**КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**

**ІМЕНІ ТАРАСА ШЕВЧЕНКА**

Факультет комп’ютерних наук та кібернетики

Кафедра інтелектуальних програмних систем

**Кваліфікаційна робота**

**на здобуття освітнього рівня бакалавра**

за спеціальністю 121 Інженерія програмного забезпечення

на тему:

**РОЗРОБКА ТЕЛЕГРАМ-БОТУ ДЛЯ ПОШУКУ ОПТИМАЛЬНИХ ПРОПОЗИЦІЙ АВТО З ВИКОРИСТАННЯМ ТЕХНОЛОГІЇ ВЕБСКРАПІНГУ**

Виконав студент 4-го курсу

Іван ІЛЬЇН

Науковий керівник:

доцент, кандидат фіз.-мат. наук

Оксана ШКІЛЬНЯК

Засвідчую, що в цій роботі немає запозичень з

праць інших авторів без відповідних

посилань.

Студент

Роботу розглянуто й допущено до захисту

на засіданні кафедри інтелектуальних

програмних систем

« 29 » травня 2024 р.,

протокол № 11

Завідувач кафедри

Олександр ПРОВОТАР

Київ – 2024**РЕФЕРАТ**

Обсяг роботи 45 сторінок, 21 рисунок, 7 джерел посилань

ВЕБСКРАПІНГ, ОТРИМАННЯ ТА АНАЛІЗ ДАНИХ, SELENIUM, SCRAPY, BEAUTIFULSOUP, ТЕЛЕГРАМ-БОТ

**Об’єкт роботи**. Процес пошуку та аналізу пропозицій вживаних авто на тематичних вебсайтах за допомогою телеграм-боту. Предметом роботи є сам телеграм-бот, який надає користувачам можливість зручного та ефективного пошуку авто на основі вказаних користувачем даних.

**Мета роботи**. Створення телеграм-бота для пошуку вживаних авто, використовуючи процес вебскрапінгу.

**Методи розробки.** Технології та інструменти програмної реалізації процесу вебскрапінгу. Інструменти для створення телеграм-бота.

**Інструменти розробки.** Мова програмування Python,інструмент для веброзробки Selenium, фреймворк Scrapy, бібліотека BeautifulSoup, фреймворк Aiogram.

**Результат роботи.** У даній роботі було розроблено телеграм-бот для пошуку вживаних авто на тематичних сайтах за допомогою вебскрапінгу. Основна увага приділялася створенню парсера, який, використовуючи інструменти Scrapy, Selenium та BeautifulSoup, збирає та аналізує дані з сайтів продажу авто. Розроблений телеграм-бот, має полегшити пошук пропозицій вживаних авто, хоча не забезпечує достовірність інформації вказаної у знайдених пропозиціях.

**Сфера застосування**: Розроблений телеграм-бот, використовується для пошуку пропозицій вживаних авто. Проте, використовуючи аналогічний підхід, можна розробити програмне забезпечення для структурованого пошуку та аналізу інформації з потрібних тематичних джерел.

**ЗМІСТ**

[СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ 5](#_Toc168509114)

[ВСТУП 6](#_Toc168509115)

[РОЗДІЛ 1 ОПИС ТА ТЕОРЕТИЧНІ ВІДОМОСТІ 8](#_Toc168509116)

[1.1 Загальна інформація 8](#_Toc168509117)

[1.2 Пов’язані терміни 9](#_Toc168509118)

[1.3 Технології вебскрапінгу 10](#_Toc168509119)

[1.4 Опис типового процесу вебскрапінгу 11](#_Toc168509120)

[1.4.1 Загальний опис 11](#_Toc168509121)

[1.4.2 Отримання даних з HTML 13](#_Toc168509122)

[1.5 Огляд популярних платформ та інструментів для вебскрапінгу 16](#_Toc168509123)

[1.5.1 Bright Data 16](#_Toc168509124)

[1.5.2 ScrapingBee 17](#_Toc168509125)

[1.5.3 Octoparse 17](#_Toc168509126)

[РОЗДІЛ 2 ТЕХНОЛОГІЇ ТА ІНСТРУМЕНТИ ДЛЯ РОЗРОБКИ 19](#_Toc168509127)

[2.1 Інструменти розробки вебскрапінг-частини додатку 19](#_Toc168509128)

[2.1.1 Мова розробки Python 19](#_Toc168509129)

[2.1.2 Асинхронність, паралельність та модуль concurrent.futures 19](#_Toc168509130)

[2.1.3 BeautifulSoup 21](#_Toc168509131)

[2.1.4 Scrapy 21](#_Toc168509132)

[2.1.5 Selenium 22](#_Toc168509133)

[2.2 Інструменти створення телеграм-боту 23](#_Toc168509134)

[2.2.1 Загальна інформація про чат-бот 23](#_Toc168509135)

[2.2.2 Aiogram 24](#_Toc168509136)

[2.2.3 Telegram Bot API 25](#_Toc168509137)

[РОЗДІЛ 3 ПРАКТИЧНА ЧАСТИНА 26](#_Toc168509138)

[3.1 Загальний опис програмної реалізації 26](#_Toc168509139)

[3.2 Програмна реалізація вебскрапінгу 28](#_Toc168509140)

[3.2.1 Власний фільтр пропозицій веб-сайту та Selenium 28](#_Toc168509141)

[3.2.2 Отримання даних авто з веб-сайту та BeautifulSoup 31](#_Toc168509142)

[3.2.3 Вебкраулінг за допомогою Scrapy 33](#_Toc168509143)

[3.2.4 Опис алгоритму процесу вебскрапінгу 35](#_Toc168509144)

[3.3 Програмна реалізація телеграм-бота 36](#_Toc168509145)

[3.3.1 Створення телеграм-боту 37](#_Toc168509146)

[3.3.2 Обробка повідомлень 38](#_Toc168509147)

[3.3.3 Процес вебскрапінгу у телеграм-боті 40](#_Toc168509148)

[ВИСНОВКИ 43](#_Toc168509149)

[ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ 44](#_Toc168509150)

# СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

CSV (Comma-Separated Values) - формат для представлення табличних даних.

JSON ( JavaScript Object Notation) – формат для обміну даними на основі JavaScript.

XML (eXtensible Markup Language) – мова розмітки для визначення даних.

XLS – формат файлів Microsoft Excel.

API (application programming interface) – набір правил та специфікацій, які дозволяють різним програмним додаткам взаємодіяти один з одним.

RFC (Request for Comments) – офіційний документ Інженерної робочої групи Інтернету.

HTTP (HyperText Transfer Protocol ) – протокол прикладного рівня для мережі Інтернет

HTML (HyperText Markup Language ) – мова розмітки для перегляду вебсторінок.

DOM (Document Object Model) інтерфейс для доступу до вмісту HTML та XML документів.

CSS (Cascading Style Sheets) – мова для опису зовнішнього вигляду документа.

DRY (Don’t repeat yourself) – принцип спрямований на мінімізацію дублювання програмного коду

# 

# ВСТУП

**Сучасний стан об'єкта дослідження або розробки.** Мережа інтернет стала невід’ємною частиною життя сучасної людини. Кількість та зміст інформації, що можливо отримати, використовуючи інтернет, вражає об’ємом та різноманітністю. Незважаючи на такі переваги, є недолік такого стану речей. Велика кількість та різноманітність заважають користувачам знайти необхідну інформацію серед потоку даних у мережі інтернет. У випадку, коли необхідно аналізувати на зберігати велику кількість інформації, зазвичай, використовують вебскрапінг.

**Актуальність роботи та підстави для її виконання.** Швидке зростання ринку вживаних автомобілів та постійне оновлення пропозицій вимагають від потенційних покупців оперативності в отриманні інформації, щоб знайти найкращі умови купівлі. Використання телеграм-бота дозволить спростити пошук, адаптувавши процес під потреби користувача з мінімальними зусиллями.

Завдяки технології вебскрапінгу, телеграм-бот може систематично збирати дані з тематичних вебсайтів, що продають автомобілі, інформацію про ціни, моделі та інші значущі параметри. Це не тільки значно спрощує процес пошуку, але й дозволяє користувачам отримувати більш актуальну інформацію, що сприяє ухваленню обґрунтованих рішень про покупку.

**Мета й завдання роботи.** Метою кваліфікаційної роботи є розробка програмного забезпечення, що використовуючи технології вебскрапінгу, надасть користувачу оптимальні пропозиції вживаних авто з вебресурсів відповідної тематики.

Для досягнення поставленої мети необхідно виконати такі завдання:

1. Дослідити сутність та методи вебскрапінгу.
2. Проаналізувати та обрати інструменти програмної реалізації процесу вебскрапінгу.
3. Проаналізувати та обрати інструменти для інструменти для створення телеграм-боту.
4. Реалізувати парсер для отримання даних з тематичних вебресурсів.
5. Створити телеграм-бот та вбудувати в нього парсер.

**Об'єкт і методи дослідження або розроблення.** Об’єктом дослідження є телеграм-бот для пошуку пропозицій автомобілів на тематичних вебсайтах.

Засобами розробки є інструменти для вебскрапінгу такі, як Selenium, Scrapy, BeautifulSoup, що дозволяють зручно збирати дані з вебсайтів

Також, інструменти для розробки телеграм-боту, а саме, Aiogram та Telegram Bot API.

**Можливі сфери застосування.** Розроблений телеграм-бот може використовуватись для зручного пошуку пропозицій авто, не завантажуючи додаткове програмне забезпечення та не передивляючись власноруч тематичні сайти. У подальшому, схожим способом, можна створити телеграм-боти для пошуку будь-якої інформації за певною темою такі, як мистецтво, маркетинг, статті та багато іншого.

# 

# РОЗДІЛ 1 ОПИС ТА ТЕОРЕТИЧНІ ВІДОМОСТІ

## 1.1 Загальна інформація

**Вебскрапінг** - це процес отримання та перетворення інформації з вебсторінок, орієнтованих на перегляд у браузері, в структуровані дані. Може застосовуватись у багатьох галузях, наприклад:

* маркетинг: компанії використовують вебскрапінг для моніторингу цін конкурентів, збору інформації про продукти та перевірки відгуків споживачів;
* академічні дослідження: університети та наукові інститути можуть застосовувати вебскрапінг для збору даних з різних вебресурсів для наукових досліджень;
* рекрутинг: HR-фахівці можуть використовувати вебскрапінг для пошуку потенційних кандидатів, збору інформації про їхній досвід і навички з профільних сайтів або соціальних мереж;
* автоматизовані системи керування вмістом: компанії можуть збирати новини, статті, блоги, відео та інший вміст для публікації на власних платформах.

Вебскрапінг включає в себе такі способи отримання інформації:

* ручне отримання інформації шляхом копіювання та зберігання у необхідному форматі;
* використання програм, що імітують роботу людини з вебсторінками: шляхом підключення до вебсерверу напряму або за допомогою інструментів керування браузером.

Зазвичай використовують програмне забезпечення, що автоматизує процес вебскрапінгу, через більшу швидкість отримання інформації та здатність обробити великий обсяг даних за відносно короткий проміжок часу.

## 1.2 Пов’язані терміни

**Вебкраулінг**: процес автоматичного перегляду вебсайтів за допомогою програми “павука” або бота, що переходить за посиланнями з однієї сторінки на іншу. Головна мета вебкраулера — систематичне переглядання вебресурсу для індексації або збору вебсторінок.

**Індексація**: процес обробки інформації, отриманої з вебсторінок, і створення даних, які дозволяє швидко і ефективно знаходити інформацію.

**Машинозчитувані дані**: дані, які представлені в такому форматі, що можуть автоматично оброблятися комп'ютерною програмою без потреби у втручанні людини, забезпечуючи при цьому збереження семантичного змісту.

Ці дані зазвичай зберігаються у форматах, які легко зчитаються програмами, а саме CSV, JSON, XML, або через API, що дозволяє різним програмам взаємодіяти з ними.

**API** (Application Programming Interface): набір правил та специфікацій, які дозволяють різним програмним додаткам взаємодіяти один з одним. В контексті вебскрапінгу — це набір правил і специфікацій, які дозволяють програмам взаємодіяти з вебсервісом. API забезпечує стандартизований спосіб доступу до функціоналу і даних вебсервісу, що може включати збір інформації, управління даними, взаємодію з сервером тощо.

**Токенізація** — це розбиття секцій рядка вхідних символів на менші одиниці, такі як окремі слова чи терміни.

**HTTP** - протокол прикладного рівня для мережі Інтернет, визначений у документах RFC та RFC . Протокол реалізується у двох частинах додатків: клієнтської та серверної. Передача даних між клієнтом і сервером відбувається шляхом передачі повідомлень. Протокол HTTP визначає структуру цих повідомлень.

Методи HTTP-запиту:

* GET – запит на отримання даних.
* HEAD – запит на отримання даних без вмісту тіла документа.
* POST – внесення нових даних.
* PUT – оновлення або заміна існуючого запису.
* DELETE – видалення певних даних.
* OPTIONS – визначає, які HTTP-методи підтримуються для конкретного URL.

**Request** - зазвичай запит клієнта до сервера.

**Response** - зазвичай відповідь сервера клієнту

## 1.3 Технології вебскрапінгу

Як було зазначено вище, процес вебскрапінгу - це не тільки автоматизований збір даних з вебресурсу, а ще й ручне копіювання та зберігання необхідної інформації. Часто технології автоматизованого вебскрапінгу не можуть повністю замінити оцінку та обробку даних людиною. Отже, при обробці невеликого обсягу даних можна використовувати технологію ручного копіювання та зберігання даних.

**Відповідність текстових шаблонів**: простий, але потужний підхід до вилучення інформації з вебсторінок може базуватися на команді UNIX grep або засобах зіставлення регулярних виразів мов програмування, таких як Python. Команда grep у UNIX дозволяє шукати текст у файлах за допомогою регулярних виразів, ефективно вибираючи рядки, які відповідають заданому шаблону. Цей інструмент широко використовується для фільтрації і пошуку специфічної інформації у великих текстових файлах або потоках даних.

**HTML-парсинг**: аналіз вихідного коду HTML вебсторінки для отримання даних. Основна мета HTML-парсингу — проаналізувати структуру HTML і вилучити з неї корисні дані. Включає в себе побудову DOM-дерева. Враховуючи, що переважна більшість вебсторінок взятих з одного вебсайту, автоматично генеруються за одним шаблоном, використання HTML парсера значно спрощує процес отримання даних з певного ресурсу.

**Побудова DOM-дерева**: перетворення вихідного HTML вебсторінки на DOM(Document Object Mode), який представляє весь вміст сторінки у вигляді структурованого дерева об'єктів. Ця модель дозволяє детально взаємодіяти з кожним елементом сторінки, таким як теги, атрибути, текст та інші. Є частиною HTML-парсингу.

## 1.4 Опис типового процесу вебскрапінгу

### 1.4.1 Загальний опис

Зазвичай процес вебскрапінгу складається з двох елементів:

* ініціатор;
* ціль.

Ініціатор - програмне забезпечення, що використовує технології та інструменти вебскрапінгу для автоматизованого отримання даних з вебсайтів.

Ціль - дані, що необхідно отримати з вебсайту. Це можуть бути вміст сторінок, контактна інформація або форми. Загалом, будь-які дані вебресурсу.

Сам процес вебскрапінгу можна схематично розбити на три кроки.

**Крок 1** (рис. 1). Ініціатор запускає процес вебскрапінгу, створюючи HTTP-запити для взаємодії з вебсайтами і збору даних з них. Методи запитів можуть бути різними, наприклад, GET, POST, PUT, DELETE, HEAD та OPTIONS, описаних раніше.

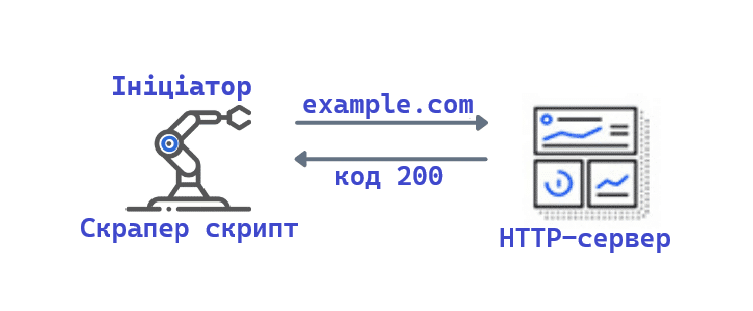


Рисунок 1 – Перший крок процесу вебскрапінгу

**Крок 2** (рис. 2). У випадку існування сторінки та вдалого підключення, вебсайт відправить у відповідь код 200. Після чого скрапер отримує HTML-код сторінки та почне процес збору неструктурованих даних.

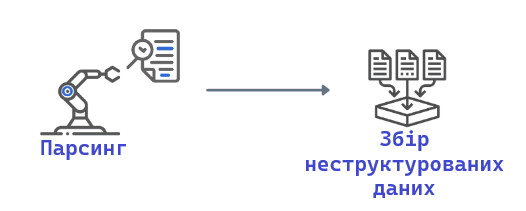


Рисунок 2 – Другий крок процесу вебскрапінгу

**Крок 3** (рис. 3). Починається процес аналізу даних з вебсторінки. Спочатку скрапер отримує необроблені дані, після чого структурує та зберігає інформацію відповідно до способу та вимог, що були прописані ініціатором. Як було описано раніше, зазвичай дані вихідні дані зберігаються у машинозчитуваних форматах: XLS, CSV, JSON.

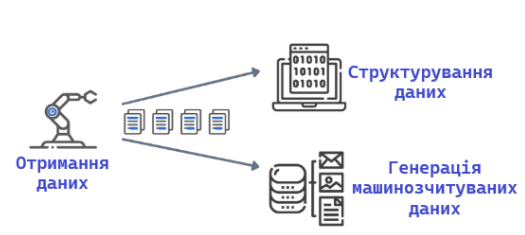


Рисунок 3 – Третій крок процесу вебскрапінгу

### 1.4.2 Отримання даних з HTML

Важлива частина вебскрапінгу - це безпосередньо вилучення інформації з вебсторінки. В свою чергу вебсторінка створюється за допомогою мови розмітки HTML або XHTML, де зберігаються класи, ідентифікатори, таблиці, списки, блоки, контейнери, тобто усі основні елементи вебсторінки. Разом з HTML використовують CSS.

CSS - мова таблиць та стилів, що описує зовнішній виду документу, написаного мовою HTML. Але може використовуватись з будь-якими XML-документами, такими як SVG.

Хоча теоретично можливо екстрагувати дані безпосередньо з сирого HTML за допомогою таких інструментів, як регулярні вирази, цей процес часто виявляється часозатратним і технічно складним. HTML, який був спеціально створений для машинного зчитування, може мати складну та різноманітну структуру. В таких умовах селектори CSS та XPath стають незамінними інструментами, оскільки вони дозволяють точно і ефективно вилучати потрібні елементи з документів HTML.

**CSS-селектори** — це шаблони, які застосовуються для ідентифікації та вибору елементів у HTML-структурі вебсторінки. Вони виявляються особливо корисними не тільки у стилізації сторінок, але й у вебскрапінгу, оскільки є більш точним і ефективним методом екстрагування даних з HTML-документів. Використання CSS-селекторів пропонує значні переваги, що робить їх переважним вибором для завдань вебскрапінгу.

CSS-селектор дає змогу ідентифікувати необхідний елемент HTML-документу за атрибутами. Найпопулярніші з них це class та id, що позначаються “.” та “#” відповідно. Наприклад, для отримання доступу до елементу, що записаний у HTML як <h1 class="card" id="text"> <h1> , необхідно використати селектори h1.card, h1#text або разом: h1.card#text.

За необхідності, отримати доступу до інформації, що знаходиться всередині інших елементів селектори записують через пробіл.

Приклад (рис. 4):

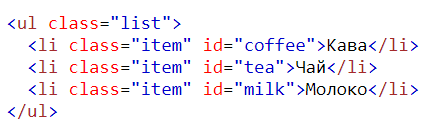


Рисунок 4 – HTML код для прикладу

CSS-селектор для отримання всіх елементів за тегом li з HTML коду: ul.list li.item

CSS-селектор для отримання елементу за тегом li, в якому міститься текст “Чай” з HTML коду: ul.list li#tea

Отже CSS-селектори - це простий та зручний інструмент ідентифікації елементів в HTML-коді вебсторінки, що широко використовується у вебскрапінгу.

**XPath-селектори** - селектори мови XML-шляхів, представляють собою мову шляхів, що використовується для визначення та навігації по елементах у документах XML та HTML.

Дозволяє вибирати конкретні вузли в HTML-структурі, використовуючи критерії, як-от розміщення в документі, назви тегів, атрибути або вміст. XPath є зручним інструментом при пошуку та виборі елементів на основі специфічних атрибутів, включаючи класи та ідентифікатори, що забезпечує точність та гнучкість у маніпуляціях з даними HTML. Існує два типи XPath, а саме абсолютний та відносний.

**Абсолютний XPath** представляє собою прямий метод для локалізації елементів у документі, але він має певний недолік: використання абсолютного XPath залежить від незмінності шляху до цільового елемента. Отже будь-які модифікації у документі, такі як додавання чи видалення елементів, можуть призвести до того, що вираз стає недієздатним, не здатний ідентифікувати вказаний елемент.

**Відносний XPath** ініціюється з будь-якої точки в структурі HTML, використовуючи подвійну косу риску (//) на початку виразу (рис. 5). Цей метод дозволяє ідентифікувати елементи на вебсторінці з будь-якого місця в її структурі, уникаючи необхідності формулювати довгі і складні шляхи. Відносний XPath є більш гнучким і переважно кращим вибором у порівнянні з абсолютним XPath, оскільки він не вимагає вказування повного шляху від кореневого елемента документа.



Рисунок 5 - Схема для побудови відносного XPath

XPath-селектор для отримання всіх елементів з HTML коду за тегом li: //ul[@class=’list’]//li[@class=’item’]

XPath-селектор для отримання елементу з HTML коду за тегом li, в якому міститься текст “Чай”: //ul[@class=’list’]//li[@id=’tea’]

## 1.5 Огляд популярних платформ та інструментів для вебскрапінгу

### 1.5.1 Bright Data

Компанія Bright Data є одним з лідерів у галузі збору вебданих, керуючи великою та різноманітною мережею проксі-серверів по всьому світу. Її мережа містить мільйони проксі, що є оптимальними для стратегій ротації IP-адрес, необхідних для ефективного вебскрапінгу.

На основі цієї інфраструктури, Bright Data пропонує широкий спектр інструментів та служб для вебскрапінгу, включаючи API для Web Scraper. Цей хмарний інструмент дозволяє використовувати налаштовувані API прикінцевих пристроїв для екстракції даних із популярних доменів, забезпечуючи масштабованість та надійність, а також усуваючи загальні технічні перешкоди, такі як протидія антибот-механізмам.

**Основні функції та переваги:**

1. Високий рівень надійності, безвідмовність роботи 99,9%.
2. Можливість необмеженого масштабування відповідно до потреб проекту.
3. Повна відповідність етичним стандартам та вимогам.
4. Цілодобова підтримка користувачів.
5. Включає багато особливостей таких як пакетна обробка запитів, автоматичний парсинг та валідація даних, ротація IP-адрес та користувацьких агентів, автоматичне вирішення капчі, рендеринг JavaScript, резидентні проксі-сервери, та доставка вебхуків.

**Недоліки:**

1. Сервіс є платним.
2. Вартість залежить від обсягу даних та типу домену, що може збільшити витрати залежно від масштабу проекту.

### 1.5.2 ScrapingBee

ScrapingBee пропонує API для вебпарсингу, яке спрощує процес екстракції даних з Інтернету. API автоматизує управління проксі-серверами та налаштування headless-браузера, дозволяючи розробникам зосередитися на самій екстракції даних. Цей інструмент орієнтований на розробників, які інтегрують функціонал парсингу у свої програмні рішення. Використання великої кількості проксі-серверів зменшує ризики обмежень та блокувань.

**Основні функції та переваги:**

1. Оплата здійснюється тільки за успішні запити, що зменшує витрати.
2. Розгалужена документація та численні публікації в блозі надають глибоке розуміння інструменту.
3. Висока ефективність на більшості сайтів.
4. Підтримка інтерактивних вебсайтів з виконанням JavaScript.
5. Автоматичне обходження антибот-заходів, включаючи рішення для капчі.

**Недоліки:**

1. Швидкість API може бути нижчою порівняно з аналогами.
2. Обмеження на паралельне виконання запитів.
3. Для ефективного використання необхідні технічні знання.

### 1.5.3 Octoparse

Octoparse — це один із провідних інструментів для вебскрапінгу, розроблений для користувачів без навичок програмування. Цей інструмент дозволяє екстрагувати неструктуровані дані з будь-яких вебсайтів і перетворювати їх у структуровані набори даних. Задачі збору даних налаштовуються через простий інтерфейс "вкажи і натисни", що робить цей інструмент доступним для користувачів без технічних знань.

**Основні функції та переваги:**

1. Інструмент для парсингу даних, що не вимагає навичок програмування.
2. Широкий вибір інтеграцій: Zapier, Google Диск, Google Таблиці, Dropbox, Slack, Hubspot, Salesforce, Airtable, Cloudmersive API.
3. Наявність безкоштовного плану та пробної версії розширених функцій.
4. Документація та довідковий центр доступні на кількох мовах, включаючи іспанську, китайську, французьку та італійську.
5. Підтримка складних дій на вебсайтах, таких як безкінечна прокрутка, навігація по сторінках, взаємодія з випадаючими списками та імітація курсору миші.

**Недоліки:**

1. Відсутність підтримки для Linux.
2. Деякі функції можуть бути складними для розуміння.

# РОЗДІЛ 2 ТЕХНОЛОГІЇ ТА ІНСТРУМЕНТИ ДЛЯ РОЗРОБКИ

## 2.1 Інструменти розробки вебскрапінг-частини додатку

### 2.1.1 Мова розробки Python

Python — це інтерпретована, об'єктно-орієнтована, високорівнева мова програмування. Вона включає високорівневі структури даних, що в поєднанні з динамічною типізацією, робить Python особливо привабливим для швидкої розробки програмного забезпечення та використання в якості скриптової мови або зв'язувальної мови для інтеграції існуючих компонентів. Простий синтаксис Python сприяє зручності навчання та читабельності коду, що значно знижує витрати на його підтримку. Python підтримує модулі та пакети, що сприяє модульності програм та повторному використанню коду. Інтерпретатор Python та широка стандартна бібліотека доступні безкоштовно для всіх основних платформ та можуть бути вільно розповсюджені.

### 2.1.2 Асинхронність, паралельність та модуль concurrent.futures

**Асинхронність**: дозволяє програмі продовжувати виконання інших завдань, не блокуючи основний потік виконання. Часто використовується для обробки операцій, які можуть тривати значний час, наприклад, читання даних із мережі або файлової системи. Асинхронний код зазвичай застосовує модель подій та виклики callbacks для повідомлення про завершення операції.

**Паралелізм**: надає можливість виконувати кілька завдань одночасно. Кожне завдання може виконуватись у власному потоці або процесі, в залежності від використовуваного механізму. Це значно підвищує ефективність обробки завдань, дозволяючи оптимально використовувати системні ресурси.

**Модуль concurrent.futures** в Python надає високорівневий інтерфейс для асинхронного програмування з використанням потоків та процесів. Цей модуль спрощує управління паралельними завданнями, дозволяючи розробникам використовувати однаковий інтерфейс для потоків і процесів, що полегшує використання.

На відміну від модулів threading або multiprocessing, concurrent.futures пропонує менш детальний контроль над низькорівневими аспектами управління потоками чи процесами, але водночас робить процес розробки більш інтуїтивним і менш схильним до помилок.

Основні класи модуля:

* **ThreadPoolExecutor**: використовується для створення пулу потоків, що дозволяє виконувати паралельні завдання в рамках одного процесу. Це корисно при виконанні великої кількості I/O-залежних завдань, таких як запити до мережі або файлової системи.
* **ProcessPoolExecutor**: використовується для створення пулу процесів, який дозволяє розподіляти обчислення. Це може значно прискорити виконання завдань, що вимагають паралельну обробку даних.

Обидва класи підтримують контекстне управління за допомогою оператора with, що спрощує управління ресурсами і забезпечує належне закриття пулів після виконання завдань. Класи також надають метод submit, який приймає функцію та аргументи і повертає Future об'єкт. Цей об'єкт дозволяє відслідковувати статус завдання та отримувати результати коли вони стають доступними.

### 2.1.3 BeautifulSoup

BeautifulSoup — це бібліотека на мові програмування Python, яка використовується для парсингу HTML та XML документів. Ця бібліотека надає інтуїтивно зрозумілі та зручні способи для пошуку та модифікації даних, що робить її популярним інструментом серед розробників для вебскрапінгу.

Однією з основних переваг BeautifulSoup є її здатність обробляти і виправляти погано сформований HTML-код, з яким часто доводиться стикатися під час збору даних з Інтернету.

BeautifulSoup підтримує різні парсери, такі як lxml та html5lib, що надає гнучкість вибору залежно від специфіки завдання. Це дозволяє вибрати найефективніший парсер для конкретного випадку використання, оптимізуючи таким чином процес парсингу.

Використання BeautifulSoup дозволяє розробникам ефективно збирати дані, що знаходяться у вільному доступі в Інтернеті, та зберігати їх у структурованій формі, що може бути використаним для подальшого аналізу та обробки. Цей інструмент є незамінним у різних областях застосування, від академічних досліджень до комерційних проектів, де потрібно швидко та ефективно обробляти великі обсяги вебінформації.

### 2.1.4 Scrapy

Scrapy — це безкоштовний та відкритий фреймворк для вебкраулінгу, написаний на Python. На даний момент його підтримує компанія Zyte (раніше відома як Scrapinghub), яка спеціалізується на розробці та послугах вебскрапінгу.

Архітектура проекту Scrapy побудована навколо "павуків" — самостійних скриптів, які отримують набір інструкцій. Відповідаючи принципам фреймворків на кшталт Django, які дотримуються DRY. Scrapy полегшує розробку та масштабування великих проектів, дозволяючи розробникам зручно та ефективно парcити вебресурси.

### 2.1.5 Selenium

Selenium — це інструмент для веброзробки з відкритим кодом, який використовується для автоматизації функцій вебперегляду. Розроблений у 2004 році, Selenium спочатку застосовувався для автоматичного тестування вебсайтів та додатків у різних браузерах, але згодом став популярним інструментом для вебскрапінгу. Selenium підтримує використання кількох мов програмування, включаючи Python, Java та C#.

Selenium складається з набору драйверів для різних браузерів та клієнтських бібліотек, написаних на різних мовах програмування, які використовуються для керування цими драйверами.

Використання Selenium полягає в створенні автоматизованого бота, який виконує всі дії з браузером, що зазвичай виконуються вручну. Цей інструмент дозволяє розробникам ефективно автоматизувати вебскрапінг та виконувати складні взаємодії з вебсайтами у рамках своїх проектів.

WebDriver — це інструмент з відкритим кодом, призначений для автоматизації тестування вебдодатків у різних браузерах. Він надає функції для переходу до вебсторінок, введення даних користувачем, виконання JavaScript та багато іншого.

## 2.2 Інструменти створення телеграм-боту

Для реалізації інтерфейсу додатку, було вирішено обрати телеграм-бот. У такого рішення є певні переваги. Наприклад відсутність необхідності інсталяції додаткового додатку на пристрій, більшість потенційної цільової аудиторії має обліковий запис та досвід користування мессенджером телеграм загалом, та телеграм-ботами зокрема. Отже, інтерфейс буде інтуїтивно зрозумілим для користувача. Також функціонал додатку дозволяє використання лише одного діалогового вікна з користувачем. Ще однією перевагою такого рішення є асинхронність роботи телеграм-боту, отже велика кількість користувачів зможе використовувати телеграм-бот.

### 2.2.1 Загальна інформація про чат-бот

На базовому рівні, чат-бот — це комп'ютерна програма, яка імітує процес спілкування з людиною (письмово або усно), дозволяючи користувачам взаємодіяти з цифровими пристроями, ніби вони спілкуються з реальною людиною. Чат-боти можуть бути простими - програми, що відповідають на прості запитання одним рядком тексту, так і складними, як цифрові асистенти, які навчаються та еволюціонують, забезпечуючи вищий рівень персоналізації, оскільки вони збирають та обробляють інформацію.

Завдяки цим характеристикам, чат-боти стають невід'ємною частиною цифрових стратегій багатьох організацій, відкриваючи нові шляхи для автоматизації обслуговування клієнтів, маркетингу та взаємодії.

Чат-бот в Telegram — це додаток, що функціонує на платформі мессенджера Telegram, налаштований на взаємодію з користувачами та автоматизацію певних завдань або надання інформації автоматично за запитом.

Користувачі можуть взаємодіяти з ботами, надсилаючи команди чи повідомлення, на які бот відповідають відповідно до вказаних розробником інструкцій.

### 2.2.2 Aiogram

Aiogram — це сучасний повністю асинхронний фреймворк для Telegram Bot API, написаний на Python за допомогою asyncio та aiohttp. Він спрощує та прискорює процес розробки ботів для Telegram.

Фреймворк Aiogram дозволяє розробникам використовувати асинхронне програмування, що підвищує продуктивність і ефективність ботів шляхом покращення обробки запитів. Завдяки інтеграції з asyncio і aiohttp, Aiogram забезпечує надійне управління асинхронними завданнями та HTTP-запитами в середовищі Python, що дозволяє розробникам створювати більш швидкі та масштабовані боти.

Крім того, Aiogram містить набір зручних інструментів для обробки вхідних повідомлень, налаштування клавіатур, обробки виключень і тому подібне, що робить процес розробки ботів більш інтуїтивно зрозумілим і менш трудомістким.

### 2.2.3 Telegram Bot API

Telegram Bot API — це потужний інтерфейс, розроблений для створення та управління ботами в мессенджері Telegram. Цей API надає розробникам широкий спектр інструментів для автоматизації взаємодій з користувачами, обробки повідомлень та інтеграції з різними зовнішніми сервісами.

Telegram Bot API дозволяє ботам виконувати такі дії, як надсилання та прийом текстових повідомлень, зображень, відео, файлів, міток на карті, та багато іншого. Боти можуть використовувати розширені клавіатури для зручності користувачів та взаємодіяти з користувачами у групових чатах та на каналах.

Однією з ключових особливостей Telegram Bot API є його висока продуктивність і здатність обробляти велику кількість запитів від користувачів, що робить його зручним інструментом для розробки ботів, призначених для масштабних застосувань і широкої аудиторії.

Telegram Bot API, також, шифрує трафік між ботом і серверами Telegram. Це забезпечує захист конфіденційності даних користувачів.

Завдяки своїй гнучкості, масштабованості та високому рівню безпеки, Telegram Bot API є популярним вибором серед розробників, які створюють інноваційні та ефективні боти для різноманітних цілей, від спрощення повсякденних завдань до забезпечення складних бізнес-процесів.

# 

# РОЗДІЛ 3 ПРАКТИЧНА ЧАСТИНА

## 3.1 Загальний опис програмної реалізації

Результатом практичної роботи є телеграм-бот, що надає користувачу посилання на пропозиції вживаних авто, отримані з тематичних вебсайтів, за допомогою вебскрапінгу.

Після того як користувач ініціює роботу з телеграм-ботом, починається діалог, у якому користувач має вказати дані авто, яку необхідно знайти.

Дані авто (рис. 6):

* марка авто,
* модель авто,
* діапазон ціни,
* діапазон років випуску,
* характеристики авто,
* опис.

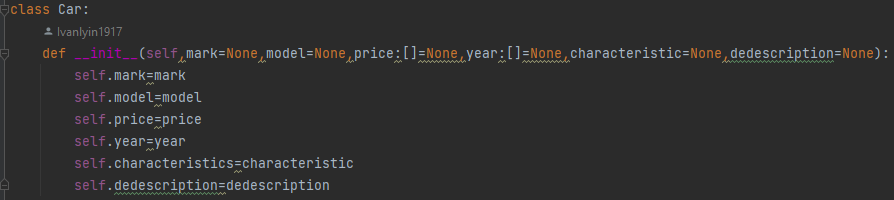


Рисунок 6 - Дані автомобіля

Характеристики (рис. 7):

* Продавець,
* коробка передач,
* тип палива,
* об’єм двигуна,
* тип приводу,
* тип кузова,
* колір,
* пробіг.

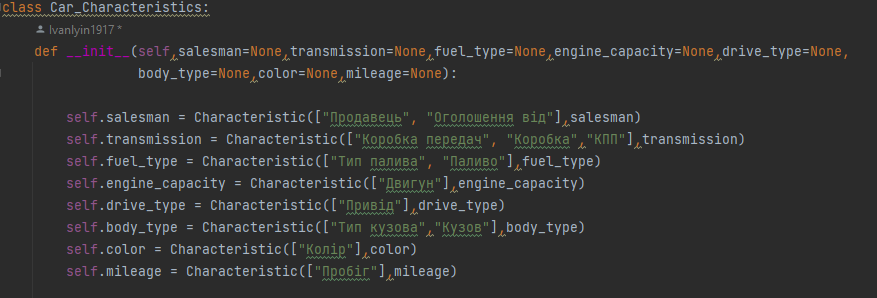


Рисунок 7 - Характеристики автомобіля

Для зручності було створено додатковий клас Characteristic (рис. 8), полями якого є ключі та значення певної характеристики.

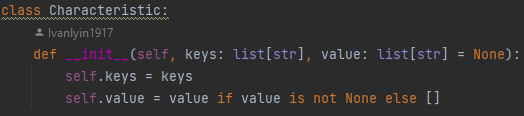


Рисунок 8 – Об’єкт характеристики

Після отримання необхідних даних, починається процес вебскрапінгу. Після чого отримуємо список, який сортується за схожістю описів, вказаних користувачем та продавцем автомобіля. Користувач отримує цей список, після чого має можливість ввести нові дані або повторно ініціювати процес пошуку.

## 3.2 Програмна реалізація вебскрапінгу

Так як кожен сайт може мати свою архітектуру та спосіб взаємодії з користувачем, а процес вебскрапінгу у телеграм-боті охоплює більше ніж один сайт, необхідно створювати окремий парсер для кожного вебсайту.

Далі описано складові парсера на прикладі сайту <https://automoto.ua/uk/>

### 3.2.1 Власний фільтр пропозицій вебсайту та Selenium

Першим кроком у загальному процесі вебскрапінгу в телеграм-боті є використання Selenium. Це дозволяє відкинути невідповідні пропозиції.

Наприклад, на сайті <https://automoto.ua/uk/> більш ніж чотириста тисяч пропозицій. Перебирання всіх варіантів потребує занадто багато часу. Тому фільтрація за допомогою Selenium дає змогу скоротити список пропозицій до більш сприятливих значень.

Процес фільтрації з Selenium реалізований як функція, що викликається у подальшому.

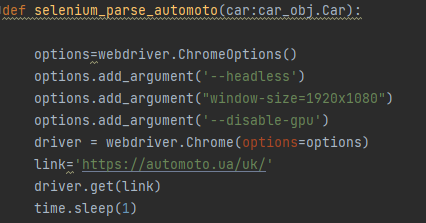


Рисунок 9 – Початок роботи методу з Selenium

Початок роботи методу (рис. 9):

1. Так як Selenium використовує WebDriver, спочатку треба вказати параметри: --headless, --disable-gpu, window-size=1920x1080

**--headless**: режим безголової роботи дозволяє запускати браузер без графічного інтерфейсу користувача. Це означає, що браузер працює у фоновому режимі, не відображаючи візуальний інтерфейс. Перевагою є підвищення продуктивності та швидкість виконання, оскільки ресурси системи не витрачаються на відтворення графічного інтерфейсу. Ідеально підходить для використання в середовищах без графічних інтерфейсів.

**--disable-gpu**: відключення візуалізації графіки, яке забороняє браузеру використовувати апаратну підтримку графічного процесора для рендеринга сторінок. Запобігає потенційним проблемам сумісності або пов'язаними з апаратним прискоренням, особливо на серверах або у нестандартних конфігураціях.

**window-size=1920x1080**: встановлює розміри вікна браузера на 1920 пікселів у ширину і 1080 пікселів у висоту. Цей параметр використовується для визначення фізичних розмірів вікна браузера при його запуску. Цей параметр особливо корисний при запуску у безголовому режимі, де візуальний розмір вікна може не бути явно визначений. Вказання розміру вікна забезпечує більш передбачувану поведінку браузера, зменшуючи ризики, пов'язані з різними візуальними режимами перегляду сторінок.

1. Створюється новий екземпляр WebDriver Chrome, передаючи в нього налаштовані параметри через об'єкт options. Цей WebDriver використовується для управління браузером.
2. Задається змінна link, яка містить URL-адресу, на яку потрібно перейти.
3. Викликається метод get на об'єкт драйвера, передаючи URL, що вказує драйверу Chrome відкрити вказану вебсторінку.
4. Зупиняється виконання на 1 секунду. Це дає час вебсторінці на завантаження перед тим, як виконувати подальші дії. Це необхідно для забезпечення того, що всі елементи на сторінці були повністю завантажені.

Надалі зупинка виконання на одну секунду буде зустрічатися постійно у методі фільтрації з Selenium з причини описаної вище.

Після відкриття стартової сторінки, вказується параметри. Далі описано процес взаємодії Selenium з браузером на прикладі встановлення значення марки авто (рис. 10).

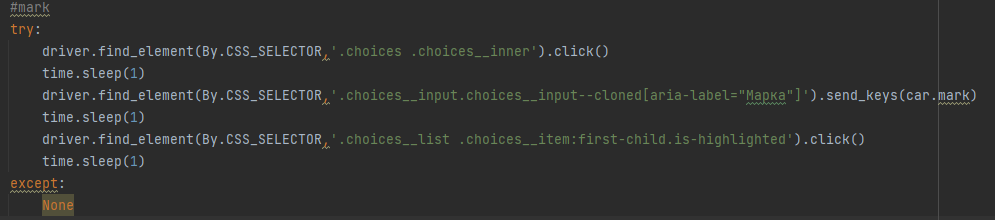


Рисунок 10 – Встановлення значення марки

Для взаємодії з елементом вебсторінки, спочатку необхідно його знайти, це можливо зробити за допомогою метода WebDriver .find\_element, як аргумент цього метода використовується спосіб пошуку, у наведеному прикладі використовується пошук за допомогою CSS-селектор, що був описаний раніше.

Взаємодія з знайденим елементом відбувається за допомогою різних методів, у наведеному прикладі це:

* .click: натискання на елемент,
* .send\_keys: ввести текст, заданий в аргументі до функції, в елемент. Зазвичай це текстове поле.

Опис процесу встановлення значення марки:

1. Знаходиться поле, що відповідає за значення марки, та відбувається натискання на цей елемент.
2. Вводиться значення задане у об’єкті, що передали в аргумент.
3. Знаходиться перший елемент у список можливих варіантів, що з'являється після заповнення текстового поля.

За схожим патерном відбувається взаємодія з іншими елементами вебсторінки. Також, важливий аспект - якщо елемент відсунутий за межі видимої області вікна браузера, взаємодія з ним неможлива. Тому необхідно виконати пролистування до необхідного елементу (рис. 11).

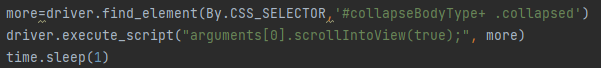


Рисунок 11 - Пролистування за допомогою Selenium

Після того, як були вказані всі необхідні характеристики автомобіля, метод припиняє роботу WebDriver повертає поточне посилання на якому завершився процес фільтрації (рис. 12).

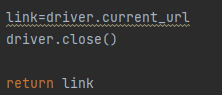


Рисунок 12 – завершення роботи методу

### 3.2.2 Отримання даних авто з вебсайту та BeautifulSoup

Необхідною частиною загального парсера є отримання даних авто з вебсайту. Для цього створені методи, що використовують бібліотеку BeautifulSoup.

Приклад методу для пропозицій з вебсайту <https://automoto.ua/uk/>:

**Початок роботи методу** (рис. 13)**:**

1. **Визначення URL**: ініціалізація змінної url містити URL адресу вебсторінки, з якої буде здійснюватися витягування даних.
2. **Відправлення HTTP запиту**: requests.get(url) виконує HTTP GET запит до вказаної URL адреси. Цей метод повертає об'єкт відповіді, який містить вміст сторінки.
3. **Парсинг вмісту сторінки**: BeautifulSoup(response.content, features="html.parser") використовується для аналізу вмісту вебсторінки.
4. **Створення об'єкту soup**: Змінна res є об'єктом BeautifulSoup. Дозволяє отримати доступ до HTML елементів цієї сторінки, щоб можна було легко знаходити та екстрагувати дані.

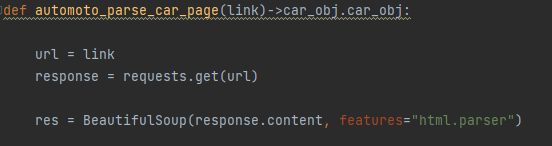


Рисунок 13 – Початок роботи методу з BeautifulSoup

**Екстракція даних в вебсторінки**:

Результатом виконання методу є об’єкт класу в якому зберігаються такі дані як назва, що на даному вебсайті містить модель, марку та рік випуску, що окремо обробляється і зберігається в окрему змінну, також характеристики та опис продавця.

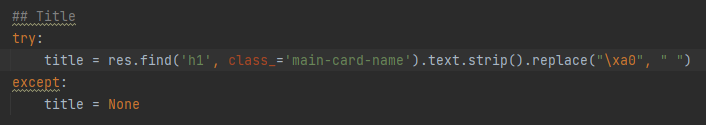


Рисунок 14 – Отримання назви пропозиції

Далі описано процес екстракції даних на прикладі отримання назви (рис. 14):

1. **Метод find()**:
   * res.find('h1', class\_='main-card-name') використовується для пошуку першого тегу <h1> з класом main-card-name у вмісті вебсторінки, який був заздалегідь оброблений і збережений у змінній res;
   * цей метод повертає перший знайдений HTML елемент, який відповідає заданим критеріям.
2. **Атрибут text**: використовується для отримання всього тексту, який міститься в обраному HTML елементі. В цьому випадку, він екстрагує текст знайденого тегу <h1>.
3. **Метод strip()**: використовується для видалення всіх початкових і кінцевих пробілів з тексту.
4. **Метод replace()**: .replace("\xa0", " ") замінює всі входження спеціального символу нерозривного пробілу (Unicode-символ U+00A0) на звичайний пробіл. Це важливо, оскільки нерозривні пробіли можуть призводити до помилок при обробці тексту або порівнянні рядків.

У такий спосіб екстрагуються всі необхідні дані, які в кінці роботи метода зберігаються у об’єкті класу, що повертається методом. Дані необхідні для подальшого зіставлення з даними, що вказав користувач.

### 3.2.3 Вебкраулінг за допомогою Scrapy

Наступним етапом є краулінг, який починається з вебсторінки по посиланню, що було отримано з методу, який описаний у пункті вище.

Краулінг реалізується за допомогою фреймворку Scrapy, а програмною реалізацією клас, який є похідним від базового класу Spider. Цей клас автоматизує процес збору даних з вебсторінок, що належать до певного домену.

**Основні атрибути класу**:

* name: Унікальна назва павука, що використовується Scrapy для ідентифікації павука;
* allowed\_domains: Список доменів, який обмежує область скрапінгу павуком до дозволених доменів. Це зменшує ризик збору даних з непризначених джерел;
* start\_urls: Початкові URL-адреси, з яких павук починає процес скрапінгу. Зазначення початкових адрес спрощує контроль над початком збору даних та орієнтацію павука на конкретних вебсторінках.

**Метод parse**. Основний метод parse автоматично викликається Scrapy при отриманні відповіді від вебсайту. Метод обробляє відповідь HTTP і вилучає необхідні дані або здійснює подальші запити. В методі parse використовуються селектори для отримання інформації із структури HTML-сторінки, що дозволяє ефективно збирати потрібні дані.

Інформацією, що екстрагується з вебсторінки, є посилання на пропозиції. У випадку, якщо присутній елемент що відповідає за перехід на наступну вебсторінку, метод викликається повторно але вже для наступної вебсторінки.

Після отримання посилання вебсторінки, на якій знаходиться пропозиція, викликається метод для парсингу вебсторінки за допомогою BeautifulSoup, що описаний раніше, та відбувається співставлення даних від користувача та даних, отриманих з поточної вебсторінки. Якщо співставлення даних має позитивний результат, то об’єкт, у якому зберігаються дані поточної пропозиції, додається до списку.

**Метод start\_requests**: перевизначений метод базового класу Spider в Scrapy, який ініціює початкові HTTP-запити до вебсайтів. Цей метод перебирає URL-адреси, вказані в атрибуті start\_urls класу, та створює для кожного з них об'єкт запиту за допомогою функції scrapy.Request.

Кожен запит конфігурується з наступними параметрами:

* callback=self.parse: Вказує метод parse як функцію зворотного виклику, яка буде обробляти відповіді на запити;
* dont\_filter=True: Запобігає фільтрації запитів дуплікатів, дозволяючи багаторазово обробляти одні й ті ж URL;
* meta: Словник з додатковими налаштуваннями, такими як відключення повторних спроб ("dont\_retry": True) та встановлення часу очікування завантаження ("download\_timeout": timeout).

### 3.2.4 Опис алгоритму процесу вебскрапінгу

Отже, програмна реалізація вебскрапінгу відбувається за таким алгоритмом:

1. За допомогою Selenium, відбувається фільтрація усіх пропозицій з вебсайту, використовуючи власний фільтр вебсайту. Результатом є посилання на вебсторінку, яка відображає пропозиції, які відповідають побажанням користувача.
2. Відбувається вебкраулінг за допомогою Scrapy, початковою сторінкою якого є вебсторінка за посиланням отриманим у минулому пункті. В процесі вебкраулінгу екстрагуються посилання на вебсторінку з пропозицією. Так як залишається можливість того, що у першому пункті фільтрація відбулась некоректно, наприклад через погане інтернет-з’єднання, відбувається додаткова фільтрація пропозицій, шляхом співставлення даних введених користувачем та даних отриманих з вебсторінки поточної пропозиції. Екстракція даних поточної пропозиції відбувається за допомогою BeautifulSoup.
3. Список об’єктів класу, що було отримано після вебкраулінгу, сортується за найбільшою схожістю опису користувача та опису пропозиції з вебсторінки.
4. Результатом роботи загального парсера є відфільтрований та відсортований список пропозицій.

## 3.3 Програмна реалізація телеграм-бота

Як було зазначено раніше, використання телеграм-бота як користувацького інтерфейсу, є вдалим рішенням в межах даного проекту, його переваги вже описквалися.

Основним алгоритмом дій телеграм-бота є:

* Запит у користувача даних авто, що необхідно знайти,
* Запуск парсера, та отримання списку пропозицій,
* Вивід списку, або повідомлення, що пропозицій не знайдено.

Перед програмною реалізацією необхідно зареєструвати бота за допомогою BotFather, який є інструментом від розробників Telegram для створення і управління ботами:

1. **Створення нового бота**. Вводиться команда /newbot у чат з BotFather. BotFather запитає про ім'я бота та його унікальне ім'я користувача (username), яке має закінчуватися на bot. Наприклад, example\_bot.
2. **Завершення реєстрації бота**. Після вибору імені для нового бота, BotFather надасть унікальний токен доступу (API Key). Цей токен є ключем для програмного доступу до бота через Telegram API.
3. **Збереження токена**. Токен необхідно зберегти в безпечному місці, оскільки він потрібний для програмного управління ботом.

Далі наведено кроки реалізації телеграм боту за допомогою бібліотеки Aiogram.

### 3.3.1 Створення телеграм-боту

Ініціалізація (рис. 15) телеграм-бота відбувається за такими кроками:

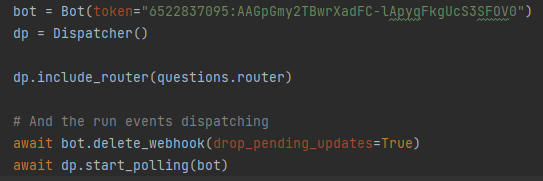


Рисунок 15 – Ініціалізація бота

1. **Ініціалізація бота**: Bot(token="...") створює новий екземпляр бота за допомогою токену, отриманого від BotFather під час реєстрації бота в Telegram.
2. **Створення диспетчера**: Dispatcher() ініціалізує диспетчер для управління потоками повідомлень і команд, які надходять від користувачів**.**
3. **Інтеграція роутера з обробниками**: dp.include\_router(questions.router) включає роутер з попередньо визначеними обробниками, що знаходяться в модулі questions. Це дозволяє боту реагувати на специфічні запити користувачів, наприклад, відповідати на запитання.
4. **Видалення вебхука**: await bot.delete\_webhook(...) видаляє вебхук, щоб переконатися, що бот не отримує подвійні оновлення та очищає всі незавершені оновлення. Це забезпечує стабільну роботу бота при запуску.
5. **Запуск опитування**: await dp.start\_polling(bot) ініціює асинхронне опитування API Telegram для отримання оновлень. Цей метод тримає бота в активному стані, слухаючи і реагуючи на нові повідомлення та команди від користувачів.

### 3.3.2 Обробка повідомлень

Початок роботи телеграм-бота відбувається після команди /start. Після чого бот пропонує користувачу почати роботу з команди /set. Починається процес введення користувачем даних авто. Бот запитує у користувача по пунктам дані. Давши відповідь на запит бота, користувач бачить наступне питання, та відповідає на нього.

Приклад реалізації (рис. 16) взаємодії користувача з ботом:

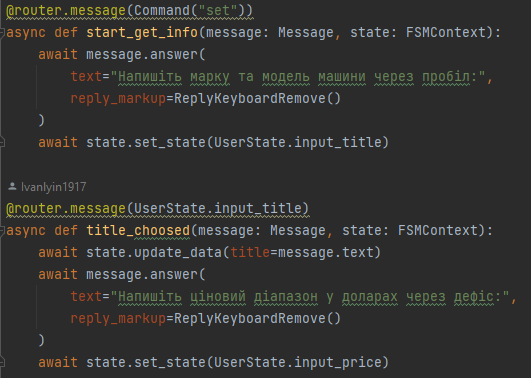


Рисунок 16 – Взаємодія з користувачем

1. Команда /set ініціює початок опитування, на початку якого бот пропоную користувачу ввести марку та модель авто. Та встановлює поточний стан розмови, що використовуються для відстеження того, що користувач знаходиться у фазі вводу інформації про марку і модель автомобіля.
2. Наступний метод запускається відразу після того, як користувач відповість. Відповідь користувача зберігається, бот пропонує ввести наступні дані - діапазон цін.

Параметр reply\_markup=ReplyKeyboardRemove() вказує на видалення будь-якої існуючої клавіатури з попередніх відповідей. Це створює чистий інтерфейс для користувача, щоб ввести потрібну інформацію без зайвих елементів на екрані.

Для зручності деякі дані пропонується ввести шляхом натискання на кнопку. Для цього вказується параметр: reply\_markup (рис. 17).

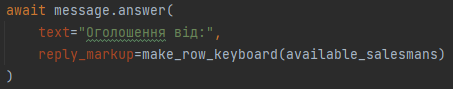


Рисунок 17 – Встановлення кнопок для відповіді

make\_row\_keyboard - метод, що створює список кнопок за заданим масивом назв (рис. 18).

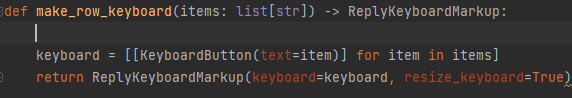


Рисунок 18 – Створення списку кнопок

Масивом назв є список найпопулярніших варіантів, та додаткові кнопки, що дозволяють ввести свій варіант та/або якщо варіантів декілька, пропустити питання. У випадку пропуску характеристика залишається порожньою, отже, на етапі вебскрапінгу, фільтрація по цій характеристиці не відбувається.

Приклад (рис. 19):

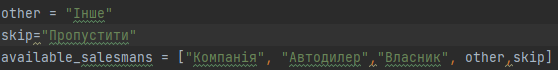


Рисунок 19 – Варіанти відповіді

Після того як, користувач введе всі запитувані дані, бот виведе їх, та запропонує пошук або повторне введення даних (рис. 20).

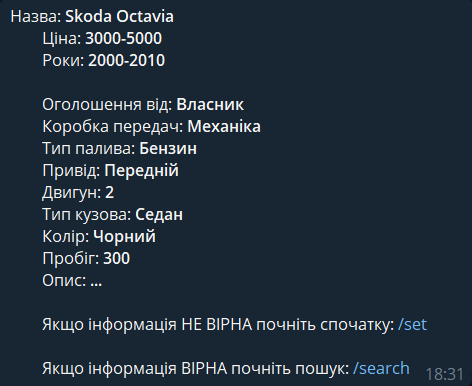


Рисунок 20 – Повідомлення з користувацькими даними

У випадку, коли користувач вибирає пошук починається процес вебскрапінгу, результат якого бот виводить користувачу. Якщо пошук відбувся успішно, але кількість пропозицій перевищує певне число n, то виводяться та видаляються перші n, а за командою /more виводяться та видаляються наступні n, так відбувається поки користувач не захоче повторно ввести дані та почати пошук або поки кількість не стане менше рівною n. Так зроблено для того, щоб не було переповнення повідомлення, що телеграм може відправити.

### 3.3.3 Процес вебскрапінгу у телеграм-боті

Використання телеграм-бота, як користувацький інтерфейс, передбачає велику кількість користувачів, які не повинні чекати своєї черги. Тому процес вебскрапінгу має бути паралельним.

Запуск функції для парсингу даних з вебсайту, використовуючи бібліотеку concurrent.futures (рис. 21) для паралельного виконання та asyncio для асинхронної інтеграції.

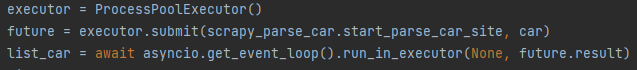


Рисунок 21 – Запуск краулера

Опис кожного кроку процесу:

1. **Створення екземпляру виконавця**: executor = ProcessPoolExecutor() ініціалізує об'єкт ProcessPoolExecutor. Цей об'єкт управляє пулом процесів, що дозволяє виконувати функції паралельно в окремих процесах, ізольованих один від одного.
2. **Запуск завдання в пулі процесів**: future = executor.submit(...). Об'єкт future представляє собою майбутній результат операції парсингу, дозволяючи перевіряти статус виконання і отримувати результат коли він буде готовий.
3. **Асинхронна інтеграція**:

asyncio.get\_event\_loop().run\_in\_executor(None,future.result) викликається для асинхронного отримання результату з future. None як перший параметр вказує на використання стандартного виконавця подій, замість пулу процесів, оскільки future.result вже є методом, який отримує результат з процесу.

Таким чином, процес вебскрапінгу буде відбуватися для кожного користувача паралельно.

Як було сказано раніше, деякі характеристики не використовуються у фільтрації пропозицій за допомогою фільтра вебсайта, якщо користувач не вказав їх. Також, можливий випадок, коли фільтрація за вказаними характеристиками призводить то великої кількості пропозицій. У будь-якому випадку час очікування користувачем відповіді зросте. Отже, було вирішено розділити процес вебкраулінгу.

Краулер, що отримує посилання на пропозиції, переходить по сторінкам з пропозиціями по порядку. Перший запуск краулера починається з вебсторінки по посиланню, отриманого з методу для фільтрації пропозицій за допомогою Selenium, проходить n сторінок, та виконує вище описані етапи процесу вебскрапінгу, виводить результат користувачу, зберігає посилання на вебсторінку з порядковим номером n+1. Наступний запуск краулера починається з вебсторінки за посиланням збереженим раніше та виконує дії описані раніше. Якщо кількість вищевказаних вебсторінок, що містять посилання на про пропозиції, менша за n, то краулер проходить їх всі за один раз.

Таким чином обробляється по n сторінок. Користувач, за бажанням, може продовжувати пошук, або почати спочатку. Варто зазначити, що подальші запуски краулера займають менше часу, ніж перший запуск, так як не використовується метод фільтрації за допомогою Selenium.

Емпіричним шляхом було визначено, що зручно проходити від трьох до п’яти сторінок за один запуск вебкраулера.

Продовжити пошук можна за командую /continue. Використання цієї команди пропонується користувачу після перегляду усіх пропозицій, отриманих після кожного запуску краулера.

### 

# ВИСНОВКИ

У роботі розроблено телеграм-бот для пошуку пропозицій вживаних авто серед тематичних вебсайтів з використанням технологій вебскрапінгу. Основна увага була приділена процесу вебскрапінгу, що дозволяє збирати та аналізувати інформацію з сайтів. Тому основною ціллю було написання парсера, який взаємодіє з користувачем через телеграм-бот, який є вдалим вибором, користувацького інтерфейсу, через доступність та зручність використання.

У роботі досліджено методи процесу вебскрапінгу. Були проаналізовані та обрані інструменти для програмної реалізації цього процесу, розглянуті інструменти для створення телеграм-боту.

В ході виконання програмної реалізації було розроблено парсер, що екстрагує дані з веб-сайтів продажу вживаних авто, за допомогою таких інструментів, як Scrapy, Selenium та BeautifulSoup. Ці інструменти були обрані через зручність їх використання. Враховуючи, що кожен сайт має свою архітектуру та особливості взаємодії, у подальшому можливе розширення пошуку шляхом написання схожих парсерів та об’єднання їх результатів.

Застосування даного телеграм-бота полегшить пошук оптимальних пропозицій вживаних авто, серед тематичних вебсайтів. Але варто враховувати, що достовірність отриманої інформації залежить, в першу чергу, від тієї особи чи осіб, хто створив пропозицію, отже телеграм-бот лише шукає, серед поданої інформації.

У подальшому, використовуючи схожий принцип роботи, можна створити подібні телеграм-боти для пошуку необхідної інформації з однієї теми, це необхідно для того, щоб її можна було структурувати та співставляти.

Загалом, можна сказати, що встановлена мета була досягнута і всі поставлені завдання були успішно виконані. Розроблений телеграм-бот може бути корисним для осіб, які планують купувати, авто, та відповідає зазначеним вимогам.

# ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Документація мови Python[Електронний ресурс] - Режим доступу до ресурсу: https://docs.python.org/3/

2. Документація інструменту Selenium [Електронний ресурс] - Режим доступу до ресурсу: https://www.selenium.dev/documentation/

3. Документація фреймворку Scrapy [Електронний ресурс] - Режим доступу до ресурсу: https://docs.scrapy.org/en/latest/

4. Документація бібліотеки BeautifulSoup [Електронний ресурс] - Режим доступу до ресурсу: https://www.crummy.com/software/BeautifulSoup/bs4/doc/

5. Документація фреймворку Aiogram[Електронний ресурс] - Режим доступу до ресурсу: https://docs.aiogram.dev/en/latest/

6. Mitchell R. Web Scraping with Python – Gravenstein Highway North, Sebastopol: 2018, Second Edition – 306 c.

Книги: два або три автори

7. J.F. Kurose, K.W. Ross Computer Networking: A Top-Down Approach – University of Massachusetts, Amherst: 2017, 7th Edition – 856 с.