ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка розподіленої системи моніторингу веб-ресурсів»

на здобуття освітнього ступеня бакалавра
зі спеціальності 123 Комп'ютерної інженерії
(код, найменування спеціальності)
освітньо-професійної програми Комп'ютерна інженерія
(назва)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ *Денис ОВЧИННІКОВ*
(підпис) *Ім'я, ПРІЗВИЩЕ здобувача*

Виконав: здобувач вищої освіти гр. КІД-42
Денис ОВЧИННІКОВ
Ім'я, ПРІЗВИЩЕ
Керівник: Віталій ВОЛОХІН
науковий ступінь, вчене звання Ім'я, ПРІЗВИЩЕ
Рецензент: _____
науковий ступінь, вчене звання Ім'я, ПРІЗВИЩЕ

Київ – 2026

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут Інформаційних технологій

Кафедра 123 Комп'ютерної інженерії

Ступінь вищої освіти бакалавр

Спеціальність 123 Комп'ютерної інженерії

Освітньо-професійна програма Комп'ютерна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри
Комп'ютерної інженерії
_____ Наталія ЛАЩЕВСЬКА
_____
«_____» _____ 2026р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Овчинніков Денис Андрійович
(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: Розробка розподіленої системи моніторингу веб-ресурсів
керівник кваліфікаційної роботи Віталій ВОЛОХІН,
(Ім'я, ПРІЗВИЩЕ, науковий ступінь, вчене звання)
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «24» лютого 2026р. № 56
2. Строк подання кваліфікаційної роботи «5» червня 2026р.
3. Вихідні дані до кваліфікаційної роботи: науково-технічна література: Python 3 documentation, Selenium Browser Automation Project Documentation, Scrapy Documentation
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Парсинг інформації веб-ресурсів
2. Створення системи доступу користувача до отриманої інформації
5. Перелік ілюстративного матеріалу: *презентація*
6. Дата видачі завдання«24» лютого 2026р.

РЕФЕРАТ

Обсяг роботи 51 сторінка, 21 рисунок, 7 джерел посилань

Мета роботи – Створення системи моніторингу вебресурсів уживаних авто та надання користувачу інформації через телеграм-бота.

Об’єкт дослідження – процес автоматизованого збору та аналізу даних з тематичних веб-ресурсів у рамках розподіленої системи моніторингу.

Предмет дослідження – розподілена система моніторингу веб-ресурсів на базі телеграм-боту.

Короткий зміст роботи: Сучасний інтернет містить величезну кількість веб-ресурсів, що постійно оновлюються. Ручний моніторинг таких ресурсів є трудомістким і неефективним, особливо коли йдеться про регулярне відстеження змін на багатьох сайтах одночасно. Тож постає питання, як автоматизувати процес збору та аналізу даних з веб-ресурсів таким чином, щоб користувач отримував актуальну інформацію без необхідності вручну переглядати кожен сайт. Це досягається шляхом побудови розподіленої системи моніторингу, компоненти якої відповідають за збір, обробку та передачу даних користувачу. Роль інтерфейсного компонента такої системи виконує телеграм-бот, що забезпечує зручну взаємодію з користувачем. Для реалізації модуля збору даних використовуються інструменти веб-скрапінгу — Scrapy, Selenium та BeautifulSoup, а для побудови інтерфейсу — фреймворк Aiogram.

На основі проведеного дослідження було з'ясовано актуальність використання розподіленого підходу при побудові систем моніторингу веб-ресурсів, розглянуто архітектуру такої системи, етапи її реалізації та приклад практичного застосування на базі телеграм-боту.

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

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ 7

ВСТУП 8

РОЗДІЛ 1 ОПИС ТА ТЕОРЕТИЧНІ ВІДОМОСТІ 11

1.1 Загальна інформація 11

1.2 Базові поняття та термінологія системи моніторингу 12

1.3 Технології вебскрапінгу 14

1.4 Опис типового процесу вебскрапінгу 15

1.4.1 Загальний опис 15

1.4.2 Механізми отримання даних з HTML-коду 17

1.5 Огляд популярних платформ та інструментів для вебскрапінгу 20

1.5.1 Bright Data 20

1.5.2 ScrapingBee 21

1.5.3 Octoparse 22

РОЗДІЛ 2 ТЕХНОЛОГІЇ ТА ІНСТРУМЕНТИ ДЛЯ РОЗРОБКИ 23

2.1 Інструменти розробки вебскрапінг-частини додатку 23

2.1.1 Мова розробки Python 23

2.1.2 Асинхронність, паралельність та модуль concurrent.futures 24

2.1.3 BeautifulSoup 25

2.1.4 Scrapy 25

2.1.5 Selenium 26

2.2 Інструменти створення телеграм-боту 26

2.2.1 Загальна інформація про чат-бот 27

2.2.2 Aiogram 27

2.2.3 Telegram Bot API 28

РОЗДІЛ 3 ПРАКТИЧНА ЧАСТИНА 30

3.1 Загальний опис програмної реалізації 30

3.2 Програмна реалізація вебскрапінгу 32

3.2.1 Власний фільтр пропозицій вебсайту та Selenium 32

3.2.2 Отримання даних авто з вебсайту та BeautifulSoup 35

3.2.3 Вебкраулінг за допомогою Scrapy 37

3.2.4 Опис алгоритму процесу вебскрапінгу 39

3.3 Програмна реалізація телеграм-бота 40

3.3.1 Створення телеграм-боту 41

3.3.2 Обробка повідомлень 42

3.3.3 Процес вебскрапінгу у телеграм-боті 46

ВИСНОВКИ 49

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ 50

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. На сьогоднішній день глобальна мережа Інтернет виступає ключовим середовищем для генерації, збереження та дистрибуції інформаційних потоків. Стрімке збільшення обсягів контенту та поява великої кількості гетерогенних веб-ресурсів створюють ситуацію інформаційного перенасичення. За таких умов кінцеві користувачі стикаються із проблемою неефективного та часомісткого пошуку релевантних даних серед масивів неструктурованої інформації. Для автоматизації процесів збору, систематизації та довготривалого зберігання динамічних веб-даних стандартні пошукові інструменти є малоефективними, що зумовлює активне впровадження технологій автоматизованого витягування даних, відомих як веб-скрапінг та веб-майнінг.

Актуальність роботи. Необхідність оперативного моніторингу веб-простору є критично важливою у сферах із високою динамікою оновлення інформації. Яскравим прикладом такого середовища є вторинний ринок автотранспорту, де актуальність комерційних пропозицій може змінюватися щохвилинно. Потенційні покупці та аналітики ринку потребують інструментів, які здатні акумулювати дані з багатьох первинних джерел у режимі реального часу.

Розробка та впровадження *розподіленої системи моніторингу* дозволяє вирішити цю задачу шляхом рознесення обчислювальних потужностей: ізольовані модулі збору даних (парсери) можуть незалежно функціонувати на різних вузлах, забезпечуючи безперебійний інтелектуальний аналіз цільових сайтів. Використання сучасних месенджерів, зокрема платформи Telegram, як інтерфейсу кінцевої доставки сповіщень, дозволяє кінцевому користувачу миттєво отримувати структуровану інформацію (ціни, технічні характеристики, параметри моделей) згідно з гнучко налаштованими фільтрами. Це оптимізує процес прийняття рішень, мінімізує рутинну пошукову роботу та демонструє практичну ефективність спроектованої розподіленої архітектури в реальних ринкових умовах.

Метою кваліфікаційної роботи є проектування, програмна реалізація та валідація розподіленої системи моніторингу гетерогенних веб-ресурсів із модульною архітектурою збору даних та інтегрованим асинхронним сервісом миттєвого сповіщення користувачів (на прикладі агрегації даних вторинного авторинку).

Для досягнення детермінованої мети було поставлено та вирішено такі **завдання**:

1. Проаналізувати теоретичні засади, патерни та сучасні методи автоматизованого витягування структурованих і неструктурованих даних із веб-джерел (веб-скрапінг та веб-майнінг).
2. Провести порівняльний аналіз та обґрунтувати вибір програмних стеків і бібліотек для побудови високоефективних асинхронних збирачів контенту.
3. Дослідити архітектурні особливості та обрати інструментарій для розробки клієнт-серверного інтерфейсу взаємодії на базі сучасних месенджерів.
4. Програмно реалізувати відмовостійкі модулі парсингу (агенти збору даних) для роботи з веб-платформами різного ступеня складності (включаючи ресурси з динамічним рендерингом контенту).
5. Здійснити інтеграцію розроблених модулів парсингу в єдину розподілену систему та налаштувати логіку передачі інформаційних пакетів через інтерфейс сповіщень.
6. Провести тестування та оцінку ефективності функціонування системи в умовах реального навантаження під час моніторингу цільових онлайн-майданчиків.

Об’єктом дослідження є архітектурні рішення та програмні процеси автоматизованого збору й дистрибуції інформаційних потоків у глобальній мережі.

Засобами розробки виступає кросплатформний стек технологій мови Python, зокрема:

- Спеціалізовані інструменти веб-девелопменту та скрапінгу: фреймворк *Scrapy* (для організації конвеєрів обробки даних), автоматизатор *Selenium* (для обходу обмежень динамічного клієнтського рендерингу) та бібліотека *BeautifulSoup* (для обробки DOM-структури документів).
- Технології побудови інтерфейсу користувача: асинхронний фреймворк *Aiogram* та інструменти взаємодії з *Telegram Bot API*.

Практичне значення та сфери застосування результатів

Розроблена розподілена система вирішує прикладну задачу оптимізації пошуку та моніторингу актуальних пропозицій на ринку вживаних автомобілів. Впровадження системи дозволяє кінцевому користувачу відмовитися від ручного моніторингу багатьох тематичних сайтів або встановлення громіздкого стороннього софту, перенісши весь процес взаємодії у звичне середовище месенджера.

Завдяки реалізованому принципу модульності та розподіленості, запропонована архітектурна модель є універсальною. Окремі компоненти збору даних можуть бути безболісно конфігуровані або замінені для аналізу абсолютно інших веб-ресурсів: від моніторингу наукових публікацій і новинних стрічок до агрегації аналітичних даних у сферах маркетингу, фінансів чи логістики.

РОЗДІЛ 1 ОПИС ТА ТЕОРЕТИЧНІ ВІДОМОСТІ

1.1. Загальна інформація

Автоматизований збір даних із веб-ресурсів (веб-скрапінг, англ. *web scraping*, або веб-майнінг контенту) є процесом синтаксичного аналізу, витягування та трансформації неструктурованого контенту веб-документів, оптимізованих для сприйняття людиною через графічний інтерфейс браузера, у структуровані формати даних (JSON, XML, CSV або реляційні БД) [6]. У контексті побудови розподілених інформаційних систем веб-скрапінг виступає як первинний агент збору сирої інформації, що забезпечує безперебійне наповнення сховищ даних для їх подальшої аналітичної обробки.

Масштабованість та універсальність технологій веб-скрапінгу зумовлюють їх широке застосування в архітектурі сучасних аналітичних платформ для вирішення таких комплексних завдань:

- **Аналіз конкурентного середовища та бізнес-моніторинг:** автоматизація відстеження цінових стратегій на маркетплейсах, моніторинг асортименту продукції конкурентів, збір та сентимент-аналіз відгуків споживачів.
- **Агрегація науково-дослідних даних:** формування репрезентативних текстових корпусів для навчання моделей машинного навчання, збір статистичних масивів для соціологічних, економічних та лінгвістичних досліджень.
- **Інтелектуальний рекрутинг та OSINT:** моніторинг профільного ринку праці, консолідація профілів потенційних кандидатів із платформ професійних соціальних мереж та дошок вакансій.
- **Синдикація контенту та інформаційний моніторинг:** автоматизоване наповнення систем керування вмістом (CMS) шляхом агрегації новинних стрічок, тематичних публікацій, блогів та мультимедійних метаданих.

За методологією реалізації та рівнем автоматизації процеси витягування контенту з глобальної мережі класифікують на такі ключові підходи:

1. **Інтерактивний (ручний) збір:** безпосереднє копіювання інформації оператором системи. Даний метод характеризується низькою ефективністю, високою ймовірністю помилок і є непридатним для систем моніторингу реального часу.
2. **Низькорівнева програмна емуляція (статичний парсинг):** пряма взаємодія з веб-серверами через HTTP-запити за допомогою спеціалізованих програмних бібліотек. Цей підхід забезпечує максимальну швидкість обробки, проте є неефективним при роботі з сучасними веб-додатками, що використовують динамічний рендеринг контенту на стороні клієнта (Single Page Applications, SPA).
3. **Високорівнева автоматизація браузерів (динамічний парсинг):** використання програмних драйверів (Headless браузерів) для імітації поведінки реального користувача, включаючи виконання JavaScript-сценаріїв, обробку подій кліку, прокрутки та заповнення форм.

У сучасних розподілених системах моніторингу пріоритет надається саме комбінованим програмним комплексам автоматизації. Це зумовлено експоненціальним зростанням обсягів інформації та критичною необхідністю забезпечення високої швидкості паралельної обробки даних, що є неможливим за умов використання ручних або неадаптованих методів збору.

1.2. Базові поняття та термінологія системи моніторингу

Для розробки системи моніторингу веб-ресурсів необхідно визначити основні терміни та технології, на яких базується її робота.

Веб-краулінг (web crawling) — це процес автоматичного обходу сайтів за допомогою спеціальних програм (роботів або «павуків»). Головна задача краулера — послідовно переходити за посиланнями з однієї сторінки на іншу, щоб вивчити структуру сайту або зібрати сторінки для подальшої індексації.

Індексація — обробка та сортування зібраної з веб-сторінок інформації. Це потрібно для створення бази даних, у якій потім можна швидко знайти необхідні матеріали.

Машинозчитувані дані — це інформація, збережена у форматі, який комп'ютерна програма може обробляти автоматично, без участі людини. Для обміну такими даними найчастіше використовують формати **JSON, XML або CSV**.

API (Application Programming Interface) — готовий набір правил та інструментів, який дозволяє різним програмам спілкуватися між собою. У нашому випадку API дозволяє модулям системи моніторингу безпечно й швидко отримувати дані безпосередньо від веб-сервісів.

Токенізація — процес розбиття тексту на дрібні логічні частини (токени), наприклад, на окремі слова чи фрази. Це допомагає системі аналізувати вміст сторінки та фільтрувати оголошення.

HTTP (Hypertext Transfer Protocol) — основний мережевий протокол для передачі даних в Інтернеті. Взаємодія відбувається через обмін двома типами повідомлень:

- **Запит (Request)** — повідомлення від нашої програми до сервера сайту (наприклад, прохання відкрити сторінку).
- **Відповідь (Response)** — те, що сервер повертає назад (код сторінки, картинки або повідомлення про помилку).

Для точного формування запитів використовуються такі основні **HTTP-методи**:

- GET – прохання надіслати дані (наприклад, відкрити сторінку з авто);
- POST – передача нових даних на сервер (наприклад, відправка заповненої форми);
- PUT / PATCH – оновлення вже існуючої інформації;
- DELETE – видалення певних даних;
- HEAD – запит тільки технічної інформації (заголовків) без завантаження всієї сторінки.

1.3. Технології вебскрапінгу

Процес збору інформації з веб-сторінок можна реалізувати різними методами — від найпростіших ручних дій до автоматизованого аналізу коду сторінки. Вибір конкретного підходу залежить від обсягу даних та складності цільового ресурсу.

Ручний збір даних

Найпростіший спосіб отримання інформації — це її самостійне копіювання користувачем у текстовий файл або таблицю. Цей метод не потребує програмування і підходить, коли потрібно обробити всього кілька сторінок. Проте ручний підхід стає абсолютно неефективним, якщо обсяг даних великий або інформація на сайті оновлюється щохвилини (як це відбувається на автомобільних майданчиках).

Пошук за текстовими шаблонами (Регулярні вирази)

Це базовий рівень автоматизації. Замість аналізу всієї структури сайту, програма шукає в його тексті конкретні збіги за допомогою регулярних виразів або інструментів типу утиліти grep в операційних системах UNIX.

- Наприклад, за допомогою шаблонів можна швидко витягнути з тексту сторінки всі номери телефонів, ціни або електронні адреси.
- Цей метод працює дуже швидко, але якщо дизайн сайту чи структура тексту мінімально зміняться, пошук за шаблоном може дати збій.

HTML-парсинг та побудова DOM-дерева

Це найбільш надійний та популярний підхід, який використовується в сучасних системах моніторингу. Оскільки більшість сторінок на одному сайті (наприклад, картки оголошень про продаж авто) створюються за єдиним шаблоном, програма може чітко орієнтуватися в їхній структурі.

Процес HTML-парсингу складається з таких етапів:

1. Програма завантажує вихідний код сторінки (HTML).
2. Браузер або спеціальна бібліотека (наприклад, BeautifulSoup) перетворює цей код на **DOM-дерево (Document Object Model)**. Це структурована ієрархія, де кожен елемент сайту (заголовок, таблиця, абзац, посилання) стає окремим об'єктом.
3. Розробник вказує програмі точний шлях у цьому дереві до потрібних даних (наприклад, «знайти таблицю з характеристиками авто та взяти звідти рік випуску»).

Завдяки побудові DOM-дерева ми можемо точно зчитувати інформацію, навіть якщо тексти оголошень написані користувачами у довільній формі.

1.4. Опис типового процесу веб-скрапінгу

1.4.1. Загальний опис

У будь-якому процесі автоматизованого збору даних з Інтернету завжди беруть участь дві основні сторони:

- **Ініціатор (агент збору)** — програмний модуль системи моніторингу (парсер), який надсилає запити та обробляє отриманий код.
- **Ціль (об'єкт моніторингу)** — конкретний веб-ресурс, з якого необхідно вилучити інформацію (текст оголошень, технічні параметри автомобілів, ціни тощо).

Сам процес взаємодії між ними можна розділити на три послідовні кроки.

Крок 1. Програма-ініціатор автоматично запускає процес моніторингу, формуючи та надсилаючи мережевий HTTP-запит до сервера цільового веб-ресурсу. На цьому етапі система вказує потрібну адресу сторінки (URL) та метод дії (найчастіше це метод GET для отримання контенту), як показано на рис. 1.



Рисунок 1 – Перший крок процесу вебскрапінгу

Крок 2. Якщо цільова сторінка існує і сервер працює без збоїв, він повертає відповідь із кодом успішного підключення (200 OK). Після цього наш модуль збору даних завантажує «сирий» неструктурований HTML-код сторінки в оперативну пам'ять комп'ютера для подальшої обробки (рис. 2).



Рисунок 2 – Другий крок процесу вебскрапінгу

Крок 3. На фінальному етапі завантажений код передається до аналітичного модуля системи моніторингу. Програма розбирає структуру сторінки, знаходить потрібні теги (наприклад, назву моделі авто, вартість чи рік випуску) та очищує їх від зайвого коду. Отримані структуровані дані зберігаються у машинозчитуваних форматах (JSON або CSV) або передаються далі по архітектурі системи (рис. 3).

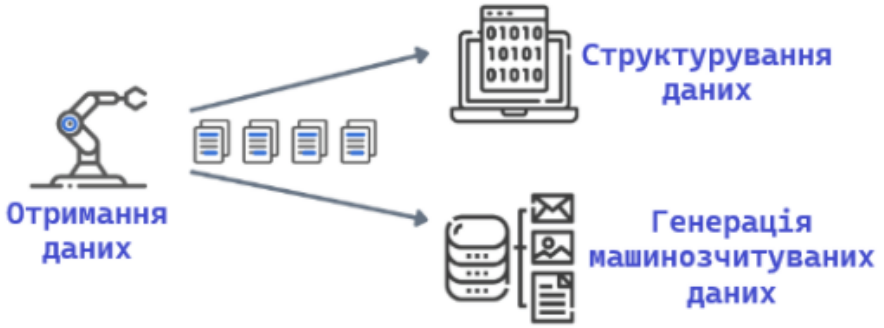


Рисунок 3 – Третій крок процесу вебскрапінгу

1.4.2. Механізми отримання даних з HTML-коду

Головним завданням на етапі безпосереднього парсингу є точне вилучення корисної інформації з коду веб-сторінки. Будь-який сайт будується за допомогою мови розмітки HTML, яка структурує контент за допомогою тегів, таблиць, списків, класів та ідентифікаторів. Разом з HTML завжди використовується CSS (каскадні таблиці стилів), що відповідає за візуальне оформлення цих елементів у браузері.

Оскільки сучасні веб-сторінки мають дуже заплутану структуру, намагатися витягнути з них дані за допомогою звичайного пошуку по тексту чи регулярних виразів — складно і ненадійно. Для швидкого й точного пошуку потрібних елементів розробники використовують спеціальні інструменти навігації: **CSS-селектори** та шляхи **XPath**.

CSS-селектори — це спеціальні шаблони-вказівники, які дозволяють системі моніторингу миттєво знаходити потрібний елемент на сторінці за його технічними атрибутами. Головна перевага CSS-селекторів перед іншими методами — це простота їхнього написання та висока швидкість обробки програмою.

Найчастіше під час скрапінгу пошук елементів (наприклад, ціни автомобіля чи його пробігу) відбувається за двома основними атрибутами:

- **Клас (class)** — позначається крапкою .
- **Ідентифікатор (id)** — позначається символом решітки #

Наприклад, якщо на сайті назва машини записана в коді як <h1 class="card" id="text">...</h1>, то програма-парсер зможе знайти цей заголовок за допомогою селекторів h1.card, h1#text або поєднавши їх: h1.card#text.

Якщо ж потрібні дані сховані глибоко всередині інших блоків чи контейнерів, селектори просто записуються послідовно через пробіл, вказуючи програмі шлях від батьківського елемента до дочірнього.

Розглянемо практичний приклад взаємодії з HTML-структурою сайту, наведений на рис. 4.

```
<ul class="list">
  <li class="item" id="coffee">Кава</li>
  <li class="item" id="tea">Чай</li>
  <li class="item" id="milk">Молоко</li>
</ul>
```

Рисунок 4 – Приклад HTML коду

Згідно з наведеною схемою:

1. Щоб вибрати всі елементи списку з тегом li, які знаходяться всередині контейнера списку, використовується комбінований селектор: ul.list li.item.
2. Якщо потрібно точково отримати конкретний елемент (наприклад, рядок із текстом "Чай"), програма орієнтується на унікальний ідентифікатор цього рядка: ul.list li#tea.

Таким чином, використання CSS-селекторів дозволяє системі моніторингу чітко й безпомилково знаходити потрібні текстові блоки в коді сторінки, ігноруючи решту непотрібного вмісту сайту.

Використання XPath-шляхів

XPath (XML Path Language) — це спеціальна мова запитів, яка спочатку створювалася для роботи з XML-документами, але зараз активно застосовується у веб-скрапінгу для навігації по HTML-коду сторінок.

На відміну від CSS-селекторів, XPath дозволяє шукати елементи сайту не тільки за класами чи ідентифікаторами, а й за багатьма іншими складними критеріями. Наприклад, за допомогою XPath можна знайти елемент за його точним текстовим вмістом (знайти кнопку, на якій написано «Показати контакти») або рухатися кодом «назад» — від дочірнього елемента до його батьківського контейнера.

За структурою побудови запитів шляхи XPath поділяють на два основні типи:

1. **Абсолютний XPath.** Будується від самого початку (кореня) HTML-документа і починається з однієї косої риски (/). Головний недолік цього підходу — висока чутливість до будь-яких змін на сайті. Якщо розробники ресурсу додадуть або видалять хоча б один блок чи рекламний банер на сторінці, весь абсолютний шлях ламається, і програма більше не зможе знайти потрібні дані (наприклад, ціну авто).
2. **Відносний XPath.** Починається з будь-якого місця в коді сторінки, що позначається подвійною косою рискою (//) на початку виразу. Програмі не потрібно знати весь ланцюжок тегів від самого верху сайту — вона просто шукає потрібний елемент з унікальним атрибутом по всьому документу.

Завдяки своїй гнучкості та стійкості до змін у дизайні сайтів, саме відносний XPath є пріоритетним вибором при розробці систем моніторингу та парсингу даних.

Приклад побудови та синтаксису відносного XPath-шляху наведено на рис. 5.

```
Xpath=//tagname[@attribute='value']
```

Рисунок 5 – Схема для побудови выдносного 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. Аналіз існуючих інструментів та платформ для автоматизованого збору даних

1.5.1. Платформа Bright Data

Bright Data — це хмарна інфраструктура корпоративного рівня, яка спеціалізується на забезпеченні доступу до веб-ресурсів та автоматизації збору даних. Основним технічним компонентом платформи є глобальна мережа проксі-серверів (включаючи резидентні, мобільні та серверні IP-адреси), що дозволяє реалізувати гнучкі стратегії ротації IP-адрес під час моніторингу великих масивів інформації.

Платформа надає розробникам готовий інструментарій Web Scraper API, який автоматично вирішує типові інженерні проблеми веб-скрапінгу. Серед ключових функціональних можливостей Bright Data варто виділити:

- **Обхід систем захисту:** автоматичне подолання антибот-систем, блокувань за географічною ознакою та динамічна ротація заголовків користувацького агента (User-Agent).
- **Автоматизація обробки контенту:** вбудований рендеринг сценаріїв JavaScript на стороні сервера, автоматичне розпізнавання та вирішення систем перевірки (Captcha).
- **Конвеєризація даних:** підтримка пакетної обробки запитів, первинна валідація отриманої інформації та доставка результатів через механізм вебхуків (Webhooks).

Переваги використання платформи:

1. Висока стабільність з'єднання та гарантований рівень безвідмовної роботи (Uptime на рівні 99.9%).
2. Можливість динамічного масштабування обчислювальних ресурсів під завдання будь-якого обсягу.
3. Доступ до легальних резидентних проксі-мереж, що мінімізує ризик блокування запитів під час збору даних.

Недоліки сервісу:

1. Повністю комерційна модель розповсюдження із високою вартістю послуг.
2. Складна тарифна сітка, де фінальна ціна залежить від обсягу трафіку та специфіки цільових доменів, що може суттєво збільшити бюджет проекту при постійному моніторингу.

1.5.2. Сервіс ScrapingBee

ScrapingBee є спеціалізованим API-рішенням, розробленим для автоматизації процесів веб-скрапінгу на рівні програмного коду. Головне технічне завдання сервісу — повне приховування складнощів керування headless-браузерами (екземплярами браузерів без графічного інтерфейсу) та проксі-мережами від кінцевого розробника. Платформа бере на себе рутинні завдання з налаштування інфраструктури, дозволяючи розробнику фокусуватися виключно на написанні логіки розбору HTML-коду.

Переваги використання ScrapingBee:

1. **Економічна оптимізація:** комерційна модель передбачає тарифікацію виключно успішно виконаних HTTP-запитів, що знижує витрати при частих збоях зв'язку.
2. **Підтримка динамічного контенту:** повноцінне виконання JavaScript-сценаріїв на цільових веб-сторінках перед вилученням коду.
3. **Автоматизований захист:** інтегровані механізми автоматичного вирішення графічних тестів (капчі) та обходу базових антибот-систем.
4. **Якісна документація:** наявність детальних інструкцій та практичних прикладів інтеграції з популярними мовами програмування, включаючи Python.

Недоліки сервісу:

1. Відносно низька швидкість відповіді API у порівнянні з прямими HTTP-запитами через додаткові накладні витрати на рендеринг сторінок.
2. Жорсткі ліміти пропускної здатності на кількість одночасно виконуваних (паралельних) потоків збору даних.
3. Орієнтація виключно на розробників, що вимагає наявності базових технічних навичок програмування для інтеграції рішення в систему.

1.5.3. Візуальна платформа Octoparse

Octoparse представляє інший клас систем збору даних — це десктопна та хмарна no-code платформа, орієнтована на користувачів, які не володіють навичками програмування. Процес налаштування збирачів даних (парсерів) тут відбувається через графічний інтерфейс за принципом "point-and-click" (вибір потрібних елементів сайту безпосередньо мишкою на екрані). Платформа самостійно трансформує візуальні дії користувача у програмні алгоритми збору неструктурованої інформації.

Переваги використання Octoparse:

1. Відсутність потреби в написанні коду для створення працюючого парсера.
2. Емуляція складних дій користувача в браузері: підтримка нескінченної прокрутки сторінок (infinite scroll), кліків за випадаючими списками та інтерактивної навігації.
3. Гнучка екосистема інтеграцій: можливість прямої передачі зібраних даних у сторонні сервіси (Google Таблиці, Dropbox, Zapier) або через власне хмарне сховище платформи.
4. Доступність базового безкоштовного тарифного плану для тестування функціоналу системи.

Недоліки сервісу:

1. Кросплатформні обмеження: відсутність офіційної підтримки та клієнтських додатків для операційних систем сімейства Linux.
2. Обмежені можливості кастомізації: при виникненні нестандартних завдань або складних блокувань на сайті користувач стає залежним від закритих внутрішніх алгоритмів платформи без можливості змінити код вручну.

РОЗДІЛ 2 ТЕХНОЛОГІЇ ТА ІНСТРУМЕНТИ ДЛЯ РОЗРОБКИ

2.1 Інструменти розробки вебскрапінг-частини додатку

2.1.1 Мова розробки Python

Головним інструментом для створення нашої системи моніторингу було обрано мову програмування Python. Вона чудово підходить для швидкої розробки програмного забезпечення та автоматизації роботи з веб-ресурсами.

Вибір Python обумовлений такими його перевагами:

- **Простий та зрозумілий синтаксис.** Код мови легко читати та змінювати. Це значно спрощує розробку системи та зменшує час на її подальшу підтримку.
- **Підтримка модульності.** Python дозволяє легко ділити програму на окремі пакети та модулі. Завдяки цьому ми можемо розділити систему на незалежні частини (наприклад, окремо модуль парсера та окремо логіку бота), що є важливим для побудови розподіленої архітектури.
- **Велика кількість готових бібліотек.** Для Python існує багато безкоштовних інструментів, які дозволяють швидко налаштувати збір, очищення та збереження даних із сайтів.
- **Кросплатформність.** Програма, написана на Python, може без проблем запускатися на різних операційних системах (Windows, Linux, macOS). Це дозволяє легко розгорнути систему на будь-якому віддаленому сервері.

Завдяки цим властивостям мова Python дозволила створити надійний програмний комплекс, який можна легко масштабувати у майбутньому.

2.1.2 Асинхронність, паралельність та модуль concurrent.futures

Щоб наш бот працював швидко і міг перевіряти багато сайтів одночасно, програма повинна вміти виконувати кілька завдань паралельно. Для цього в програмуванні є два головні підходи:

- **Асинхронність.** Вона дозволяє програмі не чекати, поки завантажиться один сайт, а одразу переходити до іншого. Тобто, поки ми чекаємо відповіді від сервера Auto.ria, програма вже може відправляти запит на інший сайт. Це економить багато часу.
- **Паралелізм.** Це коли комп'ютер виконует завдання одночасно, використовуючи різні потоки або процеси. Це дозволяє задіяти всю потужність процесора.

В мові Python для цього є зручний готовий модуль **concurrent.futures**. Він дозволяє легко запускати завдання паралельно без написання складного коду.

У цьому модулі є два основні інструменти:

1. **ThreadPoolExecutor** (пул потоків). Він запускає завдання у різних потоках всередині однієї програми. Це ідеально підходить для парсера, тому що головна затримка у нас йде через очікування відповіді від сайтів по мережі.
2. **ProcessPoolExecutor** (пул процесів). Він запускає завдання як окремі процеси в операційній системі. Його використовують для дуже важких обчислень, які сильно навантажують комп'ютер.

Обидва інструменти дуже прості у використанні завдяки оператору with (він сам закриває і чистить потоки після того, як вони закінчили роботу). Також через метод submit можна запустити функцію у фоновому режимі та перевіряти, чи вона вже виконалася і чи є результат.

2.1.3 BeautifulSoup

BeautifulSoup — це одна з найпопулярніших бібліотек у Python для розбору (парсингу) HTML-коду сторінок. Головне її завдання — перетворити завантажений із сайту текст у зручну структуру, в якій можна легко шукати потрібні дані.

Основні особливості бібліотеки BeautifulSoup:

- **Робота з помилками в коді.** Дуже часто сайти написані з помилками в HTML-розмітці (наприклад, розробники забули закрити якийсь тег). BeautifulSoup вміє автоматично виправляти такі помилки, завдяки чому програма не «падає» під час зчитування сторінки.
- **Підтримка різних парсерів.** Бібліотека може працювати разом із іншими інструментами, наприклад, lxml або html5lib. Це дозволяє вибирати між швидкістю роботи та точністю розбору сторінки.
- **Простота використання.** Знаходити інформацію на сторінці (наприклад, посилання чи таблиці з характеристиками автомобілів) можна за допомогою простих і зрозумілих команд.

Завдяки цьому BeautifulSoup чудово підходить для невеликих проектів, де потрібно швидко дістати дані з готового HTML-файлу.

2.1.4 Scrapy

Scrapy — це вже не просто бібліотека, а цілий безкоштовний фреймворк для масштабного збору даних, написаний на Python.

Головні відмінності Scrapy:

- **Архітектура на основі «павуків».** Логіка збору даних будується навколо окремих скриптів-роботів (павуків). Кожному павуку дається свій набір правил: на які сайти заходити, як переходити по сторінках і що саме шукати.
- **Принцип DRY (Don't Repeat Yourself).** Фреймворк спроектований так, щоб розробнику не доводилося писати один і той самий код кілька разів. Багато

функцій (наприклад, збереження у файл чи керування чергою запитів) уже готові «з коробки».

- **Висока швидкість.** Scrapy працює асинхронно, тому він ідеально підходить для великих проектів, де потрібно щодня сканувати тисячі веб-сторінок.

2.1.5 Selenium

Selenium — це спеціальний інструмент із відкритим кодом, який створювався для автоматичного тестування сайтів, але зараз часто використовується для веб-скрапінгу.

Головна фішка Selenium полягає в тому, що він керує реальним браузером. Програма через спеціальний драйвер (**WebDriver**) запускає, наприклад, Chrome або Firefox, і повністю імітує дії живої людини: сама клікає на кнопки, відкриває випадаючі списки, вводить текст у пошукові рядочки та прокручує сторінку вниз.

Чому Selenium важливий для систем моніторингу:

- **Робота з динамічними сайтами (JavaScript).** Багато сучасних автомобільних сайтів не віддають дані одразу в HTML-коді. Інформація на них з'являється лише після того, як на сторінці відпрацюють скрипти JavaScript. Звичайний парсер тут безсилий, а Selenium запускає браузер, чекає завантаження всіх елементів і лише тоді забирає текст.
- **Універсальність.** За допомогою Selenium можна обійти прості захисти від роботів, оскільки для сайту такий бот виглядає як звичайний користувач, який сидить у браузері.

Проте через запуск реального браузера Selenium споживає багато оперативної пам'яті та працює значно повільніше, ніж BeautifulSoup чи Scrapy.

2.2 Інструменти створення телеграм-боту

Для реалізації інтерфейсу користувача та системи сповіщень у нашому проєкті було обрано платформу Telegram-бот. Таке рішення має кілька важливих практичних переваг:

- **Зручність для користувача.** Користувачеві не потрібно встановлювати на свій телефон чи комп'ютер ніяких додаткових програм, окрім самого месенджера.

- **Знайомий інтерфейс.** Більшість людей уже мають акаунт у Telegram і знають, як працювати з ботами. Тому інтерфейс програми буде зрозумілим без додаткових інструкцій.
- **Простота реалізації.** Весь необхідний для нашої системи функціонал (налаштування фільтрів, пошук авто, отримання повідомлень про нові оголошення) можна зручно реалізувати в межах одного діалогового вікна.
- **Ефективність роботи.** Telegram-боти працюють асинхронно, завдяки чому система може одночасно обробляти запити від багатьох користувачів без затримок та зависань.

2.2.1 Загальна інформація про чат-бот

На базовому рівні чат-бот — це спеціальна програма, яка імітує текстове або усне спілкування з користувачем. За своєю складністю чат-боти бувають двох основних типів:

1. **Прості (лінійні).** Вони працюють за чітко прописаними правилами або кнопками і відповідають на конкретні команди короткими текстовими повідомленнями.
2. **Складні (інтелектуальні).** Це цифрові асистенти, які можуть використовувати елементи штучного інтелекту, збирати інформацію про користувача, навчатися в процесі роботи та підлаштовуватися під його потреби.

Бот у месенджері Telegram — це окрема програма, яка працює на базі платформи Telegram, взаємодіє з її серверами та автоматично виконує завдання, закладені розробником.

Користувач керує ботом за допомогою спеціальних текстових команд (наприклад, /start) або натисканням на інтерактивні кнопки в меню. Бот приймає ці запити, обробляє їх на нашому сервері та миттєво надсилає відповідь назад у чат у вигляді тексту, посилань чи картинок.

2.2.2 Бібліотека aiogram

Для програмної реалізації нашого чат-бота було обрано сучасну бібліотеку **aiogram**. Це повністю асинхронний фреймворк для мови Python, який створений спеціально для роботи з Telegram Bot API на базі вбудованих інструментів asyncio та aiohttp.

Головні переваги використання aiogram у нашому проєкті:

- **Повна асинхронність.** Фреймворк дозволяє боту не блокувати потік роботи під час очікування дій від користувача чи відповідей від бази даних. Це означає, що якщо ботом одночасно користуються десятки людей, кожен з них отримає відповідь миттєво, без черг та затримок.
- **Зручна обробка повідомлень (Хендлери).** Бібліотека має просту систему фільтрації та обробки вхідних даних. Можна легко налаштувати програму так, щоб вона окремо реагувала на текстові команди (наприклад, параметри пошуку авто), окремо — на натискання кнопок меню, а окремо — на надіслані файли чи картинки.
- **Керування клавіатурами.** aiogram надає прості інструменти для створення інтерактивних кнопок (як звичайних Reply-клавіатур під полем вводу, так і Inline-кнопок, які прикріплюються безпосередньо до повідомлень). Це дозволяє зробити інтерфейс бота максимально зрозумілим.

2.2.3 Інтерфейс Telegram Bot API

Telegram Bot API — це офіційний набір правил та інструментів, створений командою месенджера Telegram, який дозволяє стороннім програмам (нашому коду) спілкуватися з серверами месенджера.

Працює ця система за таким принципом:

1. Коли користувач натискає кнопку в боті або пише повідомлення, сервер Telegram фіксує цю дію.
2. Через Telegram Bot API це повідомлення у зашифрованому вигляді передається на наш сервер, де запущений код на Python.
3. Наша програма обробляє запит (наприклад, шукає в базі потрібну марку авто) і через цей же API відправляє відповідь назад користувачу.

Важливі технічні особливості Telegram Bot API:

- **Багатофункціональність.** Через цей інтерфейс бот може надсилати не лише звичайний текст, а й картинки з машинами, посилання на оголошення, файли або геомітки.
- **Висока швидкість і масштабованість.** Платформа розрахована на величезні навантаження, тому сервери Telegram швидко обробляють запити від великої кількості користувачів одночасно.
- **Безпека даних.** Весь трафік і обмін повідомленнями між нашим програмним кодом та серверами Telegram обов'язково шифруються. Це захищає особисті дані користувачів від перехоплення сторонніми особами.

РОЗДІЛ 3 ПРАКТИЧНА ЧАСТИНА

3.1 Загальний опис програмної реалізації

Результатом практичної розробки є інтерактивний Telegram-бот. Його головне завдання — збирати за допомогою веб-скрапінгу актуальні пропозиції щодо продажу вживаних автомобілів з тематичних сайтів і надавати користувачеві готові посилання на найкращі варіанти.

Процес взаємодії з системою виглядає так: користувач запускає роботу з ботом, після чого починається покроковий діалог. У цьому діалозі система просить користувача вказати основні критерії для пошуку транспортного засобу.

До базових параметрів автомобіля, які вводить користувач, належать (рис. 6):

- марка автомобіля;
- конкретна модель;
- бажаний діапазон вартості (мінімальна та максимальна ціна);
- діапазон років випуску;
- додаткові технічні характеристики;
- довільний текстовий опис (наприклад, побажання користувача щодо стану авто чи комплектації).

```
class Car:
    Ivanlyin1917
    def __init__(self,mark=None,model=None,price:[]=None,year:[]=None,characteristic=None,dedescription=None):
        self.mark=mark
        self.model=model
        self.price=price
        self.year=year
        self.characteristics=characteristic
        self.dedescription=dedescription
```

Рисунок 6 – Дані автомобіля

Для більш точного пошуку система дозволяє деталізувати технічні характеристики автомобіля. Сюди входять такі параметри (рис. 7):

- тип продавця (приватна особа чи автосалон);
- тип коробки передач (механіка, автомат тощо);
- вид палива (бензин, дизель, газ, електро);
- об'єм двигуна;
- тип приводу (передній, задній, повний);
- тип кузова (седан, хетчбек, кросовер тощо);
- колір автомобіля;
- максимальний або бажаний пробіг.

```
class Car_Characteristics:
    Ivanlyin1917 *
    def __init__(self,salesman=None,transmission=None,fuel_type=None,engine_capacity=None,drive_type=None,
                 body_type=None,color=None,mileage=None):

        self.salesman = Characteristic(["Продавець", "Оголошення від"],salesman)
        self.transmission = Characteristic(["Коробка передач", "Коробка","КПП"],transmission)
        self.fuel_type = Characteristic(["Тип палива", "Паливо"],fuel_type)
        self.engine_capacity = Characteristic(["Двигун"],engine_capacity)
        self.drive_type = Characteristic(["Привід"],drive_type)
        self.body_type = Characteristic(["Тип кузова","Кузов"],body_type)
        self.color = Characteristic(["Колір"],color)
        self.mileage = Characteristic(["Пробіг"],mileage)
```

Рисунок 7 – Характеристики автомобіля

Щоб програмі було зручно працювати з такою великою кількістю параметрів, у коді проекту було створено окремий допоміжний клас Characteristic (рис. 8). Полями цього класу є пари "ключ-значення", які описують кожну конкретну характеристику транспортного засобу. Це дозволяє уніфікувати дані, отримані з різних веб-сайтів.

```
class Characteristic:
    Ivanlyin1917
    def __init__(self, keys: list[str], value: list[str] = None):
        self.keys = keys
        self.value = value if value is not None else []
```

Рисунок 8 – Об’єкт характеристики

Щойно бот збирає всі необхідні критерії від користувача, автоматично запускається модульний процес веб-скрапінгу. Програма опитує цільові ресурси, збирає оголошення та формує підсумковий список. Перед відправкою користувачу цей список додатково сортується: спеціальний алгоритм порівнює текстовий опис, який ввів користувач, із текстом опису від продавця автомобіля, виставляючи найвищий пріоритет найбільш схожим варіантам.

У результаті користувач отримує у чаті структурований список посилань на оголошення. Після цього він може або завершити роботу, або ввести нові параметри та запустити пошук повторно.

3.2 Програмна реалізація вебскрапінгу

Оскільки кожен веб-сайт має свою власну унікальну архітектуру, дизайн та логіку взаємодії з користувачем, створити один універсальний алгоритм для збору даних неможливо. Наш Telegram-бот розрахований на роботу з декількома різними майданчиками одночасно, тому для кожного цільового ресурсу необхідно розробляти окремий модуль (парсер), який враховує специфіку конкретного сайту.

Усі ці модулі інтегруються в загальну систему моніторингу через єдині інтерфейси. Нижче детально описано внутрішню структуру та основні складові такого модуля на прикладі парсингу автомобільного порталу AutoMoto (https://automoto.ua/uk/).

3.2.1 Власний фільтр пропозицій вебсайту та Selenium

Першим етапом збору даних у нашому модулі є використання інструменту Selenium. Його головна задача на цьому кроці — відсіяти оголошення, які точно не підходять під запит користувача, безпосередньо на самому сайті.

Наприклад, на порталі AutoMoto опубліковано понад 400 тисяч оголошень. Якщо наша програма буде завантажувати та перевіряти кожне з них окремо, це займе занадто багато часу і завантажить сервер. Використання Selenium дозволяє задіяти внутрішні фільтри самого сайту, звузивши список пошуку до кількох десятків або сотень найбільш релевантних пропозицій.

Процес налаштування фільтрів через Selenium реалізовано у вигляді окремої функції.

```
def selenium_parse_automoto(car:car_obj.Car):

    options=webdriver.ChromeOptions()
    options.add_argument('--headless')
    options.add_argument("window-size=1920x1080")
    options.add_argument('--disable-gpu')
    driver = webdriver.Chrome(options=options)
    link='https://automoto.ua/uk/'
    driver.get(link)
    time.sleep(1)
```

Рисунок 9 – Початок роботи методу з Selenium

На початку роботи цієї функції (рис. 9) створюється об'єкт конфігурації браузера Chrome, де прописуються такі обов'які параметри (аргументи):

- **--headless** — запускає браузер у фоновому («безголовому») режимі без відкриття графічного вікна на екрані. Це значно економить оперативну пам'ять та ресурси процесора на сервері, оскільки комп'ютеру не потрібно візуально малювати сторінку.
- **--disable-gpu** — вимикає використання апаратного прискорення відеокарти. Цей параметр необхідний для стабільної роботи скрипта на віддалених серверах, де повноцінної відеокарти може взагалі не бути.
- **--window-size=1920x1080** — чітко задає віртуальний розмір екрана (роздільну здатність). Оскільки реального монітора у фоновому режимі немає, цей параметр гарантує, що всі елементи та кнопки на сайті завантажаться на своїх місцях і не «налізуть» один на одного.

Після налаштування параметрів створюється сам екземпляр WebDriver Chrome. Програма бере початкове посилання сайту (link) і за допомогою методу get відкриває головну сторінку.

Одразу після цього код зупиняє свою роботу на одну секунду (пауза time.sleep(1)). Такі затримки використовуються в методі постійно, адже програма працює швидше за інтернет, і якщо не зробити паузу, скрипт спробує натиснути на кнопку ще до того, як вона встигне завантажитися в браузері.

Коли стартова сторінка успішно відкрилася, бот починає заповнювати параметри автомобіля. Розглянемо цей процес на прикладі вибору марки машини (рис. 10).

```
#mark
try:
    driver.find_element(By.CSS_SELECTOR,'.choices .choices__inner').click()
    time.sleep(1)
    driver.find_element(By.CSS_SELECTOR,'.choices__input.choices__input--cloned[aria-label="Марка"]').send_keys(car.mark)
    time.sleep(1)
    driver.find_element(By.CSS_SELECTOR,'.choices__list .choices__item:first-child.is-highlighted').click()
    time.sleep(1)
except:
    None
```

Рисунок 10 – Встановлення значення марки

Щоб взаємодіяти з сайтом, програма спочатку знаходить потрібне поле за допомогою методу .find_element(), використовуючи для цього CSS-селектори. Після того як елемент знайдено, бот керує ним за допомогою двох основних команд:

- **.click()** — імітує звичайне натискання мишкою;
- **.send_keys()** — імітує введення тексту з клавіатури в текстове поле.

Алгоритм вибору марки автомобіля виглядає так: програма знаходить поле вибору марки та клікає на нього, вводить туди назву (наприклад, "Toyota"), чекає появи випадаючого списку і вибирає перший підходящий варіант.

За таким самим принципом заповнюються й інші параметри (модель, ціна, рік). Проте тут є важливий технічний нюанс: якщо потрібна кнопка чи поле сховані внизу сторінки (за межами видимості вікна), Selenium не зможе з ними взаємодіяти і видасть помилку. Тому в коді реалізовано автоматичне прокручування сторінки вниз до потрібного елемента (рис. 11).

```
more=driver.find_element(By.CSS_SELECTOR,'#collapseBodyType+ .collapsed')
driver.execute_script("arguments[0].scrollIntoView(true);", more)
time.sleep(1)
```

Рисунок 11 – Скролінг за допомогою Selenium

Після того як усі фільтри на сайті успішно виставлені, метод завершує роботу WebDriver. На виході функція повертає згенероване посилання на сторінку з уже відфільтрованими результатами пошуку (рис. 12). Саме це готове посилання передається далі наступному модулю програми для швидкого збору даних.

```
link=driver.current_url
driver.close()


return link
```

Рисунок 12 – Завершення роботи методу

3.2.2 Отримання даних авто з вебсайту та BeautifulSoup

Наступним обов'язковим кроком у роботі нашого парсера є безпосереднє зчитування інформації про автомобіль зі сторінки оголошення. Цей процес реалізовано за допомогою функцій, що використовують бібліотеку BeautifulSoup.

Розглянемо алгоритм збору даних на прикладі роботи з конкретною пропозицією на сайті AutoMoto.

```
def automoto_parse_car_page(link)->car_obj.car_obj:

    url = link
    response = requests.get(url)

    res = BeautifulSoup(response.content, features="html.parser")
```

Рисунок 13 – Початок роботи методу з BeautifulSoup

На початку роботи методу виконуються такі базові кроки (рис. 13):

1. **Налаштування адреси:** у змінну url передається пряме посилання на сторінку конкретного автомобіля.
2. **Запит до сайту:** за допомогою команди requests.get(url) програма відправляє HTTP-запит і завантажує весь вихідний HTML-код сторінки у пам'ять комп'ютера.
3. **Створення об'єкта пошуку:** команда BeautifulSoup(response.content, features="html.parser") обробляє завантажений код і створює об'єкт res (soup). Цей об'єкт дозволяє нам зручно переміщатися по структурі сайту та швидко знаходити потрібні теги.

Після того як сторінка успішно розпарсена, починається етап екстракції (вилучення) корисних даних. Нам потрібно дістати назву машини (яка на цьому сайті об'єднує в собі марку, модель і рік випуску), її технічні характеристики та текстовий опис від продавця.

Покажемо, як саме працює цей процес, на прикладі отримання назви автомобіля (рис. 14).

```
## Title
try:
    title = res.find('h1', class_='main-card-name').text.strip().replace("\xa0", " ")
except:
    title = None
```

Рисунок 14 – Отримання назви пропозиції

Для очищення та вилучення тексту з HTML-коду використовуються такі послідовні команди:

- **Пошук елемента за тегом і класом.** Функція res.find('h1', class_='main-card-name') шукає на сторінці головний заголовок першого рівня <h1>, який має унікальне ім'я класу main-card-name. Метод повертає саме той блок коду, де схована назва авто.
- **Очищення від тегів.** За допомогою атрибута .text програма відкидає всі HTML-теги і забирає з блоку лише чистий текст.
- **Видалення зайвих пробілів.** Метод .strip() автоматично прибирає випадкові пробіли або знаки переносу рядка на початку та в кінці отриманого тексту.

- **Заміна технічних символів.** Команда .replace("\xa0", " ") знаходить у тексті специфічні «нерозривні пробіли» (які часто з'являються на сайтах при верстці) і замінює їх на звичайні пробіли. Якщо цього не зробити, програма може некоректно порівнювати рядки під час подальшої фільтрації.

За таким самим простим шаблоном програма збирає й усі інші дані зі сторінки. Наприкінці роботи методу всі знайдені текстові блоки упаковуються в один спільний об'єкт класу, який повертається для подальшого аналізу та порівняння із запитом користувача в Telegram-боті.

3.2.3 Вебкраулінг за допомогою Scrapy

Наступним логічним етапом роботи системи є масовий збір посилань на оголошення (краулінг). Як початкову точку для цього процесу програма використовує саме те готове посилання з виставленими фільтрами, яке ми отримали від Selenium на першому кроці (описаному в пункті 3.2.1).

Цей процес реалізовано на базі фреймворку Scrapy. У коді проекту створено спеціальний клас-павук, який наслідує властивості базового класу Spider із бібліотеки Scrapy. Цей робот повністю автоматизує обхід сторінок у межах заданого сайту та збирає потрібні лінки.

Для правильної роботи нашого павука в коді обов'язково налаштовуються три головні атрибути:

- **name** — це унікальне ім'я нашого краулера в проекті. Воно потрібне системі Scrapy для того, щоб точно знати, якого саме павука ми хочемо запустити через термінал чи фоновий процес.
- **allowed_domains** — список дозволених доменів (наприклад, automoto.ua). Це важливе обмеження, яке не дозволяє роботу виходити за межі потрібного сайту. Якщо на сторінці буде посилання на сторонню рекламу чи соцмережі, павук просто ігноруватиме їх і не витрачатиме ресурси.
- **start_urls** — початковий список адрес, з яких робот починає свою роботу. У нашому випадку сюди динамічно підставляється посилання, згенероване на етапі

роботи з Selenium, щоб павук одразу потрапив на сторінку з уже відфільтрованими машинами.

Логіка роботи методів Scrapy

Процес збору даних у фреймворку Scrapy побудований на взаємодії двох головних методів: start_requests та parse.

Метод start_requests є початковою точкою роботи павука. Він перевизначає стандартні правила базового класу Scrapy та ініціює перші HTTP-запити до сайтів. Метод бере посилання, які ми записали в атрибут start_urls, і для кожного з них створює спеціальний об'єкт запиту scrapy.Request.

Під час створення запиту ми чітко прописуємо три важливі параметри:

- **callback=self.parse** — вказує програмі, що як тільки сторінка завантажиться, її код потрібно одразу передати в метод parse для подальшої обробки.
- **dont_filter=True** — вимикає стандартний фільтр дублікатів Scrapy. Це дозволяє системі повторно заходити на ті самі адреси, якщо користувач вирішить оновити пошук.
- **meta** — спеціальний технічний словник із налаштуваннями. Тут ми вимикаємо повторні спроби завантаження при збоях ("dont_retry": True) та обмежуємо час очікування відповіді від сайту ("download_timeout"), щоб програма не «зависала» на зламаних посиланнях.

Метод parse — це основний функціональний блок, який Scrapy викликає автоматично, щойно отримує HTML-код сторінки від сервера.

Робота методу parse ділиться на три кроки:

1. **Збір посилань на оголошення.** За допомогою вбудованих селекторів метод сканує сторінку пошуку та витягує з неї прямі лінки на конкретні автомобілі.
2. **Перехід по сторінках (пагінація).** Робот перевіряє, чи є на сайті кнопка «Наступна сторінка». Якщо така кнопка є, метод parse викликає сам себе повторно

вже для нової адреси. Так павук автоматично гортає весь список оголошень до самого кінця.

3. **Обробка та фільтрація даних.** Для кожного знайденого посилання на конкретне авто викликається метод парсингу за допомогою бібліотеки BeautifulSoup (який ми описали в пункті 3.2.2). Програма завантажує сторінку машини, забирає дані та порівнює їх із тим, що просив користувач у Telegram. Якщо автомобіль повністю підходить під критерії користувача, цей об'єкт додається до нашого підсумкового списку результатів.

3.2.4 Опис алгоритму процесу вебскрапінгу

Підсумовуючи роботу всієї системи збору даних, повну програмну реалізацію процесу веб-скрапінгу можна представити у вигляді такого чіткого покрокового алгоритму:

1. **Попередня фільтрація за допомогою Selenium.** Програма запускає фоновий браузер і автоматично виставляє потрібні параметри (марку, модель, ціну) безпосередньо у фільтрах самого сайту. Результатом цього кроку є одне готове посилання на сторінку, де відображаються лише ті машини, які потенційно підходять нашому користувачу.
2. **Масовий збір лінків через Scrapy.** Фреймворк підхоплює згенероване на першому кроці посилання і починає швидко збирати адреси конкретних оголошень, автоматично переходячи з однієї сторінки пошуку на іншу.
3. **Точковий збір та перевірка через BeautifulSoup.** Для кожного знайденого оголошення програма завантажує сторінку та витягує з неї чистий текст (характеристики та опис). Оскільки через можливі збої зв'язку або нестандартну верстку сайту первинний фільтр міг спрацювати неідеально, на цьому етапі код проводить повторну перевірку: він ще раз звіряє всі параметри машини з тими, що просив користувач у Telegram. Якщо перевірка успішна — авто додається до загального списку.
4. **Інтелектуальне сортування результатів.** Отриманий список автомобілів пропускається через алгоритм порівняння текстів. Система оцінює, наскільки сильно довільний опис від користувача схожий на опис, який продавець залишив на сайті, і сортує список так, щоб найкращі варіанти опинилися в самому верху.

Фінальним результатом роботи всього парсера є повністю очищений, перевірений та відсортований за релевантністю список автомобільних пропозицій, готовий для відправки в чат користувачу.

3.3 Програмна реалізація телеграм-бота

Практична реалізація інтерфейсу користувача у вигляді Telegram-бота дозволяє створити просту та доступну взаємодію із системою моніторингу автомобілів.

Увесь робочий цикл нашого чат-бота можна звести до трьох основних кроків:

1. Покрокове опитування користувача для збору критеріїв пошуку транспортного засобу.
2. Автоматичний запуск модулів веб-скрапінгу та очікування готових результатів.
3. Виведення сформованого списку автомобільних оголошень у чат або надсилання повідомлення про те, що за вказаними параметрами нічого не знайдено.

Першим обов'язковим етапом розробки є реєстрація самого бота всередині екосистеми Telegram. Цей процес відбувається через офіційний інструмент від розробників месенджера — батьківського бота **BotFather**.

Процедура реєстрації складається з таких послідовних дій:

- **Ініціалізація створення бота.** У чат із BotFather надсилається команда /newbot. У відповідь система просить ввести дві назви: спочатку відображуване ім'я бота (яке бачитимуть усі користувачі), а потім його унікальний системний юзернейм (username), який обов'язково повинен закінчуватися на _bot (наприклад, auto_monitor_bot).
- **Отримання токена доступу.** Після успішного вибору імені BotFather генерує унікальний API-токен (довгий рядок із символів та цифр).
- **Конфігурація безпеки.** Отриманий токен переноситься у файл конфігурації нашого проекту. Він є секретним ключем, який необхідний для автентифікації нашої програми на серверах Telegram та авторизації всіх HTTP-запитів.

Нижче детально розглянуто безпосередні кроки програмної реалізації логіки бота на мові Python за допомогою фреймворку aiogram.

3.3.1 Створення телеграм-боту

Процес запуску та первинної конфігурації (ініціалізації) нашого Telegram-бота на базі фреймворку aiogram складається з кількох обов'язкових кроків (рис. 15).

```
bot = Bot(token="6522837095:AAGpGmy2TBwrXadFC-lApygFkgUcS3SFOV0")
dp = Dispatcher()

dp.include_router(questions.router)

# And the run events dispatching
await bot.delete_webhook(drop_pending_updates=True)
await dp.start_polling(bot)
```

Рисунок 15 – Ініціалізація бота

Програмне налаштування головного файлу бота реалізовано за таким алгоритмом:

- **Створення об'єкта бота (Bot).** За допомогою команди Bot(token="...") ми створюємо безпосередній екземпляр бота. Як параметр сюди передається секретний токен, який ми раніше згенерували в BotFather. Цей рядок коду пов'язує нашу програму з конкретним аккаунтом у Telegram.
- **Налаштування диспетчера (Dispatcher).** Команда Dispatcher() створює головний менеджер проекту. Диспетчер відповідає за отримання всіх подій від Telegram (повідомлень, кліків по кнопках) і розподіляє їх між різними частинами нашого коду.
- **Підключення модульних роутерів (Router).** За допомогою конструкції dp.include_router(questions.router) ми інтегруємо в диспетчер окремий роутер із файлу questions. Це дозволяє розділити логіку бота на кілька файлів: наприклад, тримати код опитування користувача окремо від головного файлу запуску, що робить структуру проекту чистою та зручною для розробки.

- **Очищення черги повідомлень (Видалення вебхука).** Команда await bot.delete_webhook(drop_pending_updates=True) є дуже важливою для стабільного перезапуску системи. Вона видаляє старі вебхуки та «скидає» (видаляє) всі повідомлення, які користувачі встигли надіслати боту, поки той був вимкнений. Це гарантує, що при старті бот не почне спамити старими відповідями на колишні запити.
- **Запуск режиму опитування (Polling).** Рядок await dp.start_polling(bot) запускає асинхронний безкінечний цикл опитування серверів Telegram. Програма переходить в активний режим «слухання» і миттєво реагує на кожну нову дію користувача в чаті.

3.3.2 Обробка повідомлень

Повноцінна робота чат-бота з користувачем починається після введення стандартної команди /start. У відповідь бот вітається та пропонує перейти до налаштування пошуку за допомогою команди /set. Саме ця команда запускає покроковий процес введення критеріїв автомобіля. Бот послідовно, пункт за пунктом, запитує у користувача потрібні дані, і як тільки користувач надсилає відповідь на поточне питання, система автоматично переходить до наступного.

Приклад програмної реалізації такої взаємодії наведено нижче (рис. 16).

```
@router.message(Command("set"))
async def start_get_info(message: Message, state: FSMContext):
    await message.answer(
        text="Напишіть марку та модель машини через пробіл:",
        reply_markup=ReplyKeyboardRemove()
    )
    await state.set_state(UserState.input_title)


 Ivanlyin1917
@router.message(UserState.input_title)
async def title_choosed(message: Message, state: FSMContext):
    await state.update_data(title=message.text)
    await message.answer(
        text="Напишіть ціновий діапазон у доларах через дефіс:",
        reply_markup=ReplyKeyboardRemove()
    )
    await state.set_state(UserState.input_price)
```

Рисунок 16 – Взаємодія з користувачем

Команда /set запускає алгоритм опитування. На першому кроці бот просить користувача написати марку та модель автомобіля. Одночасно з цим програма фіксує і зберігає поточний стан розмови (FSM-стани). Це необхідно для того, щоб система чітко розуміла, що наступне текстове повідомлення від користувача — це саме назва машини, а не якась інша команда.

Наступний метод у коді спрацьовує автоматично, щойно користувач надсилає текст із маркою авто. Бот переносить цю інформацію в тимчасову пам'ять і одразу видає наступне питання — наприклад, просить вказати бажаний діапазон цін.

Під час надсилання деяких текстових запитань у коді використовується параметр reply_markup=ReplyKeyboardRemove(). Він потрібен для того, щоб примусово прибрати з екрана телефону чи комп'ютера текстові кнопки, які залишилися від попередніх дій. Це робить інтерфейс чату чистим, щоб користувач міг спокійно ввести потрібний текст вручну без зайвих елементів.

Для максимальної зручності, коли користувачеві треба вибрати типові параметри (наприклад, тип палива чи коробку передач), програма пропонує не писати слово руками, а просто натиснути на готову кнопку. Для цього у функції відправки повідомлення налаштовується спеціальний параметр макету клавіатури reply_markup (рис. 17).

```
await message.answer(
    text="Оголошення від:",
    reply_markup=make_row_keyboard(available_salesmans)
)
```

Рисунок 17 – Встановлення кнопок для відповіді

Функція make_row_keyboard — це спеціальний метод у коді нашого проекту, який автоматично створює та виводить у чат список інтерактивних кнопок на основі переданого в нього масиву з текстовими назвами (рис. 18).

```
def make_row_keyboard(items: list[str]) -> ReplyKeyboardMarkup:
    keyboard = [[KeyboardButton(text=item)] for item in items]
    return ReplyKeyboardMarkup(keyboard=keyboard, resize_keyboard=True)
```

Рисунок 18 – Створення списку кнопок

Цей масив зазвичай містить список найпопулярніших варіантів відповіді (наприклад, поширені типи кузовів чи палива), а також обов'язкові додаткові кнопки. Вони дозволяють користувачеві або написати свій власний варіант вручну, або взагалі пропустити поточне запитання, якщо цей параметр для нього не є важливим.

Якщо користувач вирішує пропустити крок, ця технічна характеристика автомобіля залишається в програмі порожньою. Це означає, що на етапі веб-скрапінгу система просто ігноруватиме цей параметр і не буде проводити за ним фільтрацію оголошень на сайтах.

Приклад програмної реалізації такого меню наведено нижче (рис. 19).

```
other = "Інше"
skip="Пропустити"
available_salesmans = ["Компанія", "Автодилер","Власник", other,skip]
```

Рисунок 19 – Варіанти відповіді

Після того як користувач успішно проходить усі етапи опитування та вводить усі необхідні характеристики, бот збирає їх докупи і виводить на екран у вигляді підсумкового повідомлення. Разом із цим повідомленням система пропонує користувачеві дві фінальні кнопки: або запустити процес пошуку за вказаними критеріями, або скинути все і почати введення даних заново (рис. 20).

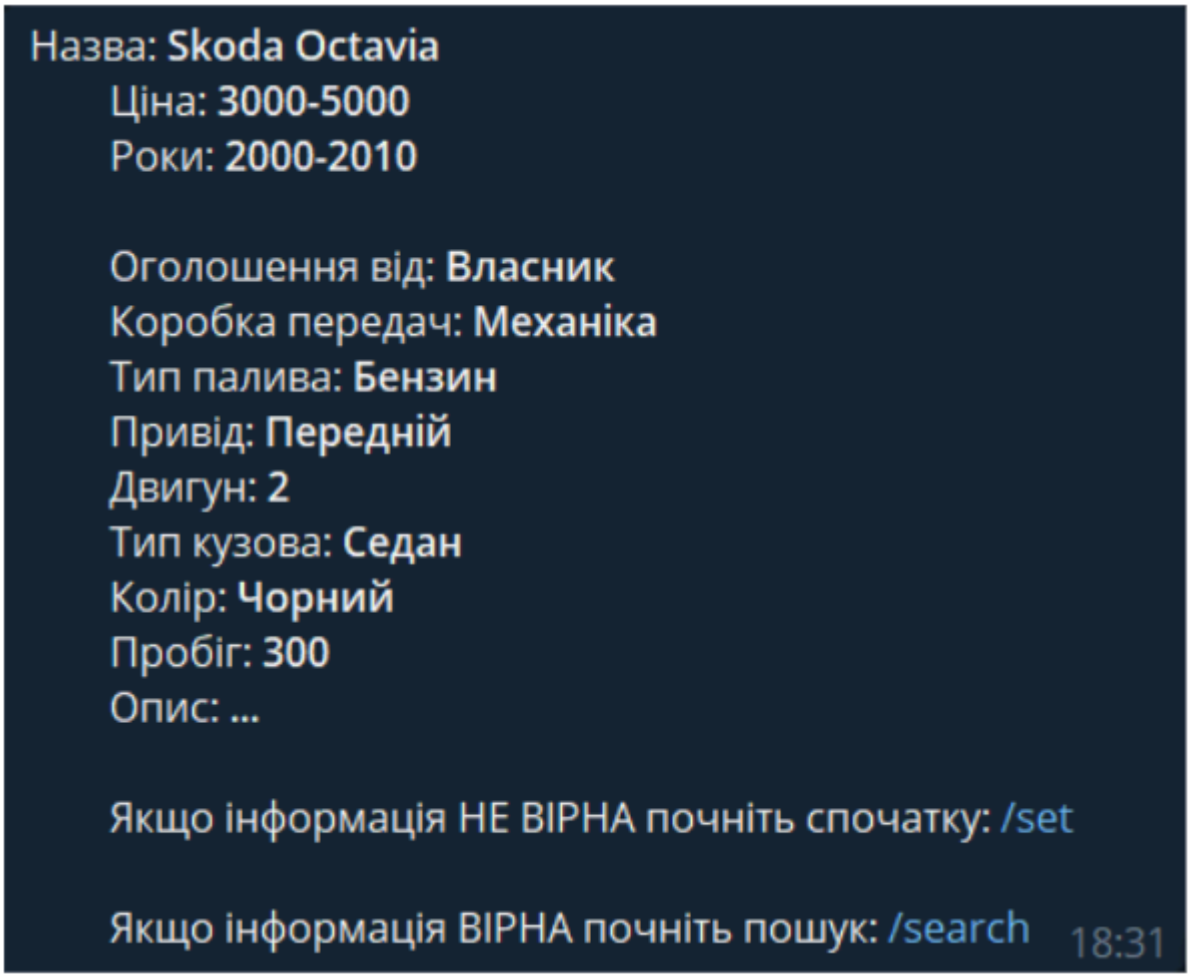


Рисунок 20 – Повідомлення користувацької відповіді

Коли користувач натискає кнопку запуску пошуку, програма активує модулі веб-скрапінгу, збирає знайдені оголошення та повертає їх у чат. Якщо система знайшла багато підходящих автомобілів, вмикається спеціальний алгоритм порційного виведення даних.

Якщо загальна кількість знайдених варіантів перевищує певне число n, бот виводить користувачеві перші n пропозицій і одразу видаляє їх із поточної черги відображення. Щоб подивитися наступну порцію машин, користувач має надіслати команду /more — тоді бот покаже наступні n варіантів. Цей цикл повторюється доти, доки список оголошень не вичерпається або поки користувач не вирішить зупинити процес і скинути фільтри для нового пошуку.

Такий підхід дозволяє уникнути технічних обмежень месенджера Telegram на максимальний розмір одного текстового повідомлення та захищає інтерфейс від переповнення даними.

3.3.3 Процес вебскрапінгу у телеграм-боті

Оскільки Telegram-бот розрахований на роботу з великою кількістю користувачів одночасно, система не повинна змушувати людей чекати своєї черги в загальному потоці. Процес веб-скрапінгу є ресурсомістким завданням (CPU-bound та I/O-bound), тому для кожного користувача він має виконуватися повністю паралельно та ізольовано.

Для реалізації паралельних обчислень у проекті було використано стандартну бібліотеку Python concurrent.futures у поєднанні з асинхронним фреймворком asyncio для безшовної інтеграції в логіку чат-бота (рис. 21).

```
executor = ProcessPoolExecutor()
future = executor.submit(scrapy_parse_car.start_parse_car_site, car)
list_car = await asyncio.get_event_loop().run_in_executor(None, future.result)
```

Рисунок 21 – Запуск краулера

Програмна реалізація цього процесу базується на таких кроках:

- **Створення пулу процесів (ProcessPoolExecutor).** За допомогою конструкції executor = ProcessPoolExecutor() ініціалізується пул окремих процесів операційної системи. На відміну від потоків (threads), які в Python обмежені механізмом GIL (Global Interpreter Lock), запуск важкого парсингу в окремому процесі дозволяє повністю задіяти багатоядерність процесора і не «зарожувати» роботу самого Telegram-бота.
- **Асинхронний запуск завдання.** Команда asyncio.get_event_loop().run_in_executor(executor, parsing_function, *args) делегує виконання функції парсингу створеному пулу процесів. Завдяки цьому асинхронний цикл подій (event loop) бота продовжує працювати у штатному режимі, моментально відповідаючи іншим користувачам, поки у фоні триває збір даних.

Ще однією серйозною проблемою під час веб-скрапінгу є час очікування. Якщо користувач вказав замало фільтрів, сайт може видати сотні сторінок з оголошеннями. Завантаження та перевірка такої кількості даних за один раз тривали б занадто довго. Щоб користувач не чекав відповіді хвилинами, було прийнято архітектурне рішення — оптимізувати процес веб-краулінгу за допомогою порційного (посторінкового) збору даних.

Логіка посторінкової обробки працює за таким алгоритмом:

1. **Перший запуск системи.** Краулер отримує від Selenium базове посилання з фільтрами, але проходить не весь сайт, а лише фіксовану кількість сторінок n. Програма збирає оголошення, сортує їх за релевантністю, виводить перші результати користувачу і запам'ятовує в системі посилання на наступну сторінку під номером n+1
2. **Подальший пошук.** Якщо користувач хоче подивитися більше варіантів, він викликає спеціальну команду /continue. Бот підхоплює збережене раніше посилання і запускає Scrapy вже безпосередньо з потрібної сторінки n+1, знову обробляючи наступну порцію з n сторінок. Якщо ж на сайті залишилося менше ніж n сторінок, краулер просто забирає всі залишки за один захід.

Емпіричним шляхом (під час тестування системи) було визначено, що оптимальним рішенням є проходження **від 3 до 5 сторінок** за один цикл скрапінгу. Цього цілком достатньо, щоб видати користувачу свіжу порцію актуальних машин, а сам процес збору триває лічені секунди.

Варто також підкреслити, що всі наступні запуски краулера через команду /continue відбуваються в рази швидше, ніж перший запуск. Це пов'язано з тим, що програмі більше не потрібно запускати важкий фоновий браузер Selenium для налаштування фільтрів — павук відразу йде за готовими прямими посиланнями пагінації сайту.

ВИСНОВКИ

У кваліфікаційній роботі успішно вирішено актуальну науково-практичну задачу — розроблено Telegram-бот для автоматизованого моніторингу та пошуку пропозицій вживаних автомобілів на тематичних веб-ресурсах із використанням технологій вебпарсингу.

Основні результати та висновки роботи полягають у наступному:

1. **Обґрунтування архітектурного рішення.** Вибір Telegram-бота як користувацького інтерфейсу повністю виправдав себе на практиці. Це дозволило створити доступну та інтуїтивно зрозумілу систему, яка не потребує встановлення додаткового програмного забезпечення на пристрої користувачів і забезпечує високу швидкість обміну даними.
2. **Аналіз та вибір технологічного стеку.** В ході дослідження методів веб-скрапінгу було проаналізовано та комбіновано три ключові інструменти, кожен з яких ефективно вирішує своє коло завдань у межах єдиного парсера:
    - **Selenium** (із інструментом WebDriver) успішно застосовано для первинної взаємодії з динамічними елементами сайтів та ініціалізації внутрішніх фільтрів у фоновому (headless) режимі.
    - Фреймворк **Scrapy** забезпечив високу швидкість, паралельність та асинхронність під час масового збору посилань на оголошення.
    - Бібліотека **BeautifulSoup** дозволила виконати точкову екстракцію, очищення текстових даних від HTML-тегів та виправлення помилок розмітки.
3. **Оптимізація та масштабованість системи.** Для захисту від блокування потоків (GIL) та забезпечення паралельної роботи багатьох користувачів було успішно впроваджено ізольовані обчислення за допомогою пулу процесів (ProcessPoolExecutor). Окрім цього, розроблено алгоритм посторінкового (порційного) краулінгу, що дозволило суттєво скоротити час очікування відповіді користувачем та уникнути технічних обмежень Telegram щодо максимального обсягу повідомлень.

4. **Практична цінність та перспективи розвитку.** Створений програмний продукт дозволяє значно автоматизувати та прискорити процес пошуку транспортних засобів, зводячи результати з різних платформ у єдине вікно. При цьому архітектура системи розроблена із дотриманням принципу DRY, що дозволяє у майбутньому легко масштабувати проект шляхом підключення нових автомобільних сайтів без зміни загальної логіки роботи бота.

Проте варто зауважити, що фінальна достовірність та якість отриманих даних напряму залежать від інформації, яку автори оголошень первинно вказують на сайтах.

Загалом, поставлена мета бакалаврської роботи повністю досягнута, а всі задекларовані технічні та функціональні завдання виконані в повному обсязі. Розроблений Telegram-бот готовий до практичного використання та може стати корисним інструментом для потенційних покупців на вторинному автомобільному ринку.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Python 3 documentation : official website. URL: https://docs.python.org/3/ .
2. Selenium Browser Automation Project Documentation : official website. URL: https://www.selenium.dev/documentation/ .
3. Scrapy documentation : official website. URL: https://docs.scrapy.org/en/latest/ .
4. Beautiful Soup Documentation : official website. URL: https://www.crummy.com/software/BeautifulSoup/bs4/doc/ .
5. aiogram documentation : official website. URL: https://docs.aiogram.dev/en/latest/.
6. Mitchell R. Web Scraping with Python: Collecting More Data from the Modern Web. 2nd ed. Sebastopol : O'Reilly Media, 2018. 306 p.
7. Kurose J. F., Ross K. W. Computer Networking: A Top-Down Approach. 7th ed. Amherst : Pearson, 2017. 856 p.

Додаток 1 “Демонстраційні матеріали”

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Розробка розподіленої системи моніторингу веб-ресурсів

Виконав: студент 4 курсу
групи КІД-42
Овчинніков Денис

Науковий керівник:
Віталій Волохін

Київ-2026

Мета роботи та актуальність

Мета роботи: створення телеграм-бота для пошуку вживаних авто, використовуючи процес вебскрапінгу.

Актуальність: необхідність в автоматизації пошуку інформації серед вебданих, у контексті швидкого розвитку ринку вживаних автомобілів.



Завдання

- Дослідити процес вебскрапінгу.
- Проаналізувати та обрати інструменти програмної реалізації процесу вебскрапінгу.
- Обрати інструменти для створення телеграм-бота.
- Реалізувати парсер для отримання даних з тематичних веб-ресурсів.
- Створити телеграм-бот та вбудувати в нього парсер.

Використані технології

Мова програмування



Вебскрапінг





Scrapy

Beautiful Soup

Телеграм-бот



Алгоритм процесу вебскрапінгу

1. Фільтрація пропозицій за допомогою фільтра вебсайту (Selenium)

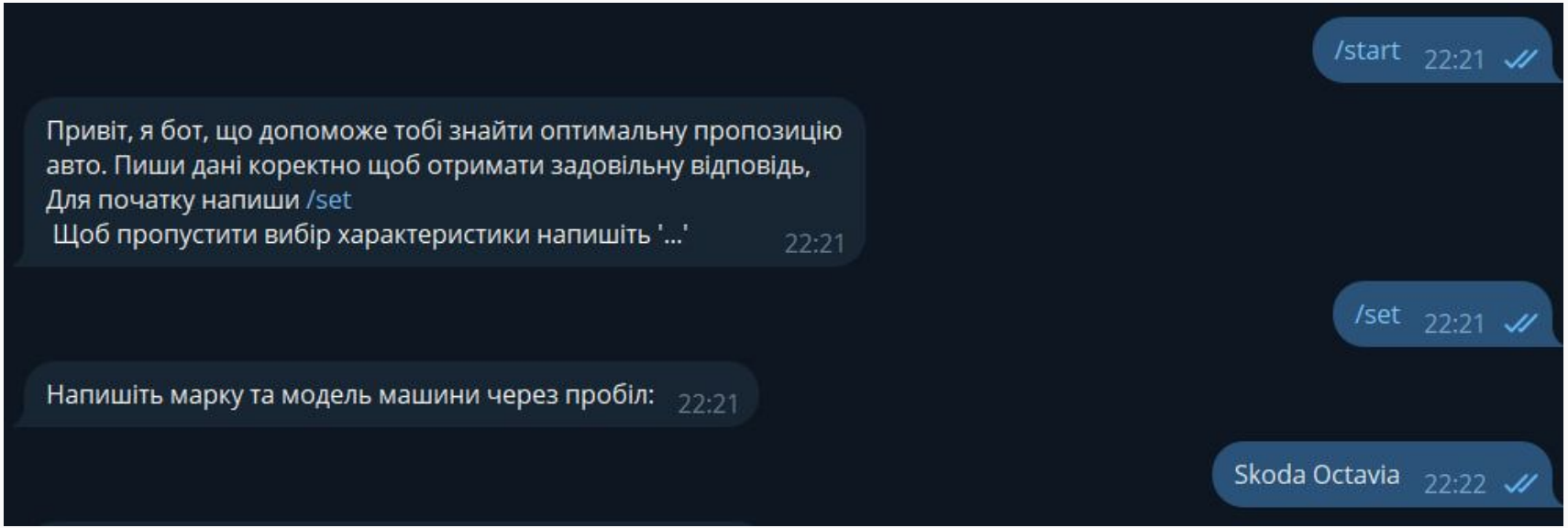
2. Вебкраулінг, отримання посилань на вебсторінки пропозицій (Scrapy)

    2.1 Екстрагування даних з вебсторінки пропозиції (BeautifulSoup)

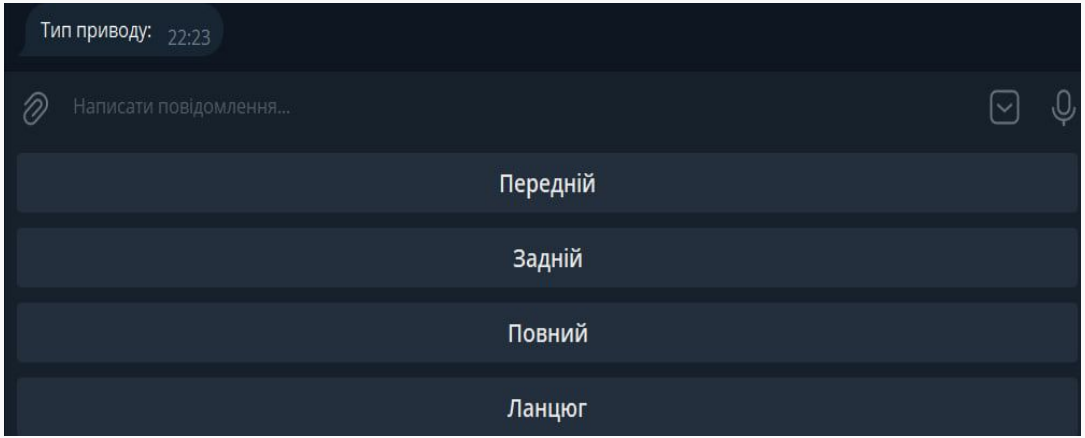
    2.2 Співставлення даних (користувацькі та поточні)

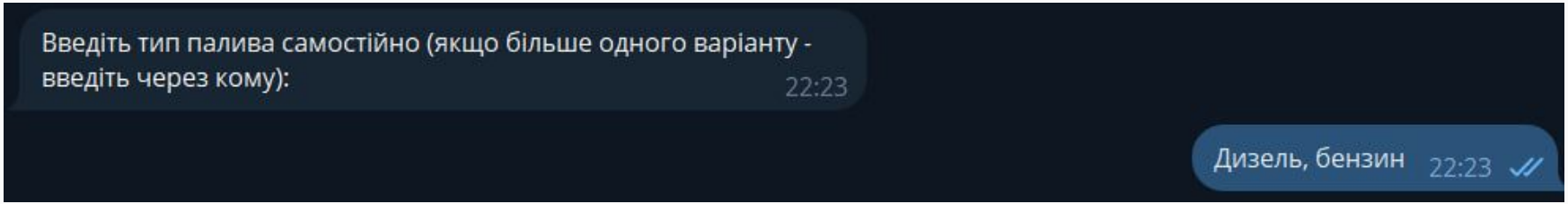
3. Сортування за схожістю описів (користувацький та поточний)

Початок роботи телеграм-бота

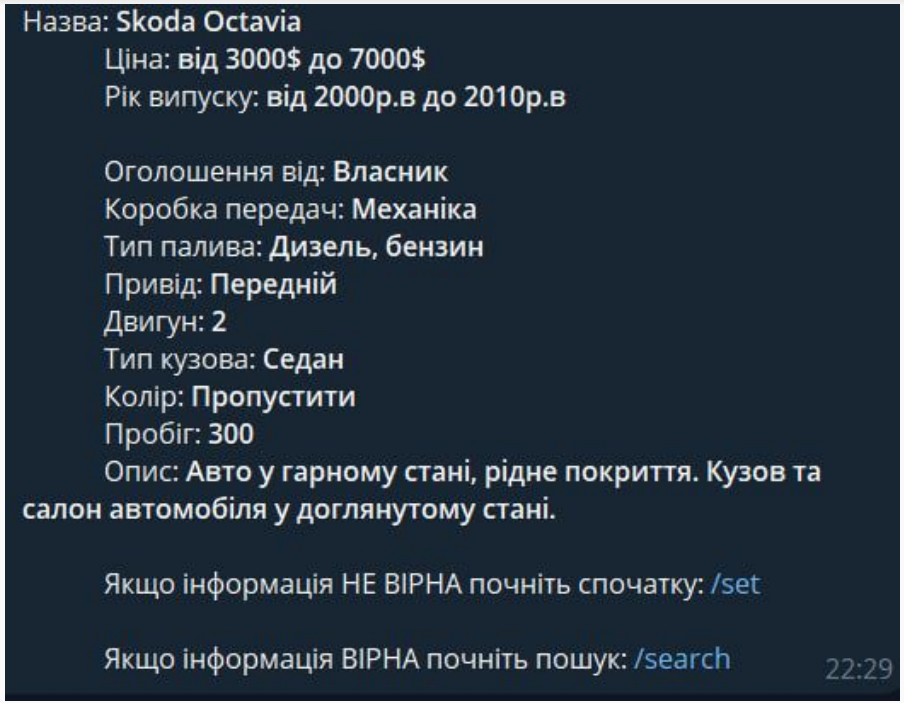


Введення даних авто

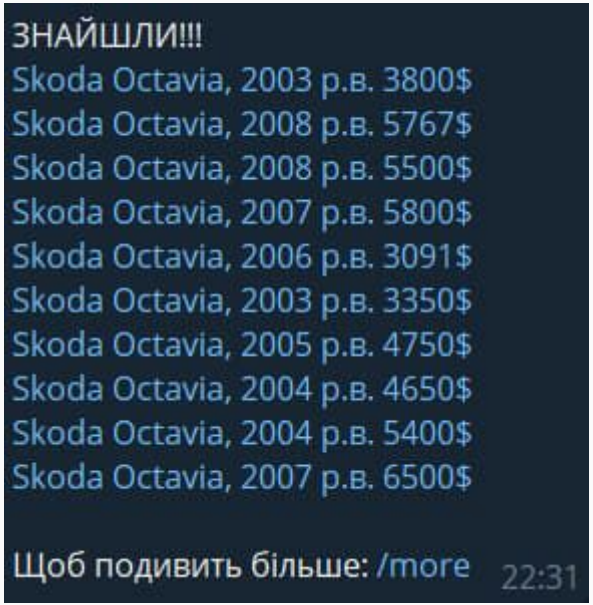




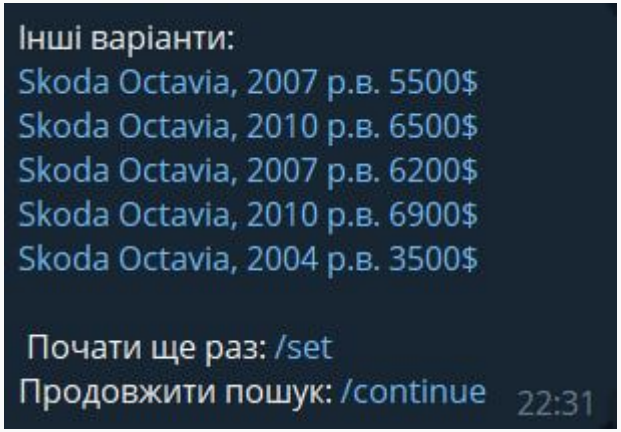
Початок пошуку



Результат пошуку







Висновки

- Досліджено процес вебскрапінгу
- Обрано інструменти для програмної реалізації процесу вебскрапінгу
- Обрано інструменти для створення телеграм-бота
- Створено телеграм-бот для пошуку пропозицій вживаних авто

Розроблений телеграм-бот може бути корисним для осіб, які планують купувати вживане авто.

Подальші плани

- Покращити процес вебскрапінгу та телеграм-бот
- Збільшити кількість веб-сайтів для пошуку

У подальшому, використовуючи схожий принцип роботи, можна створити подібні телеграм-боти для пошуку необхідної інформації з певної теми.

Дякую за увагу

Питання та відповіді