Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: “Розробка додатку для автоматичного утворення Google Merchant Feed та збору даних з веб-сторінок”

Виконав(ла): студент(ка) IV курсу, групи СП-41
спеціальності 121 “Інженерія програмного забезпечення”
(шифр і назва спеціальності)

		Караванський В.В.
	(підпис)	(прізвище та ініціали)
Керівник		Петрик М.Р.
	(підпис)	(прізвище та ініціали)
Нормоконтроль		Стоянов Ю.М.
	(підпис)	(прізвище та ініціали)
Завідувач кафедри		Петрик М.Р.
	(підпис)	(прізвище та ініціали)
Рецензент		
	(підпис)	(прізвище та ініціали)

Тернопіль
2024

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра програмної інженерії
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри

(підпис) (прізвище та ініціали)

« »
20___ р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавр
(назва освітнього ступеня)

за спеціальністю 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

студенту Караванському Владиславу Володимировичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка додатку для автоматичного утворення Google Merchant Feed та збору даних з веб-сторінок

Керівник роботи д. ф. - м. н., професор Петрик Михайло Романович
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «_____» _____ 20___ року №

2. Термін подання студентом завершеної роботи

3. Вихідні дані до роботи Успішний та безперебійний збір даних з веб-сторінок та формування з них Google Merchant Feed.

4. Зміст роботи (перелік питань, які потрібно розробити)
Словник предметної області, аналіз вимог до системи, опис методології проєктування та технологій системи, проєктування відношень між акторами і прецедентами, виявлення класів сутності, проєктування програмного забезпечення, визначення класів системи, опис роботи системи, розгортання, тестування та верифікація вимог, безпека життєдіяльності та основи охорони праці.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
Діаграми: типовий приклад використання системи, діаграма варіантів використання, діаграма класів, діаграма послідовності. Рисунки: скріншоти з результатами виконання завдання.

6. Консультанти розділів роботи

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

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка

Студент _____ _____
(підпис) (прізвище та ініціали)

Керівник роботи _____ _____
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Розробка додатку для автоматичного утворення Google Merchant Feed та збору даних з веб-сторінок // Кваліфікаційна робота освітнього рівня “Бакалавр” // Караванський Владислав Володимирович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп’ютерно-інформаційних систем і програмної інженерії, кафедра програмної інженерії, група СП-41 // Тернопіль, 2024 // С. 89, рис. – 25, табл. – 1 , додат. – 3, бібліогр. – 16.

Ключові слова: Python, Google Cloud, Google Merchant Feed, Visual Studio Code, Gitlab, BigQuery, веб-сайт, дані, парсинг, скрапінг, база даних, лістинг, діаграма.

В першому розділі кваліфікаційної роботи було створено словник системи, досліджено актуальність завдання, проаналізовано вимоги до системи, описано методології проєктування та технологій системи, проєктовано відношення між акторами і прецедентами, виявлено класи сутності.

В другому розділі кваліфікаційної роботи було визначено класи системи та описано роботу системи.

В третьому розділі кваліфікаційної роботи було проведено процес тестування застосунку.

ANNOTATION

Bachelor's qualification work on the topic “Development of an application for automatic generation of Google Merchant Feed and data collection from web pages” // Karavansky Vladyslav Volodymyrovych // Ternopil Ivan Puluj National Technical University, Faculty Of Computer Information Systems And Software Engineering, 121 Software Engineering, group SP-41 // Ternopil, 2024 // pages – 89, figures – 25, tables – 1, appendices – 3, bibliography – 16.

Keywords: Python, Google Cloud, Google Merchant Feed, Visual Studio Code, Gitlab, BigQuery, website, data, parsing, scraping, database, listing, chart.

In the first section of the qualification work, a dictionary of the system was created, the relevance of the task was investigated, requirements for the system were analyzed, design methodologies and system technologies were described, the relationship between actors and precedents was designed, and entity classes were identified.

In the second section of the qualification work, the classes of the system were defined and the operation of the system was described.

In the third section of the qualification work, the application testing process was described.

ЗМІСТ

ВСТУП 7
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ 9
1.1. Словник предметної області 9
1.2. Огляд предметної області 11
1.3. Аналіз вимог до системи 12
1.4. Опис методології проєктування та технологій системи 16
1.5. Проєктування відношень між акторами і прецедентами 19
1.6. Виявлення класів сутності 22
2 ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 24
2.1. Визначення класів системи 24
2.2. Опис роботи системи 32
3 РОЗГОРТАННЯ, ТЕСТУВАННЯ ТА ВЕРИФІКАЦІЯ ВИМОГ 35
3.1. Процес тестування 35
4 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ ТА ОСНОВИ ОХОРОНИ ПРАЦІ 43
4.1. Соціальні та психологічні фактори ризику 43
4.2. Гігієнічні вимоги до організації та обладнання робочих місць з ВДТ 46
ВИСНОВОК 49
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 50
ДОДАТКИ 52
ДОДАТОК А 53
ДОДАТОК Б 54
ДОДАТОК В 89

ВСТУП

Обсяг даних зростає швидше, ніж будь-коли. 90% даних у світі було створено за останні два роки. І кожні два роки він подвоюється у всьому світі.

Людство має необмежений доступ до величезної кількості різноманітних та унікальних джерел даних. В нас час уже немає поняття “відсутності інформації”. Вона є всюди, де тільки можна зазирнути на просторах всесвітньої мережі інтернет, проте не кожен вміє її знайти. Адже дані знаходяться не лише на видному місці, де їх заклали розробники веб-сайтів.

Візуальна складова – це лише частина всього того, що приховує в собі веб-сторінка. Насправді ж, велика кількість корисної інформації переховується у вихідному коді, скриптах, запитах до API інтерфейсів, метаданих та багатьох інших джерелах даних, які не є інтуїтивно зрозумілими звичайному користувачу. Проте там, де пересічна людина бачить незрозумілий код та купу “страшних” елементів, з якими нічого не може вдіяти, людина сфери Web Data Scraping бачить велику кількість можливостей для того, що зібрати дані, які потім можна буде використовувати як в аналітичних, так і бізнес цілях. Можливості є необмеженими, адже якщо відомо, що якісь дані існують, то зупиняє професіонала в цій сфері лише поріг власних можливостей, які можна покращувати протягом усього свого життя.

Сучасна ера цифрової трансформації вимагає від нас не тільки вміння швидко адаптуватися до нових технологій, але й ефективно використовувати можливості, які вони надають. Величезний обсяг даних, що постійно зростає, відкриває нові горизонти для досліджень та інновацій. Розуміння та аналіз цих даних дозволяє бізнесам приймати обґрунтовані рішення, прогнозувати ринки та тренди, а також оптимізувати операційні процеси.

В умовах, коли конкуренція на ринку стає все більш жорсткою, можливість швидко отримувати та аналізувати дані з різноманітних джерел стає вирішальним фактором успіху. Веб скрапінг дозволяє збирати інформацію з тисяч веб-сторінок

за лічені хвилини, забезпечуючи доступ до найактуальніших даних у реальному часі. Цей підхід відкриває безмежні можливості для бізнес-розвідки, маркетингових досліджень, моніторингу конкурентів та багатьох інших сфер.

Однак, зі збільшенням обсягів даних зростає і складність їх обробки та аналізу. Необхідно не лише вміти отримувати дані, але й ефективно їх структурувати, очищати та аналізувати. Використання передових методів машинного навчання та штучного інтелекту стає незамінним інструментом у процесі обробки великих обсягів даних. Автоматизація процесів збору та аналізу даних дозволяє значно підвищити продуктивність та точність аналітики.

Водночас, важливо пам’ятати про етичні аспекти та правові норми, пов’язані з використанням скрапінгу. Дотримання законодавства про захист персональних даних та конфіденційності є ключовим елементом у роботі з даними. Забезпечення прозорості та відповідальності у процесі збору та використання інформації є необхідною умовою для підтримки довіри користувачів та партнерів.

Мета роботи: створити застосунок для збирання даних з веб-сторінок, який буде зберігати усе в базі даних Google BigQuery, а також буде створювати, перевіряти та надсилати на FTP сервер сформований згідно вимог Google Merchant Feed.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Словник предметної області

Таблиця 1 – Словник системи

Іменник	Тлумачення
Десктоп-клієнт	Застосунок для персональних комп’ютерів
Sitemap	Файл, у якому надається інформація про сторінки, інші файли на сайті, зв’язки між ними.
Веб-сторінка	Документ в Інтернеті, доступ до якого здійснюється у веб-браузері
Веб-браузер	Програма для доступу до веб-сайтів
Веб-сайт	Сукупність веб-сторінок
Проксі	Проксі-сервер діє як агент або посередник для користувача, пропускаючи всі веб-запити та відповіді
Cookies	Невеликі блоки даних, створені веб-сервером під час перегляду веб-сайту та розміщені на комп’ютері користувача
API (Application Programming Interface)	Інтерфейс прикладного програмування
Метадані	Дані, які надають інформацію про інші дані, але не вміст самих даних

Продовження таблиці 1

Іменник	Тлумачення
Data Scraping/Скрапінг	Процес, під час якого комп’ютерна програма витягує дані з результатів іншої програми
Data parsing/Парсинг	Процес перетворення даних з одного формату на інший
FTP	Стандартний протокол зв'язку, який використовується для передачі файлів від сервера до клієнта
Сервер	Комп'ютер, який надає інформацію іншим комп'ютерам, які називаються “клієнтами”
Клієнт	Комп’ютер або програма, яка покладається на надсилання запиту до іншої програми, і отримує доступ до послуги, наданої сервером
Google Merchant Feed/Фід	Це файл, який містить список продуктів. Потрібен як для наповнення веб-сторінки, так і для реклами.
Логування	Це процес ведення журналу подій, які відбуваються в комп'ютерній системі
Лог	Подія, в журналі подій

1.2. Огляд предметної області

Величезні обсяги вихідної інформації, доступної у Всесвітній павутині, все ще знаходяться у форматі сторінок на мові гіпертекстової розмітки (HTML). Автоматизоване отримання даних є складним процесом, оскільки запланованим споживачем завжди є пересічна людина, а не програміст. Збір даних з онлайн-джерел став одним із методів дослідження ринку. Він пропонує набагато швидший результат, ніж пошук даних вручну.

Великі компанії, в яких є онлайн магазини, та які хочуть надати користувачеві можливість купувати товар з будь якої точки світу та будь якого пристрою, зазвичай надають майже усю необхідно інформацію про товар, щоб задовільнити споживача. Веб-скрапінг є одним із методів збору таких даних. Цільовий збір даних з онлайн магазинів допомагає отримувати необхідні дані чи то для аналізу, чи то в прибуткових цілях. В даному випадку конкретного завдання, все базується на описі товару, його цінах та наявності, останні два з яких часто та динамічно змінюються.

Веб-сайти не завжди надають свої дані у зручних форматах, таких як файли “.csv” або “.json”. HTML-сторінки створюються сервером у відповідь на запит користувача. На цьому етапі програмне забезпечення сервера не має значення, важливий вихід у браузері. За допомогою скрипту на обраній мові програмування або інструменту для веб-скрапінгу, кожну сторінку з однаковим шаблоном в межах одного (інколи декількох веб-сайтів) можна використовувати як джерело даних.

Проте не завжди статична HTML сторінка може містити необхідний набір даних, інколи потрібні дані, такі як: наявність товару, ціна на кожен розмір, унікальний ідентифікаційний номер та інші, приховані набагато глибше. Вони можуть міститися у JavaScript коді, чи взагалі у запитах до API інтерфейсів, якщо такі є доступними.

Стандартний HTTP-запит, надісланий до кінцевої точки API, повертає відповідь із сервера. Кожен API має власну специфікацію та параметри. Формат відповіді може бути встановлений в параметрах запиту. Найпоширенішим форматом відповіді API є JSON.

Проте під час скрапінгу веб-сторінок варто лише сподіватися, що такий варіант, як API, доступний, адже на багатьох сайтах доступ до них зачинений. А на веб-сайтах, де доступ таки є, сподіватися на якусь документацію стосовно формату відповіді, з детальним описом кожного поля, взагалі ніколи не варто. Завжди потрібно співставляти логічні зв’язки між тим, що ти бачиш на веб-сторінці, і між тим, що ти бачиш у відповіді API інтерфейсу.

Стосовно законності автоматизованого отримання даних з веб-сторінок, завжди потрібно отримувати дозвіл від власника веб-сайту, щоб використовувати ці дані у власних цілях.

1.3. Аналіз вимог до системи

Постановка завдання.

Консольний застосунок для запуску процесу скрапінгу даних з обраних веб-сайтів, який збирає дані, обробляє їх, зберігає у базу даних, а потім формує з цих даних Google Merchant Feed, який буде зберігатися на FTP сервері веб-сайту.

Функціональні вимоги.

Додаток передбачає функції, які будуть створювати для користувача, який по сумісництву відповідає за оновлення даних на веб-сайті, зручний набір для керування цим процесом, а саме:

1. Вибір веб-сайтів для скрапінгу.
2. Глибоке налаштування параметрів скрапінгу.
3. Запуск скрапінгу та його виконання завдяки написаному алгоритму.

4. Логування усього процесу витягання даних з веб-сайту.
5. Відображення логів для контролю над успіхом процесу.
6. Якісна автоматична обробка отриманих даних, їх валідація згідно наданих вимог.
7. Зберігання та оновлення зібраних даних у базі даних.
8. Автоматична система виявлення відсутності успіху скрапінгу.
9. Експорт та конвертування даних з бази даних в фід формату XML.
10. Завантаження конвертованих даних на FTP сервер.
11. Зупинка процесу скрапінгу.

В процесі розробки програмного забезпечення наведений перелік функціональних можливостей може бути розширений та доповнений.

Можливі точки розширення:

1. Інтеграція API інтерфейсу, яка дозволить оновлювати дані веб-сайту одразу під час того, як відбувається скрапінг, не використовуючи фіди, як джерело даних.
2. Можливість обирати декілька сайтів для скрапінгу одночасно, додавши систему потоків.
3. Повноцінний користувацький інтерфейс.
4. Додавання системи аутентифікації користувачів.
5. Розширення системи авторизації для керування доступом до різних функцій додатку на основі ролей користувачів.
6. Додавання можливості експорту даних у формати JSON, CSV, Excel.
7. Додавання підтримки для скрапінгу JavaScript-згенерованого контенту за допомогою таких інструментів, як Selenium.
8. Інтеграція зі службами сповіщень (електронна пошта, SMS, месенджери) для повідомлень про успішність або проблеми під час скрапінгу.
9. Створення системи автоматичних завдань для скрапінгу за розкладом.
10. Інтеграція з сервісами машинного навчання для автоматичного аналізу та покращення якості даних.

Вихідними даними є Google Merchant Feed з усією необхідною інформацією про товари.

Нефункціональні вимоги.

1. Надійність:

- зручність використання. Застосунок повинен володіти зручним та інтуїтивним для користувача консольним інтерфейсом;
- постійна робота. Застосунок повинен працювати постійно та безперебійно на фоні, як додатковий процес.
- правильні дані. Отримані дані не повинні бути пошкодженими, необробленими чи “сирими”. Кожне поле повинно бути перевіреним та відповідати вимогам.

2. Ефективність:

- час відгуку. Запит до однієї сторінки та обробка даних на ній не повинен перевищувати 5 секунд;
- високий відсоток успішних запитів. На 50 паралельних запитів не повинно бути більше 10-ти помилок.

3. Придатність до експлуатації:

- масштабованість. Застосунок не повинен займати велику кількість оперативної пам’яті, навіть при умові обробки великого об’єму даних.

4. Обмеження проєктування:

- вимоги до середовища виконання. Програмний засіб повинен коректно та швидко функціонувати на персональних комп’ютерах з ОС Windows/MacOS/Linux.

У цій архітектурі вся система організована в такі рівні:

1. Рівень програми (Back end).
2. Рівень презентації (Front end)

Вимоги до захисту та доступу.

Враховуючи наведені вище особливості роботи програмного забезпечення можна сформувати наступні вимоги до захисту:

1. Доступ користувачів до застосунку повинен здійснюватися завдяки його попередній інсталяції на персональний комп'ютер з ОС Windows/MacOS/Linux.

2. Застосунок повинен коректно реагувати на усі можливі виключні ситуації.

Вимоги до інтерфейсу системи.

Перелік вимог до інтерфесу програмного забезпечення:

- програма повинна коректно відображатися у системах Windows/MacOS/Linux, як консольний додаток;
- інтерфейс повинен бути мінімалістичним, зрозумілим та функціональним;
- користувач повинен знати всі свої можливі дії;
- налаштування параметрів скрапінгу повинно бути простим та ефективним;
- логи повинні бути організованими, чіткими, мінімалістичними та читабельними.

Вимоги до програмного забезпечення:

Операційні системи: Windows/MacOS/Linux.

Доступ до інтернету.

1.4. Опис методології проєктування та технологій системи

Була обрана методологія аналізу вимог і гнучкий процес розробки ПЗ ICONIX, що ґрунтується на варіантах використання. ICONIX надає більше вимог та проєктної документації і прагне уникнути паралічу аналізу. Процес ICONIX використовує лише чотири діаграми на основі UML у чотири-етапному процесі, який перетворює текст прецеденту на робочий код.

Принциповою відмінністю ICONIX є використання аналізу надійності, методу подолання розриву між аналізом та проєктуванням. Аналіз стійкості зменшує неоднозначність в описах варіантів використання, гарантуючи, що вони написані в контексті предметної області, що супроводжується моделями. Цей процес значно спрощує проєктування, тестування та оцінку варіантів використання.

Процес ICONIX поділено на чотири етапи. На кожному етапі робота з попередньої віхи переглядається та оновлюється.

Етап 1: Огляд вимог. Перед початком процесу ICONIX необхідно провести певний аналіз вимог. На основі цього аналізу можна визначити варіанти використання, створити модель предметної області та створити кілька прототипів графічних інтерфейсів.

Етап 2: Попередній огляд проєкту. Після визначення варіантів використання можна написати текст, що описує взаємодію користувача та системи. Виконується аналіз надійності, щоб знайти потенційні помилки у тексті варіанта використання, та відповідним чином оновлюється модель предметної області. Текст прецеденту важливий для визначення того, як користувачі будуть взаємодіяти з системою. Вони також надають розробнику перевірити правильність результатів аналізу вимог.

Етап 3: Детальний аналіз проєкту. На цьому етапі процесу ICONIX модель предметної області та текст прецеденту з етапу 2 використовуються для проєктування системи, що будується. Діаграма класів створюється з

моделі предметної області, а текст варіантів використання використовується створення діаграм послідовності.

Етап 4: Розгортання. Модульні тести написані для перевірки того, що система відповідатиме тексту варіанта використання та діаграм послідовності. Нарешті, код пишеться з використанням діаграм класів та послідовностей в якості настанови.

Ця методологія була обрана тому, що дозволить швидко та ефективно виконати задачі поставлені перед розробником проєкту, йому не доведеться витратити більшу частину свого часу на формування діаграм, він зможе якісно описати усі вимоги та спроєктувати систему згідно них, після чого протестувати її. Ця методологія не є найлегшою, проте знаходиться в золотій середині між швидкістю та ефективністю. Після того як проєкт пройде через усі чотири необхідні етапи, він буде завершеним та готовим до експлуатації.

Для побудови застосунку було обрано Visual Studio Code приєднаний до Gitlab.

Gitlab – це веб-сховище Git, яке надає безкоштовні відкриті та приватні сховища, можливості відстеження проблем і вікі. Це платформа DevOps, яка дозволяє професіоналам виконувати всі завдання в проєкті – від планування проєкту та керування вихідним кодом до моніторингу та безпеки. Крім того, це дозволяє командам співпрацювати та створювати краще програмне забезпечення.

Visual Studio Code, також відомий як VS Code, – це редактор вихідного коду, розроблений Microsoft для Windows, Linux, macOS і веб-браузерів. Функції включають підтримку налагодження, підсвічування синтаксису, інтелектуальне завершення коду, рефакторинг коду та вбудований контроль версій за допомогою Git. Він містить базову підтримку для більшості поширених мов програмування. Ця базова підтримка включає підсвічування синтаксису, зіставлення дужок, згортання коду та настроювані фрагменти. Visual Studio Code також постачається з IntelliSense для JavaScript, TypeScript, JSON, CSS і HTML, а також підтримує налагодження Node.js. Підтримку

додаткових мов можна отримати за допомогою безкоштовних розширень на VS Code Marketplace [1].

Це середовище було обране в першу чергу тому, що додаток буде написаний мовою програмування Python і буде мати лише консольний інтерфейс. Працювати з проєктами такого масштабу у цьому середовищі найзручніше, особливо коли справа стосується роботи з веб-сайтами, адже воно має вбудовану підтримку JavaScript, JSON та HTML, що є вкрай необхідним інструментарієм під час скрапінгу даних з веб-сторінок.

Також вбудований контроль версій за допомогою Git є неабиякою перевагою середовища перед конкурентами, адже дозволяє використовувати систему контролю версій в тому ж місці, де і відбувається розробка, що сильно підвищує швидкість, ефективність та надійність процесу.

Для написання застосунку у Visual Studio Code було обрано високорівневу мову програмування Python.

Python є багатопарадигмальною мовою програмування. Об’єктно-орієнтоване програмування та структуроване програмування повністю підтримуються, і багато їхніх функцій підтримують функціональне програмування та аспектно-орієнтоване програмування. Багато інших парадигм підтримуються за допомогою розширень. Python був розроблений таким чином, щоб бути розширюваним за допомогою модулів. Ця компактна модульність зробила його особливо популярним як засіб додавання програмованих інтерфейсів до існуючих програм. Python стверджує, що прагне до простішого, менш захараченого синтаксису та граматики, надаючи розробникам можливість вибору методології кодування [2].

Це ідеальна мова програмування для скрапінгу, адже дозволяє дуже швидко та динамічно вирішувати велику кількість проблем, які можу виникнути під час обробки веб-сторінок. В нагоді стає велика кількість бібліотек, багато з яких є офіційними бібліотеками великих компаній, таких як Google. Вона також дозволяє легко працювати з веб-сторінками, завдяки

бібліотеці BeautifulSoup та нативною підтримкою JSON файлів, а також при створенні фідів нативна підтримка XML файлів є дуже корисною.

Як база даних була обрана Google BigQuery. BigQuery – це дуже гнучка платформа аналітики даних із підтримкою штучного інтелекту, яка допомагає отримати максимальну користь від даних і розроблена для роботи з різними форматами. BigQuery надає єдиний уніфікований робочий простір, який включає SQL, логування та інтерфейс Canvas на основі NL для спеціалістів із обробки даних, які володіють різними навичками кодування, щоб спростити аналітичні робочі процеси від прийому та підготовки даних до дослідження та візуалізації даних до створення та використання моделі ML. Безсерверна архітектура BigQuery дозволяє використовувати запити SQL для аналізу даних [3].

Ця база даних зручна тим, що легко піддається масштабуванню і динамічно формує ціну за використання сховища. Ти платиш тільки за ті дані, якими користуєшся. Також у Python є бібліотека від Google для роботи з цією базою даних, яка сильно спрощує комунікацію з BigQuery.

1.5. Проєктування відношень між акторами і прецедентами

В описаній системі виділено два актори:

1) Користувач – може обирати сайти зі списку, запускати процес їх скрапінгу, зупиняти його, слідувати за ним, а також виставляти його параметри, задля покращення його ефективності.

2) База даних – зберігає та оновлює дані з веб-сайтів, а також повертає їх у вигляді сформованих XML фідів. Вона також зберігає та оновлює логи, які утворюються під час скрапінгу.

Типовий приклад використання продемонстрований на рисунку 1.

Рисунок 1 – Типовий приклад використання

Варіанти використання системи:

1. Вибір веб-сайтів для скрапінгу.

Користувач буде мати меню з доступними на даний момент веб-сайтами для веб-скрапінгу. Він успішно обирає веб-сайт.

2. Глибоке налаштування параметрів скрапінгу.

Користувач зможе обирати параметри, які будуть впливати на поведінку скрапінгу веб-сайту. Він успішно зберігає обрані налаштування скрапінгу.

3. Запуск скрапінгу та його виконання завдяки написаному алгоритму.

Користувач підтверджує, що готовий запустити скрапінг та запускає його. Скрапінг успішно запускається.

4. Логування усього процесу витягання даних з веб-сайту.

Цей процес відбувається автоматично. Під час скрапінгу усі успішні та неуспішні запити записуються.

5. Відображення логів для контролю над успіхом процесу.

Користувач може обрати, чи відображати йому логи. Якщо він обрав відображення, то під час скрапінгу він буде бачити всю необхідну інформацію для відслідковування помилок.

6. Якісна автоматична обробка отриманих даних, їх валідація згідно наданих вимог.

Цей процес відбувається автоматично, якщо користувач обрав такий варіант в налаштуваннях. Відбувається успішний скрапінг, під час якого дані ретельно шукаються та перевіряються.

7. Зберігання та оновлення зібраних даних у базі даних.

Цей процес відбувається автоматично, якщо користувач обрав такий варіант в налаштуваннях. Отримані дані успішно зберігаються у базі даних

8. Автоматична система виявлення відсутності успіху скрапінгу.

Цей процес відбувається автоматично. Застосунок визначає відсоток успішних записів, і якщо він низький, то процес скрапінгу зупиняється.

9. Експорт та конвертування даних з бази даних в фід формату XML.

Після вибору веб-сайту, або після завершення скрапінгу, користувач може обрати опцію експорту даних з бази даних у Google Merchant Feed формату XML. Фід успішно формується.

10. Завантаження конвертованих даних на FTP сервер.

Якщо користувач обрав опцію створити фід, то йому буде запропоновано завантажити конвертовані дані на FTP сервер. Після згоди, файли успішно завантажуються на виділений сервер.

11. Зупинка процесу скрапінгу.

Користувач може в будь який момент зупинити процес скрапінгу комбінацією клавіш. Він успішну зупинить його виконання та повернеться до головного меню.

Для кращого візуального сприйняття зв’язків між акторами та варіантами використанням можна створити діаграму варіантів використання. Вона продемонстрована на рисунку 2.

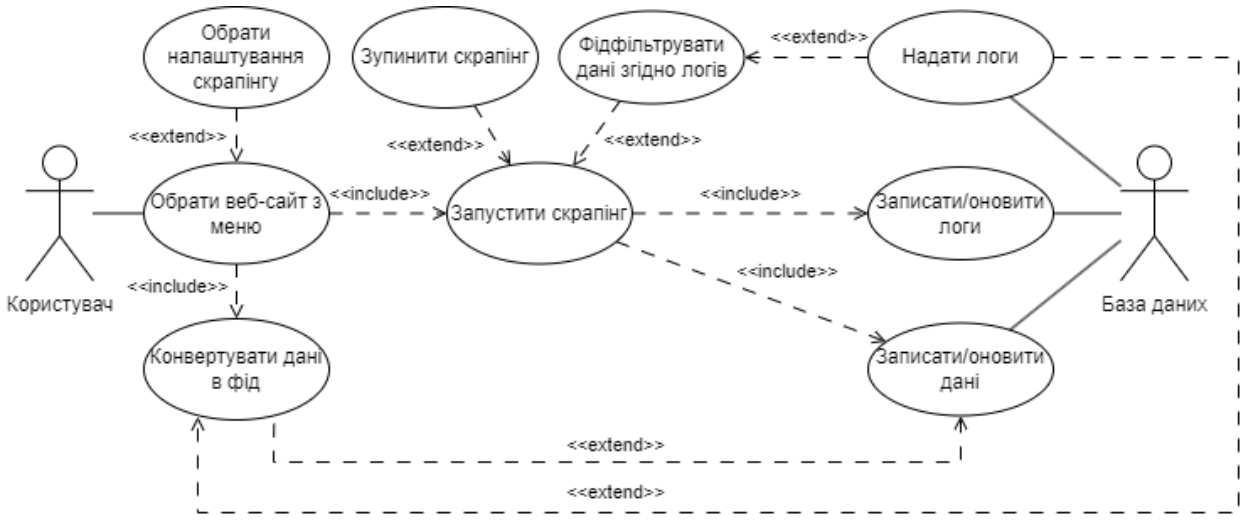


Рисунок 2 – Діаграма варіантів використання

1.6. Виявлення класів сутності

Проаналізувавши опис даної системи, з урахуванням точок розширення, ми можемо виявити наступні сутності предметної області.

Сутність Scraper – призначена для виконання процесу скрапінгу даних. Це одна з основних сутностей, що містить у собі основні методи для роботи зі скрапінгом, без яких цей процес був би неможливим. Вона також зберігає в собі усіх необхідні дані, яких потребує процес веб-скрапінгу. Без цих даних він був би неможливим.

Сутність Parameters – призначена для збереження параметрів. Вона містить у собі усі необхідні поля, що дозволяються усім іншим сутностям успішно комунікувати між собою і динамічно змінювати значення усіх параметрів за потреби. Таким чином усі сутності мають доступ до загального переліку параметрів і кожна з них може впливати на них, що йде на користь продуктивності та ефективності скрапінгу.

Сутність BigQuery_Handler – призначена для роботи з базою даних Google BigQuery. Вона містить у собі усі необхідні поля та методи, що дозволяють класу

Scraper записувати в BigQuery дані та отримувати їх. Також вона відповідає за конвертування даних у фід необхідного формату.

Сутність Helper – призначена для усіх додаткових операцій, які стосуються скрапінгу. Ці операції є опціональними, проте покращують ефективність та оптимізовують наявний алгоритм скрапінгу завдяки додатковим полям та методам, які доповнюють наявний функціонал.

Сутність Proxy_Handler – призначена для роботи з проксі. Вона містить у собі усі необхідні поля та методи, що дозволяють класу Scraper користуватися проксі. В загальному надає їх динамічно або статично, коли це потрібно.

2 ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1. Визначення класів системи

Маючи всі відомості про систему з першого розділу, можна приступити до проєктування архітектури системи, а саме визначення, якими класами буде володіти система. Визначивши всі класи системи побудуємо діаграму класів за допомогою мови UML.

1) Scraper – основний клас, в якому зберігаються усі необхідні поля та методи для скрапінгу веб-сторінок. Він також містить в собі по екземпляру кожного з решти класів в якості атрибутів.

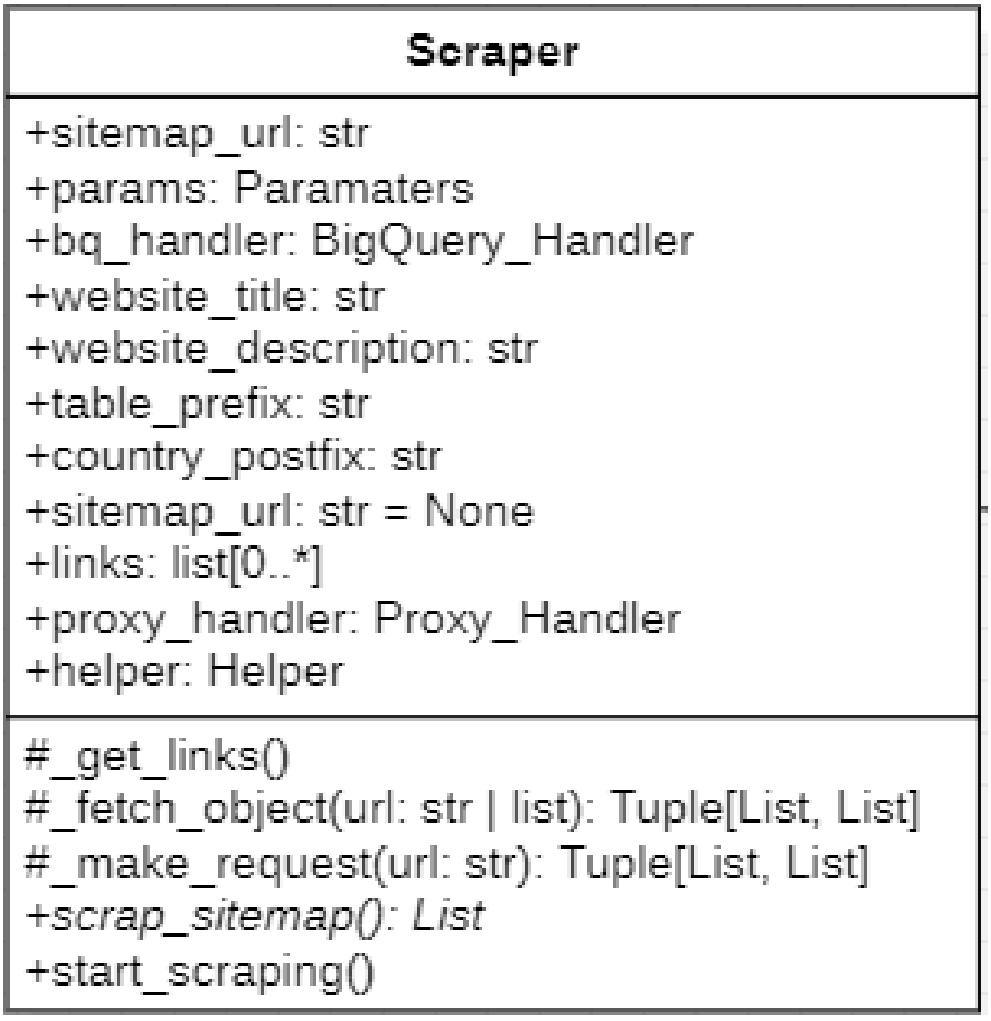


Рисунок 3 – Клас “Scraper”

- sitemap_url: str – посилання на sitemap веб-сайту. За замовчуванням має значення None;
- params: Paramaters – сутність класу Paramaters, в якій зберігаються усі параметри, які визначають поведінку скрапінгу;
- bq_handler: BigQuery_Handler – сутність класу BigQuery_Handler, яка виконує усі операції, пов’язані з базою даних Google BigQuery;
- website_title: str – назва веб-сайту;
- website_description: str – опис веб-сайту;
- website_link: str – посилання на веб-сайт;
- table_prefix: str – назва таблиці в Google BigQuery;
- country_postfix: str – код країни веб-сайту. За замовчуванням має значення “unspecified”;
- links: list[0..*] – набір посилань для скрапінгу;
- proxy_handler: Proxy_Handler – сутність класу Proxy_Handler для роботи з проксі;
- helper: Helper – сутність класу Helper для додаткових операцій скрапінгу;
- _get_links() – функція для ітерації по отриманих посиланнях з метою отримання з них даних;
- _fetch_object(in url:str | list): Tuple[List, List] – функція для обробки отриманого посилання;
- _make_request(in url:str): Tuple[List, List] – функція для виконання запиту до посилання та отримання даних з нього;
- *scrap_sitemap(): List*: – функція для отримання посилань з sitemap веб-сайту;
- start_scraping() – функція, яка запускає процес скрапінгу веб-сайту.

2) Parameters – клас, який представляє набір параметрів для скрапінгу веб-сайту.

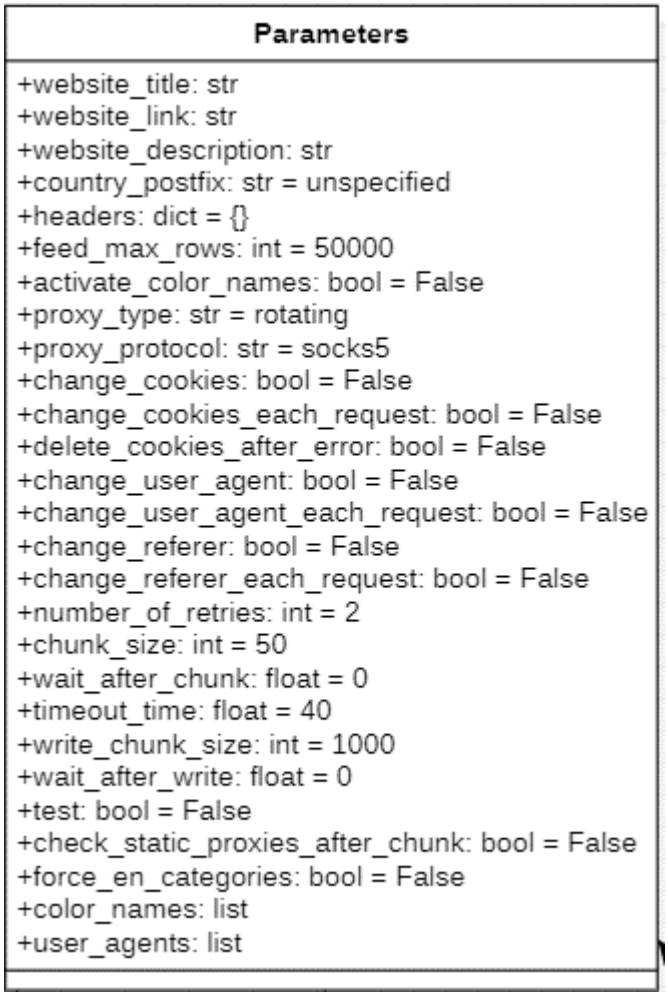


Рисунок 4 – Клас “Parameters”

- website_title: str – назва веб-сайту;
- website_link: str – посилання на веб-сайт;
- website_description: str – опис веб-сайту;
- table_prefix: str – назва таблиці в Google BigQuery;
- country_postfix: str – код країни веб-сайту. За замовчуванням має значення “unspecified”;
- headers: dict = {} – зміна в якій зберігаються заголовки запиту;
- feed_max_rows: int = 50000 – максимальна кількість рядків в одному фіді. Значення по замовчуванню рівне 50000;

- activate_color_names: bool = False – чи активовувати власну систему кольорів товару. Значення по замовчуванню рівне False;
- proxy_type: str = rotating – тип проксі. Значення по замовчуванню рівне “rotating”;
- proxy_protocol: str = socks5 – протокол проксі. Значення по замовчуванню рівне “socks5”;
- change_cookies: bool = False – зміна cookies. Значення по замовчуванню рівне False;
- change_cookies_each_request: bool = False – зміна cookies кожного запиту. Значення по замовчуванню рівне False;
- delete_cookies_after_error: bool = False – видаляти cookies після невдалої спроби запиту. Значення по замовчуванню рівне False;
- change_user_agent: bool = False – зміна агенту користувача. Значення по замовчуванню рівне False;
- change_user_agent_each_request: bool = False – зміна агенту користувача кожен запит. Значення по замовчуванню рівне False;
- change_referer: bool = False – зміна джерела звертання. Значення по замовчуванню рівне False;
- change_referer_each_request: bool = False – зміна джерела звертання кожен запит. Значення по замовчуванню рівне False;
- number_of_retries: int = 2 – кількість повторних запитів. Значення по замовчуванню рівне 2;
- chunk_size: int = 50 – кількість одночасних запитів. Значення по замовчуванню рівне 50;
- wait_after_chunk: float = 0 – пауза після виконання одночасних запитів. Значення по замовчуванню рівне 0;
- timeout_time: float = 40 – час очікування відповіді від сервера. Значення по замовчуванню рівне 40;

- write_chunk_size: int = 1000 – кількість запитів, після якої дані запишуться в базу даних. Значення по замовчуванню рівне 1000;
- wait_after_write: float = 0 – пауза після виконання запису в базу даних. Значення по замовчуванню рівне 0;
- test: bool = False – запити є тестовими. Значення по замовчуванню рівне False;
- check_static_proxies_after_chunk: bool = False – якщо проксі статичні, перевіряти їх працездатність. Значення по замовчуванню рівне False;
- force_en_categories: bool = False – використовувати англійські категорії товару, незалежно від країни. Значення по замовчуванню рівне False;
- color_names: list – список власних кольорів товару;
- user_agents: list – список агентів користувачів.

3) Proxy_Handler – клас для роботи з проксі.

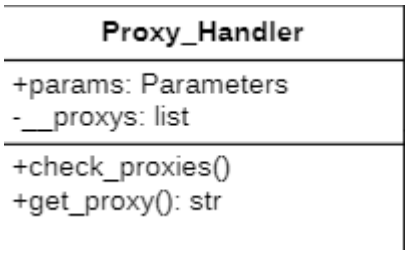


Рисунок 5 – Клас “Proxy_Handler”

- params: Paramaters – сутність класу Paramaters, в якій зберігаються усі параметри, які визначають поведінку скрапінгу;
- __proxys: list – список проксі;
- check_proxies() – метод для перевірки проксі;
- get_proxy(): str – метод для отримання проксі.

4) BigQuery_Handler – клас для роботи з базою даних Google BigQuery.

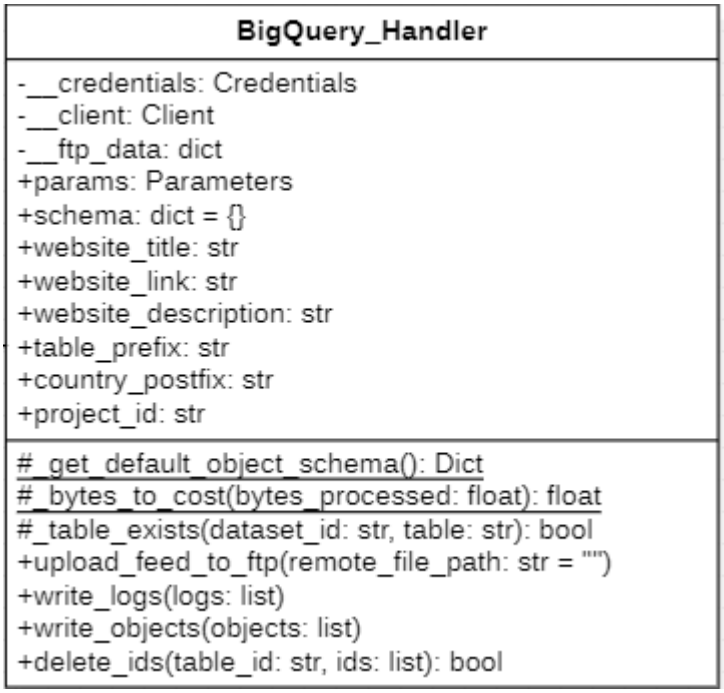


Рисунок 6 – Клас “BigQuery_Handler”

- __credentials: Credentials – сутність класу Credentials з даними авторизації до Google Cloud;
- __client: Client – сутність класу Client, яка є клієнтом Google Cloud;
- __ftp_data: dict – дані авторизації до FTP сервера;
- params: Parameters – сутність класу Paramaters, в якій зберігаються усі параметри, які визначають поведінку скрапінгу;
- schema: dict = {} – схема таблиці для BigQuery;
- website_title: str – назва веб-сайту;
- website_link: str – посилання на веб-сайт;
- website_description: str – опис веб-сайту;
- table_prefix: str – назва таблиці в Google BigQuery;
- country_postfix: str – код країни веб-сайту. За замовчуванням має значення “unspecified”;
- project_id: str – ідентифікатор проєкту в BigQuery;

- _get_default_object_schema(): Dict – метод для отримання стандартної схеми таблиці в BigQuery;
- _bytes_to_cost(in bytes_processed:float): float – метод конвертування байтів обробки даних BigQuery в ціну запиту;
- _table_exists(in dataset_id:str, in table:str): bool – метод для перевірки, чи таблиця в BigQuery існує;
- upload_feed_to_ftp(in remote_file_path:str = “”) – метод конвертації даних з BigQuery в фід та завантаження їх на FTP сервер;
- write_logs(in logs:list) – метод запису логів в базу даних;
- write_objects(in objects:list) – метод запису об’єктів в базу даних;
- delete_ids(in table_id:str, in ids:list): bool – метод для дедуплікації даних в таблиці.

5) Helper – клас з допоміжними методами для скрапінгу даних з веб-сайтів.

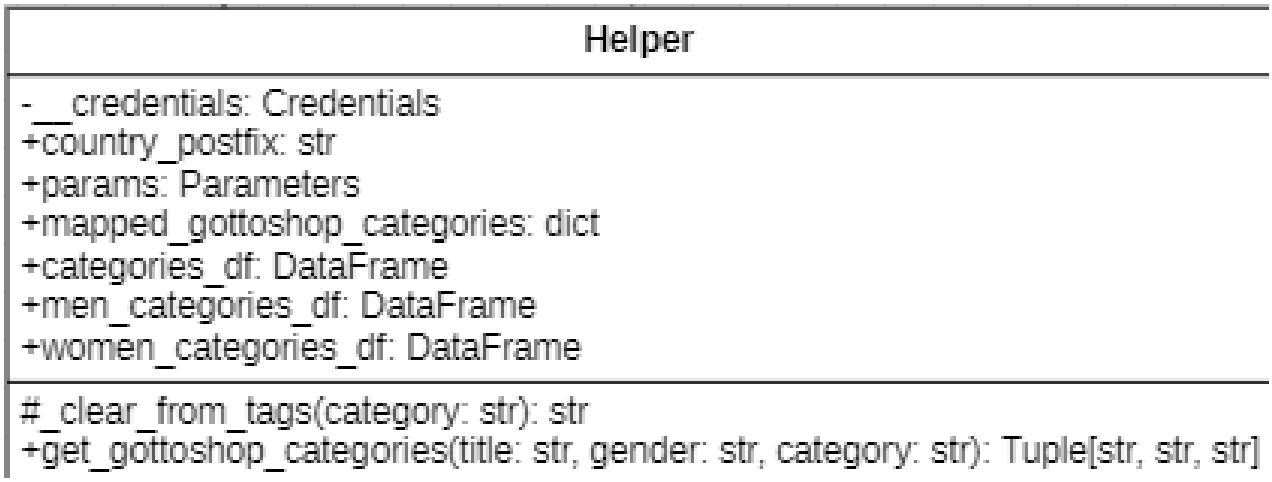


Рисунок 7 – Клас “Helper”

- __credentials: Credentials – сутність класу Credentials з даними авторизації до Google Cloud;
- country_postfix: str – код країни веб-сайту. За замовчуванням має значення “unspecified”;

– params: Parameters – сутність класу Paramaters, в якій зберігаються усі параметри, які визначають поведінку скрапінгу;
– mapped_gottoshop_categories: dict – словник категорій товарів;
– categories_df: DataFrame – DataFrame з повними категоріями товарів;
– men_categories_df: DataFrame – DataFrame з повними чоловічими категоріями товарів;
– women_categories: DataFrame – DataFrame з повними жіночими категоріями товарів.

Побудуємо діаграму класів, на основі усіх вищеописаних класів, яку можна буде використати під час розробки застосунку:

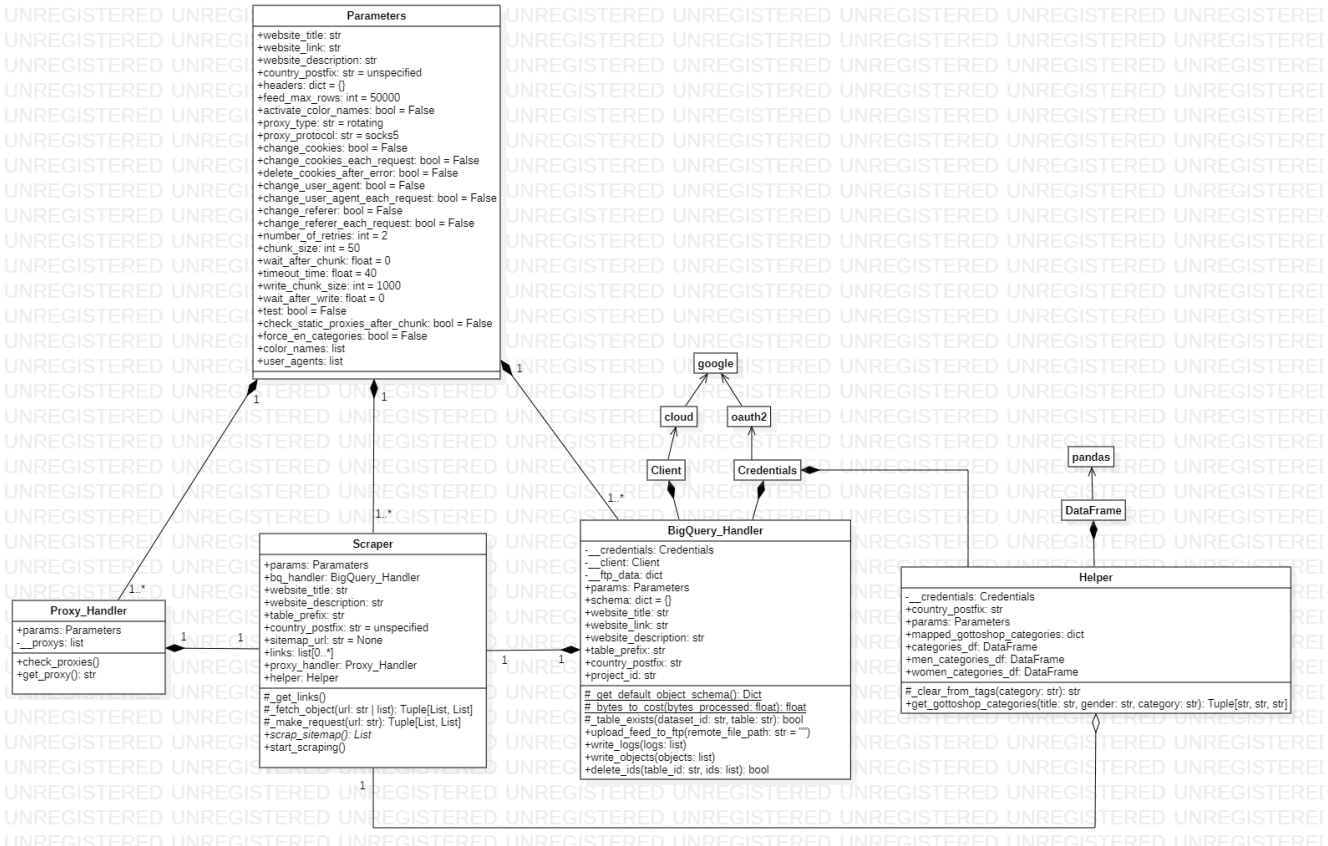


Рисунок 8 – Діаграма класів (Додаток А)

2.2. Опис роботи системи

Розглянемо найскладніші процеси, які будуть відбуватися у застосунку.

Користувач використовує додаток для скрапінгу сайту без зміни параметрів.

Головний сценарій (успішний):

Користувач відкриває додаток. Обирає сайт зі списку доступних сайтів для оновлення. Сайт починає оброблятися додатком. Після завершення скрапінгу, користувачу пропонується конвертувати зібрані дані в фід і завантажити їх на FTP сервер. Користувач погоджується або відмовляється.

Альтернативні сценарії:

1) Під час скрапінгу виникла критична неочікувана помилка. Додаток сповістив про неї.

2) Під час скрпінгу кількість неуспішних запитів перевищила допустиму. Скрапінг призупиняється і користувач обирає, чи продовжити процес, чи зупинити повністю.

3) Після завершення скрапінгу, користувач зробив нерелевантний вибір на запитання про конвертацію даних і їх завантаження на FTP сервер. Додаток сповіщає його про це.

4) Під час конвертації даних виникла критична помилка. Додаток сповіщає користувача про це.

5) Під час завантаження даних на FTP сервер виникла критична помилка. Додаток сповіщає користувача про це.

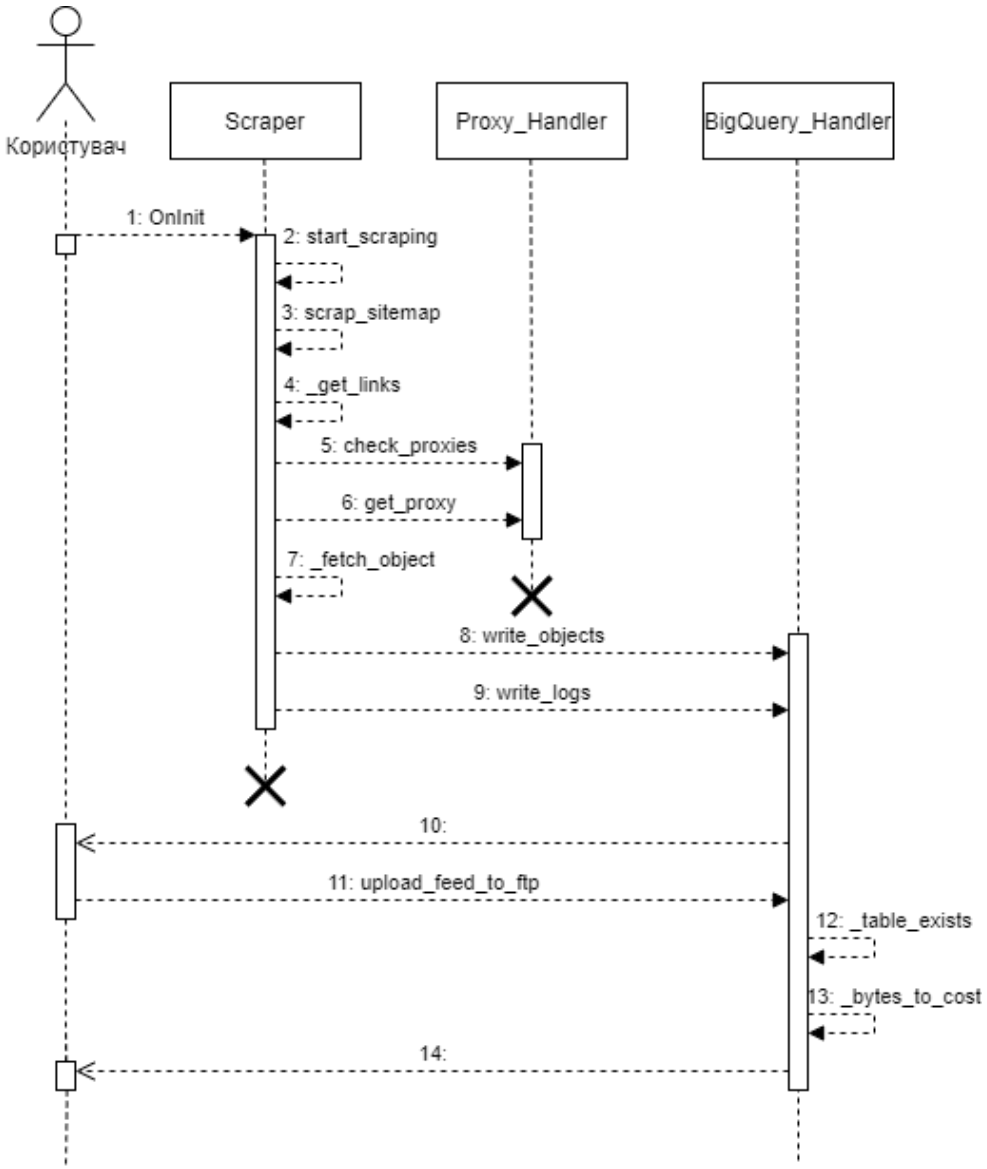


Рисунок 9 – Діаграма послідовності скрапінгу веб-сайту

Обробка даних отриманих від запиту. Цей процес відрізняється для кожного з веб-сайтів, проте загальну суть можна зрозуміти, обравши для прикладу один з вже оброблених веб-сайтів.

Передумови: користувач повинен запустити скрапінг.

Якщо варіант використання завершиться успішно, дані, отримані від запиту, обробляться і буде створено об’єкт, який потім можна зберегти у базу даних Google BigQuery.

Головний сценарій (успішний):

У методі _fetch_object(in url:str | list): Tuple[List, List] відбувається запит до посилання та обробка даних з нього. Після обробки, створюється лог, в якому відмічено успіх операції. Він не виводиться на екран.

Альтернативні сценарії:

1) Запит до посилання був неуспішний. Утворюється лог помилки про неуспішний запит.

2) Під час обробки даних сталася критична помилка. Утворюється лог помилки про неуспішну обробку даних.

3) Під час обробки даних було знайдено пошкоджені дані, або якісь дані були відсутні. Утворюється лог помилки про неуспішну обробку даних з поясненням причини помилки.

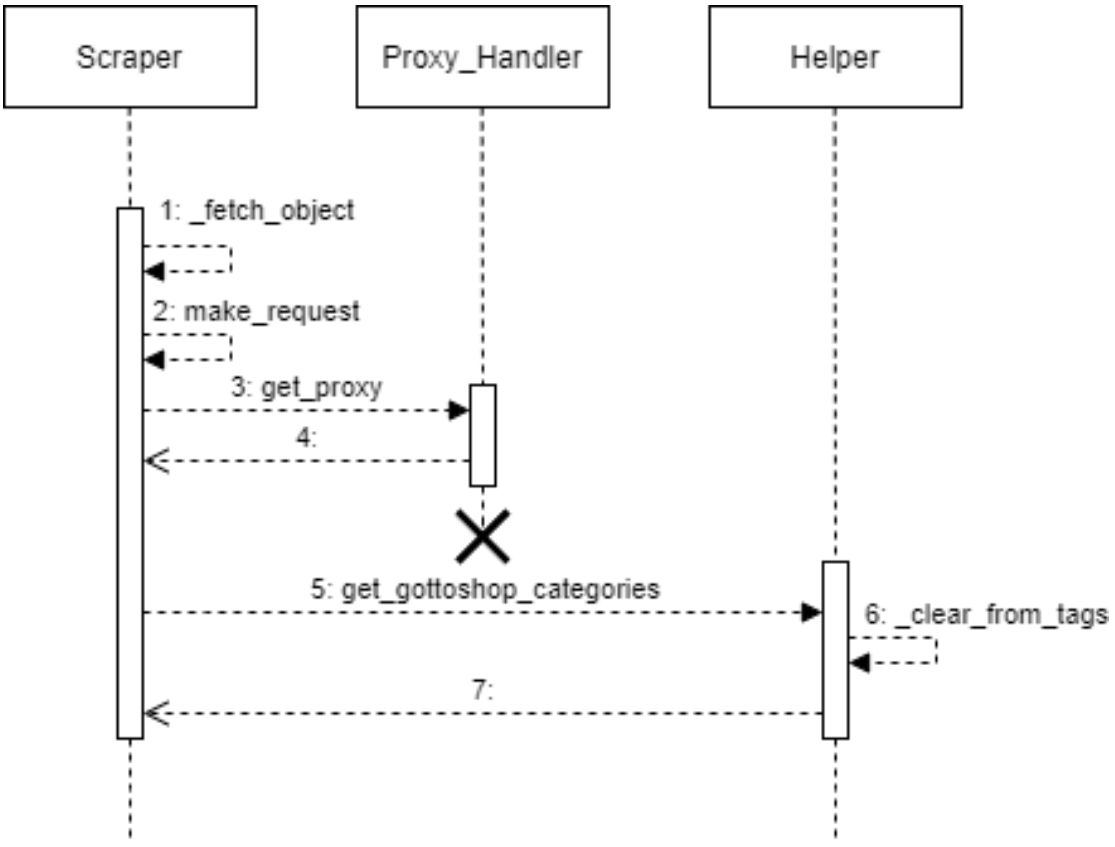


Рисунок 10 – Діаграма послідовності обробки даних запиту

3 РОЗГОРТАННЯ, ТЕСТУВАННЯ ТА ВЕРИФІКАЦІЯ ВИМОГ

3.1. Процес тестування

В даному проєкті було проведено базове тестування оскільки проєкт націлений на початкові стадії ICONIX і проводиться однією людиною, що ускладнює весь процес проєктування та розробки програмної системи.

Консольний додаток повинен мати наступні характеристики:

- зручність використання. Інтерфейс застосунку повинен бути інтуїтивно зрозумілим, з чіткими підказками та інструкціями для користувача. Команди та параметри повинні бути логічними та легко запам’ятовуватися;
- обробка помилок. Програма повинна коректно обробляти помилки, надаючи користувачам зрозумілі повідомлення про помилки та, за можливості, рекомендації щодо їх виправлення;
- масштабованість. Застосунок повинен легко адаптуватися до змін в обсязі даних;
- продуктивність. Програма повинна працювати ефективно, швидко обробляючи команди та виконуючи завдання;
- безпека. Консольний застосунок повинен забезпечувати належний рівень безпеки, включаючи захист від шкідливих даних та забезпечення конфіденційності користувацьких даних;
- кросплатформеність. Застосунок повинен працювати на різних операційних системах (Windows, macOS, Linux) без необхідності значних змін в коді;
- модульність. Код застосунку повинен бути розділений на незалежні модулі, що дозволяє легко вносити зміни та додавати нові функції;
- чітке та інформативне логування. Програма повинна вести журнали подій для полегшення діагностики проблем та відстеження роботи застосунку;

– відповідність стандартам. Застосунок повинен дотримуватися стандартів кодування та найкращих практик програмування, що забезпечує його надійність та майбутню підтримку.

При розробці консольного застосунку для даного проєкту було надано увагу ясноті та зручності для безпроблемного користування застосунку будь-яким користувачем з операційними системи Windows/MacOS/Linux:

Код застосунку був адаптований до усіх перелічених операційних систем. Приклад такої адаптації зображений на рисунку 11.

```python
import os

def clear_terminal():
    # Check if the system is Windows or not
    if os.name == 'nt':
        os.system('cls')  # Clear command for Windows
    else:
        os.system('clear')  # Clear command for macOS and Linux
```

Рисунок 11 – Приклад адаптації коду під різні операційні системи

Як ми можемо спостерігати, метод був написаний з урахуванням особливостей терміналів різних операційних систем.

Головне меню застосунку продемонстроване на рисунку 12:

```
Hi! This is a test Scraping application! Let's see what we got here...

List of websites for scraping:

1. Website 1

Press Ctrl + C to exit the app.

Choose the number of a website you want to scrap:
```

Рисунок 12 – Головне меню застосунку

Примітка. Даний застосунок був розроблений в межах справжнього прикладного завдання, яке виникло під час роботи безпосередньо за спеціальністю. Саме тому інтерфейс є простим, щоб інші розробники могли користуватися ним швидко та зручно.

На рисунку ми бачимо перелік веб-сайтів, доступних користувачу для скрапінгу. В цілях збереження приватних даних, кількість веб-сайтів була зменшена до одного, та його одного було названо “Website 1”.

Як ми можемо спостерігати, користувач може вибрати цифрою веб-сайт для скрапінгу, а також вийти з застосунку комбінацією клавіш “Ctrl + C”.

При виборі веб-сайту ми можемо спостерігати наступне меню, продемонстроване на рисунку 13:

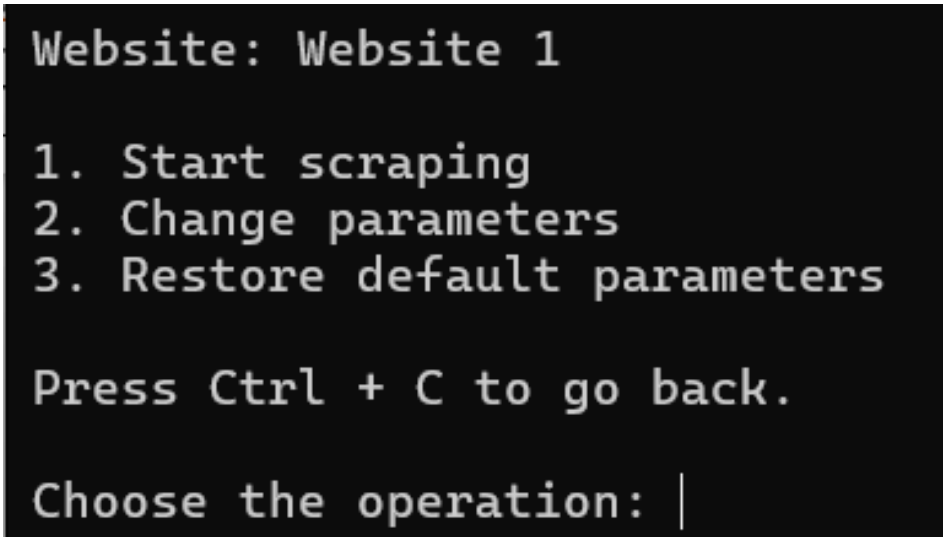


Рисунок 13 – Меню для роботи з веб-сайтом

У цьому меню користувачу представлені усі можливі функції, які він може виконати з веб-сайтом. Все розставлено згідно пріоритетності функцій. Спочатку основна, потім дві додаткових

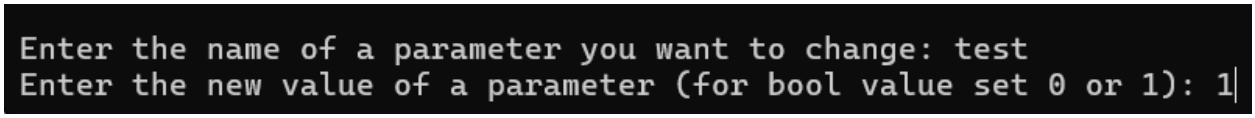
У користувача не може виникнути плутанини, адже усе стоїть на своїх місцях та виконує свої задачі.

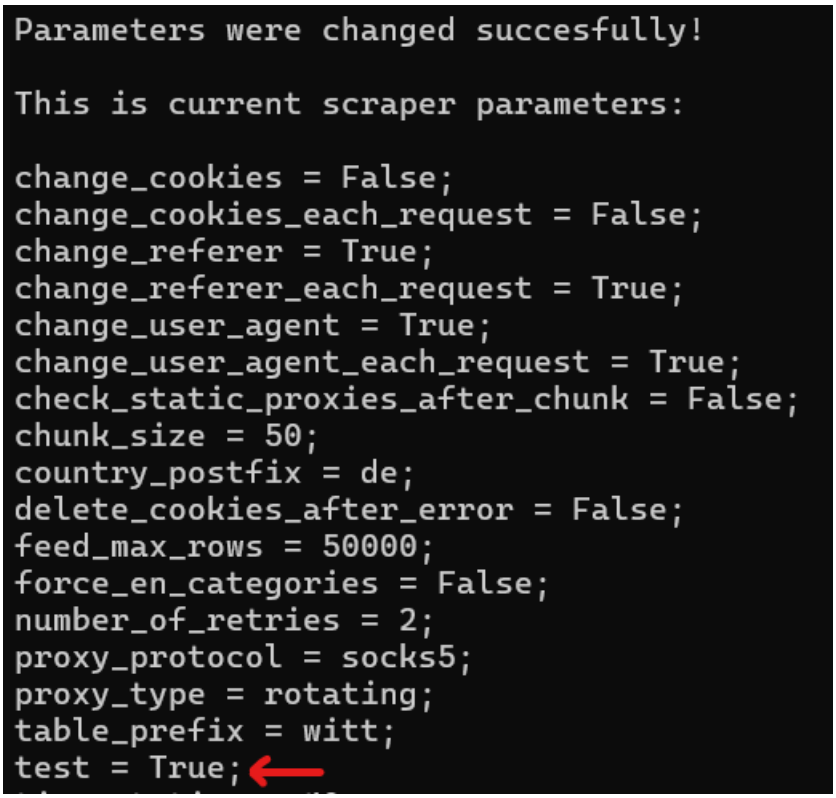
При виборі другого пункту ми отримуємо наступний вивід, продемонстрований на рисунку 14.

Рисунок 14 – Меню для встановлення параметрів скрапінгу

Користувач може ввести назву параметра, який хоче змінити, та задати йому певне значення. Можна встановити тільки той тип значення, яким вже є параметр. Тобто користувач не може ввести текстове значення в числовий параметр, числове в логічний параметр тощо.

Зміна параметра “test” продемонстрована на рисунках 15 та 16.

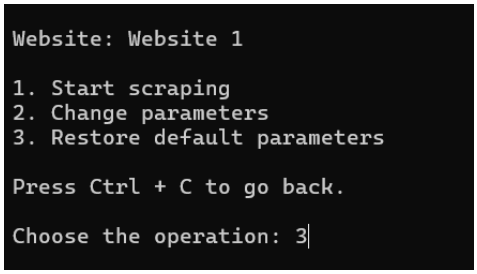




Рисунки 15 та 16 – Зміна параметру test

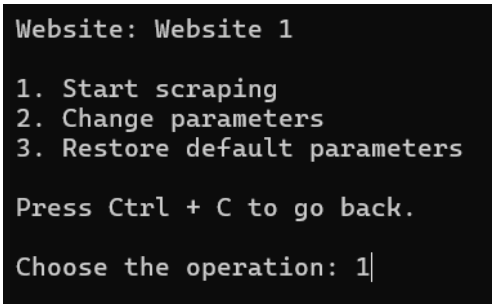
Як ми бачимо, параметр успішно набув нового значення.

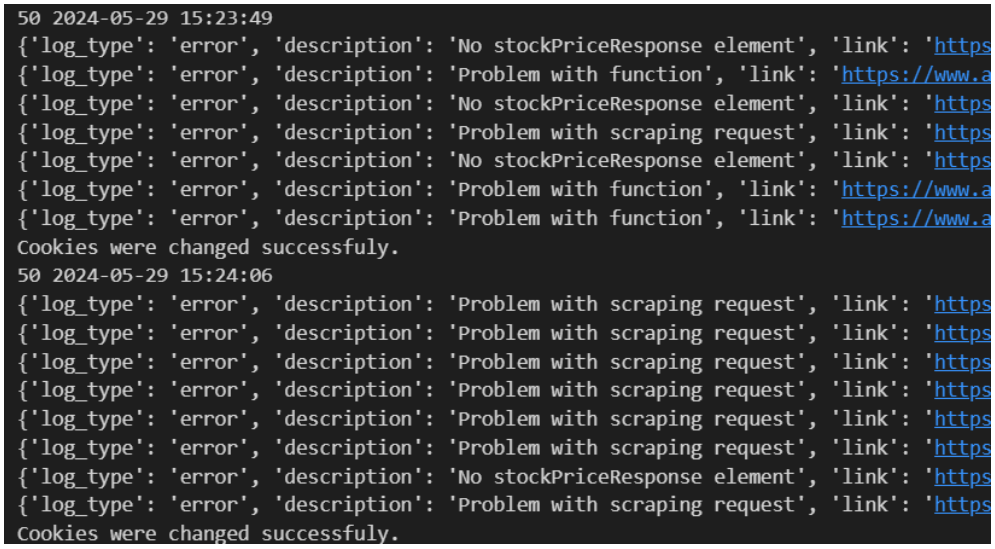
При виборі третього пункту з меню роботи з веб-сайтом, всі параметри повернуться до стандартних, визначених у класі Parameters. Це продемонстровано на рисунках 17 та 18.

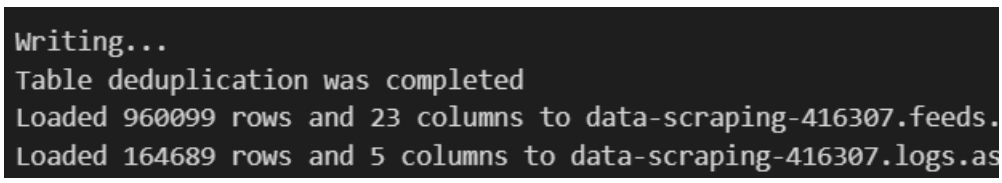


Рисунки 17 та 18 – Виставлення стандартних параметрів

При виборі третього пункту з меню роботи з веб-сайтом, запуститься скрапінг з обраними параметрами. Процес виглядає наступним, і продемонстрований на рисунках 19-21:







Рисунки 19-21 – Процес скрапінгу веб-сайту

Ми бачимо, що процес скрапінгу відбувається успішно. Дані витягаються, обробляються асинхронно, і потім завантажуються у BigQuery. Логи демонструють усю необхідну інформацію про помилки. Як ми бачимо, виникає лише 7-8 помилок на 50 паралельних запитів, що є хорошим результатом і не перевищує кількість допустимих помилок. Логи також успішно записуються у базу даних.

Після завершення скрапінгу, дані можна завантажити на FTP сервер. Це виглядає так, як на рисунку 22.

```
Estimated bytes processed: 41730923 KB
In megabytes: 41.730923 MB
In gigabytes: 0.041730923 GB
Estimated cost of a query: $0.0002372128
Number of chunks: 1
Chunk 1 uploaded successfully to es/jd_s
All chunks uploaded successfully.
```

Рисунок 22 – Завантаження зібраних даних на FTP сервер

Застосунок інформує нас про обсяг та вартість цієї операції за розцінками Google Cloud Platform. Це необхідна інформація для розробників, щоб слідкувати за бюджетом проєкту.

Отримані дані, та їх порівняння з даними з джерела продемонстровані на рисунку 23:

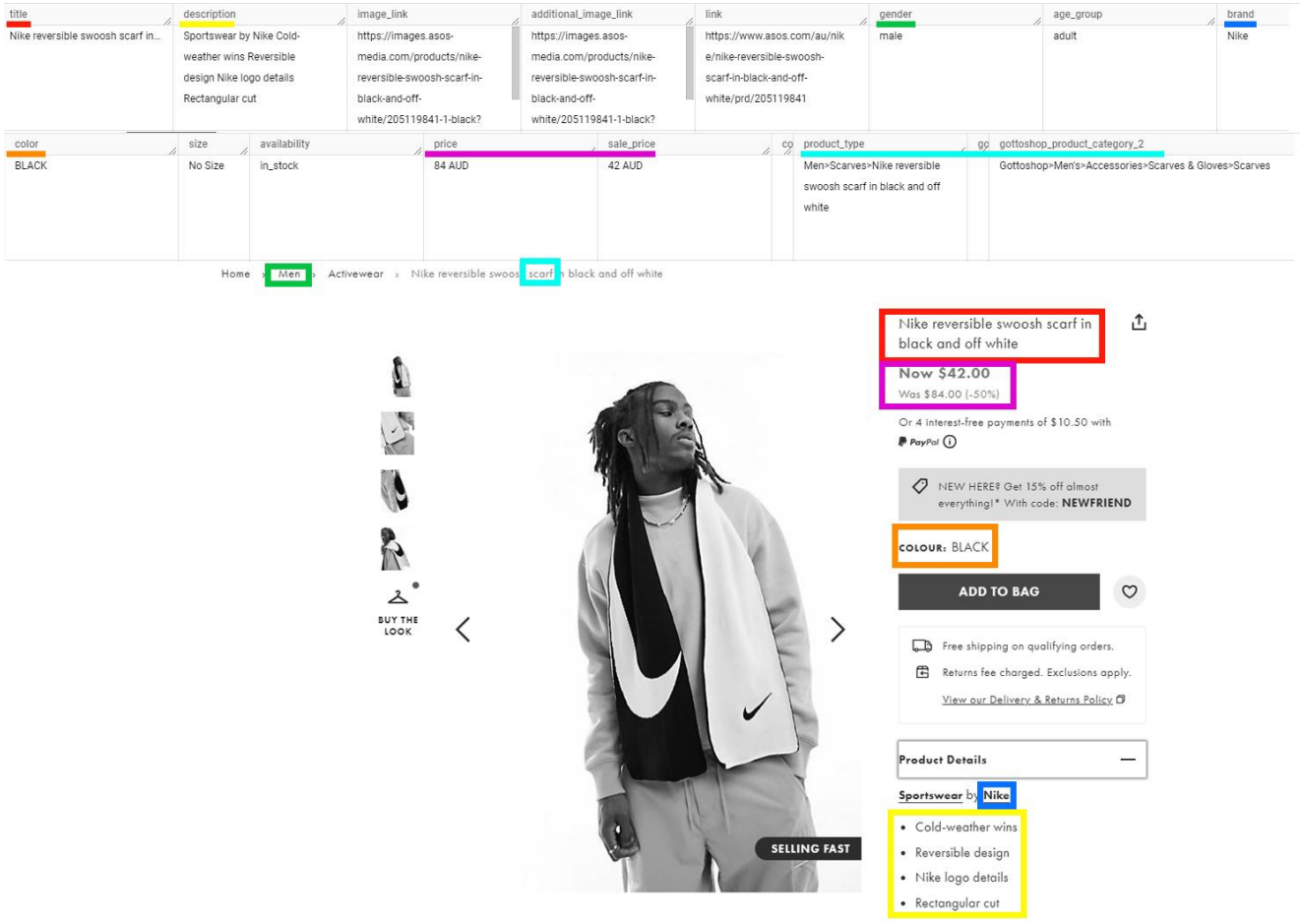


Рисунок 23 – Порівняння отриманих даних з оригінальними

Відповідними кольорами позначено отримані дані та дані з веб-сторінки. Дані фіда і дані веб-сайту повністю співпадають, що означає, що усі необхідні дані були отримані та правильно оброблені, що й було ціллю застосунку. Усі інші дані, які можна спостерігати на рисунку 23 та які не були підкресленими, були отримані з іншого джерела, а не напряму з веб-сторінки.

Приклад обробки виключної ситуації зображений на рисунках 24 та 25:

```
try:
    async with aiohttp.ClientSession(headers=self.params.headers) as session_object:
        async with session_object.get(self.website_link, proxy=f"{self.params.proxy_protocol}://" + temp_proxy, timeout =
            await resp.text()
            status_code = resp.status
            cookies = resp.cookies
    await session_object.close()

    if status_code == 200:
        self.params.headers['Cookie'] = '; '.join([f'{name}={value}' for name, value in cookies.items()])
        print("Cookies were changed successfuly.")
    else:
        if self.params.delete_cookies_after_error:
            if 'Cookie' in self.params.headers:
                del(self.params.headers['Cookie'])
                print('Cookies were deleted.')
        print("Cookies weren't changed!")
except:
    if self.params.delete_cookies_after_error:
        if 'Cookie' in self.params.headers:
            del(self.params.headers['Cookie'])
            print('Cookies were deleted.')
    print("Error. Cookies weren't changed!")
```

```
Cookies were changed successfuly.
50 2024-06-06 23:16:07
{'log_type': 'error', 'description
{'log_type': 'error', 'description
{'log_type': 'error', 'description
{'log_type': 'error', 'description
Server disconnected
Cookies weren't changed!
```

Рисунки 24 та 25 – Обробка виключних ситуацій

Завдяки такій обробці виключних ситуацій, процес скрапінгу є стабільним та безперебійним, що є дуже важливим фактором успіху для поставленої задачі.

4 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ ТА ОСНОВИ ОХОРОНИ ПРАЦІ

4.1. Соціальні та психологічні фактори ризику

Історично сформовані форми спільної діяльності людей, які характеризуються певним типом відношень між людьми утворюють суспільство, або соціум.

Соціум – це особлива система, певний організм, що розвивається за своїми специфічними законами, які характеризуються надзвичайною складністю. У соціумі взаємодіє велика кількість людей. Результатом цих зв'язків є особлива обстановка, що створюється в окремих соціальних групах, яка може впливати на інших людей, які не входять у дані групи.

Процеси, що відбуваються в суспільстві загалом і в окремих громадських групах, вивчає соціологія. Закономірності поведінки та діяльності людей, обумовлені їх належністю до соціальних груп, а також психологічні характеристики цих груп вивчає соціальна психологія.

Вплив соціальних факторів на стан здоров'я суспільства вивчає соціальна гігієна.

Соціальні небезпеки. Соціальними називаються небезпеки, що отримали широке розповсюдження у суспільстві та ті, що загрожують життю й здоров'ю людей. Носіями соціальних небезпек є люди, які утворюють певні соціальні групи. Особливість соціальних небезпек полягає в тому, що вони загрожують великій кількості людей. Розповсюдження соціальних небезпек обумовлене особливостями поведінки людей окремих соціальних груп. Соціальні небезпеки дуже численні. Наприклад, до соціальних відносяться усі протиправні (незаконні) форми насильства, вживання речовин, що порушують психічну та фізіологічну рівновагу людини (алкоголь, наркотики), паління, самовбивства, шахрайство, шарлатанство, здатні завдати шкоди здоров'ю людей.

У своїй основі соціальні небезпеки породжуються соціально-економічними процесами, які протікають у суспільстві. У той же час слід відзначити суперечливий характер причин, наслідком яких є соціальні небезпеки.

Недосконаліть людської природи – головна передумова появи соціальних небезпек. Наявність адекватної правової системи може виявитися основною умовою попередження та захисту від соціальних небезпек.

Розглянемо деякі види соціальних небезпек.

Шантаж у юридичній практиці розглядається як злочин, що полягає у погрозі викриття, розголошення ганебних відомостей із метою домогтися яких-небудь вигод. Шантаж, як небезпека, справляє негативну дію на нервову систему.

Шахрайство – це злочин, що полягає у заволодінні державним, громадським чи особистим майном (або у придбанні прав на майно) шляхом обману або зловживання довірою. Очевидно, що людина, яка стала жертвою шахрайства, зазнає сильного психофізіологічного потрясіння.

Бандитизм за карним правом – це організація озброєних банд з метою нападу на державні та громадські установи або на окремих осіб, а також участь у таких бандах та скоєних ними нападах.

Розбій – це злочин, що полягає у нападі з метою заволодіння державним, громадським або особистим майном, що поєднується з насильством або погрозою насильства, небезпечними для життя і здоров’я особи, що зазнала нападу.

Згвалтування – це статевий акт із застосуванням фізичного насильства, погроз або з використанням безпомічного стану потерпілої. Карне право передбачає суворе покарання за згвалтування, аж до страти (при обтяжливих обставинах).

Терор – фізичне насильство аж до фізичного знищення.

Наркоманія – залежність людини від прийому наркотиків, захворювання, яке виражається у тому, що життєдіяльність організму підтримується на певному рівні тільки за умови прийому наркотичної речовини і веде до глибокого насичення фізичних та психічних функцій.

Виникнення наркоманії пов’язане із ефектом ейфорії, стимулювання або приємно оглушливим ефектом наркотику. Чим сильніше виражений ефект ейфорії, тим скоріше настає звикання. Розвиток наркоманії може настати як результат цікавості, експериментування, як наслідок прийому знеболюючих, снодійних засобів. Розповсюдженню наркоманії сприяє нездорове мікросоціальне середовище, відсутність у людини інтелектуальних та соціально-позитивних установок. У всі часи наркоманія переслідувалася.

Алкоголізм – хронічне захворювання, обумовлене систематичним вживанням спиртних напоїв. Виявляється фізична та психологічна залежність від алкоголю, психічна та соціальна деградація, патологія внутрішніх органів, обміну речовин, центральної та периферичної нервової системи. Нерідко виникають алкогольні психози.

Суїцид. Історія людства свідчить про те, що насильство, агресивність, жорстокість поширені серед людей так само як любов, доброта, милосердя. Особлива жорстокість – це агресія, спрямована на себе (аутоагресія). Вона виявляється в актах самопринижения, самозвинувачення, у нанесенні собі тілесних пошкоджень та у самогубстві – суїциді. Особливість самогубства у тому, що смерть є ділом рук самого потерпілого і завжди являє собою акт насильства. Потрібно однак чітко визнати, що завжди є обставини, які доводять людину до самогубства. Тому цей вираз (самогубство) носить умовне значення. За деякими даними ВООЗ, у світі кожного року здійснюється більше 500 тис. самогубств і приблизно 7 млн намагань до самогубства. Рівень самогубств оцінюється за кількістю здійснених суїцидів на 100 тис населення.

Існує переконання, що покінчити з собою можуть тільки хворі люди. Насправді вони складають лише 25-27 %, ще 19 % – це алкоголіки. Більша частина самовбивць – це здорові люди. Спеціалісти переконані, що суїциди – це результат впливу соціального середовища, яке підриває віру людини. Намір позбавити себе життя з’являється у людини в умовах, коли вона оцінює ситуацію як конфлікт, який не має вирішення. Причин самогубств багато. Це хвороба,

зрада, важкі умови життя, проблема батьків та дітей, любовні взаємини, релігійний вплив тощо.

Намагань до самогубства більше у жінок у 8-10 разів, завершених суїцидів – у чоловіків, у чотири рази. Зараз спостерігається зростання самогубств серед дітей та підлітків. Профілактика самогубств полягає у психологічних, педагогічних та соціальних заходах, спрямованих на відновлення втраченої психологічної та фізичної рівноваги людини. Захист від соціальних небезпек полягає у профілактичних заходах, спрямованих на ліквідацію цих небезпек. Крім цього, потрібна відповідна підготовка людини, що дає змогу адекватно діяти у небезпечних ситуаціях. Потрібна юридична, психологічна, інформаційна та силова підготовка.

4.2. Гігієнічні вимоги до організації та обладнання робочих місць з ВДТ

ВДТ – це відеодисплейний термінал. У цьому розділі вказано, якими документами необхідно користуватися при обладнанні робочих місць користувачів ПК, як правильно і на якій відстані необхідно знаходитися користувачу ПК від екрана ВДТ.

Обладнання і організація робочого місця з ВДТ мають забезпечувати відповідність конструкції всіх елементів робочого місця та їх взаємного розташування ергономічним вимогам з урахуванням характеру і особливостей трудової діяльності (ДСТУ 8604:2015, ДСТУ 7299:2013, ДСТУ 7951:2015).

Конструкція робочого місця користувача ВДТ має забезпечити підтримання оптимальної робочої пози.

Робочі місця з ВДТ слід так розташовувати відносно світлових прорізів, щоб природне світло падало збоку, переважно зліва.

При розміщенні робочих столів з ВДТ слід дотримуватись таких відстаней: між бічними поверхнями ВДТ – 1,2 м; від тильної поверхні одного ВДТ до екрана іншого – 2,5 м.

Екран ВДТ має розташовуватися на оптимальній відстані від очей користувача, що становить 600…700 мм, але не ближче ніж за 600 мм з урахуванням розміру літерно-цифрових знаків і символів.

Розташування екрана ВДТ має забезпечувати зручність зорового спостереження у вертикальній площині під кутом +30 до нормальної лінії погляду працюючого.

Клавіатуру слід розташовувати на поверхні столу на відстані 100…300 мм від краю, звернутого до працюючого. У конструкції клавіатури має передбачатися опорний пристрій (виготовлений із матеріалу з високим коефіцієнтом тертя, що перешкоджає мимовільному її зсуву), який дає змогу змінювати кут нахилу поверхні клавіатури у межах 5…15°.

Для забезпечення захисту і досягнення нормованих рівнів комп’ютерних випромінювань необхідно застосовувати приекранні фільтри, локальні світлофільтри (засоби індивідуального захисту очей) та інші засоби захисту, що пройшли випробування в акредитованих лабораторіях і мають щорічний гігієнічний сертифікат.

При оснащенні робочого місця з ВДТ лазерним принтером параметри лазерного випромінювання повинні відповідати вимогам ДСанПІН 3.3.2.007-98.Екран ВДТ має розташовуватися на оптимальній відстані від очей користувача, що становить 600…700 мм, але не ближче ніж за 600 мм з урахуванням розміру літерно-цифрових знаків і символів.

Розташування екрана ВДТ має забезпечувати зручність зорового спостереження у вертикальній площині під кутом +30 до нормальної лінії погляду працюючого.

Клавіатуру слід розташовувати на поверхні столу на відстані 100…300 мм від краю, звернутого до працюючого. У конструкції клавіатури має передбачатися опорний пристрій (виготовлений із матеріалу з високим коефіцієнтом тертя, що

перешкоджає мимовільному її зсуву), який дає змогу змінювати кут нахилу поверхні клавіатури у межах 5… 15°.

Для забезпечення захисту і досягнення нормованих рівнів комп’ютерних випромінювань необхідно застосовувати приекранні фільтри, локальні світлофільтри (засоби індивідуального захисту очей) та інші засоби захисту, що пройшли випробування в акредитованих лабораторіях і мають щорічний гігієнічний сертифікат.

ВИСНОВОК

В результаті виконаної роботи було спроєктовано додаток для автоматичного утворення Google Merchant Feed та збору даних з веб-сторінок. Було проаналізовано актуальність теми в сучасних реаліях, описано її важливість та шляхи для вирішення. Визначено, яким чином відбувається скрапінг веб-сторінок.

За допомогою StarUML було спроєктовано зручний та ефективний додаток, який містить у собі усі необхідні функції для зручного та ефективного скрапінгу веб-сторінок.

Застосунок містить у собі такі корисні функції, як:

- глибоке і детальне налаштування параметрів скрапінгу;
- асинхронний скрапінг веб-сайтів з виставленими параметрами;
- повноцінне логування для кожного запиту;
- меню з вибором сайтів;
- автоматична обробка даних від запитів;
- завантаження логів і отриманих даних в базу даних;
- конвертація отриманих даних в фід правильного формату та їх завантаження на FTP сервер.

Однією з основних задач було зручне, надійне та безперебійне користування застосунком, адже скрапінг веб-сторінок є дуже об’ємною задачею, під час якої будь яка зупинка є великою втраю часу та ресурсів. Ці умови були виконані завдяки простому та інтуїтивному консольному інтерфейсу. Також було створено різноманітні перевірки, які не дозволяють будь яким поширеним помилкам під час веб-скрапінгу вивести застосунок з ладу, отже надійність, зручність та безперебійність були досягнуті.

Враховуючи усе вищеперераховане, можна зробити висновок, що всі поставлені завдання кваліфікаційної роботи були виконані в повному обсязі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Microsoft. Visual Studion Code Documentation [Електронний ресурс] // Microsoft. – 2024. – Режим доступу до ресурсу: https://code.visualstudio.com/docs.

2. Python 3.12.4 documentation [Електронний ресурс] // Python Software Foundation. – 2024. – Режим доступу до ресурсу: https://docs.python.org/3/.

3. BigQuery [Електронний ресурс] // Google. – 2024. – Режим доступу до ресурсу: https://cloud.google.com/bigquery.

4. Best Practices and Guidelines for Scraping [Електронний ресурс] // Pluralsight. – 2019. – Режим доступу до ресурсу: https://www.pluralsight.com/resources/blog/guides/best-practices-and-guidelines-for-scraping.

5. Käming P. What is Web Scraping and How to Scrape Data? [Електронний ресурс] / Philipp Käming // La Growth Machine. – 2023. – Режим доступу до ресурсу: https://lagrowthmachine.com/web-scraping-guide/.

6. Rosenberg D. Use Case Driven Object Modeling with UML: Theory and Practice / D. Rosenberg, M. Stephens. – Berkeley: Apress, 2007. – 440 с.

7. Rosenberg D. Agile Development with ICONIX Process / D. Rosenberg, M. Stephens. – Berkeley: Apress, 2005. – 261 с.

8. Петрик М.Р. Проєктування програмного забезпечення на основі аналізу вимог та інструментальних засобів розробки IBM Rational Software Architect (від Вимог до коду) Науково-методичний посібник. Тернопіль: Вид-во ТНТУ ім. Івана Пулюя. – 2022. – 560с.

9. Безпека життєдіяльності та охорона праці : підручник : у 2 ч. / Я. О. Сєріков, Л. Ф. Коженевські, М. В. Хворост ; Харків. нац. ун-т міськ. госп-ва ім. О. М. Бекетова [та ін.]. – Харків : ХНУМГ ім. О. М. Бекетова ; Краків : ЄАС, 2021. Ч. 1 : Безпека життєдіяльності. – 2021. – 255 с.

10. Заплатинський В. М. Полімовний тлумачний словник з безпеки. Підручник. – К.: Центр учбової літератури, 2009. – 120 с.

11. Іванова І.В., Заплатинський В.М., Гвоздій С.П. “Безпека життєдіяльності” навчально-контролюючі тести. – Київ: “Саміт-книга”, 2005. – 148 с.

12. Гандзюк М.П., Желібо Є.П., Халімовський М.О. Основи охорони праці: Підручник. 5-е вид. / За ред. М.П. Гандзюка. – К.: Каравела, 2011. – 384 с.

13. ДСТУ 8604:2015 Дизайн і ергономіка. Робоче місце для виконання робіт у положенні сидячи. Загальні ергономічні вимоги [Електронний ресурс] // Київ: Український науково-дослідний інститут дизайну та ергономіки НАУ. – 2015. – Режим доступу до ресурсу: https://online.budstandart.com/ua/catalog/doc-page.html?id_doc=71028.

14. ДСТУ 7299:2013 Дизайн і ергономіка. Робоче місце оператора. Взаємне розташування елементів робочого місця. Загальні вимоги ергономіки [Електронний ресурс] // Київ: Український науково-дослідний інститут дизайну та ергономіки НАУ. – 2013. – Режим доступу до ресурсу: https://online.budstandart.com/ua/catalog/doc-page?id_doc=57364.

15. ДСТУ 7951:2015 Дизайн і ергономіка. Крісло оператора. Загальні ергономічні вимоги [Електронний ресурс] // Київ: Український науково-дослідний інститут дизайну та ергономіки НАУ. – 2015. – Режим доступу до ресурсу: https://online.budstandart.com/ua/catalog/doc-page?id_doc=66671.

16. Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин [Електронний ресурс] // Міністерство охорони здоров'я України. Головне санітарно-епідеміологічне управління. – 1998. – Режим доступу до ресурсу: https://zakon.rada.gov.ua/rada/show/v0007282-98#Text.

ДОДАТКИ

ДОДАТОК А

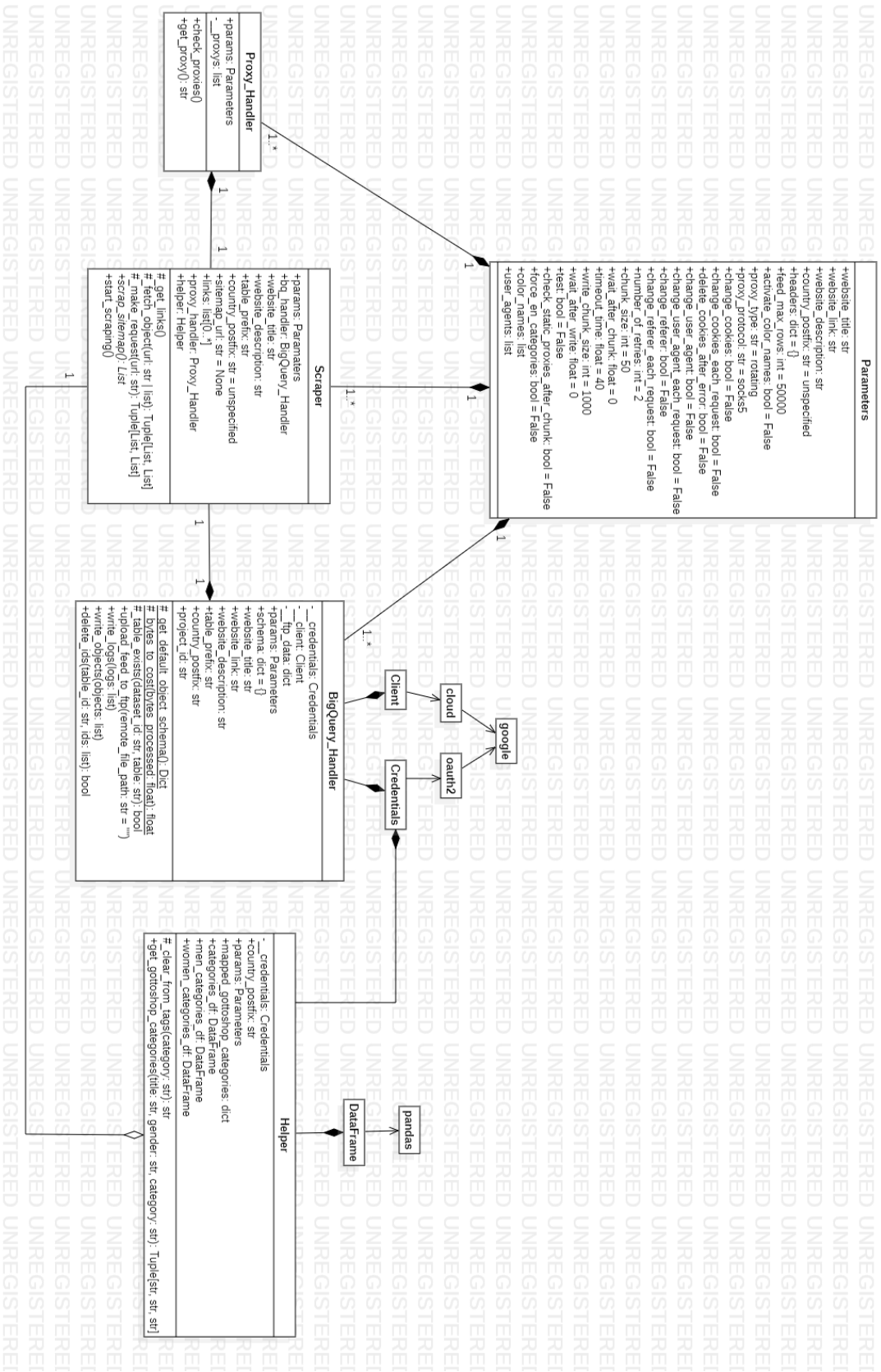


Рисунок 8 – Діаграма класів

ДОДАТОК Б

Лістинг коду

Файл “scraping.py”

```
import sys
import os
import time

from abc import ABC, abstractmethod
import requests
import pandas as pd
import json
import asyncio
import aiohttp
from io import BytesIO
from datetime import datetime
from dict2xml import dict2xml
from ftplib import FTP

from google.cloud import bigquery
from google.oauth2 import service_account
from googleapiclient.discovery import build

from IPython.display import clear_output
from typing import List, Tuple, Dict
import random
import math

class Parameters():
    """
    A class to hold configuration parameters for web scraping.

    Attributes:
        website_title (str): The title of the website to scrape.
        website_link (str): The URL of the website to scrape.
        website_description (str): A description of the website.
        country_postfix (str): The country postfix for the scraping.
        headers (dict): The HTTP headers to use for requests.
        feed_max_rows (int): The maximum number of rows in the feed.
        activate_color_names (bool): Flag to activate color names.
        proxy_type (str): The type of proxy to use ('static_all',
'static_country', 'rotating').
        proxy_protocol (str): The protocol for the proxy ('socks5').
        change_cookies (bool): Flag to change cookies.
        change_cookies_each_request (bool): Flag to change cookies
for each request.
        delete_cookies_after_error (bool): Flag to delete cookies
after an error.
        change_user_agent (bool): Flag to change the user agent.
        change_user_agent_each_request (bool): Flag to change the
user agent for each request.
        change_referer (bool): Flag to change the referer.
```

```
        change_referer_each_request (bool): Flag to change the
referer for each request.
        number_of_retries (int): Number of retries for requests.
        chunk_size (int): The chunk size for processing.
        wait_after_chunk (float): The wait time after processing a
chunk.
        timeout_time (float): The timeout time for requests.
        write_chunk_size (int): The chunk size for writing.
        wait_after_write (float): The wait time after writing a
chunk.
        test (bool): Flag for test mode.
        check_static_proxies_after_chunk (bool): Flag to check static
proxies after each chunk.
        force_en_categories (bool): Flag to force English categories.
        color_names (list): List of color names.
        user_agents (list): List of user agents.
    """
    def __init__(self,
        website_title: str = "",
        website_link: str = "",
        website_description: str = "",
        country_postfix: str='unspecified',
        headers: dict = {
            # 'Accept': '*/*',
            # 'Accept-Encoding': 'gzip, deflate, br, zstd',
            # 'Accept-Language': 'en-US,en;q=0.9',
        },

        feed_max_rows: int = 50000,
        activate_color_names: bool = False,

        proxy_type: str = 'rotating',
        proxy_protocol: str = 'socks5',

        change_cookies: bool  = False,
        change_cookies_each_request: bool  = False,
        delete_cookies_after_error: bool = False,

        change_user_agent: bool = False,
        change_user_agent_each_request: bool = False,

        change_referer: bool = False,
        change_referer_each_request: bool = False,

        number_of_retries: int = 2,

        chunk_size: int = 50,
        wait_after_chunk: float = 0,
        timeout_time: float = 40,

        write_chunk_size: int = 1000,
        wait_after_write: float = 0,
        test: bool = False,

        check_static_proxies_after_chunk: bool = False,

        force_en_categories: bool = False,
    ):
```

```
        # Scraping config
        self.website_title = website_title
        self.website_link = website_link
        self.website_description = website_description

        self.table_prefix  =  website_title.lower().strip().replace('
', '_')
        self.country_postfix = country_postfix

        if  proxy_type  not  in  ['static_all',  'static_country',
'rotating']:
            raise  Exception(f"Wrong  proxy  type:  {proxy_type}.
Available proxy types: 'static_all', 'static_country', 'rotating'.")

        self.headers = headers
        self.proxy_type = proxy_type
        self.proxy_protocol = proxy_protocol

        self.change_cookies = change_cookies
        self.change_cookies_each_request                          =
change_cookies_each_request
        self.delete_cookies_after_error = delete_cookies_after_error

        self.change_user_agent = change_user_agent
        self.change_user_agent_each_request                       =
change_user_agent_each_request

        self.change_referer = change_referer
        self.change_referer_each_request                          =
change_referer_each_request

        self.number_of_retries = number_of_retries

        self.chunk_size = chunk_size
        self.wait_after_chunk = wait_after_chunk

        self.write_chunk_size = write_chunk_size
        self.wait_after_write = wait_after_write
        self.test = test

        self.check_static_proxies_after_chunk                     =
check_static_proxies_after_chunk

        self.timeout_time = timeout_time

        self.feed_max_rows = feed_max_rows

        self.force_en_categories = force_en_categories
        if activate_color_names:
            with open("other/colornames.json", "r") as file:
                self.color_names  =  [i['name'].lower()  for  i  in
json.load(file)]

        with open('other/user_agents.txt', 'r') as file:
            self.user_agents = file.readlines()
        # End of scraping config

class BigQuery_Handler():
```

```
    """
    A class to handle interactions with Google BigQuery.

    Methods:
        _get_default_object_schema() -> Dict: Get the default object
schema.
        _bytes_to_cost(bytes_processed: float) -> float: Convert
bytes processed to cost.
        _table_exists(dataset_id: str, table: str) -> bool: Check if
a table exists.
        upload_feed_to_ftp(remote_file_path: str = ""): Upload feed
data to FTP.
        write_logs(logs: list): Write logs to BigQuery.
        delete_ids(table_id: str, ids: list) -> bool: Delete IDs from
a table.
        write_objects(objects): Write objects to BigQuery.
    """
    @staticmethod
    def _get_default_object_schema() -> Dict:
        """
        Get the default object schema from a JSON file.

        Returns:
            dict: The default object schema.
        """
        file_path = "other/default_object_schema.json"
        try:
            with open(file_path) as file:
                schema = json.load(file)
            return schema
        except FileNotFoundError:
            raise FileNotFoundError("Default schema file not found.")
        except json.JSONDecodeError:
            raise ValueError("Error decoding default schema JSON.")

    @staticmethod
    def _bytes_to_cost(bytes_processed: float) -> float:
        """
        Convert bytes processed to cost in USD.

        Args:
            bytes_processed (float): The number of bytes processed.

        Returns:
            float: The cost in USD.
        """
        # Convert bytes to terabytes
        terabytes_processed = bytes_processed / (1024 ** 4)

        # BigQuery on-demand cost per terabyte for us-west1 region
        cost_per_terabyte = 6.25

        # Calculate cost
        cost = terabytes_processed * cost_per_terabyte

        return cost

    def _table_exists(self, dataset_id: str, table: str) -> bool:
```

```
        """
        Check if a table exists in a dataset.

        Args:
            dataset_id (str): The ID of the dataset.
            table (str): The name of the table.

        Returns:
            bool: True if the table exists, False otherwise.
        """
        table_ref    =    self.__client.dataset(dataset_id=dataset_id,
project=self.__client.project).table(table)
        # Check if the table exists
        try:
            self.__client.get_table(table_ref)
            return True
        except Exception as e:
            return False

    def upload_feed_to_ftp(self, remote_file_path: str = ""):
        """
        Upload feed data from BigQuery to an FTP server.

        Args:
            remote_file_path (str): The remote file path on the FTP
server.
        """
        try:
            if not remote_file_path:
                remote_file_path = self.table_prefix + '.xml'

            upload_choice = input(f"Do you want to export data from
BigQuery to: {self.country_postfix}/{remote_file_path}? (y/n)")
            if upload_choice.lower() == 'y':
                if              self._table_exists(dataset_id="feeds",
table=self.table_prefix):
                    sql_query       =       f"SELECT       *       FROM
`{self.__client.project}.feeds.{self.table_prefix}`              WHERE
gottoshop_product_category_1        IS        NOT        NULL        OR
gottoshop_product_category_2 IS NOT NULL"
                    query       =       self.__client.query(sql_query,
job_config=bigquery.QueryJobConfig(dry_run=True))

                    # Get the estimated bytes processed and cost
                    estimated_bytes_processed                        =
query.total_bytes_processed

                    print("Estimated        bytes        processed:",
estimated_bytes_processed, "KB")
                    print("In                                megabytes:",
estimated_bytes_processed/1000000, "MB")
                    print("In                                gigabytes:",
estimated_bytes_processed/1000000000, "GB")
                    print(f"Estimated    cost    of    a    query:
${self._bytes_to_cost(estimated_bytes_processed)}")

                    proceed_option  =  input("Do  you  want  to  proceed
(y/n): ")
```

```
                    if proceed_option.lower() != 'y':
                        return

                    query = self.__client.query(sql_query)

                    item                                                =
query.result().to_dataframe().to_dict('records')
                    # Calculate the number of chunks
                    num_chunks      =      math.ceil(len(item)      /
self.params.feed_max_rows)

                    print("Number of chunks:", num_chunks)

                    # Iterate over chunks and upload them
                    for i in range(num_chunks):
                        start_index = i * self.params.feed_max_rows
                        end_index     =     min((i     +     1)     *
self.params.feed_max_rows, len(item))
                        chunk = item[start_index:end_index]

                        feed = {
                            "title": self.website_title,
                            "link": self.website_link,
                            "description": self.website_description,
                            "item": chunk,
                        }

                        chunk_remote_file_path                        =
f'{remote_file_path.split(".xml")[0]}_{i + 1}.xml'

                        chunk_xml_data        =        dict2xml(feed,
wrap='channel', indent="   ")
                        chunk_xml_bytes = chunk_xml_data.encode('utf-
8')

                        with FTP(self.__ftp_data['host']) as ftp:
                            ftp.login(self.__ftp_data['login'],
self.__ftp_data['pass'])
                            ftp.cwd(self.country_postfix)
                            ftp.storbinary(f'STOR
{chunk_remote_file_path}', BytesIO(chunk_xml_bytes))

                            print(f"Chunk    {i    +    1}    uploaded
successfully to {self.country_postfix}/{chunk_remote_file_path}")

                    print("All chunks uploaded successfully.")
                    return
                else:
                    print(f"There      is      no      feeds      to
load/`{self.__client.project}.feeds.{self.table_prefix}`    may    not
exist.")
                    return
            except Exception as e:
                raise e

    def write_logs(self, logs: list):
        """
```

```
        Write logs to BigQuery.

        Args:
            logs (list): The logs to write.
        """
        try:
            logs_df = pd.DataFrame(logs)
            logs_df['log_type'] = logs_df['log_type'].apply(lambda x:
str(x) if x else None)
            logs_df['description']                                    =
logs_df['description'].apply(lambda x: str(x) if x else None)
            logs_df['link'] = logs_df['link'].apply(lambda x: str(x)
if x else None)
            logs_df['timestamp'] = logs_df['timestamp'].apply(lambda
x: pd.Timestamp(x) if x else None)
            logs_df['additional_data']                                =
logs_df['additional_data'].apply(lambda x: str(x) if x else None)

            schema=[
                {
                    "name": "log_type",
                    "mode": "NULLABLE",
                    "type": "STRING",
                    # "description": "",
                    "fields": []
                },
                {
                    "name": "description",
                    "mode": "NULLABLE",
                    "type": "STRING",
                    # "description": "",
                    "fields": []
                },
                {
                    "name": "link",
                    "mode": "NULLABLE",
                    "type": "STRING",
                    # "description": "",
                    "fields": []
                },
                {
                    "name": "timestamp",
                    "mode": "NULLABLE",
                    "type": "TIMESTAMP",
                    # "description": "",
                    "fields": []
                },
                {
                    "name": "additional_data",
                    "mode": "NULLABLE",
                    "type": "STRING",
                    # "description": "",
                    "fields": []
                },

            ]

            project = self.__credentials.project_id
```

```
            dataset = 'logs'
            table = f"{self.table_prefix}_logs"
            table_id = "{}.{}.{}".format(project, dataset, table)
            tableRef = self.__client.dataset(dataset).table(table)
            job_config = bigquery.job.LoadJobConfig(
                schema=schema,
                write_disposition="WRITE_APPEND",
                autodetect=True,
                schema_update_options=["ALLOW_FIELD_ADDITION"]
            )

            job = self.__client.load_table_from_dataframe(logs_df,
tableRef, job_config=job_config)  # Make an API request.
            job.result()  # Wait for the job to complete.

            table = self.__client.get_table(tableRef)  # Make an API
request.
            echo = "Loaded {} rows and {} columns to
{}".format(table.num_rows, len(table.schema), tableRef)
            print(echo)

            # Delete duplicates
            job_config                                              =
bigquery.QueryJobConfig(destination=table_id,
write_disposition="WRITE_TRUNCATE")

            # Write your SQL statement here:
            sql = """
                #standardSQL
                SELECT event.* FROM (
                SELECT ARRAY_AGG(
                t ORDER BY timestamp DESC LIMIT 1
                )[OFFSET(0)]  event
                FROM `{}` t
                # GROUP BY the id you are de-duplicating by
                GROUP BY t.link
            )
            """.format(table_id)

            # Start the query, passing in the extra configuration.
            query_job           =           self.__client.query(sql,
job_config=job_config)  # Make an API request.
            query_job.result()  # Wait for the job to complete.
            echo2   =   "Query   results   loaded   to   the   table
{}".format(table_id)
            print(echo2)
        except Exception as e:
            print("Problem with writing logs:", e)
            raise(e)

    def delete_ids(self, table_id: str, ids: list) -> bool:
        """
        Delete IDs from a table.

        Args:
            table_id (str): The ID of the table.
            ids (list): The IDs to delete.
```

```
        Returns:
            bool: True if deletion was successful, False otherwise.
        """
        # Deduplicate table by id
        str_ids = ", ".join([f'"{i}"' for i in ids])
        sql = f"""
            DELETE FROM `{table_id}` WHERE id IN ({str_ids})
        """
        query_job = self.__client.query(sql)
        query_job.result()
        return query_job.done()

    def write_objects(self, objects):
        """
        Write objects to BigQuery.

        Args:
            objects (list): The objects to write.
        """
        if self.schema is None:
            self.schema = self._get_default_object_schema()
        try:
            obj_df = pd.DataFrame(objects)
            obj_df['id'] = obj_df['id'].apply(lambda x: str(x) if x
else None)
            obj_df['item_group_id']                                 =
obj_df['item_group_id'].apply(lambda x: str(x) if x else None)
            obj_df['mpn'] = obj_df['mpn'].apply(lambda x: str(x) if x
else None)
            obj_df['title'] = obj_df['title'].apply(lambda x: str(x)
if x else None)
            obj_df['description']                                   =
obj_df['description'].apply(lambda x: str(x) if x else None)
            obj_df['image_link'] = obj_df['image_link'].apply(lambda
x: str(x) if x else None)
            obj_df['converted_image_link']                          =
obj_df['image_link'].apply(lambda x: str(x) if x else None)
            obj_df['link'] = obj_df['link'].apply(lambda x: str(x) if
x else None)
            obj_df['gender']   =   obj_df['gender'].apply(lambda   x:
str(x) if x else None)
            obj_df['age_group'] = obj_df['age_group'].apply(lambda x:
str(x) if x else None)
            obj_df['brand'] = obj_df['brand'].apply(lambda x: str(x)
if x else None)
            obj_df['color'] = obj_df['color'].apply(lambda x: str(x)
if x else None)
            obj_df['size'] = obj_df['size'].apply(lambda x: str(x) if
x else None)
            obj_df['availability']                                  =
obj_df['availability'].apply(lambda x: str(x) if x else None)
            obj_df['price'] = obj_df['price'].apply(lambda x: str(x)
if x else None)
            obj_df['condition'] = obj_df['condition'].apply(lambda x:
str(x) if x else None)
            obj_df['product_type']                                  =
obj_df['product_type'].apply(lambda x: str(x) if x else None)
```

```
            obj_df['gottoshop_product_category_1'] =
obj_df['gottoshop_product_category_1'].apply(lambda x: str(x) if x
else None)
            obj_df['gottoshop_product_category_2'] =
obj_df['gottoshop_product_category_2'].apply(lambda x: str(x) if x
else None)

            schema = self.schema

            for key in schema:
                if key['name'] not in obj_df:
                    obj_df[key['name'] ] = None

            project = self.__credentials.project_id
            dataset = 'feeds'
            table = self.table_prefix
            table_id = "{}.{}.{}".format(project, dataset, table)
            tableRef = self.__client.dataset(dataset).table(table)
            job_config = bigquery.job.LoadJobConfig(
                schema=schema,
                write_disposition="WRITE_APPEND",
                autodetect=True,
                schema_update_options=["ALLOW_FIELD_ADDITION"]
            )

            if self._table_exists(dataset, table):
                ids = obj_df['id'].to_list()
                delete_result = self.delete_ids(table_id, ids)
                if not delete_result:
                    raise Exception("Problem with the table
deduplication.")
                print("Table deduplication was completed")

            job = self.__client.load_table_from_dataframe(obj_df,
tableRef, job_config=job_config)  # Make an API request.
            job.result()  # Wait for the job to complete.

            table = self.__client.get_table(tableRef)  # Make an API
request.
            echo = "Loaded {} rows and {} columns to
{}".format(table.num_rows, len(table.schema), tableRef)
            print(echo)

        except Exception as e:
            print(f"An error occurred with processing objects: {e}")
            raise(e)

    def __init__(
        self, params: Parameters,
        schema: dict = {},
    ):
        self.params = params
        self.website_title = params.website_title
        self.website_link = params.website_link
        self.website_description = params.website_description

        self.table_prefix = params.table_prefix
        self.country_postfix = params.country_postfix
```

```
        self.schema = schema

        if self.country_postfix != 'en':
            self.table_prefix += f"_{self.country_postfix}"

        self.__credentials                                          =
service_account.Credentials.from_service_account_file('other/gc_confi
g.json')
        self.project_id = 'your_project_id'
        self.__client                                               =
bigquery.Client(credentials=self.__credentials,
project=self.project_id)

        with open("other/ftp.json", "r") as file:
            self.__ftp_data = json.load(file)

class Proxy_Handler():
    """
    A class to handle proxy operations, including checking and
retrieving proxies.
    """
    def check_proxies(self):
        """
        Reads the configuration file and checks proxies based on the
country postfix and proxy type.
        Downloads the proxy list if applicable and formats the
proxies.
        """
        with open("other/config.json", "r") as file:
            config_data = json.load(file)
        if self.params.country_postfix == 'us':
            if 'static' in self.params.proxy_type:
                if self.params.proxy_type == 'static_country':
                    proxy_requests                                  =
requests.get(f"https://proxy.webshare.io/api/v2/proxy/list/download/{
config_data['proxy_token']}/us/any/username/backbone/-/")
                elif self.params.proxy_type == 'static_all':
                    proxy_requests                                  =
requests.get(f"https://proxy.webshare.io/api/v2/proxy/list/download/{
config_data['proxy_token']}/-/any/username/backbone/-/")

                proxys = proxy_requests.text.split('\r\n')
                proxys = [i for i in proxys if i]
                self.__proxys = proxys
                for i in range(len(self.__proxys)):
                    proxy = self.__proxys[i]
                    proxy = proxy.split(":")
                    self.__proxys[i] = ":".join([proxy[2], proxy[3]])
+ "@" + ":".join([proxy[0], proxy[1]])
            elif self.params.proxy_type == 'rotating':
                self.__proxys = [config_data['rotate_proxy_url']]
        else:
            self.__proxys = [config_data['rotate_proxy_url']]

    def get_proxy(self) -> str:
        """
        Retrieves a proxy from the list of proxies.
```

```
        If static proxies are used, it randomly selects one from the
list.
        """
        if 'static' in self.params.proxy_type:
            if self.__proxys:
                proxy = random.choice(self.__proxys)
                return proxy
            else:
                raise Exception("There are no valid proxies. Try to
call 'check_proxies' method for proxies validation.")
        else:
            return random.choice(self.__proxys)

    def __init__(self,
        params: Parameters
    ):
        """
        Initializes the Proxy_Handler with the provided parameters
and checks for available proxies.

        :param params: Parameters object containing configuration for
proxies.
        """
        self.params = params
        self.check_proxies()

class Helper():
    """
    A helper class to provide utility functions for category handling
and mapping.
    """
    def _clear_from_tags(self, category: str) -> str:
        """
        Clears the category string from tags.

        :param category: The category string to clear.
        :return: The cleaned category string.
        """
        if category:
            category = category.split('gotto-tag')[-1].strip()
        return category

    def  get_gottoshop_categories(self,  title:  str,  gender:  str,
category: str) -> Tuple[str, str, str]:
        """
        Maps the given category to Gottoshop categories based on the
title and gender.

        :param title: The product title.
        :param gender: The gender for the product.
        :param category: The initial category.
        :return: A tuple containing the cleaned category and two
mapped Gottoshop categories.
        """
        gottoshop_product_category_1 = None
        gottoshop_product_category_2 = None
        split_keys = []
        temp_splits = []
```

```
        if ' & ' in category and ',' not in category:
            temp_splits = category.split(' & ')
        elif ' & ' in category and ',' in category:
            first_split = category.split(' & ')
            second_split = [i.split(',') for i in first_split]
            for element in second_split:
                temp_splits.extend(element)
        elif ' and ' in category and ',' not in category:
            temp_splits = category.split(' and ')
        elif ' and ' in category and ',' in category:
            first_split = category.split(' and ')
            second_split = [i.split(',') for i in first_split]
            for element in second_split:
                temp_splits.extend(element)
        split_keys.extend(temp_splits)
        split_keys    =    list(set([i.lower().strip()    for    i    in
split_keys]))
        split_keys.sort()

        if split_keys:
            for key in split_keys:
                if key in title.lower():
                    category = key
                elif key[-1] == 's':
                    if key[0:-1] in title.lower():
                        category = key
                    elif key[-2:] == 'es':
                        if key[0:-2] in title.lower():
                            category = key

        for condition in ['Sport', '']:
            if  not  gottoshop_product_category_1  or  'Unspecified'  in
str(gottoshop_product_category_1):
                if gender == 'female':
                    for col_idx in range(4, 1, -1):
                        col_name = 'UK: {} рівень'.format(col_idx)
                        mask                                          =
self.women_categories_df[col_name].str.lower() == category
                        if mask.any():
                            result                                    =
self.women_categories_df.loc[mask, ['UK: 1 рівень', 'Структура всіх
рівнів']].reset_index().drop(columns=['index'])
                            if  condition  in  result['Структура  всіх
рівнів'][0]:
                                pass
                            else:
                                gottoshop_product_category_1          =
result['Структура всіх рівнів'][0]
                                break
            if  not  gottoshop_product_category_2  or  'Unspecified'  in
str(gottoshop_product_category_2):
                if gender == 'male':
                    for col_idx in range(4, 1, -1):
                        col_name = 'UK: {} рівень'.format(col_idx)
                        mask                                          =
self.men_categories_df[col_name].str.lower() == category
                        if mask.any():
```

```
                            result = self.men_categories_df.loc[mask,
['UK:          1          рівень',          'Структура          всіх
рівнів']].reset_index().drop(columns=['index'])
                            if  condition  in  result['Структура  всіх
рівнів'][0]:
                                pass
                            else:
                                gottoshop_product_category_2        =
result['Структура всіх рівнів'][0]
                                break
            if gender == 'unisex':
                if not gottoshop_product_category_1 or 'Unspecified'
in str(gottoshop_product_category_1):
                    for col_idx in range(4, 1, -1):
                        col_name = 'UK: {} рівень'.format(col_idx)
                        mask                                        =
self.women_categories_df[col_name].str.lower() == category
                        if mask.any():
                            result                                  =
self.women_categories_df.loc[mask, ['UK: 1 рівень', 'Структура всіх
рівнів']].reset_index().drop(columns=['index'])
                            if  condition  in  result['Структура  всіх
рівнів'][0]:
                                pass
                            else:
                                gottoshop_product_category_1        =
result['Структура всіх рівнів'][0]
                                break
                if not gottoshop_product_category_2 or 'Unspecified'
in str(gottoshop_product_category_2):
                    for col_idx in range(4, 1, -1):
                        col_name = 'UK: {} рівень'.format(col_idx)
                        mask                                        =
self.men_categories_df[col_name].str.lower() == category
                        if mask.any():
                            result = self.men_categories_df.loc[mask,
['UK:          1          рівень',          'Структура          всіх
рівнів']].reset_index().drop(columns=['index'])
                            if  condition  in  result['Структура  всіх
рівнів'][0]:
                                pass
                            else:
                                gottoshop_product_category_2        =
result['Структура всіх рівнів'][0]
                                break
            if    not    gottoshop_product_category_1    or    not
gottoshop_product_category_2:
                continue
            else:
                break

        if    not    gottoshop_product_category_1    and    not
gottoshop_product_category_2:
            gottoshop_product_category = None
            if category in self.mapped_gottoshop_categories:
                gottoshop_product_category                          =
self.mapped_gottoshop_categories[category]
                if gender == 'female':
```

```
                    gottoshop_product_category_1                    =
gottoshop_product_category['w']
                    gottoshop_product_category_2 = None
                elif gender == 'male':
                    gottoshop_product_category_1 = None
                    gottoshop_product_category_2                    =
gottoshop_product_category['m']
                elif gender == 'unisex':
                    gottoshop_product_category_1                    =
gottoshop_product_category['w']
                    gottoshop_product_category_2                    =
gottoshop_product_category['m']
            if not gottoshop_product_category:
                if gender == 'female' or gender == 'unisex':
                    gottoshop_product_category_1                    =
"Gottoshop>Women's>Unspecified"
                if gender == 'male' or gender == 'unisex':
                    gottoshop_product_category_2                    =
"Gottoshop>Men's>Unspecified"
        elif        not        gottoshop_product_category_1        and
gottoshop_product_category_2:
            if gender == 'unisex':
                if category in self.mapped_gottoshop_categories:
                    gottoshop_product_category                      =
self.mapped_gottoshop_categories[category]
                    gottoshop_product_category_1                    =
gottoshop_product_category['w']
        elif        not        gottoshop_product_category_2        and
gottoshop_product_category_1:
            if gender == 'unisex':
                if category in self.mapped_gottoshop_categories:
                    gottoshop_product_category                      =
self.mapped_gottoshop_categories[category]
                    gottoshop_product_category_2                    =
gottoshop_product_category['m']

        category = self._clear_from_tags(category)
        return         category,         gottoshop_product_category_1,
gottoshop_product_category_2

    def __init__(self,
        country_postfix: str,
        params:Parameters
    ):
        self.country_postfix = country_postfix
        self.params = params
        self.__credentials                                              =
service_account.Credentials.from_service_account_file('other/gc_confi
g.json')
        with open("other/config.json", "r") as file:
            config_data = json.load(file)

        if self.country_postfix not in ['en', 'ie', 'ca', 'za', 'us',
'unspecified'] and not self.params.force_en_categories:
            with
open(f"other/categories_{self.country_postfix}.json", "r") as file:
                self.mapped_gottoshop_categories = json.load(file)
        else:
```

```
            with open("other/categories.json", "r") as file:
                self.mapped_gottoshop_categories = json.load(file)

        service           =           build('sheets',           'v4',
credentials=self.__credentials)

spreadsheet_id=config_data['gottoshop_categories_spreadsheet_id']
        result                                                        =
service.spreadsheets().values().get(spreadsheetId=spreadsheet_id,
range="Дерево").execute()
        values = result.get('values', [])
        values = [i[0:7] for i in values]
        columns = values[0]
        categories_df = pd.DataFrame(values[1:], columns=columns)
        categories_df  =  categories_df.drop(columns=["5",  "Приблизний
результат в H1. Для первинної перевірки"])

        self.categories_df         =         pd.DataFrame(values[1:],
columns=columns)
        self.categories_df   =   self.categories_df.drop(columns=["5",
"Приблизний результат в H1. Для первинної перевірки"])
        self.men_categories_df                                        =
self.categories_df[self.categories_df['UK: 1 рівень'] == "Men's"]
        self.women_categories_df                                      =
self.categories_df[self.categories_df['UK: 1 рівень'] == "Women's"]

class Scraper():
    """
    An  abstract  base  class  for  web  scraping.  It  provides  the
structure for scraping
    a  sitemap,  managing  proxies,  and  handling  scraping  tasks
asynchronously.
    """
    @abstractmethod
    def scrap_sitemap(self) -> List:
        """
        Abstract method to scrape the sitemap and return a list of
links.
        Must be implemented by subclasses.
        """
        pass

    async def start_scraping(self):
        """
        Starts the scraping process by retrieving links from the
sitemap, processing the links,
        and uploading the resulting data to an FTP server.
        """
        try:
            self.links = await self.scrap_sitemap()
            await self._get_links()

            remote_file_path = self.table_prefix

self.bq_handler.upload_feed_to_ftp(remote_file_path=f'{remote_file_pa
th}.xml')
        except Exception as e:
            print("Scraping error:", e)
```

```
            raise(e)

    async def _get_links(self):
        """
        Processes the retrieved links, handles proxy settings, and
performs scraping tasks.
        Manages changes to cookies, user agents, and referers as
specified in parameters.
        """
        if 'static' in self.params.proxy_type:
            self.proxy_handler.check_proxies()
        temp_proxy = self.proxy_handler.get_proxy()

        # Change cookies
        if self.params.change_cookies:
            try:
                async                                          with
aiohttp.ClientSession(headers=self.params.headers) as session_object:
                    async  with  session_object.get(self.website_link,
proxy=f"{self.params.proxy_protocol}://"  +  temp_proxy,  timeout  =
self.params.timeout_time) as resp:
                        await resp.text()
                        status_code = resp.status
                        cookies = resp.cookies
                await session_object.close()

                if status_code == 200:
                    self.params.headers['Cookie']                  =';
'.join([f'{name}={value}' for name, value in cookies.items()])
                    print("Cookies were changed successfuly.")
                else:
                    if self.params.delete_cookies_after_error:
                        if 'Cookie' in self.params.headers:
                            del(self.params.headers['Cookie'])
                            print('Cookies were deleted.')
                    print("Cookies weren't changed!")
            except:
                if self.params.delete_cookies_after_error:
                    if 'Cookie' in self.params.headers:
                        del(self.params.headers['Cookie'])
                        print('Cookies were deleted.')
                print("Error. Cookies weren't changed!")

        # Change user agent
        if self.params.change_user_agent:
            user_agent                                             =
random.choice(self.params.user_agents).strip()
            self.params.headers['User-Agent'] = user_agent

        # Change referer
        if self.params.change_referer:
            self.params.headers['Referer']                         =
random.choice(self.links)


        tasks_object = []
        counter = 0
        write_counter = 0
```

```
        print("Start",
pd.Timestamp(datetime.now()).strftime(format="%Y-%m-%d %H:%M:%S"))
        obj_results = []
        logs = []

        parsing_links = self.links

        for link in parsing_links:
            if counter == self.params.chunk_size:
                print(counter,
pd.Timestamp(datetime.now()).strftime(format="%Y-%m-%d %H:%M:%S"))
                if tasks_object:
                    results = await asyncio.gather(*tasks_object)
                    if results:
                        for result in results:
                            if result:
                                if result[0]:
                                    logs.extend(result[0])
                                if result[1]:
                                    obj_results.extend(result[1])

                print("Number of logs:", len(logs))
                print("Number of results:", len(obj_results))
                if self.params.wait_after_chunk:
                    await asyncio.sleep(self.params.wait_after_chunk)

                if 'static' in self.params.proxy_type:
                    if self.params.check_static_proxies_after_chunk:
                        self.proxy_handler.check_proxies()

                # Change cookies
                if self.params.change_cookies:
                    try:
                        async                                    with
aiohttp.ClientSession(headers=self.params.headers) as session_object:
                            async                                with
session_object.get(self.website_link,
proxy=f"{self.params.proxy_protocol}://"   +   temp_proxy,   timeout   =
self.params.timeout_time) as resp:
                                await resp.text()
                                status_code = resp.status
                                cookies = resp.cookies
                        await session_object.close()

                        if status_code == 200:
                            self.params.headers['Cookie']          =';
'.join([f'{name}={value}' for name, value in cookies.items()])
                            print("Cookies        were        changed
successfuly.")
                        else:
                            if
self.params.delete_cookies_after_error:
                                if 'Cookie' in self.params.headers:

del(self.params.headers['Cookie'])
                                    print('Cookies were deleted.')
                            print("Cookies weren't changed!")
```

```
                    except:
                        if self.params.delete_cookies_after_error:
                            if 'Cookie' in self.params.headers:
                                del(self.params.headers['Cookie'])
                                print('Cookies were deleted.')
                        print("Error. Cookies weren't changed!")

                # Change user agent
                if self.params.change_user_agent:
                    user_agent                                          =
random.choice(self.params.user_agents).strip()
                    self.params.headers['User-Agent'] = user_agent
                    self.params.headers['Referer']                      =
random.choice(self.links)

                # Change referer
                if self.params.change_referer:
                    self.params.headers['Referer']                      =
random.choice(self.links)

                tasks_object = []
                counter = 0

            if write_counter == self.params.write_chunk_size:
                if not self.params.test:
                    clear_output()
                    print("Writing...")
                    if obj_results:
                        self.bq_handler.write_objects(obj_results)
                    if logs:
                        self.bq_handler.write_logs(logs)

                obj_results = []
                logs = []

                write_counter = 0
                if 'static' in self.params.proxy_type:
                    self.proxy_handler.check_proxies()
                    temp_proxy = self.proxy_handler.get_proxy()
                if self.params.wait_after_write:
                    await asyncio.sleep(self.params.wait_after_write)

            counter += 1
            write_counter += 1

tasks_object.append(asyncio.create_task(self._fetch_object(link)))

        if tasks_object:
            results = await asyncio.gather(*tasks_object)
            if results:
                for result in results:
                    if result:
                        if result[0]:
                            logs.extend(result[0])
                        if result[1]:
                            obj_results.extend(result[1])
        print("Number of logs:", len(logs))
        print("Number of results:", len(obj_results))
```

```
        if not self.params.test:
            if obj_results:
                self.bq_handler.write_objects(obj_results)
            if logs:
                self.bq_handler.write_logs(logs)

        obj_results = []
        logs = []

        return True

    @abstractmethod
    async def _fetch_object(self, url) -> Tuple[List, List]:
        """
        Abstract method to fetch and parse an object from the given
URL.
        Must be implemented by subclasses.

        :param url: The URL to fetch.
        :return: A tuple containing logs and the parsed object list.
        """
        if type(url) == list:
            # Link
            link = url[1]

            url = url[0]

        logs = []

        # Here you make requests, set request_url to whatever value
you need
        request_url = url

        logs, response = await self._make_request(request_url)
        if not response:
            return logs, []

        # This is the response you are going to work with. There can
be more than one, just copy and paste previous block
        html_code = response

        try:
            # Here goes yours parsing script
            pass

        except Exception as e:
            error_log = {
                "log_type": "error",
                "description": "Problem with scraping element",
                "link": url,
                "timestamp":
pd.Timestamp(datetime.now()).strftime(format="%Y-%m-%d %H:%M:%S"),
                "additional_data": e
            }
            print(error_log)
            logs.append(error_log)

            return logs, []
```

```
        return

    async def _make_request(self, url)-> Tuple[List, List]:
        """
        Makes an asynchronous HTTP request to the given URL and
handles proxy and header settings.

        :param url: The URL to request.
        :return: A tuple containing logs and the response text.
        """
        temp_headers = self.params.headers
        if self.params.change_user_agent_each_request:
            user_agent                                          =
random.choice(self.params.user_agents).strip()
            temp_headers['User-Agent'] = user_agent
        if self.params.change_referer_each_request:
            temp_headers['Referer'] = random.choice(self.links)

        temp_proxy = self.proxy_handler.get_proxy()
        try:
            session_timeout                                     =
aiohttp.ClientTimeout(total=None,sock_connect=self.params.timeout_tim
e,sock_read=self.params.timeout_time)
            async   with   aiohttp.ClientSession(headers=temp_headers,
timeout = session_timeout) as session_object:
                logs = []
                status_code = 0
                for i in range(0, self.params.number_of_retries):
                    if status_code in ['404', '410']:
                        error_log = {
                            "log_type": "error",
                            "description":   "Problem   with   scraping
request",
                            "link": url,
                            "timestamp":
pd.Timestamp(datetime.now()).strftime(format="%Y-%m-%d %H:%M:%S"),
                            "additional_data": f"{status_code}"
                        }

                        # print(error_log)
                        logs.append(error_log)
                        await session_object.close()
                        return logs, None

                    if self.params.test:
                        async with session_object.get(url, timeout =
self.params.timeout_time) as resp:
                            response = await resp.read()
                            status_code = resp.status
                            cookies = resp.cookies
                    else:
                        async       with       session_object.get(url,
proxy=f"{self.params.proxy_protocol}://"   +   temp_proxy,   timeout   =
self.params.timeout_time) as resp:
                            response = await resp.text()
                            status_code = resp.status
                            cookies = resp.cookies
```

```
                    if status_code == 200:
                        break

                if status_code != 200:
                    error_log = {
                        "log_type": "error",
                        "description":    "Problem    with    scraping
request",
                        "link": url,
                        "timestamp":
pd.Timestamp(datetime.now()).strftime(format="%Y-%m-%d %H:%M:%S"),
                        "additional_data": f"{status_code}"
                    }

                    print(error_log)
                    logs.append(error_log)
                    return logs, None

                if self.params.change_cookies_each_request:
                    self.params.headers['Cookie']                  ='
'.join([f'{name}={value}' for name, value in cookies.items()])
                await session_object.close()
                return logs, response

        except Exception as e:
            error_log = {
                "log_type": "error",
                "description": "Problem with scraping function",
                "link": url,
                "timestamp":
pd.Timestamp(datetime.now()).strftime(format="%Y-%m-%d %H:%M:%S"),
                "additional_data": e
            }
            print(error_log)
            logs.append(error_log)
            # await session_object.close()
            return logs, None

    # Constructor with parameters
    def __init__(self, params: Parameters,
        sitemap_url:str = None,
    ):
        """
        Constructor   to   initialize   the   Scraper   with   parameters,
BigQuery handler, and proxy handler.

        :param   params:   The   Parameters   object   containing   scraping
settings.
        :param sitemap_url: The URL of the sitemap to scrape.
        """
        self.params = params

        self.bq_handler = BigQuery_Handler(params=params)

        self.website_title = self.bq_handler.website_title
        self.website_link = self.bq_handler.website_link
        self.website_description                                        =
self.bq_handler.website_description
```

```
        self.table_prefix = self.bq_handler.table_prefix
        self.country_postfix = self.bq_handler.country_postfix

        self.sitemap_url = sitemap_url

        self.links = None

        self.proxy_handler = Proxy_Handler(params=params)
        self.helper              =              Helper(params=params,
country_postfix=self.country_postfix)
```

Файл “website_scraper.py”

```
from typing import List, Tuple
import aiohttp
import json

import pandas as pd
from datetime import datetime
from bs4 import BeautifulSoup

from scraping import Scraper, Parameters


class Website_Scraper(Scraper):
    async def scrap_sitemap(self) -> List:
        print("Scraping sitemap:", self.sitemap_url)
        if 'static' in self.params.proxy_type:
            self.proxy_handler.check_proxies()

        temp_proxy = self.proxy_handler.get_proxy()
        async with aiohttp.ClientSession(headers=self.params.headers)
as session_object:
            async      with      session_object.get(self.sitemap_url,
proxy=f"{self.params.proxy_protocol}://" + temp_proxy) as resp:
                response = await resp.text()

            sitemap_df = pd.read_xml(response)
            sitemap_df                                                =
sitemap_df[sitemap_df['loc'].str.contains("sitemap-products")]

            products_df = pd.DataFrame()
            for url in sitemap_df['loc']:
                print(url)
                async           with           session_object.get(url,
proxy=f"{self.params.proxy_protocol}://" + temp_proxy) as resp:
                    response = await resp.text()
                temp_df = pd.read_xml(response)
                products_df   =   pd.concat([products_df,   temp_df],
ignore_index=True)

            all_links = set(products_df['loc'].to_list())
            parsed_links = []
            if       self.bq_handler._table_exists(dataset_id="logs",
table=self.params.table_prefix + "_logs"):
```

```
                query = f'DELETE FROM
`{self.bq_handler.__client.project}.logs.{self.params.table_prefix}_l
ogs` WHERE log_type = "error" and additional_data = "403";'
                #    query    =    f'SELECT    DISTINCT(link)    FROM
`{self._Scraper__client.project}.logs.{self.table_prefix}_logs`;'
                query_result =
self.bq_handler.__client.query(query).result()
                if query_result:
                    print("All temporary errors were deleted.")

                query     =     f'SELECT     DISTINCT(link)     FROM
`{self.bq_handler.__client.project}.logs.{self.params.table_prefix}_l
ogs` WHERE timestamp >= TIMESTAMP_SUB(CURRENT_TIMESTAMP(), INTERVAL 3
DAY) or (log_type = "error" and additional_data != "403");'
                #    query    =    f'SELECT    DISTINCT(link)    FROM
`{self._Scraper__client.project}.logs.{self.table_prefix}_logs`;'
                query_result =
self.bq_handler.__client.query(query).result()
                parsed_links =
set(query_result.to_dataframe()['link'].to_list())

                links = list(all_links - parsed_links)
            else:
                links = list(all_links)

            print(f"All: {len(all_links)}\nParsed:
{len(parsed_links)}\nLeft: {len(links)}")
            return links

    async def _fetch_object(self, url) -> Tuple[List, List]:
        if type(url) == list:
            # Link
            link = url[1]

            url = url[0]

        logs = []

        # Here you make requests, set request_url to whatever value
you need
        request_url = url

        logs, response = await self.make_request(request_url)
        if not response or self.params.test:
            return logs, []
        html_code = response

        try:
            soup = BeautifulSoup(html_code, 'html.parser')
            json_script = soup.find('script', id="__NEXT_DATA__",
type="application/json")
            if not json_script:
                error_log = {
                    "log_type": "error",
                    "description": "No product json data",
                    "link": link,
                    "timestamp":
pd.Timestamp(datetime.now()).strftime(format="%Y-%m-%d %H:%M:%S"),
```

```
                    "additional_data": None
                }

                print(error_log)
                logs.append(error_log)
                return logs, []

            if not json_script.contents:
                error_log = {
                    "log_type": "error",
                    "description": "No product json data",
                    "link": link,
                    "timestamp":
pd.Timestamp(datetime.now()).strftime(format="%Y-%m-%d %H:%M:%S"),
                    "additional_data": None
                }

                print(error_log)
                logs.append(error_log)
                return logs, []

            all_product_data                                                =
json.loads(json_script.contents[0])['props']['pageProps']['product']

            # Item group id
            item_group_id = all_product_data['canonicalId']

            # Title
            title = None
            if 'mainTitle' in all_product_data:
                if all_product_data['mainTitle']:
                    title = all_product_data['mainTitle']
            if not title:
                title = all_product_data['name']

            # Description
            description = None
            if 'description' in all_product_data:
                if all_product_data['description']:
                    description = all_product_data['description']

            # Brand
            brand = None
            if 'brand' in all_product_data:
                if all_product_data['brand']:
                    brand = all_product_data['brand']['name']

            # Color
            color = None
            if 'color' in all_product_data:
                color = all_product_data['color']

            # Link
            link = url

            # Additional image link
            image_base_url = ""
            image_params = "?width=720"
```

```
            additional_image_link = [image_base_url + i['hash'] +
image_params for i in all_product_data['images']]

            # Image link
            image_link = additional_image_link[0]

            breadcrumbs_values = [i['name'].lower() for i in
all_product_data['breadcrumb']]

            # Gender and Age group
            gender_data = breadcrumbs_values[0]
            if gender_data == 'damenmode':
                gender = 'female'
                age_group = 'adult'

            elif gender_data == 'herrenmode':
                gender = 'male'
                age_group = 'adult'
            else:
                error_log = {
                    "log_type": "error",
                    "description": "Redundant element",
                    "link": link,
                    "timestamp":
pd.Timestamp(datetime.now()).strftime(format="%Y-%m-%d %H:%M:%S"),
                    "additional_data": None
                }

                # print(error_log)
                logs.append(error_log)
                return logs, []

            # Gottoshop product category
            category = None
            gottoshop_product_category_1 = None
            gottoshop_product_category_2 = None
            for category_value in reversed(breadcrumbs_values):
                last_category, last_gpc1, last_gpc2 =
self.helper.get_gottoshop_categories(title, gender, category_value)
                if last_gpc1 or last_gpc2:
                    if ('Unspecified' not in str(last_gpc1) and
last_gpc1) or ('Unspecified' not in str(last_gpc2) and last_gpc2):
                        if not gottoshop_product_category_1 and not
gottoshop_product_category_2:
                            gottoshop_product_category_1 = last_gpc1
                            gottoshop_product_category_2 = last_gpc2
                            category = last_category
                        # Check for level of category
                        elif gottoshop_product_category_1 or
gottoshop_product_category_2:
                            # Women
                            if gottoshop_product_category_1 and
last_gpc1 and 'Unspecified' not in str(last_gpc1):
                                if last_gpc1.count('>') >
gottoshop_product_category_1.count('>'):
                                    gottoshop_product_category_1 =
last_gpc1
                                    category = last_category
```

```
                            elif last_gpc1:
                                gottoshop_product_category_1        =
last_gpc1
                                category = last_category
                            # Men
                            if    gottoshop_product_category_2    and
last_gpc2 and 'Unspecified' not in str(last_gpc2):
                                if       last_gpc2.count('>')       >
gottoshop_product_category_2.count('>'):
                                    gottoshop_product_category_2    =
last_gpc2
                                    category = last_category
                            elif last_gpc2:
                                gottoshop_product_category_2        =
last_gpc2
                                category = last_category

            if (not gottoshop_product_category_1 or 'Unspecified' in
str(gottoshop_product_category_1))              and              (not
gottoshop_product_category_2        or        'Unspecified'        in
str(gottoshop_product_category_2)):
                breadcrumbs_values    =    [i.lower()    for    i    in
all_product_data['masterCategory'].split("|")]
                category = None
                gottoshop_product_category_1 = None
                gottoshop_product_category_2 = None
                for category_value in breadcrumbs_values:
                    last_category,     last_gpc1,     last_gpc2     =
self.helper.get_gottoshop_categories(title, gender, category_value)
                    if last_gpc1 or last_gpc2:
                        if  ('Unspecified'  not  in  str(last_gpc1)  and
last_gpc1) or ('Unspecified' not in str(last_gpc2) and last_gpc2):
                            if  not  gottoshop_product_category_1  and
not gottoshop_product_category_2:
                                gottoshop_product_category_1        =
last_gpc1
                                gottoshop_product_category_2        =
last_gpc2
                                category = last_category
                            # Check for level of category
                            elif    gottoshop_product_category_1    or
gottoshop_product_category_2:
                                # Women
                                if  gottoshop_product_category_1  and
last_gpc1 and 'Unspecified' not in str(last_gpc1):
                                    if     last_gpc1.count('>')     >
gottoshop_product_category_1.count('>'):
                                        gottoshop_product_category_1
= last_gpc1
                                        category = last_category
                                elif last_gpc1:
                                    gottoshop_product_category_1    =
last_gpc1
                                    category = last_category
                                # Men
                                if  gottoshop_product_category_2  and
last_gpc2 and 'Unspecified' not in str(last_gpc2):
```

```
                                    if     last_gpc2.count('>')     >
gottoshop_product_category_2.count('>'):
                                        gottoshop_product_category_2
= last_gpc2
                                        category = last_category
                                elif last_gpc2:
                                    gottoshop_product_category_2     =
last_gpc2
                                    category = last_category

            if (not gottoshop_product_category_1 or 'Unspecified' in
str(gottoshop_product_category_1))              and              (not
gottoshop_product_category_2        or        'Unspecified'        in
str(gottoshop_product_category_2)):
                category,               gottoshop_product_category_1,
gottoshop_product_category_2                                        =
self.helper.get_gottoshop_categories(title, gender, "Unspecified")

            # Product type
            product_type    =    ">".join([i.title()    for    i    in
breadcrumbs_values])

            # Condition
            condition = None

            ALL_ELEMENTS = []
            for dimension in all_product_data['dimensions']:
                for variant in dimension['variants']:
                    # ID
                    id = variant['id']

                    # Size
                    size = variant['size']['label']

                    # Price & Sale price
                    sale_price = None
                    price_value = variant['price']['withTax']
                    if variant['price']['reductions']:
                        sale_price_value = price_value
                        for               reduction               in
variant['price']['reductions']:
                            price_value += reduction['withTax']

                        if sale_price_value == 0 or price_value == 0:
                            continue

                        if price_value > sale_price_value:
                            sale_price                              =
str(float(sale_price_value/100)) + " " + variant['price']['currency']
                    price  =  str(float(price_value/100))  +  "  "  +
variant['price']['currency']

                    # Availability
                    if variant['availability']['quantity'] > 0:
                        availability = 'in_stock'
                    else:
                        availability = 'out_of_stock'
```

```
                    ELEMENT = {
                        "id": id,
                        "item_group_id": item_group_id,
                        "mpn": item_group_id,
                        "title": title,
                        "description": description,
                        "image_link": image_link,
                        "additional_image_link":
additional_image_link,
                        "link": link,
                        "gender": gender,
                        "age_group": age_group,
                        "brand": brand,
                        "color": color,
                        "size": size,
                        "availability": availability,
                        "price": price,
                        "sale_price": sale_price,
                        "condition": condition,
                        "product_type": product_type,
                        "gottoshop_product_category_1":
gottoshop_product_category_1,
                        "gottoshop_product_category_2":
gottoshop_product_category_2
                    }
                    required_columns    =    ['id',    'item_group_id',
'title', 'image_link', 'link', 'gender', 'age_group', 'availability',
'price', 'product_type']
                    for key in required_columns:
                        if not ELEMENT[key]:
                            error_log = {
                                "log_type": "error",
                                "description": f"No {key} data",
                                "link": url,
                                "timestamp":
pd.Timestamp(datetime.now()).strftime(format="%Y-%m-%d %H:%M:%S"),
                                "additional_data": None
                            }
                            print(error_log)
                            logs.append(error_log)

                            return logs, []
                    ALL_ELEMENTS.append(ELEMENT)

            log = {
                "log_type": "success",
                "description": "Object was successfuly created",
                "link": url,
                "timestamp":
pd.Timestamp(datetime.now()).strftime(format="%Y-%m-%d %H:%M:%S"),
                "additional_data": None
            }
            logs.append(log)

            return logs, ALL_ELEMENTS

        except Exception as e:
            error_log = {
```

```
                "log_type": "error",
                "description": "Problem with scraping element",
                "link": url,
                "timestamp":
pd.Timestamp(datetime.now()).strftime(format="%Y-%m-%d %H:%M:%S"),
                "additional_data": e
            }
            print(error_log)
            logs.append(error_log)

            return logs, []

    def __init__(self, params: Parameters, sitemap_url: str = None):
        super().__init__(params, sitemap_url)
```

Файл “app.py”

```
from website_scraper import Website_Scraper, Parameters
import asyncio
import os

def clear_terminal():
    """
    Clears the terminal screen.
    If the system is Windows, it uses 'cls', otherwise it uses
'clear' for macOS and Linux.
    """
    # Check if the system is Windows or not
    if os.name == 'nt':
        os.system('cls')  # Clear command for Windows
    else:
        os.system('clear')  # Clear command for macOS and Linux


def get_all_attributes(obj, prefix=''):
    """
    Recursively retrieves all attributes of an object, excluding
certain special attributes.

    :param obj: The object from which to retrieve attributes.
    :param prefix: The prefix to prepend to attribute names.
    :return: A dictionary containing attribute names and their
values.
    """
    all_attributes = {}
    for attribute in dir(obj):
        if attribute.startswith('__') or attribute == 'user_agents'
or attribute == 'headers':
            continue
        value = getattr(obj, attribute)
        if hasattr(value, '__dict__'):
            all_attributes         =         get_all_attributes(value,
f"{prefix}{attribute}.")
        else:
            all_attributes[f"{prefix}{attribute}"] = value

    return all_attributes
```

```
def show_choice(name, scraper: Website_Scraper):
    """
    Displays a menu of options for the user to choose from and returns the user's choice.

    :param name: The name of the website.
    :param scraper: An instance of the Website_Scraper class.
    :return: The user's choice as a string.
    """
    print(f"Website: {name}\n")

    print(f"1. Start scraping\n2. Change parameters\n3. Restore default parameters\n")
    print(f"Press Ctrl + C to go back.\n")
    scraping_choice = input("Choose the operation: ").strip()
    return scraping_choice

async def perform_operation(scraping_choice, name, params, scraper: Website_Scraper) -> Website_Scraper:
    """
    Performs the operation chosen by the user.

    :param scraping_choice: The user's choice of operation.
    :param name: The name of the website.
    :param params: An instance of the Parameters class containing scraper parameters.
    :param scraper: An instance of the Website_Scraper class.
    :return: The modified scraper instance.
    """
    if scraping_choice == '1':
        try:
            print(f"Scraping of {name}\n")
            await scraper.start_scraping()
            return scraper
        except KeyboardInterrupt:
            print(f"Cancelled scraping!\n")
            return scraper
        except Exception as e:
            print(f"Scraping failed after an error: {e}\n")
            return scraper
    elif scraping_choice == '2':
        changing_params = True
        try:
            while changing_params:
                attributes = get_all_attributes(scraper.params)
                print("This is current scraper parameters:\n")
                for key in attributes:
                    print(f"{key} = {attributes[key]};")
                print(f"\nPress Ctrl + C to go back.\n")

                param_choice = input("Enter the name of a parameter you want to change: ").strip()
                if param_choice in attributes:
                    new_value = input("Enter the new value of a parameter (for bool value set 0 or 1): ").strip()
                    if param_choice in [
                        "change_cookies",
```

```
                        "change_cookies_each_request",
                        "change_referer",
                        "change_referer_each_request",
                        "change_user_agent",
                        "change_user_agent_each_request",
                        "check_static_proxies_after_chunk",
                        "delete_cookies_after_error",
                        "force_en_categories",
                        "test",
                    ]:
                        if new_value in ['0', '1']:
                            setattr(scraper.params,      param_choice,
bool(int(new_value)))
                            clear_terminal()
                            print(f"Parameters      were      changed
succesfully!\n")
                            continue
                        else:
                            clear_terminal()
                            print(f'Wrong value were set!\n')
                            continue

                    elif param_choice in [
                        "chunk_size",
                        "feed_max_rows",
                        "number_of_retries",
                        "write_chunk_size",
                    ]:
                        if new_value.isdigit():
                            setattr(scraper.params,      param_choice,
int(new_value))
                            clear_terminal()
                            print(f"Parameters      were      changed
succesfully!\n")
                            continue
                        else:
                            clear_terminal()
                            print(f'Wrong value were set!\n')
                            continue

                    elif param_choice in [
                        "timeout_time",
                        "wait_after_chunk",
                        "wait_after_write",
                    ]:
                        if          new_value.isdecimal()          or
new_value.isdigit():
                            setattr(scraper.params,      param_choice,
float(new_value))
                            clear_terminal()
                            print(f"Parameters      were      changed
succesfully!\n")
                            continue
                        else:
                            clear_terminal()
                            print(f'Wrong value were set!\n')
                            continue
```

```
                    elif param_choice in [
                        "country_postfix",
                        "proxy_protocol",
                        # "proxy_type",
                        "table_prefix",
                        "website_description",
                        "website_link",
                        "website_title",
                    ]:
                        if type(new_value) == str:
                            setattr(scraper.params,      param_choice,
new_value)
                            clear_terminal()
                            print(f"Parameters      were      changed
succesfully!\n")
                            continue
                else:
                    print("Wrong choice! Try again.\n")
                    clear_terminal()
                    continue
        except KeyboardInterrupt:
            clear_terminal()
            return scraper
        except Exception as e:
            print(f"During  changing  parameters  next  error  occured:
{e}\n")
            input("Press 'Enter' to continue...")
            clear_terminal()
            return scraper
    elif scraping_choice == '3':
        scraper           =           Website_Scraper(params=params,
sitemap_url=scraper.sitemap_url)
        print("Parameters were restored to default!\n")
        return scraper


async def main():
    """
    The main function that runs the web scraper application.
    Initializes scrapers, handles user input, and performs scraping
operations.
    """
    names = ['Website 1']
    params  = [
            Parameters(
            website_title="",
            website_link="",
            website_description="",
            headers = {
                "Accept": "*/*",
                "Accept-Encoding": "gzip, deflate, br, zstd",
                "Accept-Language": "en-US,en;q=0.9",
                "Referer": "",
                "Sec-Fetch-Dest": "document",
                "Sec-Fetch-Mode": "navigate",
                "Sec-Fetch-Site": "same-origin",
                "Sec-Fetch-User": "?1",
                "Upgrade-Insecure-Requests": "1",
```

```
                "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64;
x64)   AppleWebKit/537.36   (KHTML,   like   Gecko)   Chrome/123.0.0.0
Safari/537.36",
            },
            country_postfix="de",
            change_user_agent = True,
            change_user_agent_each_request = True,
            change_referer = True,
            change_referer_each_request = True,
            write_chunk_size=50,
            # test = True,
        )
    ]
    try:
        scrapers = [
            Website_Scraper(
                params = Parameters(),
                sitemap_url="",
            )
        ]
    except Exception as err:
        print(f"Startup  error:  {err}\nRestart  the  application  or
contact   the   developer   with   the   screenshot   of   an   error:
developer@gmail.com")

    if len(names) == len(params) == len(scrapers):
        print("Hi! This is a test Scraping application! Let's see
what we got here...\n")
        if len(scrapers) <= 0:
            print(f"Sorry, but there is no available websites for
scraping :(\nContact your developer at \"developer@gmail.com\" or try
again later! Have a nice day.\n")
            exit()
        else:
            try:
                program_execution = True
                while program_execution:
                    print(f"List of websites for scraping:\n")
                    for i in range(len(scrapers)):
                        print(f"{i + 1}. {names[i]}")
                    print(f"\nPress Ctrl + C to exit the app.\n")
                    website_choice = None
                    website_choice = input(f"Choose the number of a
website you want to scrap: ").strip()
                    try:
                        if           int(website_choice)           in
list(range(len(scrapers) + 1))[1:]:
                            scraping_execution = True
                            scrapers[int(website_choice)    -    1]    =
Website_Scraper(params=params[int(website_choice)         -         1],
sitemap_url=scrapers[int(website_choice) - 1].sitemap_url)
                        else:
                            raise Exception("Bad choice!")
                    except:
                        print(f'Wrong choice! Try again\n')
                        clear_terminal()
                        continue
                    try:
```

```
                        clear_terminal()
                        while scraping_execution:
                            scraping_choice                                 =
show_choice(names[int(website_choice)                -                1],
scrapers[int(website_choice) - 1])
                            if  scraping_choice  not  in  ['1',  '2',
'3']:
                                print(f'Wrong choice! Try again\n')
                                continue

                            clear_terminal()
                            scrapers[int(website_choice) - 1] = await
perform_operation(scraping_choice,  names[int(website_choice)  -  1],
params[int(website_choice) - 1], scrapers[int(website_choice) - 1])

                    except KeyboardInterrupt:
                        scraping_execution = False
                        clear_terminal()
                        continue

            except KeyboardInterrupt:
                program_execution = False
                print("\nHave a nice day!")
                input("Press 'Enter' to close the window...")
                exit()
    else:
        raise  Exception(f"Critical  error  in  code.  Bad  number  of
elements.\n№ of names: {len(names)}; № of params: {len(params)}; № of
scrapers: {len(scrapers)}")

if __name__ == "__main__":
    asyncio.run(main())
```

ДОДАТОК В

Диск із кваліфікаційною роботою бакалавра