVC можна описати наступним чином:

- Model: керує даними та бізнес-логікою.
- View: обробляє макет сторінки і відображення даних (Front-end).
- Controller: направляє команди до деталей моделі та Front-end [17].

Reverse engineering пошукових систем та проектування коду веб-скраперів буде наведено у пункті 3.3.

3.2. Моделювання процесу розробки

З метою визначення чіткого вектору дій та кращого розуміння задач, було розроблено модель процесу розробки, а саме діаграму послідовності (Sequence Diagram). Моделювання здійснено за допомогою програмного засобу StarUML.

UML, скорочено від Unified Modeling Language – це стандартизована мова моделювання, що складається з інтегрованого набору діаграм, розроблених для надання розробникам програмного забезпечення конкретизації, візуалізації, побудови та документування артефактів програмних систем, а також для бізнес-моделювання та інших задач. UML являє собою сукупність найкращих інженерних практик, які виявилися успішними в моделюванні великих і складних систем; ця мова використовує переважно графічні позначення для вираження дизайну програмних проектів [19].

Діаграми послідовності UML – це діаграми взаємодії, які детально описують спосіб виконання операцій. Вони фіксують взаємодію між об'єктами в контексті співпраці. Діаграми послідовності орієнтовані на час, і вони візуально показують порядок синергії, використовуючи вертикальну вісь діаграми, щоб представити час, які повідомлення надсилаються та коли [20].

На моделі послідовності (Рис. 3.1) зображений основний актор – користувач, що робить запит до веб-серверу додатку, та послідовність дій самого веб-серверу у разі запиту. Послідовність операцій виглядає наступним чином:

1. Основний актор «користувач» взаємодіє з об’єктом «контролер веб-серверу» - виконує запит на веб-сервер додатку з визначеним текстовим запитом до пошукової системи;
2. Об’єкт «контролер веб-серверу» взаємодіє з об’єктом «сервіс веб-скраперу» - запускає їх роботу;
3. Об’єкт «сервіс веб-скраперу» виконує підготовку текстового запиту для пошукової системи – формує запит у правильній для конкретного скраперу формі;
4. Об’єкт «сервіс веб-скраперу» робить запит до об’єкту «пошукова система»;
5. Пошукова система, в залежності від певного переліку умов, повертає результати пошуку або сторінку помилки до об’єкта «сервіс веб-скраперу»;
6. Сервіс веб-скраперу виконує обробку отриманої сторінки й свою основну роботу – деструктурує необхідні дані з отриманої HTML-сторінки;
7. Сервіс веб-скраперу користується відокремленими модулями та методами – «помічниками веб-скраперу», або хелперами, серіазілує дані формуючи хеш-таблицю вигляду «ключ – значення» та відпрявляє їх до об’єкту «контроллер веб-серверу»;
8. Контроллер веб-серверу, завершивши свою роботу, надсилає серіалізовану у вигляді JSON-об’єкту відповідь до користувача.

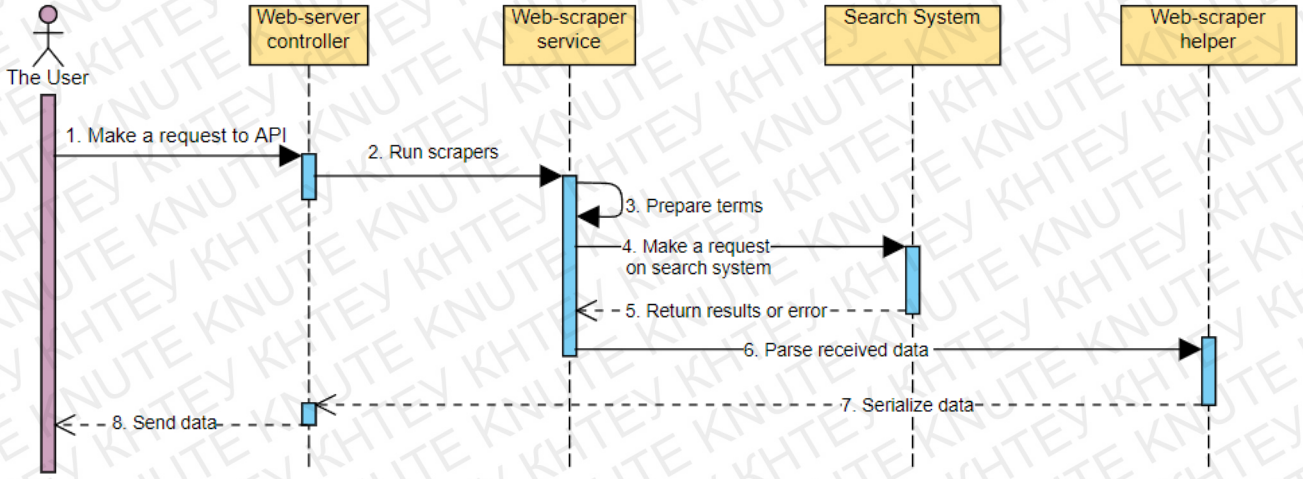


Рис. 3.1. Діаграма послідовності (авторська розробка)

Основний прецедент використання додатку полягає у запуску веб-скраперів для отримання інформації.

На стороні сервера, веб-скрапери займаються обробкою отриманих від запиту даних: дістають інформацію з HTML-тегів або із JSON-відповіді запиту, серіалізують їх й віддають дані користувачу. У випадку, якщо скрапер повернув сторінку помилки, контролер серверу відповість кодом статусу 500.

3.3. Reverse engineering пошукових систем та створення веб-скраперів

Не дивлячись на розглянутий у пункті 2. приклад використання reverse engineering для аналізу роботи Google Patents, існує ряд методів захисту від веб-скрапінгу. Подібні заходи необхідні додаткам для захисту даних а також для запобігання перенавантаження серверної частини веб-сайту, оскільки програми для автоматичного збору даних як правило виконує велику кількість запитів надзвичайно швидко. Серед основних методів захисту можна виділити наступні:

- Уникнення простих посилань виду «/topic/11», «/topic/12»;
- Обмеження результатів пошуку (10, 20 на сторінці пошуку);
- **Обмеження активності з однієї IP-адреси** (кількість та тривалість сесій);

- **CAPTCHA** - Completely Automated Public Turing test to tell Computers and Humans Apart. Повністю автоматизований публічний тест Тьюринга, щоб розрізняти комп'ютери та людей;
- Автоматична перевірка, як відвідувачі працюють з UI (як швидко заповнюється форма, куди безпосередньо клацають по кнопці, чи завантажує браузер CSS/статику, чи скрапер підвантажує тільки HTML);
- Перевірка User Agent**;**
- Перевірка Referer;
- Перевірка **cookies** файлів (шляхом присвоєння відповідному cookies-файлу спеціальні ключі);
- **Авторизація для API**, навіть для внутрішнього;
- Унікальна HTML-розмітка – різна розмітка в залежності від сторінки, локації користувачів, часу доби, що переглядається;
- Під'єднання спеціальних сервісів захисту від ботів, таких як **CloudFlare** [15].

Найпотужнішим заходом є обмеження активності з однієї IP-адреси у поєднанні з будь-якими іншими методами запобігання веб-скрапінгу, оскільки це напряму блокує IP-адресу з якої йде трафік на веб-додаток, що надзвичайно значно ускладнює процес парсингу даних.

У файлі robots.txt містяться інструкції, які говорять пошуковим роботам, які URL-адреси на сайті їм дозволено обробляти. З його допомогою можна обмежити кількість запитів на сканування і цим знизити навантаження на сайт [16], що надає достатньо інформації щодо можливості обробки сторінки, певних шляхів та використання певних URL-параметрів веб-сайту.

Нижче наведено такий перелік для Google Patents (рис. 3.2), у якому зазначено, що даний сайт приймає будь-який User-agent (що відповідає за версію браузера, операційної системи, тощо), проте не дозволяє ботам використовувати будь-які URL-параметри та похідні шляхи за адресою

«patents.google.com/», що фактично є забороною використання HTML-скраперів для даного сайту.

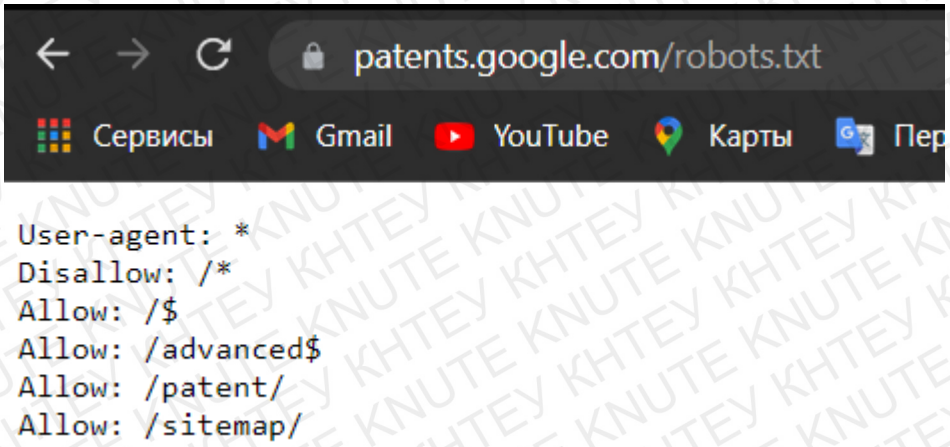


Рис. 3.2. robots.txt для сайту Google Patents

При виконанні ручного пошуку з використанням браузера на сайті Google Patents можна побачити, що необхідним контрактом отримання результатів пошуку є саме URL-параметри запиту (рис. 3.3)

Рис. 3.3. URL-параметри запиту на сайт Google Patents

Проте, у ході виконання дослідження роботи сайту Google Patents у розділі II п. 2.3 було виявлено, що для отримання результатів буде достатньо зробити запит саме на API сайту патентного пошуку, який не передбачає авторизації.

При проведенні подібних аналітичних маніпуляцій з такими пошуковими гігантами як Google та Bing видно, що у файлі robots.txt вони

також забороняють веб-скраперам та іншим ботам здійснювати пошук напряму, через GET-запит (Рис. 3.4)

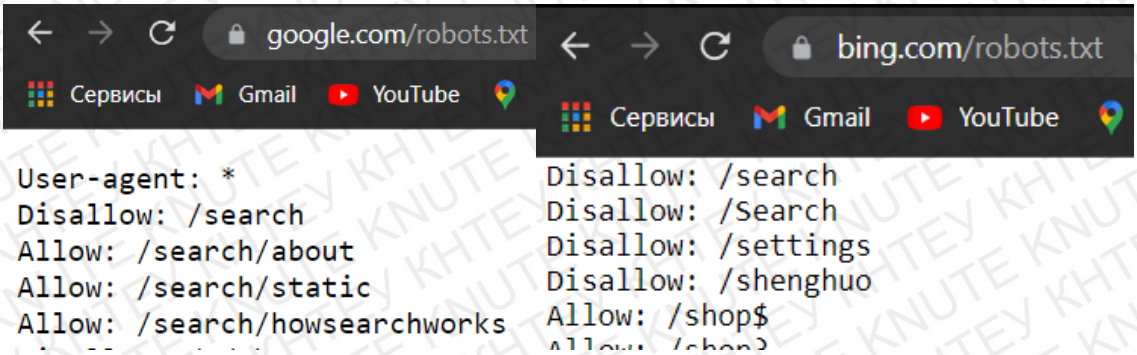


Рис 3.4. Обмеження для ботів на сайтах Google та Bing

В цілях верифікації обмежень, необхідно також подивитись у ChromeDevTools Network, чи наявні на сайтах xhr-запити як у випадку з Google Patents (Рис.3.5).

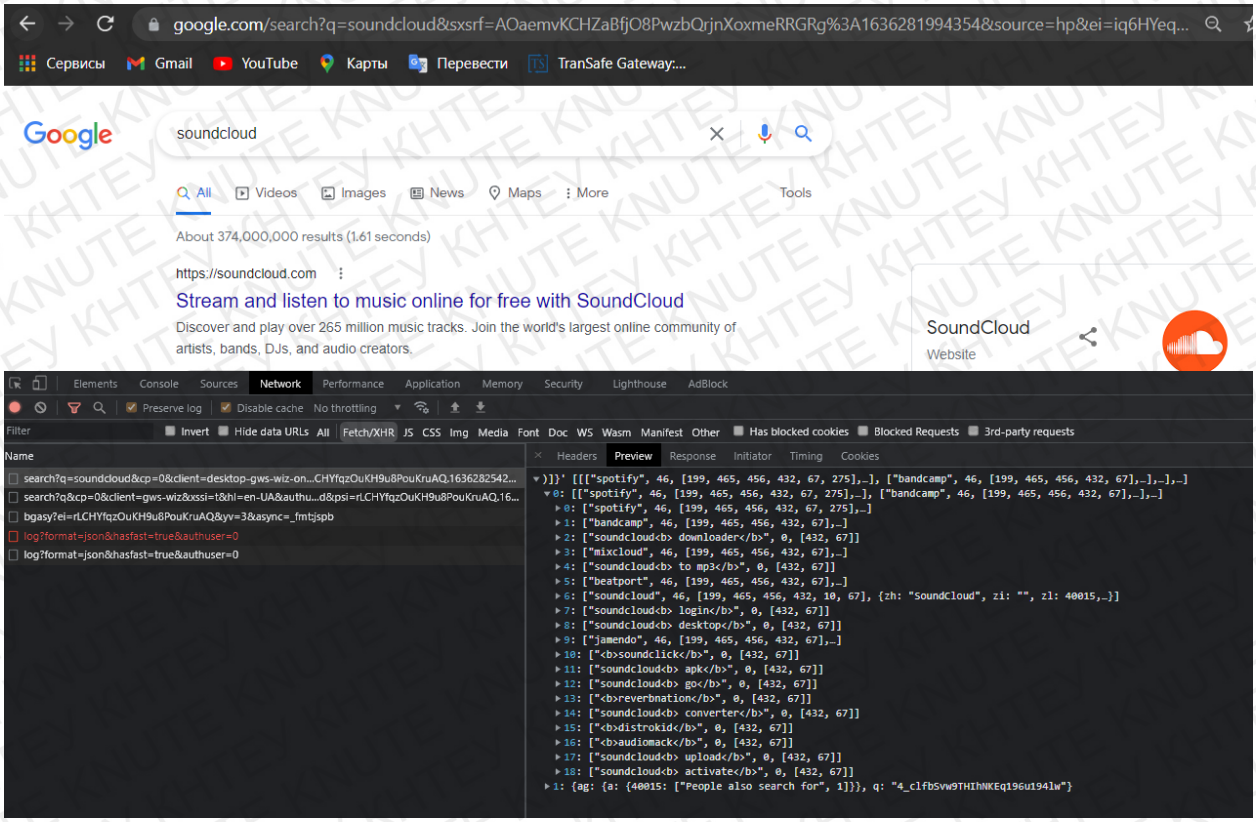


Рис. 3.5. Журнал мережевих запитів після запиту на Google.com

Як видно за результатами, наявні xhr-запити з цілком інтуїтивними назвами, проте при більш детальному огляді видно що інформація, що повернена з даних запитів, ніяк не пов’язана з виданими на сторінці

пошуковику результатами. Повернені дані більше схожі на дані нейромережі «люди також шукають» (рис. 3.6)

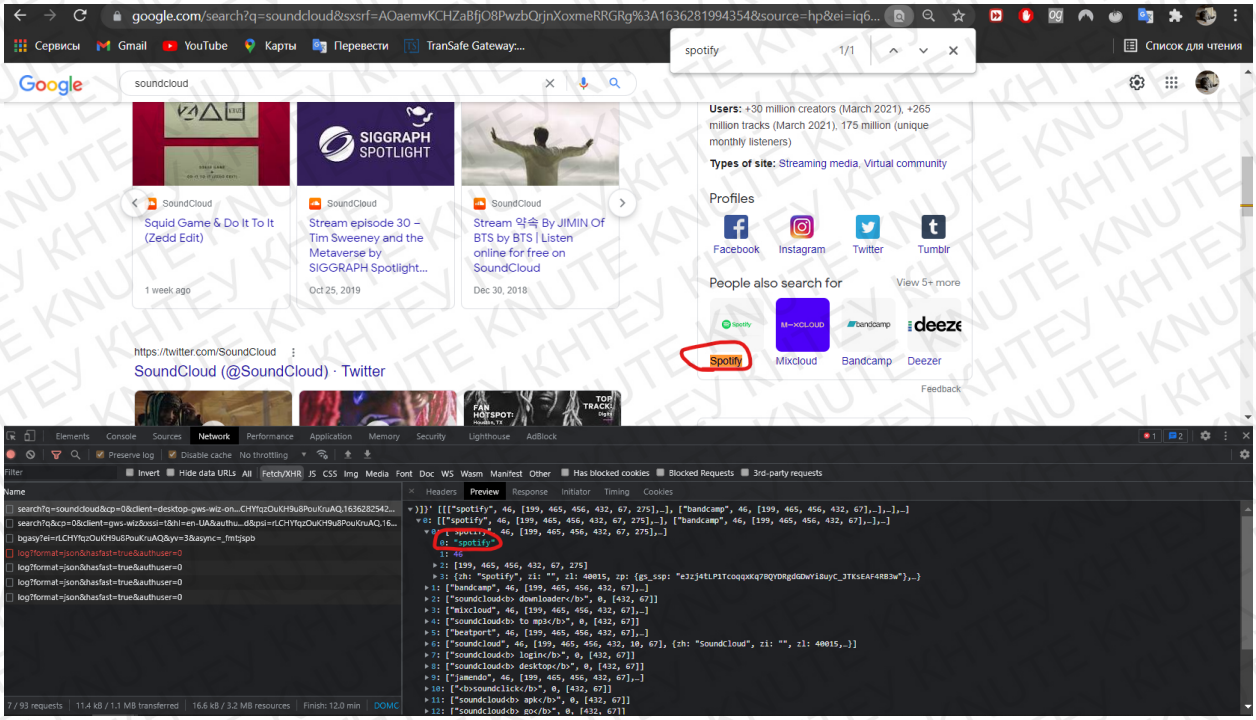
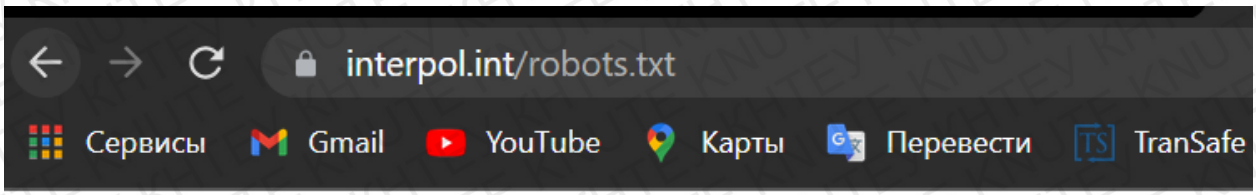


Рис. 3.6. “People also search for”

Пошукові системи не обов’язково мають контекст великих корпорацій, оскільки вони також широко використовуються й на різноманітних сайтах-провідниках спеціалізованої інформації, таких як онлайн-енциклопедія, науково-популярний блог, сайт з навчальними матеріалами, тощо. В якості неординарного прикладу такого сайту з наявною пошуковою системою та великою кількістю динамічних даних є сайт Інтерполу, а саме його сторінка зі списком розшукуваних людей.

Перш за все варто дослідити файл robots.txt на сайті, що повідомить про заборону веб-скрапінгу певних сторінок (рис. 3.7). Як видно на зображенні, сайт не має подібного файлу, проте це ще не означає що він не є захищеним від веб-скрапінгу.



Not Found

The requested URL /robots.txt was not found on this server.

Рис 3.7. robots.txt сайту Інтерполу

На сайті наявна форма з різноманітними фільтрами, такі як ім’я, фамілія, національність, гендер, вік, якою країною розшукується та ключові слова. Першим ділом слід відкрити ChromeDevTools вкладку Network, аби усі мережеві запити були записані у журнал, оскільки будь-яка форма на сайті передбачає xhr-запити на API додатку. Після даної дії, можна скористатися формою пошуку на сайті. Як видно на рисунку 3.10, пошукова форма дійсно відправляє ajax(xhr)-запит на додаток сайту, й у своїй відповіді віддає результати що відображені на сторінці.

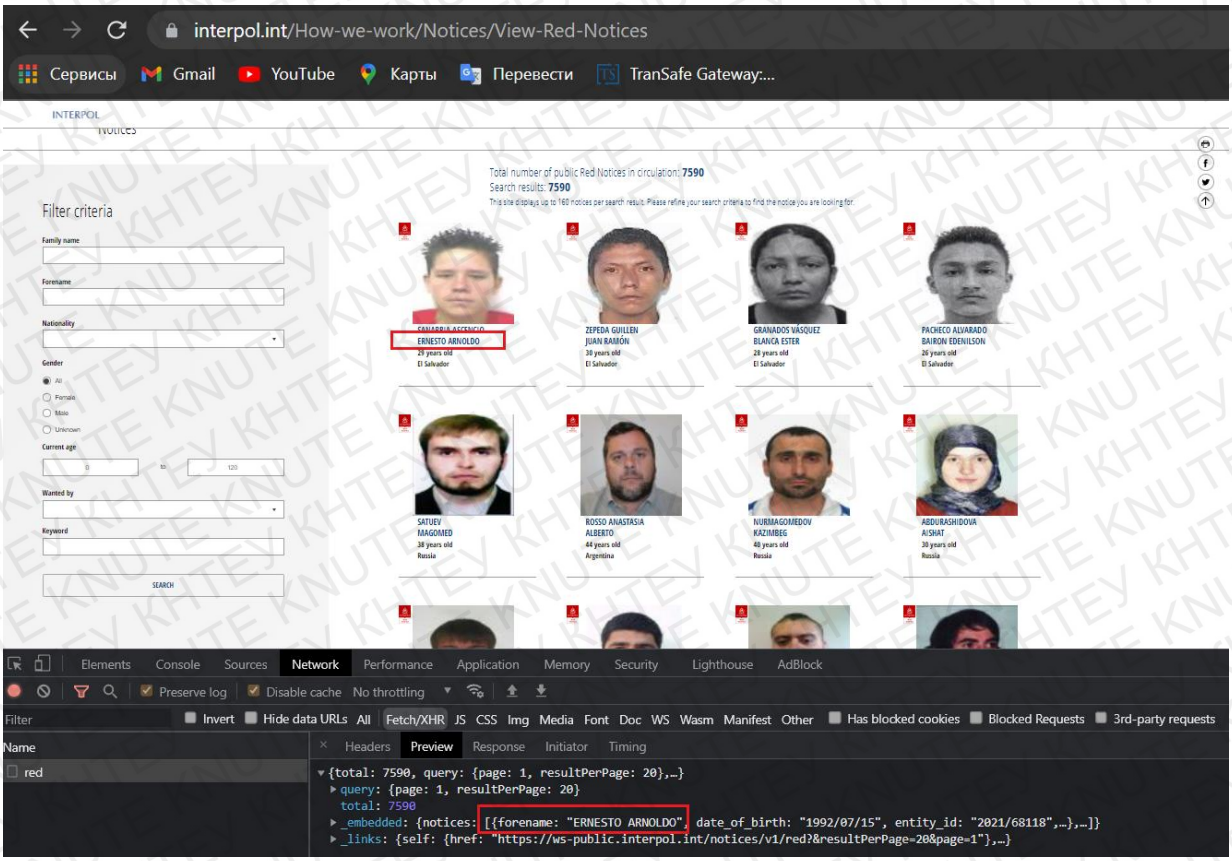


Рис. 3.7. Журнал мережевих запитів після відправлення форми

При переході на наступну сторінку, можна помітити url-параметри в новому xhr-запиті на вкладці Network (Рис.3.8).

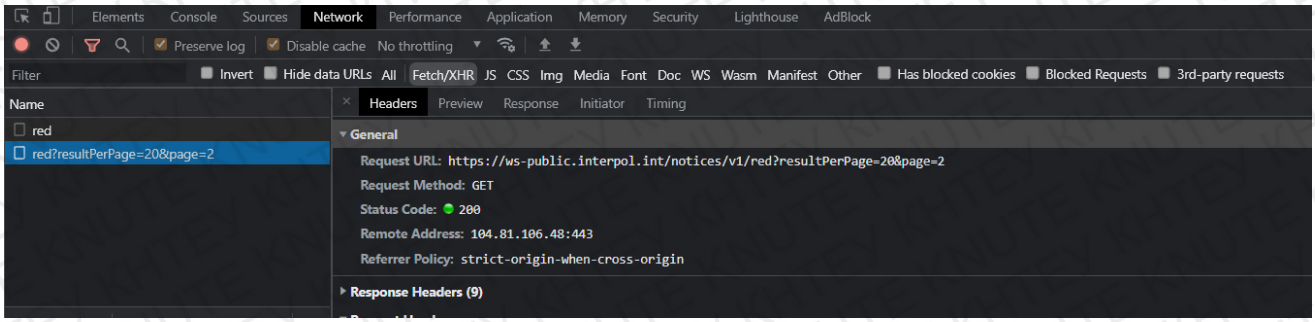


Рис. 3.8. url-параметри запиту наступної сторінки

Ця інформація відкриває дорогу емпіричним дослідам: якщо в url-параметрах наявна назва змінної, що дійсно відповідає кількості результатів що запитується у сервера, в такому разі необхідно перевірити чи можливо отримати результатів більше, ніж 20 за один запит. Слід також брати до уваги повідомлення на сторінці сайту, яке попереджає що максимальна кількість результатів на сторінці – 160. Можливо, це є обмеженням додатку сайту.

Отже, для перевірки гіпотези про більшу кількість результатів буде взята остання url з її параметрами та відкрита в іншій вкладці браузера, де і будуть проводитися досліди (Рис.3.9). Як і у вкладці Network, відповіддю сервера повинна бути у вигляді JSON-об’єкту.

Як показав експеримент, з додатку сайту дійсно можна витягнути більшу кількість результатів – було проведено спробу дістати 160, 400 і 1000 – проте не дивлячись на url-параметри у запиті, згідно з попередженням на сайті максимально можлива кількість результатів становить 160.

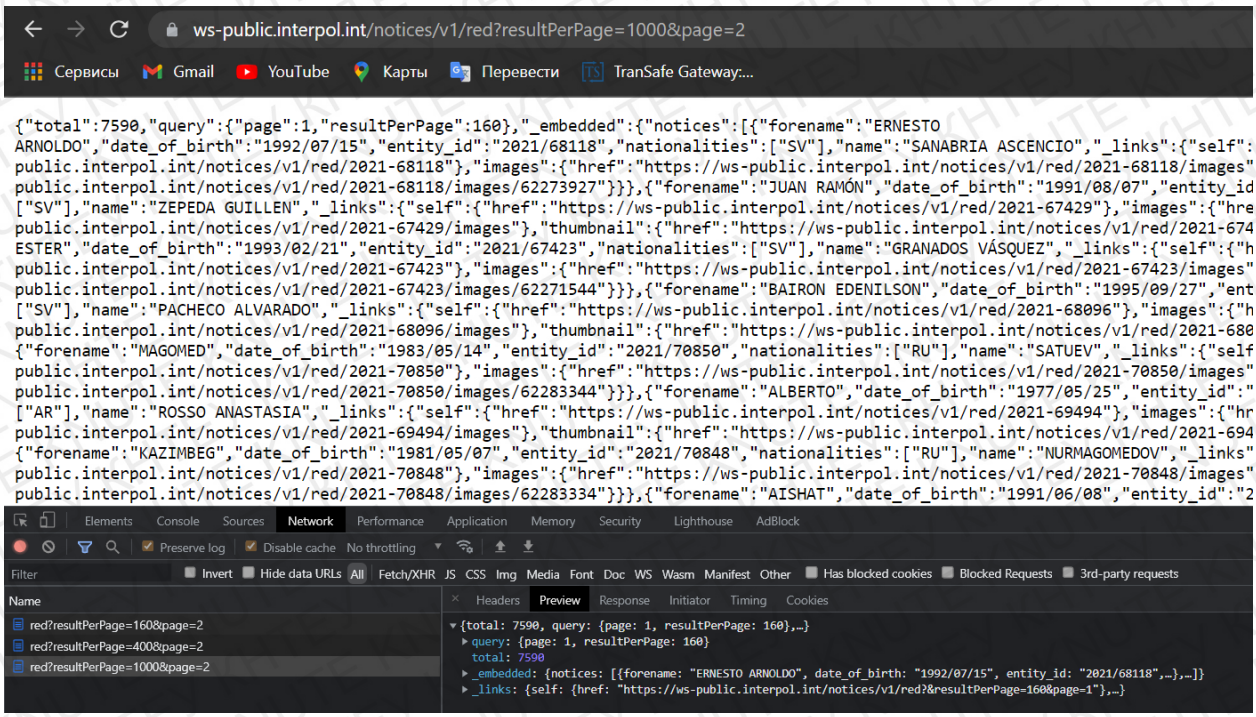


Рис. 3.9. Відповідь сервера із зміненими url-параметрами

Отримана у ході дослідження інформація приводить до висновку, що для отримання усіх результатів із сайту необхідно буде використовувати доступні на сайті фільтри та їх комбінації (що передаються на сайт в якості url-параметрів), що у свою чергу означає використання рекурсії з великим рівнем вкладених циклів перебору фільтрів. Існує й вірогідність того, що хоча переважна більшість даних і дістанеться таким чином, проте таке обмеження на сайті може створити невеликий відсоток втрачених при веб-скрапінгу даних, оскільки об’єкт розшукуваних людей може, наприклад, не мати дати народження, національності та будь-яких інших даних, наявність яких передбачається використанням фільтрів.

Не дивлячись на розглянуті заборони, навіть пошукові гіганти як правило не можуть з першого ж запиту виявити, чи є запит на сайт від клієнтської частини браузера чи це автоматизований скрипт, тому буде створено скрапер результатів пошуку сайту Google та Bing. Як засоби захисту від блокування, буде використано User-Agent власного браузера (що надає боту-скраперу вигляд людського запиту, зроблений через UI веб-браузера), та у разі використання більше ніж одного запита на сайт буде

використано часову затримку між запитами, що надійно позбавить скрапер від підозри за перевищення ліміту запитів у секунду.

Для забезпечення роботи додатку, необхідно створити сервер, який зможе приймати за певним маршрутом (або роутом, від англ. Route – «шлях») запити й давати на них відповіді. У контексті роботи даного додатку – відповідь буде являти собою оброблені та серіалізовані дані веб-скраперів, а саме результати пошуку. У якості зручного у використанні та швидкому у розгорненні варіанта буде використовуватися фреймворк Flask.

Flask — це легкий WSGI каркас веб-додатку (WSGI - Web Server Gateway Interface, з англ. «Інтерфейс шлюзу веб-сервера»). Він розроблений для того, щоб розпочати роботу швидко та легко, з можливістю масштабування до складних програм. Даний фреймворк починався як проста обгортка навколо Werkzeug (бібліотека шифрування) і Jinja (шаблонізатор User Interface на стороні Back-End серверу) і стала однією з найпопулярніших фреймворків веб-додатків Python [21].

Модель проектування MVC є досить популярною та простою у розумінні, тому як було зазначено у попередньому пункті, буде використовуватись саме вона. У випадку даної роботи, буде створено проект з наступною архітектурою (Рис.3.10):

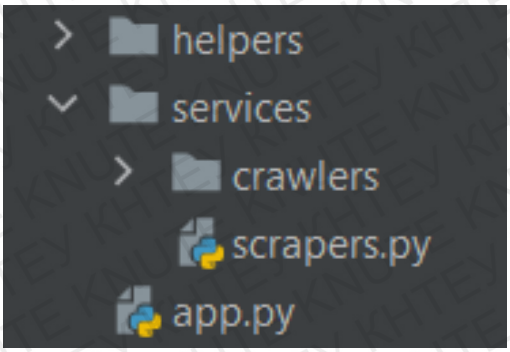


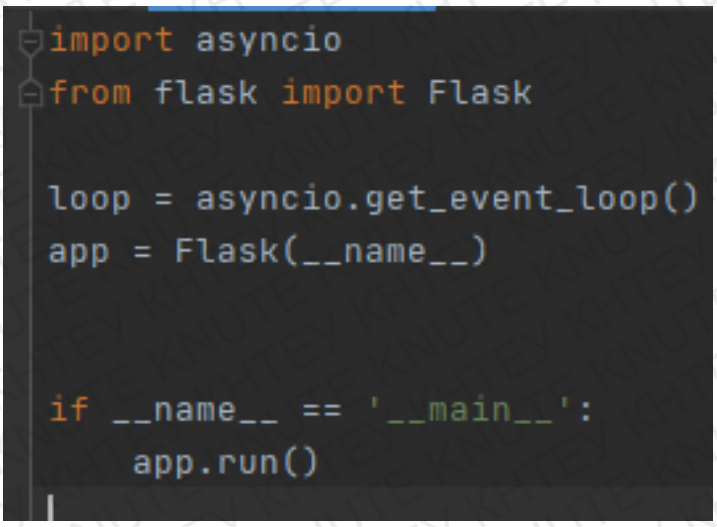
Рис. 3.10. Дерево ресурсів проекту

1) Файл додатку: в ньому буде міститися базовий сервер Flask, петля подій Asyncio (Event loop) та, оскільки проект не буде мати велику кількість маршрутів, прямо у файлі серверу буде присвоєно основні маршрути що будуть відповідати результатами веб-скраперів;

2) Папка сервісів: дотримуючись MVC, дана модель передбачає файлове розмеження контролерів та їх функціоналу – сервісів. Контролер імпортує основні функції та методи з сервісів які у свою чергу запускають ланцюг ще більш розгалужених подій, й таким чином досягається чистота, розгалуженість й інтуїтивна зрозумілість коду;
3) Папка краулерів: з точки зору семантики, веб-скрапери і є основними сервісами та бізнес-логікою проекту. Для більш чіткого розгалуження задач, буде створено дочірню папку краулерів у сервісах;
4) Папка хелперів: дані ресурси як правило означають що вони не є бізнес-логікою додатку, просто *допомагають* з роботою основних сервісів. Звідси й назва.

Отже, першим кроком є створення серверу – з використанням Python Flask це займає всього кілька рядків коду (Рис.3.11). За замовчуванням, сервер буде створено за адресою localhost або 127.0.0.1, що означає *локальний* сервер, з портом 5000. Для створення додатку, необхідно встановити пакет Flask у інтерпретатор або у інтерпретатор Python віртуального оточення проекту, імпортувати з даної бібліотеки клас Flask, та зробити новий екземпляр класу, передавши у якості аргумента magic method мови програмування Python, назву файлу поточного файлу з розширенням .py – «__name__».

Також, необхідно декларувати Event loop Asyncio, для того щоб реалізувати асинхронне програмування у проекті.



```python
import asyncio
from flask import Flask

loop = asyncio.get_event_loop()
app = Flask(__name__)


if __name__ == '__main__':
    app.run()
```

Рис. 3.11. Базовий сервер Flask

Як було зазначено вище, роути будуть декларовані прямо у файлі серверу. Тому наступним кроком буде створення маршруту запуску одного конкретного скраперу, та запуску усіх скраперів (Рис.3.12).

```python
@app.route('/api/crawlers', methods=['POST'])
def crawlers():
    if request.method == 'POST':
        terms = request.form['terms']

        items = loop.run_until_complete(run_scrapers(terms))
        return {'success': True, "results": items}


@app.route('/api/crawlers/run_one', methods=['POST'])
def run_one():
    if request.method == 'POST':
        terms = request.form['terms']
        scr_title = request.form['scr_title']

        items = loop.run_until_complete(run_separate_scraper(terms, scr_title))
        return {'success': True, "results": items}
```

Рис. 3.12. Маршрути додатку

Для декларування роутів, використовується синтаксис декораторів у мові Python бібліотеки Flask. У даному випадку, це метод перехоплення запитів на сервер, який слугує обгорткою для певної функції-колбека, що виконається при запиті на поточний маршрут. Приймає у себе HTTP-методи, що буде обслуговувати даний маршрут, а також url маршруту.

У коді намічено дані, які буде приймати даний роут, такі як «пошуковий запит» або «терміни», в обох роутах, та назва скраперу, у другому роуті відповідно. Намічено також використання певних функцій «run» та «run_separate» які поки що не існують, проте будуть імпортованими функціями з сервісу scrapers. Передбачаючи, що обидві функції – для запуску всіх скраперів та запуску одного конкретного відповідно – будуть асинхронними, використання методу event loop «run_until_complete» є необхідною умовою для коректної роботи асинхронності у Python.

Наступним є створення сервісу скраперів, проте окрім як функцій-заглушок на даний момент там нічого не буде, оскільки у даному модулі будуть імпортуватися скрапери, які необхідно також спроектувати. В такому випадку, доцільним буде намітити назви методів, що буде викликано у класів веб-скраперів, передбачаючи результат.

Веб-скрапери мають потребу у кількох однакових блоках коду, тому керуючись хорошою практикою DRY (Don’t repeat yourself – з англ. «не повторюйся» буде спроектовано *базовий* клас краулерів, від якого будуть успадковуватись усі поля та методи класу. Отже, однаковими у будь-якому веб-скрапері будуть:

1) HTTP-headers, які будуть містити «User-Agent» та «Accept-Language», що є першорядною превентивною мірою щодо захисту веб-сайтів від автоматичного збирання інформації;
2) Мова – скоріше за все, знадобиться у мережевих запитах.
3) Тестовий url – для тестування працездатності скраперів та усього маршруту;
4) Функція XHR-запиту на сторонні ресурси – будь-який скрапер буде використовувати мережеві запити, тому слушною думкою буде налаштувати метод у батьківському класі, що також буде використовувати декларовані раніше HTTP-headers.
5) Базовий метод, який буде запускати пошук, отримувати та відправляти оброблену інформацію.

User-Agent взято з браузера суб'єкта даної випускної кваліфікаційної роботи. Метод fetch буде використовувати клієнтську сесію бібліотеки Aiohttp, з впровадженими заголовками для кожного запиту, та віддавати HTML-сторінку з GET-запиту.

У розділі II розповідається про використання корутин та генераторних виразів у мові програмування Python, й хоча даний випадок не передбачає роботу з великими об'ємами даних, проте використання даного синтаксису є хорошою практикою у розробці будь-якого роду парсерів, тому в цілях ознайомлення буде використано саме підхід генераторних функцій (Рис.3.13).

```
import aiohttp
import async_timeout


class BaseCrawler:
    DEFAULT_HEADERS = [
        ('User-Agent', "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) "
                       "Chrome/95.0.4638.69 Safari/537.36"),
        ("Accept-Language", "en-US,en;q=0.5"),
    ]
    language = "en"
    test_url = 'https://knute.edu.ua/'

    @classmethod
    async def fetch(cls, url):
        async with aiohttp.ClientSession() as session, async_timeout.timeout(10):
            async with session.get(url, headers={**dict(cls.DEFAULT_HEADERS)}) as response:
                return await response.text()

    @classmethod
    async def search(cls, terms):
        items = []
        async for row in cls._search(cls, terms):
            items.append(row)

        return {cls.title: items}
```

Рис. 3.13. Базовий клас краулера

Наступним кроком є розробка веб-скраперу пошукової системи Bing. В якості результатів обробки HTML-сторінки, буде представлено масив словників, ключами яких будуть: заголовок результату; url-посилання; snippet – короткий опис; та HTML-snippet – короткий опис, загорнутий у HTML-теги.

Оскільки було прийнято рішення використання генераторних функцій, метод пошуку буде реалізацією даного синтаксису, що забезпечується використанням ключового слова «yield».

Для проведення GET-запиту необхідно проаналізувати url-параметри типового запиту до пошукової системи (Рис.3.14).

Рис. 3.14. url-параметри пошукового запиту Bing

Як видно, «q» відповідає тексту пошукового запиту (query), «sourceid» -- судячи з назви, джерелу запиту, «ie» (encoding) – тип кодування, «hl» (home language) – використовувана мова, та інші 2. При тестуванні було виялено, що необхідними є усі, окрім останніх двох. Також у випадку, якщо це запит не на першу сторінку результатів, додається ще параметр «first», що відповідає за номер результату, з якого починати відлік – так реалізована пагінація. Таким чином встановлено встановлено, що ці дані можна занести у поля класу веб-скраперу. Наступним є визначення HTML-розмітки, за якою буде діставатися інформація зі сторінки. Використовуючи ChromeDevTools,

емпіричним шляхом встановлюємо необхідні для обробки класи, що будуть слугувати орієнтиром для парсингу (Рис.3.15).

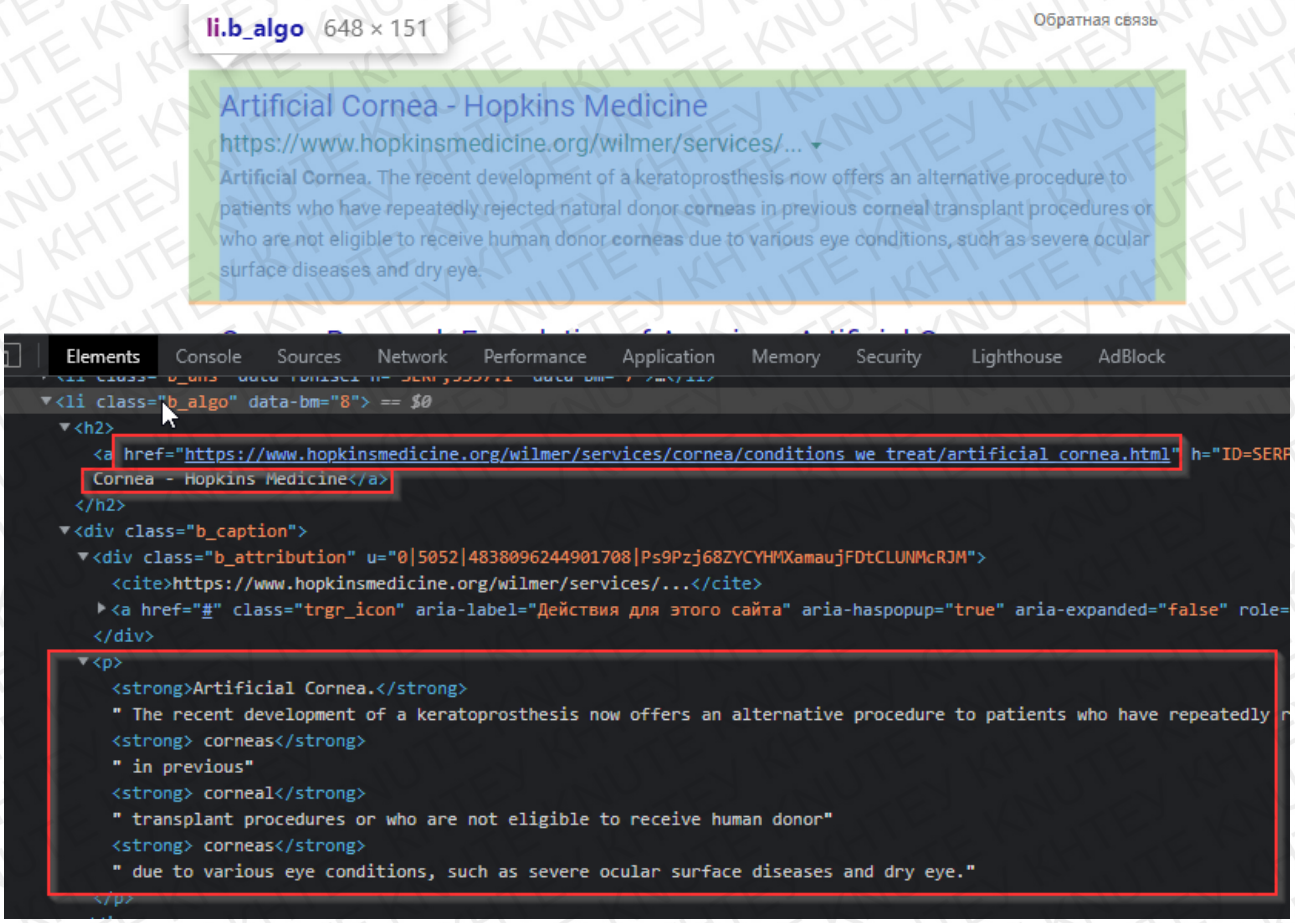


Рис. 3.15. Аналіз необхідних елементів HTML-сторінки

З отриманими даними, можна створити поля у класі, що будуть відповідати шаблону по котрому буде необхідно провести обробку інформації. Буде використано бібліотеку BeautifulSoup. Beautiful Soup — це бібліотека, яка дозволяє легко витягувати інформацію з веб-сторінок. В основі його роботи лежить аналізатор HTML або XML розмітки, надаючи методи для ітерації, пошуку та модифікації дерева елементів [22].

Створивши необхідні поля класу та під’єднавши бібліотеку BS4, залишається тільки користуватися усіма інструментами. Також, буде створено цикл for, який буде слугувати регулятором кількості бажаних результатів: оскільки на сторінку може прийти усього 10 результатів, використання циклу буде передбачати нові запити до сайту, а при їх великій кількості – висока вірогідність блокування. Проте, слушною думкою буде

регулювання таких речей, отже цикл слід додати. На рисунку 3.16 наведено спроектований скрипт веб-скраперу Bing, з використанням деяких полів з базового класу та методу fetch.

```
class BingCrawler(BaseCrawler):
    title = 'Bing'
    paragraph_re = re.compile('</?p.*?>', re.A + re.S)
    SEARCH_URL = "https://bing.com/search"
    RESULT_SELECTOR = '#b_results > .b_algo'
    NO_RESULT_SELECTOR = '#b_results > .b_no'
    max_document_count = 10

    async def _search(self, terms):
        for i in range(0, self.max_document_count, 10):
            url = self.SEARCH_URL + "?q=" + urllib.parse.quote(terms) +\
                  "&sourceid=chrome&ie=UTF-8&hl=" + self.language
            if i > 0:
                url += f"&first={i}"
            data = await self.fetch(url)
            page_dom = BeautifulSoup(data, "lxml")
            if page_dom.select(self.NO_RESULT_SELECTOR):
                return

            for result in page_dom.select(self.RESULT_SELECTOR):
                link = result.select('a')[0]['href']
                h2 = result.select('h2')
                title = h2[0].text if h2 else result.text
                b_cap = result.select('.b_caption')
                snippet = b_cap and b_cap[0].p and b_cap[0].p.text or ''
                html_snippet = snippet and str(b_cap[0].p) or '<br/>'.join(
                    [li.text for ol in result.select('ol') for li in ol.select('li')])
                yield {
                        'htmlSnippet': self.paragraph_re.sub('', html_snippet),
                        'snippet': snippet,
                        'link': link,
                        'title': title
                    }
            await asyncio.sleep(1.0)
```

Рис.3.16. Bing crawler

Наступним є перевірка даного скраперу та під’єднання його спочатку до сервісу, а далі й використання сервісу у роуті.

Зручним буде використання файлу __init__ для скраперів, який би зібрав до однієї купи усі доступні для використання скрапери. Створені екземпляри класів краулерів можуть бути зручно імпортовані у будь-яке

місце в проекті. Хоча й можна було перебрати усі доступні скрапери більш автоматично, наприклад використовуючи глобальну змінну globals та маніпулюючи синтаксисом list comprehention (генератор списків), проте у даному випадку достатньо буде й ручний запис екземплярів скраперів до змінної (рис.3.17).

```
from services.crawlers.google import GoogleCrawler
from services.crawlers.bing import BingCrawler

crawlers = [GoogleCrawler(), BingCrawler()]
```

Рис.3.17. Список скраперів

Далі, імпортуючи скрапери, прийшов час до написання сервісу запуску створеного скраперу. Слід пам’ятати про контракти бажаної функції, оскільки окрім тексту пошукового запиту буде присутня також і назва скраперу, який необхідно запустити. Тому, для автоматичного пошуку запитаного скраперу буде використано фільтрацію через генераторний вираз створення списку, що буде перебирати кожний краулер та дивитись на його назву: якщо назва скраперу співпадає із запитаною – даний скрапер і буде запущено. Це практично декларативний підхід, без нагромадження коду використанням синтаксису «if \ else».

Отримавши потрібний краулер, його необхідно запустити в Event loop – використання asyncio.gather створить об’єкт future, який буде виконувати свою роботу «на задньому фоні». Проте це усе що необхідно від проектованої функції, тому наступним рядком є очікування результатів з об’єкту future (рис. 3.18).

```python
import asyncio

from services.crawlers import crawlers


async def run_separate(terms, scr_title):

    crawler = [c for c in crawlers if c.title == scr_title][0]
    future = asyncio.gather(crawler.search(terms))
    results = await future

    return results
```

Рис. 3.18. Проектування сервісу запуску конкретного краулера

Далі, потрібно описати роут, що буде приймати та зчитувати дані від запиту на сервер, та запускати в Event loop створену функцію сервісу (Рис. 3.19)

```python
@app.route('/api/crawlers/run_one', methods=['POST'])
def run_one():
    if request.method == 'POST':
        terms = request.form['terms']
        scr_title = request.form['scr_title']

        items = loop.run_until_complete(run_separate(terms, scr_title))
        return {'success': True, "results": items}
```

Рис. 3.19. Роут для запуску конкретного скрапера

Для перевірки зробленої роботи, буде створено тестовий файл, що буде імітувати (фактично *здійснювати*) реальний запит до запущеного серверу. Буде здійснено запит за необхідним роутом, із запитом до Bing «штучна роговиця ока». Також, знаючи структуру майбутніх результатів, можна зробити відображення заголовків пошукових результатів у терміналі запущеного скрипту (Рис. 3.20).

```python
import json
import requests

app_url = 'http://127.0.0.1:5000/'

resp = requests.post(app_url + 'api/crawlers/run_one', data={'terms': 'Artificial Cornea',
                                                             'scr_title': 'Bing'})
data = json.loads(resp.text)
results = data.get('results')
print(results, '\n')

for res in results:
    for items in res.values():
        for item in items:
            print([v for k, v in item.items() if k == 'title'][0])
        print('\n')
```

Рис. 3.20. Скрипт тестового запиту на сервер

Як видно у терміналі та у відкритому дебагері на рисунку 3.21, сервер та веб-скрапер спрацювали без помилок.

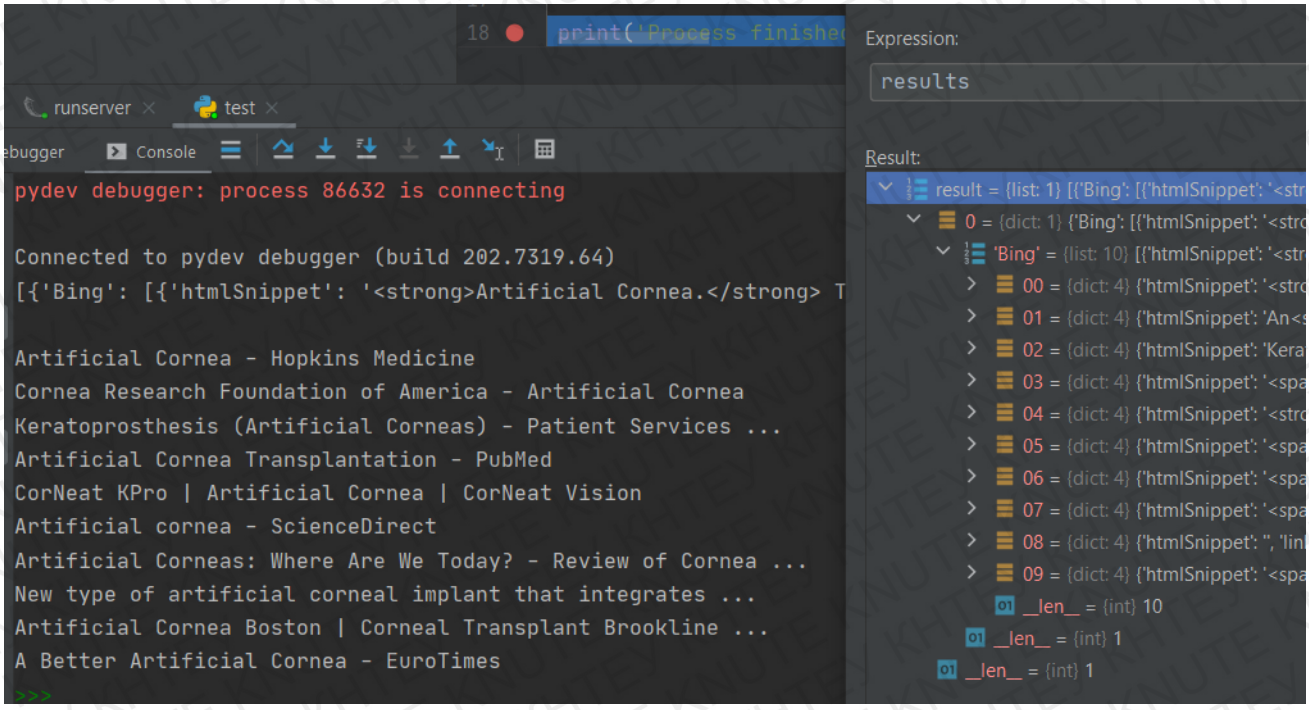


Рис. 3.21. Результати запиту до серверу додатку скраперу Bing

Наступним етапом є створення веб-скраперу пошукової системи Google. Для подальшої роботи, необхідно провести reverse engineering веб-сторінки.

Перш за все як і у випадку з Bing, необхідно подивитися на url-параметри запиту. На рисунку 3.22 зображено, які параметри використовує

гугл для своїй запитів, проте емпіричним шляхом, з використанням VPN-сервісів для запобігання блокуванню IP-адреси, було виявлено що деякі з них не обов’язково використовувати, проте «q», «hl» та «start» (при запиті не на першу сторінку) – є необхідними.

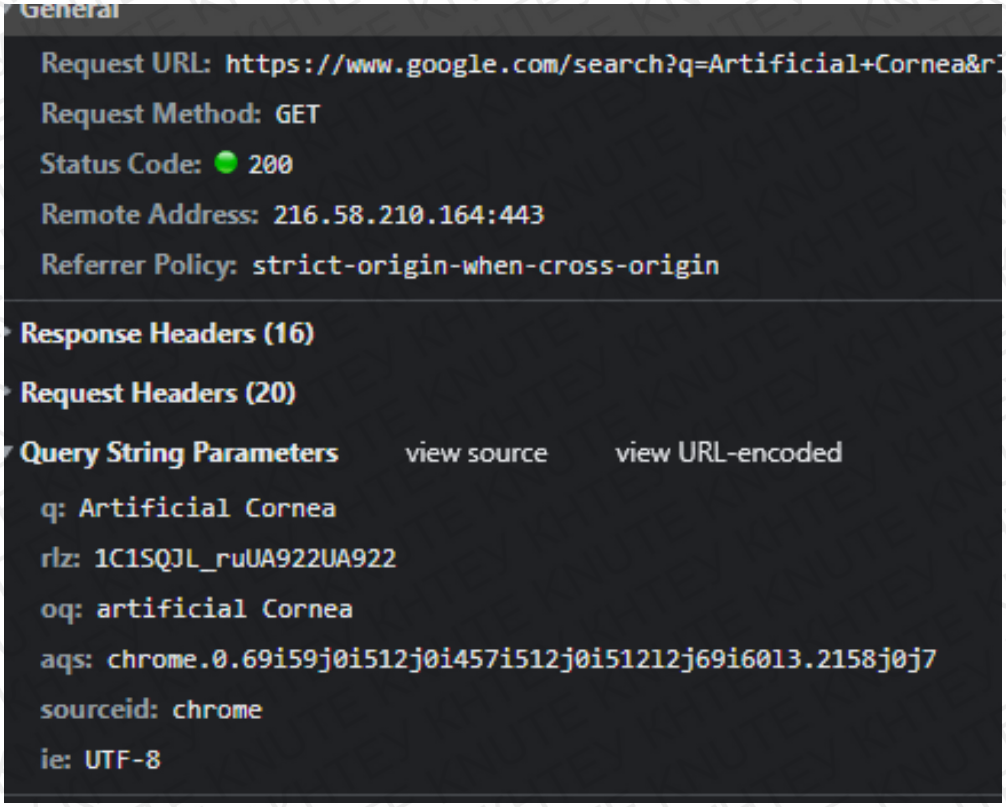


Рис. 3.22. url-параметри пошукового запиту до Google

Тепер аналіз верстки сторінки, для створення орієнтирів пошуку інформації (рис. 3.33).

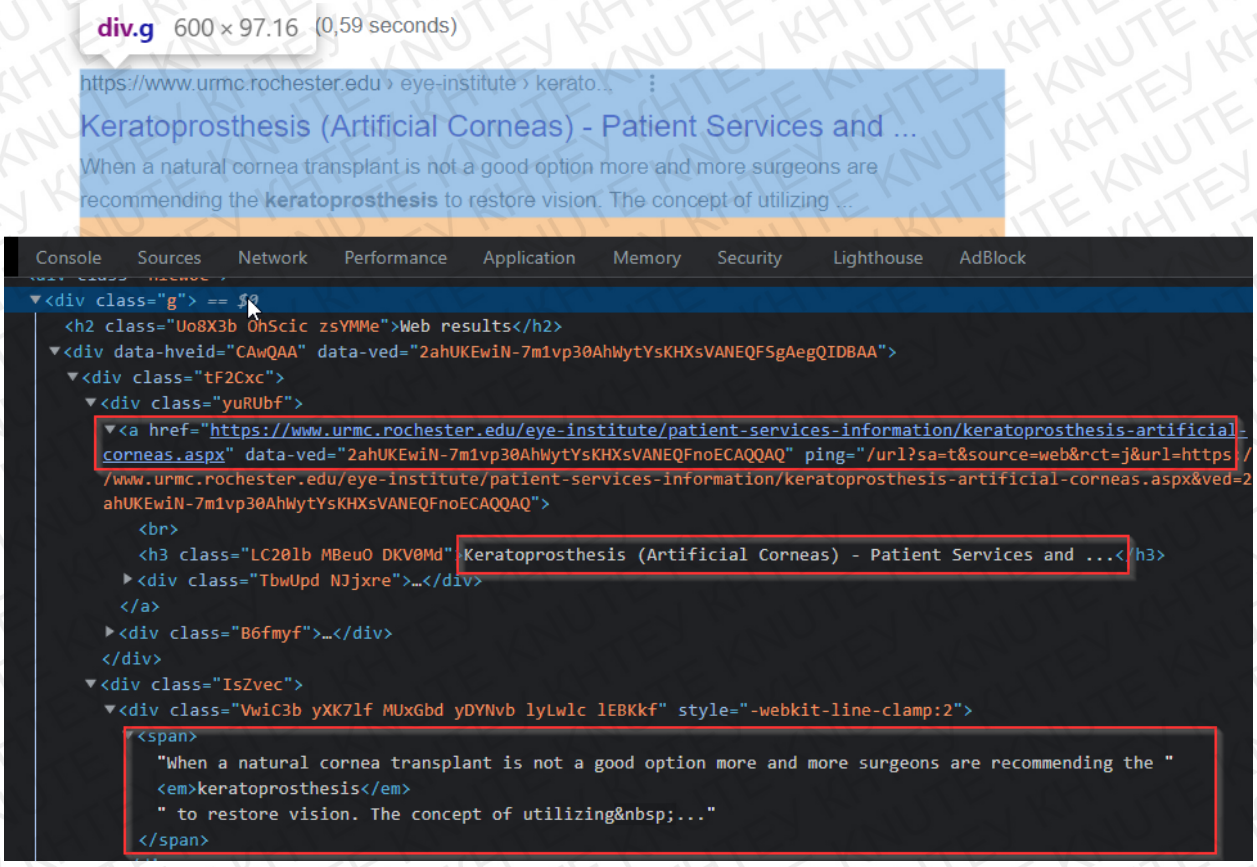


Рис.3.33. Reverse engineering елементів веб-сторінки результатів пошуку Google

В цей раз, буде використана технологія регулярних виразів, що присутня у будь-якій мові програмування а також значно швидше працює ніж будь-які бібліотеки для взаємодії із типом даних string.

У класі скраперу Google буде використано метод компіляції шаблонів регулярних виразів, за якими можна буде у майбутньому здійснити пошук.

Також, для розмежування функціоналу веб-скрапера та окремого блоку коду парсингу, буде створено файл text.py у папці helpers, основною задачею якого буде використання своїх методів для обробки тексту та діставання блоку інформації за певним шаблоном (Рис. 3.34).

```python
import re


class TextHelper:
    tags_re = re.compile('<.*?>|&\w+;', re.S+re.I)

    @classmethod
    def strip_tags(cls, html):
        return cls.tags_re.sub('', html)

    @staticmethod
    def between(text, start, end, strip=False):
        text = text.split(start, 1)[-1]
        if end:
            text = text.split(end, 1)[0]
        return text.strip() if strip else text
```

Рис. 3.34. Клас TextHelper

Хорошим тоном буде використання нещодавно розроблених у мові програмування Python інструментів, як-от наприклад використання walrus operator, що дозволяє економити одну строку коду декларуючи змінну прямо в конструкції «if \ else».

Створення коду скраперу передбачає собою правильне форматування url-посилання, використання методу fetch що було розроблено раніше у базовому класі, а також використання скомпільованих шаблонів регулярних виразів, використання методів TextHelper для роботи з текстом та серіалізація даних для відправки у якості відповіді сервера. Метод роботи через регулярні вирази хоча і швидший, проте передбачає певний рівень ерудиції, оскільки методи роботи з ними є не достатньо очевидними для запам’ятовування, проте це є дуже потужним інструментом. Спроектований скрапер зображено нижче (Рис.3.35).

Рис. 3.35. Google search скрапер

Практично завершуючим кроком кроком є створення сервісу, що буде запускати усі скрапери, створити відповідний роут, а також перевірити як роботу скрапера Google, так і роботу нового маршруту.

Причина, з якої збирання до купи усі скрапери було хорошою ідеєю, полягає у наступній маніпуляції: перебираючи список краулерів за допомогою синтаксису генератору списку, у кожного краулера буде викликатися базовий метод .search – що також є причиною створення єдиного методу батьківського класу скраперів, що буде викликати дочірній метод – а ті, у свою чергу, будуть відправлені у Event loop asyncio в якості корутин, які у результаті й дадуть оброблені дані (рис. 3.36).

```
async def run_all(terms):
    futures = asyncio.gather(*[c.search(terms) for c in crawlers])
    results = await futures

    return results
```

Рис. 3.36. Функція запуску усіх скраперів

Нижче наведено роут, що приймає запит до серверу й запускає усі веб-скрапери (Рис. 3.37).

```python
@app.route('/api/crawlers', methods=['POST'])
def crawlers():
    if request.method == 'POST':
        terms = request.form['terms']

        items = loop.run_until_complete(run_all(terms))
        return {'success': True, "results": items}
```

Рис. 3.37. Роут для отримання результатів з усіх доступних скраперів

Перевірка тестовим скриптом показала, що скрапер Google працює нормально (рис. 3.38), хоча результатів не 10 як у ситуації с Bing, це через те що деяка кількість результатів на сторінці є рекламою.

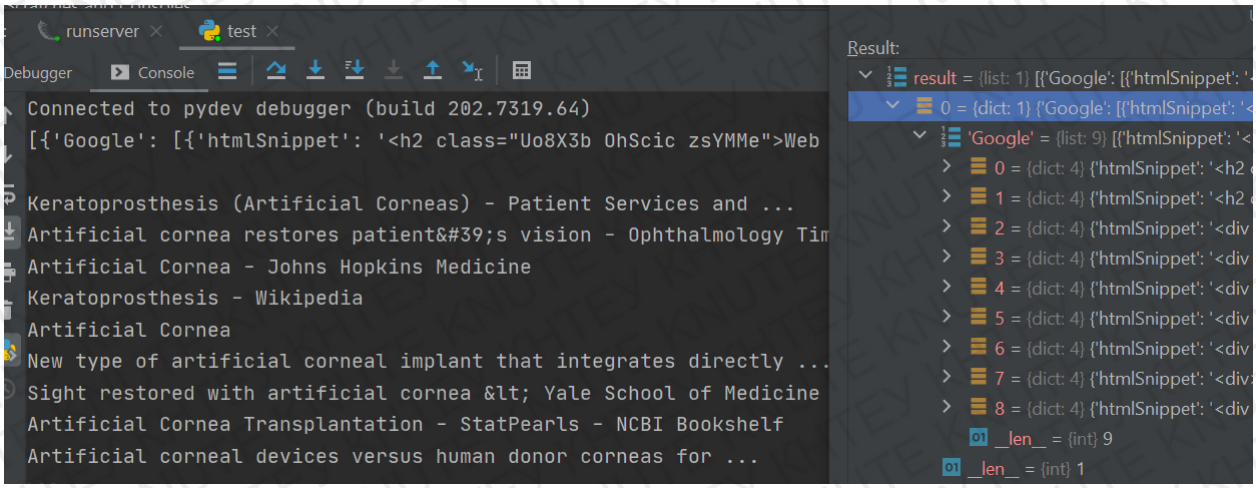


Рис. 3.38. Результат запуску скраперу Google

Запит на сервер віддав результати першої сторінки пошукового запиту з усіх доступних скраперів (рис. 3.39). Таким чином, завдання є виконаним.

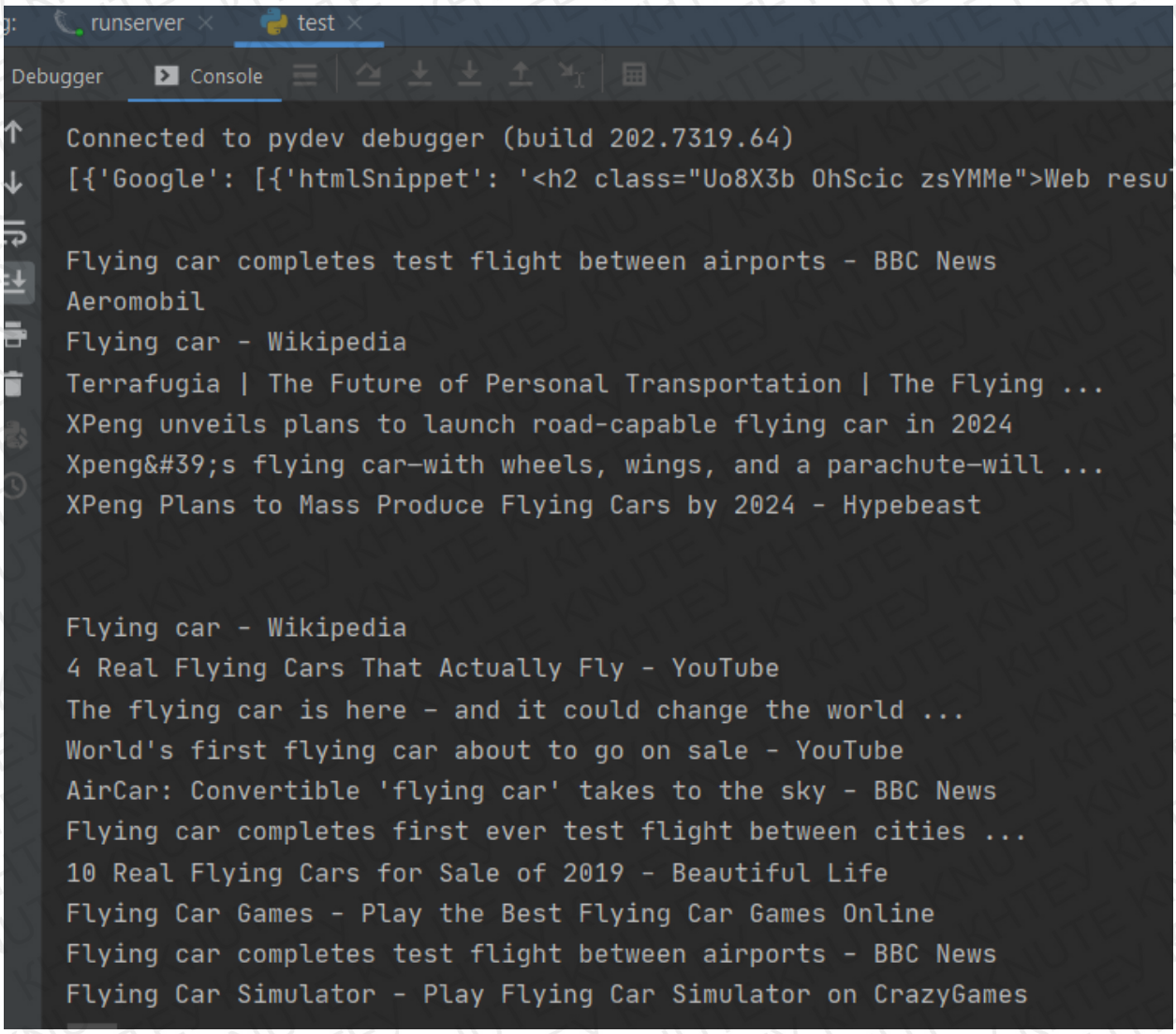


Рис. 3.39. Результат запиту на додаток

Емпіричним шляхом досліджено, що з застосовані методи захисту дали змогу зпарсити перші сторінки результатів, проте при швидкому або частому виконанні запитів Google починає повертати HTML-сторінку з CAPTCHA, й таким чином блокує подальші можливості для скрапінгу результатів.

Враховуючи такі факти, що файл robots.txt має заборону на використання запиту до нейромережі пошуку а також те що Google – корпорація неймовірних масштабів, можна зробити висновок що використовувати Google Search для веб-скрапінгу є задачею досить трудомісткою, для якої є два варіанти рішення:

1) Робити довгі паузи між запитами та виконувати їх синхронно, жертвуючи швидкістю;

2) Створити ProxyPool.

Перше рішення надає перевагу синхронному програмуванню та штучне створення затримок між запитами, що у сумі є затратним у часі процесом, який унеможливлює створення додатку що буде в реальному часі робити запити.

Другий варіант передбачає наявність великої кількості proxy-серверів, почерговим під'єднанням до яких можна запобігти блокування по IP-адресі або видачею CAPTCHA. Проте даний спосіб є ресурсо-затратним та трудомістким.

Віддзеркалення даної ситуації також властиво й пошуковій системі Bing: наявна така ж сама проблема, й такі ж самі рішення.

РЕЗУЛЬТАТИ ТА ВИСНОВКИ

- Вивчено особливості інструментів зворотної інженерії в контексті імплементації сторонніх сервісів;
- Проаналізовано реальний, практичний приклад використання методології Reverse Engineering який можна використовувати в комерційній діяльності;
- Досліджено особливості web-scraping як у глобальному, теоретичному плані, так і на прикладі практичної реалізації використовуючи мову програмування Python;
- Проаналізовано та застосовано на практиці інструменти аналізу ChromeDevTools;
- Здійснено Reverse Engineering пошукових систем Google Search, Google Patents, Bing Search та пошукової системи сайту Інтерполу;
- Розглянуто популярні методи захисту веб-сайтів від ботів та веб-скраперів а також наведено та застосовано деякі приклади їх обходження;
- Створено додаток, що використовує веб-скрапери Google Search та Bing Search в якості основного сервісу серверу.
- Як підсумок, методологія reverse engineering є сукупністю орієнтирів спостереження, які і являють собою інформацію про можливості та обмеження конкретного явища або процесу. Даний вид спостереження можна використовувати у будь-якій сфері діяльності, оскільки поза контекстом IT-індустрії корінь даного явища йде від дедуктивного метода мислення та рішення задач, що передбачає собою лише ерудицію у конкретній сфері та досвід у будуванні логічних ланцюгів.
- Використання мов програмування для автоматизації процесів, потребуючих великі об’єми часу, є надзвичайно поширеною практикою на момент захисту даної роботи. Фактично, будь-який процес яким займається людина, може бути не тільки повністю автоматизовано, а й у

неймовірно великому співвідношенні підвищити ефективність таких процесів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Chikofsky, E.J. & Cross, J.H., II (1990). "Reverse Engineering and Design Recovery: A Taxonomy". IEEE Software. С. 13 - 17
2. Common Uses for Reverse Engineering May Not Be As Sinister As You Expect [Електронний ресурс]. – Режим доступу: https://www.arlingtoncardinal.com/2020/06/common-uses-for-reverse-engineering-may-not-be-as-sinister-as-you-expect/
3. DevTools overview [Електронний ресурс]. – Режим доступу: https://developer.chrome.com/docs/devtools/network/reference/
4. Web scraping [Електронний ресурс]. – Режим доступу: https://www.zyte.com/learn/what-is-web-scraping/
5. Web Scraping [Електронний ресурс]. – Режим доступу: https://www.imperva.com/learn/application-security/web-scraping-attack/
6. Web scraping protection: How to protect your website against crawler and scraper bots [Електронний ресурс]. – Режим доступу: https://datadome.co/bot-management-protection/scraper-crawler-bots-how-to-protect-your-website-against-intensive-scraping/
7. Web scraping bots are 46% of web traffic [Електронний ресурс]. – Режим доступу: https://www.enterprisetimes.co.uk/2016/08/31/web-scraping-bots-46-web-traffic/
8. Retail e-commerce sales worldwide from 2014 to 2024 [Електронний ресурс]. – Режим доступу: https://www.statista.com/statistics/379046/worldwide-retail-e-commerce-sales/
9. Асинхронное программирование [Електронний ресурс]. – Режим доступу: https://light-electric.com/yazyki-programmirovaniya/asinhronnoe-programmirovanie-eto.html
10. DevTools для «чайников» [Електронний ресурс]. – Режим доступу: https://habr.com/en/post/548898/

11. ChromeDevTools [Електронний ресурс]. – Режим доступу: https://developer.chrome.com/docs/devtools
12. Coroutines in Python [Електронний ресурс]. – Режим доступу: https://betterprogramming.pub/coroutines-in-python-building-blocks-of-asynchronous-programming-40c39d9ed420
13. Асинхронность в Python: как Twitter обрабатывает миллиарды сеансов в день [Електронний ресурс]. – Режим доступу: https://highload.today/asinhronnost-v-python-kak-twitter-obrabatyvaet-milliardy-seansov-v-den/
14. Coroutines and Tasks [Електронний ресурс]. – Режим доступу: https://docs.python.org/3/library/asyncio-task.html
15. Методы защиты от веб скрапинга [Електронний ресурс]. – Режим доступу: https://bit.ly/31tQSgJ
16. Introduction to robots.txt [Електронний ресурс]. – Режим доступу: https://developers.google.com/search/docs/advanced/robots/intro
17. MVC [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/en-US/docs/Glossary/MVC
18. Build fast, responsive sites with Bootstrap [Електронний ресурс]. – Режим доступу: https://getbootstrap.com/
19. What is Unified Modeling Language (UML)? [Електронний ресурс]. – Режим доступу: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/
20. What is Sequence Diagram? [Електронний ресурс]. – Режим доступу: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-sequence-diagram/
21. Welcome to Flask’s documentation [Електронний ресурс]. – Режим доступу: https://flask.palletsprojects.com/en/2.0.x/
22. beautifulsoup4 4.10.0 [Електронний ресурс]. – Режим доступу: https://pypi.org/project/beautifulsoup4/