Київський національний університет імені Тараса Шевченка
Факультет інформаційних технологій
Кафедра програмних систем і технологій

УДК 004.54 *На правах рукопису*

ВИПУСКНА КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

Тема: «Розроблення програмного забезпечення здійснення аналізу та пошуку оптимального варіанту закупівель в інтернеті»

Спеціальність 121 “Інженерія програмного забезпечення”

ПОЯСНЮВАЛЬНА ЗАПИСКА

Студент
_____ Шевченко Р.С.
(підпис) (розшифровка підпису) (дата)

Науковий керівник
доц. _____ Супрун О.М.
(посада) (підпис) (розшифровка підпису) (дата)

Допускається до захисту
з питань нормоконтролю
Завідувач кафедри

_____ Бичков О.С.
(підпис) (розшифровка підпису) (дата)

2021

Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій

Кафедра програмних систем і технологій

Спеціальність 121 “Інженерія програмного забезпечення”

ЗАТВЕРДЖУЮ:
Завідувач кафедри
програмних систем і технологій
_____(О.С.Бичков)
„___”_____ 20___р..

ЗАВДАННЯ

НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ МАГІСТЕРСЬКУ РОБОТУ

СТУДЕНТУ

Шевченку Руслану Станіславовичу

---

1. Тема випускної кваліфікаційної магістерської роботи “Розроблення програмного забезпечення здійснення аналізу та пошуку оптимального варіанту закупівель в інтернеті”

Керівник проекту: Супрун Ольга Миколаївна

Затверджені протоколом кафедри №____ від _____

2. Строк подання студентом роботи: _____

3. Вихідні дані до проекту: програмне забезпечення для здійснення аналізу та пошуку оптимального варіанту закупівель в інтернеті.

4. Зміст пояснювальної записки:

1) Аналітична частина:

Дослідити зчитування даних із інтернет ресурсів. Визначити найкращі засоби для здійснення аналізу та пошуку оптимального варіанту закупівель в інтернеті в наші дні.

2) Практична частина:

Визначити особливості архітектурного рішення програмного забезпечення для здійснення аналізу та пошуку оптимального варіанту закупівель в інтернеті, розробити таку системи з необхідним функціоналом.

5. Дата видачі завдання “ ” _____20 р.

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

Номер і назва етапів роботи	Термін виконання етапів роботи	Примітка
1. Визначення постановки задачі		
2. Уточнення постановки задачі		
3. Підбір та аналіз літератури		
4. Проектування рішення 4.1. Документування та обґрунтування запропонованого рішення 4.2. Створення робочого прототипу		
5. Дослідження проблематики та уточнення рішення		
6. Програмна реалізація 6.1. Уточнення кожного блоку, модуля та документування деталізованих вимог 6.2. Розроблення та тестування кожного блоку, модуля 6.3. Внесення правок в документацію за необхідності		
7. Створення пакету програмного забезпечення та інструкції із використання		
8. Оформлення і друк пояснювальної записки		
9. Отримання рецензій		
10. Оформлення презентацій		
11. Захист		

Студент – магістр ______________ _________________________

Керівник роботи ________________ _______________________

АНОТАЦІЯ

Випускна кваліфікаційна магістерська робота: 69 с., 14 рис., 1 додат., 14 джерел.

Тема: “Розроблення програмного забезпечення здійснення аналізу та пошуку оптимального варіанту закупівель в інтернеті”

Об’єкт дослідження: принципи, методології та засоби для зчитування даних із веб-сайтів.

Предмет дослідження: система для здійснення аналізу та пошуку оптимального варіанту закупівель в інтернеті.

Мета роботи: оптимізація процесу вибору товару в інтернеті.

Результати дослідження: під час досліджень визначено методи пошуку оптимального варіанту закупівель в інтернеті.

Висновок:

Проаналізовано способи та засоби для зчитування даних із веб-сайтів. Базуючись на отриманих даних, було реалізовано систему для здійснення аналізу та пошуку оптимального варіанту закупівель в інтернеті, яку можна інтегрувати із іншими застосунками.

АННОТАЦИЯ

Выпускная квалификационная магистерская работа: 68 с., 14 рис., 1 прилож., 14 источников.

Тема: “Разработка ПО осуществления анализа и поиска оптимального варианта закупок в интернете”

Объект исследования: принципы, методологии и средства для считывания данных с веб-сайтов.

Предмет исследования: система для осуществления анализа и поиска оптимального варианта закупок в интернете.

Цель работы: оптимизация процесса выбора товара в интернете.

Результаты исследования: во время исследований было определено методы для поиска оптимального варианта закупок в интернете.

Вывод:

Проанализированы способы и средства для считывания данных с веб-сайтов. Основываясь на полученых данных, было реализовано систему для осуществления анализа и поиска оптимального варианта закупок в интернете, которую можно интегрировать с другими приложениями.

ANNOTATION

Graduation Baccalaureate Work: 68 pages, 14 pictures, 1 addition, 14 sources.

Theme: “Development of software for analysis and search of online purchases optimal variant”

An object of the research: : principles, methodologies and tools for reading data from websites.

The subject of the research system for analysis and search of online purchases optimal variant.

The purpose of the work: optimization of the a product purchases process in the Internet.

Research results: during the research, there were identified to find the best option for online purchases.

Conclusion:

There were analyzed methods and instruments for reading data from websites. Based on the obtained data, there was implemented a system for analysis and search of online purchases optimal variant, which can be integrated with other applications.

ЗМІСТ

ПЕРЕЛІК ОСНОВНИХ ПОЗНАЧЕНЬ

Скрипт - послідовність команд, інструкцій на сценарній мові, що використовується для автоматизації ручних завдань.

Фреймворк – готовий до використання комплекс програмних рішень, що охоплює дизайн, логіку та базову функціональність системи чи підсистеми.

Баг – дефект в програмі.

БД – база даних.

Рефакторинг – процес оптимізації програмного забезпечення, усунення недоліків та виправлення помилок.

Модуль(в Python) – файл з роширенням .py, який може зберігати в собі всі елементи мови програмування Python (класи, методи, поля, конструктори і т.д.)

Логування – ведення та зпис інформативних повідомлень про стан виконання програми.

GUI(Graphic User Interface) – графічний інтерфейс користувача.

IntelliSense - це загальний термін для різних функцій редагування коду, включаючи: завершення коду, інформацію про параметри, швидку інформацію про методи та функції.

DOM - це поедставлення документу у вигляді дерева об'єктів, доступного для зміни через JavaScript.

HTML (від англ. HyperText Markup Language - «мова гіпертекстової розмітки») - стандартизована мова розмітки документів у Всесвітній павутині. Більшість веб-сторінок містять опис розмітки на мові HTML (або XHTML). Мова HTML інтерпретується браузерами; отриманий в результаті інтерпретації форматований текст відображається на екрані монітора, комп'ютера або мобільного пристрою.

CLI (Command line interface) - Інтерфейс командного рядка, різновид текстового інтерфейсу між людиною і комп'ютером, в якому інструкції комп'ютеру даються в основному шляхом введення з клавіатури текстових рядків.

ORM (Object-Relational Mapping) - технологія програмування, яка зв'язує бази даних з концепціями об'єктно-орієнтованих мов програмування, створюючи «віртуальну об'єктну базу даних».

XPath (XML Path Language) - мова запитів до елементів XML-документа. Розроблена для організації доступу до частин документа XML в файлах трансформації XSLT і є стандартом консорціуму W3C. XPath використовується для навігації по DOM в XML. У XPath використовується компактний синтаксис, відмінний від прийнятого в XML.

ВСТУП

Нині більшість покупок здійснюється через інтернет. Існує безліч онлайн магазинів з доступом до будь-яких видів товарів. Сьогодні онлайн можна придбати абсолютно все, але не кожен може дозволити собі замовити товар за поточною ціною. Для вибору оптимального варіанту існує не так багато програмного забезпечення і зазвичай воно заточене під конкретний інтернет-магазин. Саме тому було прийняте рішення розробити програмне забезпечення, яке дозволяє здійснювати аналіз та пошук оптимального варіанту закупівель в інтернеті в різних інтернет-магазинів, в подальшому легко додавати нові магазини, а також за потреби інтегрувати розроблене програмне забезпечення в інші застосунки.

Актуальність роботи полягає в тому, що закупівлі в інтернеті є найпопулярнішим способом придбати товари, особливо в період пандемії. При великій кількості пропозицій в різних інтернет-магазинах засіб для оптимального вибору допоможе робити закупівлі за найбільш вигідною ціною.

Мета і задачі дослідження

Метою роботи є оптимізація процесу вибору товару в інтернеті.

Досягнення мети включало розв’язання таких задач:

1) дослідження способів зчитування даних із веб-сайтів;
2) виявлення найбільш продуктивних та популярних способів зчитування даних із веб-сайтів;
3) проектування системи здійснення аналізу та пошуку оптимального варіанту закупівель в інтернеті;
4) реалізація системи здійснення аналізу та пошуку оптимального варіанту закупівель в інтернеті.

Об’єктом дослідження є принципи, методології та засоби для зчитування даних із веб-сайтів.

Предметом дослідження є система для здійснення аналізу та пошуку оптимального варіанту закупівель в інтернеті.

Методом дослідження є аналіз особливостей розробки систем для здійснення аналізу та пошуку оптимального варіанту закупівель в інтернеті.

Новизна одержаних результатів полягає у тому, що розроблене ПЗ можна використовувати як у готовому вигляді, так і інтегрувати в іншому ПЗ.

Практичне значення одержаних результатів полягає у тому, що користувачі даного програмного забезпечення можуть зручно відслідковувати ціни на товари та оптимально обирати, де робити закупівлі.

Структура та обсяг роботи

Робота викладена на 68 сторінках друкованого тексту, який складається із вступу, двох розділів, висновків та списку використаних джерел (14 найменувань). Робота містить 14 рисунків та 1 додаток.

1. ЗЧИТУВАННЯ ДАНИХ ІЗ ВЕБ-САЙТІВ ТА ЙОГО ЗАСТОСУВАННЯ У РОЗРОБЦІ ПЗ

Постійна оптимізація інтернет-ресурсів під мобільні пристрої, зростаюча швидкість інтернету, технологічні рішення на рівні "заліза" та програмного коду, а також дизайнерські пошуки зробили всесвітню павутину такою, якою бачимо її сьогодні. А саме - барвистою, контрастною, переповненій потрібної і абсолютно марної (часом навіть шкідливою) інформації. Ця різноманітність технік і технологій в реалізації веб-сайтів і є основною проблемою, яку необхідно вирішити при організації доступу до даних.

Доступ до цільового контенту може бути організований за двома основними напрямками:

З використанням API, коли власники інформації передають її користувачам на основі особистої зацікавленості: підписки, розсилки, партнерські програми і т.д.

Без використання API: шляхом вичитування даних із веб-сторінок методом скрапінгу (якщо, звичайно, це передбачено їх кодом).

Перший напрямок обмежується тільки грошовими ресурсами, і споживач не відчуває труднощів у плані технічної реалізації. Дані, що отримуються за допомогою API, чітко структуровані і нормалізовані. Наприклад, в форматах XML або JSON.

Другий напрямок (веб-скрапінг) заслуговує на більшу увагу і є свого роду "викликом" для розробників і математиків. Автоматична обробка тексту з використанням штучного інтелекту, семантичний аналіз і т.д. - все це можна назвати колосальним технологічним проривом, що вимагає засвоєння, усвідомлення і компетентної оцінки.

1.1. Веб-скрапінг

Веб-скрапінг - це зчитування даних, яке використовується для вилучення даних з веб-сайтів [1]. Програмне забезпечення для скрапінгу веб-сторінок може отримати доступ до всесвітньої павутини безпосередньо за допомогою протоколу передачі гіпертексту або через веб-браузер. Хоча веб-скрапінг може бути здійснено вручну користувачем програмного забезпечення, даний термін зазвичай стосується автоматизованих процесів, реалізованих за допомогою бота або веб-сканера. Це форма копіювання, в якій конкретні дані збираються та копіюються з Інтернету, як правило, в центральну локальну базу даних або електронну таблицю для подальшого пошуку або аналізу.

Розшифровка веб-сторінки включає в себе її вилучення та вилучення з неї. Вилучення сторінки - це завантаження її (що робить браузер під час перегляду сторінки). Отже, сканування веб-сторінки є головним компонентом скрапінгу для отримання інформації і її подальшої обробки. Отрману сторінку можна проаналізувати, знайти потрібні дані, переформатувати, скопіювати в електронну таблицю тощо. Веб-скрапери зазвичай зчитують певну інформацію із сторінки, щоб використовувати її з іншою метою в іншому місці. Прикладом може бути пошук та копіювання імен і номерів телефонів або компаній та їх URL-адрес до списку (скрапінг контактів).

Веб-скрапінг в основному являється компонентом програм, що використовуються для веб-індексації, видобутку даних, онлайн-моніторингу зміни цін та порівняння цін, зчитування інформації про товари (для спостереження за конкуренцією), збору списків нерухомості, моніторингу погодних даних, виявлення змін на веб-сайтах, дослідження, відстеження присутності та репутації в Інтернеті, веб-розміщення та інтеграції веб-даних.

1.2. Історія веб-скрапінгу

Історія скрапінгу веб-сайтів сягає майже того часу, коли народився Інтернет. Після появи Всесвітньої павутини в 1989 році в червні 1993 року був створений перший веб-робот, World Wide Web Wanderer, який був призначений лише для вимірювання розміру мережі.

У грудні 1993 року була запущена перша веб-пошукова система на базі сканерів - JumpStation. Оскільки в Інтернеті було не так багато веб-сайтів, пошукові системи на той час покладалися на своїх адміністраторів веб-сайтів для збирання та редагування посилань у певному форматі. Для порівняння, JumpStation приніс новий стрибок, будучи першою пошуковою системою WWW, яка ґрунтувалась на веб-роботі.

У 2000 році з'явився перший веб-сканер і веб-API. API розшифровується як інтерфейс програмування прикладних програм. Це інтерфейс, який значно спрощує розробку програми, надаючи «будівельні блоки» для імплементації програмного забезпечення. У 2000 році Salesforce та eBay запустили власний API, за допомогою якого програмістам було надано доступ до завантаження деяких даних, доступних для населення. З того часу багато веб-сайтів пропонують веб-API для доступу людей до їх публічної бази даних.

1.3. Технології веб-скріпінгу

Веб-скрапінг - це процес автоматичного видобутку даних або збору інформації з всесвітньої павутини. Це галузь з активними розробками, що ділиться спільною метою із семантичним веб-баченням, амбітною ініціативою, яка все ще потребує прориву в обробці тексту, семантичному розумінні, штучному інтелекті та взаємодії людини та комп'ютера. Поточні рішення щодо зчитування веб-сторінок варіюються від спеціальних, що вимагають людських

зусиль, до повністю автоматизованих систем, які здатні перетворювати цілі веб-сайти в структуровану інформацію з обмеженнями [2].

1.4. Ручний скрапінг

Ручний скрапінг передбачає копіювання та вставлення веб-контенту, що вимагає великих зусиль і постійних повторень. Це ефективний спосіб зчитування вмісту, коли захисні механізми веб-сайту налаштовані лише на виявлення автоматизованих ботів для скрапнігу. Однак ручний скрапінг рідко зустрічається на практиці, оскільки автоматичний набагато швидший і дешевший у застосуванні.

Переваги ручного скрапінгу:

- Висока якість контенту і його цільова адаптація під потреби споживача.
- Висока швидкість пошуку.

Недоліки ручного скрапінгу:

- Можливості людини істотно обмежені його фізичними ресурсами, знаннями цільової сфери і банальними навичками пошуку в Інтернеті (далеко не всі вміють ефективно використовувати інструменти пошукових сервісів).
- Людина схильна до різних зовнішніх впливів (психологічним, фізичним і т.д.), а це негативно позначається стабільності її роботи і вартості послуг.
- Обмеженість скрапінгу до декількох сотень якісних результатів в день.

1.5. Автоматизований скрапніг

1.5.1. Розбір HTML

Синтаксичний аналіз HTML здійснюється за допомогою JavaScript. Цей швидкий і надійний метод використовується для вилучення тексту, вилучення посилань (таких як вкладені посилання або адреси електронної пошти), зчитування екрана, вилучення ресурсів тощо.

Переваги розбору HTML:

- Дозволяє отримати оригінал сторінки у вигляді HTTP-відповіді.

- Величезна кількість результатів, обмежених тільки ресурсами серверів і швидкістю Інтернету.

Недоліки розбору HTML:

- Потрібна обробка отриманих відповідей - результати можуть містити багато зайвого.
- Багато сайтів оснащені захистом від подібних "роботів" (в якості виходу з ситуації слід генерувати додаткову службову інформацію в заголовку HTTP-запиту, втім, не всі сайти можна обдурити таким чином).
- Висока ймовірність опинитися заблокованим адміністратором цільового сайту або автоматизованою системою захисту при появі стороннього методично повторюваного "інтересу" до ресурсу. Практика показує, що кількість і частота запитів може перевершувати людські можливості.
- Віддалений сервер може бути відключений або зайнятий на момент відправлення запиту. В результаті з'являється можливість великої кількості помилок типу TimeOut.

1.5.2. Розбір DOM

Модель об'єкта документа або DOM визначає стиль, структуру та вміст у файлах XML. DOM-аналізатори зазвичай використовуються скраперами, які хочуть отримати глибоке уявлення про структуру веб-сторінки. Зчитані дані можуть використовувати аналізатор DOM для отримання вузлів, що містять інформацію, а потім використовувати інструмент, такий як XPath, для скрапінгу веб-сторінок. XPath - мова запитів, яка працює над XML-документами. Оскільки документи XML засновані на деревоподібній структурі, XPath можна використовувати для навігації по дереву, вибираючи вузли на основі різноманітних параметрів.

Повноцінні веб-браузери, такі як Internet Explorer або Firefox, можуть бути вбудовані для вилучення всієї веб-сторінки або лише її частин, навіть якщо створений вміст динамічний за своєю суттю.

Переваги розбору DOM:

- Отримання динамічного контенту.
- Автоматизація. Вона дозволяє отримувати якісний контент у великих обсягах.
- Можливість реалізації комерційних рішень. Метод дозволяє безмежно користуватися їх підтримкою для вирішення проблем з купленим / орендованим ПЗ.

Недоліки розбору DOM:

- Складність і завантаженість сервера при автоматизації робить процес досить ресурсномістким, як в розробці, так і в серверних витратах.
- Складність реалізації. Для нефахівців вона практично неможлива, тому що вимагає докладних знань "заліза", основ веб-розробки і досконале володіння хоча б одною з серверних мов програмування.
- Велика частина реалізацій цього методу поширюється тільки на комерційній основі, а вартість таких продуктів поки не має тенденції до зниження.

1.5.3. Методи штучного інтелекту та онтологій

Іноді стоїть завдання скрапінгу сотень або тисяч сайтів. При цьому вони мають різну верстку і написані на різних мовах і фреймворк. У такій ситуації раціональніше всього буде інвестувати кошти в розробку складних систем штучного інтелекту і / або онтологій (цей метод заснований на теорії про те, що всі сайти можна розбити на класи і групи з подібною структурою і набором технологій).

Переваги методів штучного інтелекту:

- Одного разу створена складна система дозволяє отримати максимально якісний контент від величезного числа доменів, навіть незважаючи на невеликі зміни на сторінках (інтелектуальна система підправить можливі неточності). Оцінка якості для 150 тис. Доменів в середньому буде становити від 75% до 93% (перевірено на дослідженнях JetRuby в реалізованої системі).
- Метод дозволяє нормалізувати результат, отриманий з усіх джерел під вашу структуру бази даних.
- Незважаючи на те, що така система потребує постійної підтримки (на рівні моніторингу), при можливих збоях вона вимагає незначного втручання в код

Недоліки методів штучного інтелекту:

- Складна реалізація "двигуна", що вимагає високого рівня знань в математиці, статистиці, сфері нечіткої логіки.
- Висока вартість розробки.
- Супутні витрати на підтримку і навчання системи.
- Практика підписок на готові комерційні проекти. Мається на увазі обмежене число запитів і їх висока вартість.
- Необхідність передбачати модулі відстеження помилок, валідності даних і резервні сервери для можливого обходу "чорного списку" цільового сайту.

1.5.4. Зчитування за шаблоном

Це використання регулярних виразів за допомогою команди grep UNIX і, як правило, поєднується з популярними мовами програмування, такими як Perl або Python.

Переваги зчитування за шаблоном:

- Якщо ви вже знайомі з регулярними виразами хоча б однієї мови програмування, то реалізація цього рішення займе мінімум часу.

• Регулярні вирази дозволяють швидко вилучати велику кількість непотрібних мінорних деталей з тіла результату, не поламавши основний контент (наприклад, очистити залишки HTML коду).

• Регулярні вирази підтримуються багатьма мовами програмування. І, що найголовніше, їх синтаксис від мови до мови майже не змінюється. Це дозволяє здійснювати безболісну міграцію проектів на мови з більшою продуктивністю і ясністю коду (наприклад, з PHP на Ruby).

Недоліки зчитування за шаблоном:

• Регулярні вирази можуть перетворитися в головоломку для тих, хто їх раніше не використовував. В такому випадку краще відразу звернутися до фахівців. Як правило, проблеми виникають при інтеграції рішень на одній мові в іншу або при міграції проектів на іншу мову програмування.

• Регулярні вирази часто дуже складні для читання і аналізу. Іноді, виходячи із специфіки оброблюваної інформації, вони надмірно розтягуються.

• Якщо на цільовому ресурсі був змінений HTML код або доданий новий тег, швидше за все доведеться міняти і регулярний вираз (інакше є великий ризик отримати "битий" контент).

У Інтернеті доступні кілька інструментів та служб веб-скрапінгу. Існують такі інструменти, як cURL, Wget, HTTrack, Import.io, Node.js та кілька інших, які високо автоматизовані.

1.6. Цілі застосування веб-скрапінгу

Веб скрапінг має широкий спектр застосувань:

1. Відстеження цін - збираючи інформацію про товари та їх цінах на Amazon і інших платформах, можна стежити за конкурентами і адаптувати свою цінову політику.

2. Ринкова і конкурентна розвідка – якщо стоїть ціль проникнути на новий ринок і оцінити можливості, аналіз даних допоможе зробити зважене і адекватне рішення.

3. Моніторинг соцмереж - YouScan, Brand Analytics і інші платформи для моніторингу соцмереж використовують скрапінг.

4. Машинне навчання - з одного боку, машинне навчання і AI використовуються для збільшення продуктивності скрапінга. З іншого боку, дані, отримані з його допомогою, використовують в машинному навчанні. Інтернет - це важливе джерело даних для алгоритмів машинного навчання.

5. Модернізація сайтів - компанії переносять застарілі сайти на сучасні платформи. Для того щоб швидко і легко експортувати дані, вони можуть використовувати скрапінг.

6. Моніторинг новин - скрапінг даних з новинних сайтів і блогів дозволяє відстежувати цікаві теми і економить час.

7. Аналіз ефективності контенту - блогери або творці контенту можуть використовувати скрапінг для вилучення даних про пости, відео, твіти і т. д. в таблицю.

1.7. Способи захисту від веб-скрапінгу

Існують методи для запобігання сайтами веб-скрапінгу, такі як виявлення і блокування від обходу (перегляду) ботами сторінок сайту. Як відповідь на це існують системи веб-скрапінгу, які покладаються на використання методів аналізу DOM, комп'ютерного зору і обробки природної мови для імітації перегляду людиною, щоб забезпечити збір вмісту веб-сторінки для автономного аналізу [3].

Адміністратори можуть блокувати програми веб-скрапінгу, щоб інформація не була використана конкурентами. Програми скрапінгу можуть бути розпізнані за такими ознаками:

- незвичайна поведінка користувача (наприклад, сотні переходів на нову сторінку сайту кожну секунду);
- повторювані безрезультатні дії (користувач не буде виконувати одні й ті ж завдання раз по раз);

- використання посилань, які містяться тільки в коді веб-сайту і не видно звичайним користувачам.

Способи блокування:

- Заборонити доступ на сайт з вказаної IP-адреси (наприклад, коли ботом пройдено понад 100 сторінок за сесію);
- Заборонити ідентифікатор користувача, який є з точки зору адміністратора сайту зловмисником, що заходить на сайт з аутентифікації.

1.8. Способи обходу блокування

1.8.1. Випадкові інтервали між запитами

Жодна реальна людина не стане відправляти запити на сайт кожну секунду протягом 24 годин - такий збір інформації дуже легко виявити. Потрібно використовувати випадкові затримки (наприклад близько 2-10 секунд), щоб уникнути блокування. Не слід відправляти запити занадто часто, інакше сканування сайту буде походити на мережеву атаку.

На деяких сайтах варто перевірити файл robots.txt (як правило, знаходиться на http://<адреса сайту>/robots.txt). Іноді там можна знайти параметр Crawl-delay, який говорить, скільки секунд потрібно почекати між запитами, щоб не шкодити роботі сервера.

```
#
# robots.txt
#
# This file is to prevent the crawling and indexing of certain parts
# of your site by web crawlers and spiders run by sites like Yahoo!
# and Google. By telling these "robots" where not to go on your site,
# you save bandwidth and server resources.
#
# This file will be ignored unless it is at the root of your host:
# Used:    http://example.com/robots.txt
# Ignored: http://example.com/site/robots.txt
#
# For more information about the robots.txt standard, see:
# http://www.robotstxt.org/robotstxt.html

User-agent: *
Crawl-delay: 10
# CSS, JS, Images
Allow: /misc/*.css$
Allow: /misc/*.css?
Allow: /misc/*.js$
Allow: /misc/*.js?
Allow: /misc/*.gif
```

Рис.1.1. Приклад файлу robots.txt

1.8.2. Налаштування User Agent

User Agent - HTTP-заголовок, який повідомляє відвідуваному веб-сайту інформацію про браузер користувача. Якщо не налаштувати User Agent, скрапер буде дуже легко виявити. Крім того, сайти іноді блокують запити користувацьких агентів від невідомих браузерів. Тому іноді доводиться встановити один з популярних користувацьких агентів.

Також можна спробувати встановити свій агент на Googlebot User Agent - пошуковий робот Google. Більшість веб-сайтів, очевидно, хочуть потрапити в видачу Google і пропускають Googlebot.

Гарною практикою буде також чергування різних User Agent.

1.8.3. Налаштування проксі

Справжній користувач не зможе зможе запитувати 20 сторінок в секунду на одному і тому ж сайті. Щоб обійти веб-сервер, потрібно зробити видимість, що всі ці запити приходять з різних місць. Цю проблему вирушує використання проксі.

Для цього є безліч варіантів, наприклад сервіси SmartProxy, Luminati Network, Blazing SEO. Безкоштовні проксі не завжди підійдуть для таких цілей: вони часто повільні і ненадійні. Також можна створити свою проксі-мережу на сервері, наприклад за допомогою Scrapoxy - API з відкритим вихідним кодом.

1.8.4. Додавання referrer

Referer - заголовок HTTP-запиту, який дає зрозуміти, з якого сайту надіслався запит. Найефективнішим способом буде зробити так, щоб він показував, ніби користувач перейшли з Google:

Referer: https://www.google.com/

Варто міняти referer для веб-сайтів в різних країнах: наприклад для України використовувати https://www.google.ua/, а не https://www.google.com/. Замість Google можна підставити адреси соцмереж: Youtube, Facebook, Twitter. Referer допоможе зробити так, щоб запити виглядали як трафік з того сайту, звідки зазвичай приходить найбільше відвідувачів.

1.8.5. Використання headless-браузерів

Особливо технічно продвинуті сайти можуть відстежувати веб-шрифти, розширення, файли cookie, цифрові відбитки (фінгерпрінт). Іноді вони навіть вбудовують JavaScript-код, що відкриває сторінку тільки після його запуску - так часто можна визначити, чи надходить запит з браузера. Для обходу таких ресурсів необхідний headless-браузер. Він емулює поведінку справжнього браузера і підтримує програмне управління. Найчастіше для цих цілей вибирають Chrome Headless.

Якщо ресурс відстежує цифровий відбиток браузера, то навіть багаторазова зміна IP і очищення cookie не завжди допомагають, так як все одно можна впізнати по фінгерпрінту. За часту зміну IP при одному і тому ж відбитку цілком можуть заблокувати, і одне із завдань Chrome Headless - не допустити цього.

Найпростіший спосіб працювати з Chrome Headless - використовувати фреймворк, який об'єднує всі його функції в зручний API. Найбільш відомі рішення можна знайти в інтернеті. Але деякі веб-ресурси намагаються відстежувати і їх: йде постійна гонка між сайтами, які намагаються виявити headless-браузери, і headless-браузерами, які видають себе за справжні.

1.8.6. Підключення програми для вирішення CAPTCHA

Існують веб-сайти, які систематично просять підтвердити, що користувач не робот, за допомогою капч. Зазвичай капчі відображаються тільки для підозрілих IP-адрес, і з цим допоможуть проксі. В інших же випадках використовуються автоматичні вирішувачі CAPTCHA - наприклад, 2Captcha або AntiCaptcha.

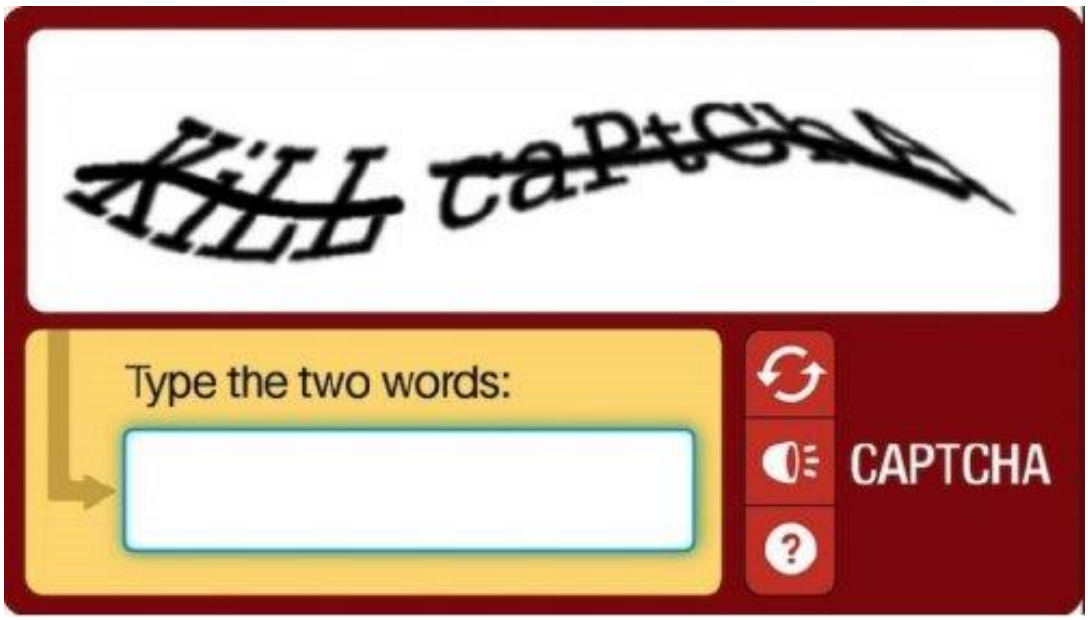

Рис.1.2. Приклад перевірки CAPTCHA

Розпізнавачі найчастіше платні, тому що капчі вручну вирішують реальні люди. Тому варто зрозуміти, чи виправдають витрати поставлену мету.

1.8.7. Уникання honeypot-пасток

«Honeypot» - це фальшиве посилання, яке невидиме для звичайного користувача, але присутнє в HTML-коді. Як тільки скрапер починає аналізувати сайт, honeypot може перенаправити на порожні і непотрібні сторінки-приманки.

Тому варто перевіряти, чи встановлені для посилання CSS-властивості «display: none», «visibility: hidden» або «color: #fff;» (В останньому випадку потрібно враховувати колір фону сайту).

1.9. Готові сервіси для веб-скрапінгу

Програми веб-скрапнігу не розраховані на звичайних користувачів, з ними працюють програмісти, які в більшості випадків пишуть код під конкретні завдання. В Інтернеті можна знайти різні засоби та інструменти для веб-скрапінгу: бібліотеки, додатки, online-сервіси, хмарні сервіси, сервіси типу DaaS, плагіни до браузерів. Один з популярних засобів скрапінгу Scrapy (це безкоштовний фреймворк з відкритим кодом). Серед комерційних популярною є платформа Import.IO [4].

Існують розробки, наприклад, Nokogiri, який створений спеціально для мови програмування Ruby, скрапери, які виконують певне завдання з безлічі можливих: Outwit Hub збирає текстову інформацію і розподіляє по осередках. Нові форми веб-скрапінгу включають прослуховування каналів даних з веб-серверів. Наприклад, JSON зазвичай використовується в якості транспортного механізму зберігання даних між клієнтом і веб-сервером.

Отримання даних з сайтів за допомогою доступу до API також ефективний. Такі компанії, як Amazon AWS і Google (API Discovery service), надають кінцевим користувачам безкоштовні інструменти, сервіси та загальнодоступні дані для парсинга.

Скрапінг вимагає правильного парсинга вихідного коду сторінки, рендеринга JavaScript, перетворення даних в читаємий вигляд і, в разі потреби, фільтрації. Тому існує безліч готових сервісів для виконання скрапінгу.

1.9.1. Octoparse

Octoparse - це простий у використанні скрапер для програмістів і не тільки. У нього є безкоштовний тарифний план і платна підписка.

Особливості Octoparse

- працює на всіх сайтах: з нескінченним скроллом, пагінацією, авторизацією, випадаючими меню, AJAX і т.д. ;
- зберігає дані в Excel, CSV, JSON, API або БД;
- дані зберігаються в хмарі;
- скрапінг за розкладом або в реальному часі;
- автоматична зміна IP для обходу блокування;
- блокування реклами для прискорення завантаження і зменшення кількості HTTP запитів;
- можна використовувати XPath і регулярні вирази;
- підтримка Windows і macOS;
- безкоштовний для простих проектів, 75 $ / місяць - стандартний, 209 $ / місяць - професійний і т. д.

1.9.2. ScrapingBee

ScrapingBee Api використовує «безголовий браузер» і зміну проксі. Також має API для скрапінгу результатів пошуку Google.

Особливості ScrapingBee:

- рендеринг JS;
- ротація проксі;
- можна використовувати з Google Sheets і браузером Chrome;
- безкоштовний до 1000 викликів API, 29 $ / місяць - для фрілансерів, 99 $ / місяць - для бізнесу і т.д.

1.9.3. ScrapingBot

ScrapingBot надає кілька API: API для необробленого HTML, API для сайтів роздрібної торгівлі, API для скрапінгу сайтів нерухомості.

Особливості ScrapingBot:

- рендеринг JS;

- якісний проксі;
- до 20 одночасних запитів;
- геотеги (можливість зчитувати локації);
- додаток Prestashop, інтегрується на сайт для моніторингу цін конкурентів;
- безкоштовний тариф на 100 кредитів, 47 $ / місяць для фрілансерів, 120 $ / місяць для стартапів, 361 $ / місяць для для бізнесу і т. д.

1.9.4. Scrapestack

Scrapestack - це REST API для веб скрапінгу в реальному часі. Він дозволяє збирати дані з сайтів за мілісекунди, використовуючи мільйони проксі і обходячи капчу.

Особливості Scrapestack:

- одночасні API запити;
- рендеринг JS;
- шифрування HTTPS;
- більше 100 геолокації;
- безкоштовний тариф до 1000 запитів, базовий тариф за 19.99 $ / місяць, професійний тариф за 79.99 $ / місяць і т. д.

1.9.5. Scraper API

Scraper API працює з проксі, браузерами і капчечю. Його легко інтегрувати. Слід лише відправити GET запит до API з вашим API ключем і URL.

Особливості Scraper API:

- рендеринг JS;
- геотеги (можливість зчитувати локації);
- має пул резидентних \ мобільних проксі для скрапінгу цін, результатів пошуку, моніторингу соцмереж і т. д.
- 1000 викликів API безкоштовно, тариф для хобі - 29 $ \ місяць, 99 $ \ місяць - для стартапів і т. Д.

1.9.6. ParseHub

ParseHub - це сервіс для веб скрапінгу, що не вимагає навичок програмування.

Особливості ParseHub:

- зрозумілий графічний інтерфейс;
- експорт даних в Excel, CSV, JSON або доступ через API;
- XPath, регулярні вирази, CSS селектори;
- безкоштовний тариф, стандартний тариф - 149 $ / місяць і т. д.

1.9.7. Xtract.io

Xtract.io - це гнучка платформа, яка використовує технології AI, ML і NLP.

Її можна налаштувати для скрапінгу і структурування даних сайтів, постів з соцмережах, PDF файлів, текстових документів, історичних даних і електронної пошти.

Особливості Xtract.io:

- скрапінг даних з каталогів, фінансових даних, даних про оренду, геолокаційні даних, даних по компаніях і контактних даних, оглядів і рейтингів.
- Передналаштована система для автоматизації всього процесу отримання даних;
- очищення та валідація даних за заданими правилами;
- експорт в JSON, текст, HTML, CSV, TSV і т. д.
- ротація проксі і проходження капчи для скрапінгу даних в реальному часі.
- гнучка цінова політика.

1.10. Висновок до розділу

Веб-скрапінг надає безліч можливостей для реалізації та покращення власного програмного забезпечення. Він дозволяє добувати необхідні дані

швидко, структуровано та дешево або взагалі безкоштовно. Веб-скрапінг оптимізує та налагоджує роботу програмного забезпечення у багатьох сферах, таких як; машинне навчання, ринкова і конкурентна розвідка, модернізація сайтів і т.д. Саме тому подальша практична робота базується на технологіях веб-скрапінгу.

2. РЕАЛІЗЦАЦІЯ ПРОГРМАНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ОПТИМІЗАЦІЇ ПРОЦЕСУ ВИБОРУ ТОВАРУ В ІНТЕРНЕТІ

В даному розділі повністю описується реалізоване програмне забзпечення здійснення аналізу та пошуку оптимального варіанту закупівель в інтернеті. Визначені функціональні та нефункціональні вимоги. Спроектована архітектура системи. Описані засоби, за допомогою яких розроблено систему, а також інструкції щодо використання.

2.1. Визначення вимог до програмного забезпечення

Вимоги до програмного забезпечення - сукупність тверджень щодо атрибутів, властивостей або якостей програмної системи, яка підлягає реалізації.

До функціональних вимог належать бізнес-вимоги та вимоги користувача.

Бізнес-вимоги описують переваги та основні деталі, які отримає замовник після реалізації проекту.

Сформовано такі бізнес-вимоги:

1. Легке додавання підтримки нових інтернет-магазинів;
2. Невеликий (менше мегабайту) розмір програмного забезпечення.
3. Можливість інтеграції в іншому програмному забезпеченні.

Для задоволення потреб користувачів створюються вимоги користувачів. Вони описують яким чином користувач буде взаємодіяти із системою, та які проблеми будуть вирішуватись застосунком.

Отже, було визначено такі вимоги користувача:

1. Легка взаємодія із ПЗ для відслідковуванян цін та зручний інтуїтивний інтерфейс користувача;
2. Запис результатів у БД для збереження товарів із попередніх запусків ПЗ;
3. Отримання інформації про товар лише з посилання на нього.

4. Можливість перегляду історії змін ціни товару.
5. Можливість перегяду ціни на один товар в декількох магазинах.

Після функціональних вимог доцільно перейти до нефункціональних. Після визначення функціоналу, який повинен бути впроваджений, виділено деякі системні вимоги, а саме:

1. Наявність підключення до мережі інтернет;
2. Встановлений на системі Python 3.6 або вище

2.2. Опис обраних технологій

Для реалізації програмного забезпечення проаналізовано багато засобів та обрано Python, як основну мову програмування, базу даних SQLite, графічний фреймворк PyQt для розроблення інтерфейсу користувача та середовище розробки Visual Studio Code.

2.2.1. Python

Python - високорівнева мова програмування загального призначення, орієнтований на підвищення продуктивності розробника і читання коду. Синтаксис ядра Python мінімалістичний. У той же час стандартна бібліотека включає великий обсяг корисних функцій [5].

Python підтримує структурне, об'єктно-орієнтоване, функціональне, імперативне і аспектно-орієнтоване програмування. Основні архітектурні риси - динамічна типізація, автоматичне керування пам'яттю, механізм обробки виключень, підтримка багатопоточних обчислень, високорівневі структури даних. Підтримується розбиття програм на модулі, які, в свою чергу, можуть об'єднуватися в пакети.

Python - мова програмування, яка активно розвивається, нові версії з додаванням і зміною мовних властивостей виходять приблизно раз в два з половиною роки.

Був обраний через ряд переваг:

Низький поріг входження: людині, знайомій з програмуванням, потрібно дуже мало часу, щоб почати писати на ньому корисні для себе скрипти, а не знайомому - Python дозволяє легко відкрити для себе програмування і спробувати свої сили в ньому.

Добре спроектований: Python увібрав в себе сучасні тенденції в програмуванні «з нуля». Крім того, він динамічно розвивається: процес включення нових конструкцій в мову добре налагоджений, і він продовжує вбирати в себе прийоми функціонального програмування, аспектно-орієнтованого програмування та іншого, залишаючись при цьому назад-сумісним і внутрішньо несуперечливим.

Легко читається синтаксис (в порівнянні з C ++, Perl, PHP): дозволяє легко читати чужий код, розбиратися в давно написаному власному коді. У поєднанні зі сказаним вище це налаштовує творців бібліотек на простоту і логічність інтерфейсів.

Величезна кількість бібліотек з кодом на будь-який випадок: від роботи з таблицями Excel та зображеннями до соціальних мереж.

Переносимість: Python реалізований під усіма поширеними операційними системами і на безлічі архітектур - Windows, Linux, MacOS, навіть на міні-комп'ютерах Arduino. Система залежностей добре продумана, і розгортання застосунків на іншій машині відбувається легко і без непередбачуваних ситуацій.

2.2.2. SQLite

SQLite - система управління реляційними базами даних , що міститься в бібліотеці C. На відміну від багатьох інших систем управління базами даних, SQLite не є двигуном бази даних клієнт-сервер. Скоріше вона вбудована в кінцеву програму [6].

SQLite є популярним вибором як вбудоване програмне забезпечення для баз даних для локального / клієнтського зберігання в прикладному

програмному забезпеченні, такому як веб-браузери. Це, мабуть, найбільш широко розгорнутий двигун бази даних, оскільки його сьогодні використовують декілька широко розповсюджених браузерів, операційних систем та вбудованих систем (таких як мобільні телефони). SQLite має прив'язку до багатьох мов програмування.

SQLite підтримує динамічне типізування даних. Кожне значення в будь-якому полі будь-якого запису може бути будь-якого типу, незалежно від типу, зазначеного при оголошенні полів таблиці. Зазначений при оголошенні поля тип зберігається для довідки в його вихідному написанні, і використовується в якості основи для вибору переваг (так зване «type affinity»: це підхід, що рідко зустрічається в інших СУБД) при виконанні неявних перетворень типів на підставі схожості цієї назви типу на що-небудь, знайоме SQLite. В цей алгоритм зашитий великий перелік варіантів назв типів даних, які практикуються в інших СУБД. Якщо безпечного перетворення записуваного значення в бажаний тип не виходить, SQLite записує значення в його початковому вигляді. Для отримання значень з бази є ряд функцій для кожного з типів, і якщо тип значення, що зберігається не відповідає запитуваному, воно теж, по можливості, перетворюється.

2.2.3. Sqlalchemy

SQLAlchemy - це програмна бібліотека для Python для роботи з реляційними СУБД із застосуванням технології ORM. Служить для синхронізації об'єктів Python і записів у реляційну базу даних. SQLAlchemy дозволяє описувати структури баз даних і способи взаємодії з ними на мові Python без використання SQL. Бібліотека була випущена в лютому 2006 під ліцензією відкритого ПЗ MIT [8].

Працює із базами даних: MySQL, PostgreSQL, SQLite, Oracle та інші, між якими можна перемикатися зміною конфігурації.

Основні можливості Sqlalchemy:

- Використання ORM не є обов'язковим.
- Стандартизована архітектура.
- Можливість використовувати SQL, що написаний вручну.
- Підтримка транзакцій.
- Створення запитів з використанням функцій і виразів Python.
- Модульність і розширюваність.
- Додаткова можливість роздільного визначення об'єктного відображення і класів.
- Підтримка складових індексів.
- Підтримка відносин між класами, в тому числі «один-до-багатьох» і «багато-до-багатьох».
- Підтримка об'єктів, що посилаються на себе.
- Попередня і наступна обробка даних (параметрів запиту, результату)

2.2.4. PyQt

PyQt - бібліотека Python, яка реалізує фреймворк для розробки графічних інтферфейсів QT. PyQt - це відкрите програмне забезпечення, розроблене британською компанією Riverbank Computing. Він доступний за аналогічними умовами як версії Qt. PyQt підтримує Microsoft Windows, а також різні системи UNIX, включаючи Linux та MacOS (або Darwin) [9].

PyQt реалізує близько 440 класів та понад 6000 функцій та методів для розробки графічного інтерфейсу, а також включає:

- існуючий набір віджетів графічного інтерфейсу;
- стилі віджетів;
- доступ до баз даних за допомогою SQL (ODBC, MySQL, PostgreSQL, Oracle);
- QScintilla, заснований на Scintilla віджет текстового редактора;

- підтримку інтернаціоналізації (i18n);
- парсер XML;
- підтримку SVG;
- інтеграцію з WebKit, движком рендеринга HTML;
- підтримку відтворення відео і аудіо.

2.2.5. Visual Studio Code

Visual Studio Code - редактор вихідного коду, розроблений Microsoft для Windows, Linux і macOS. Позиціонується як «легкий» редактор коду для кроссплатформенної розробки веб і хмарних застосунків. Включає в себе відладчик, інструменти для роботи з Git, підсвічування синтаксису і засоби для рефакторингу.

2.3. Життєвий цикл програмного забезпечення

Життєвий цикл програмного забезпечення - це період часу, який починається з моменту прийняття рішення про створення програмного продукту і закінчується в момент його повного вилучення з експлуатації. Цей цикл - процес побудови і розвитку ПЗ.

Для розробки програмного забезпечення здійснення аналізу та пошуку оптимального варіанту закупівель в інтернеті було обрано каскадну модель життєвого циклу. Каскадна модель - модель процесу розробки програмного забезпечення, життєвий цикл якої виглядає як потік, що послідовно проходить фази аналізу вимог, проектування. реалізації, тестування, інтеграції та підтримки. Було обрано саме каскадну модель, адже вимоги до даного типу програмного забезпечення були зразу сформовані чітко, а етапи створення йшли послідовно без повернень на минулі кроки.

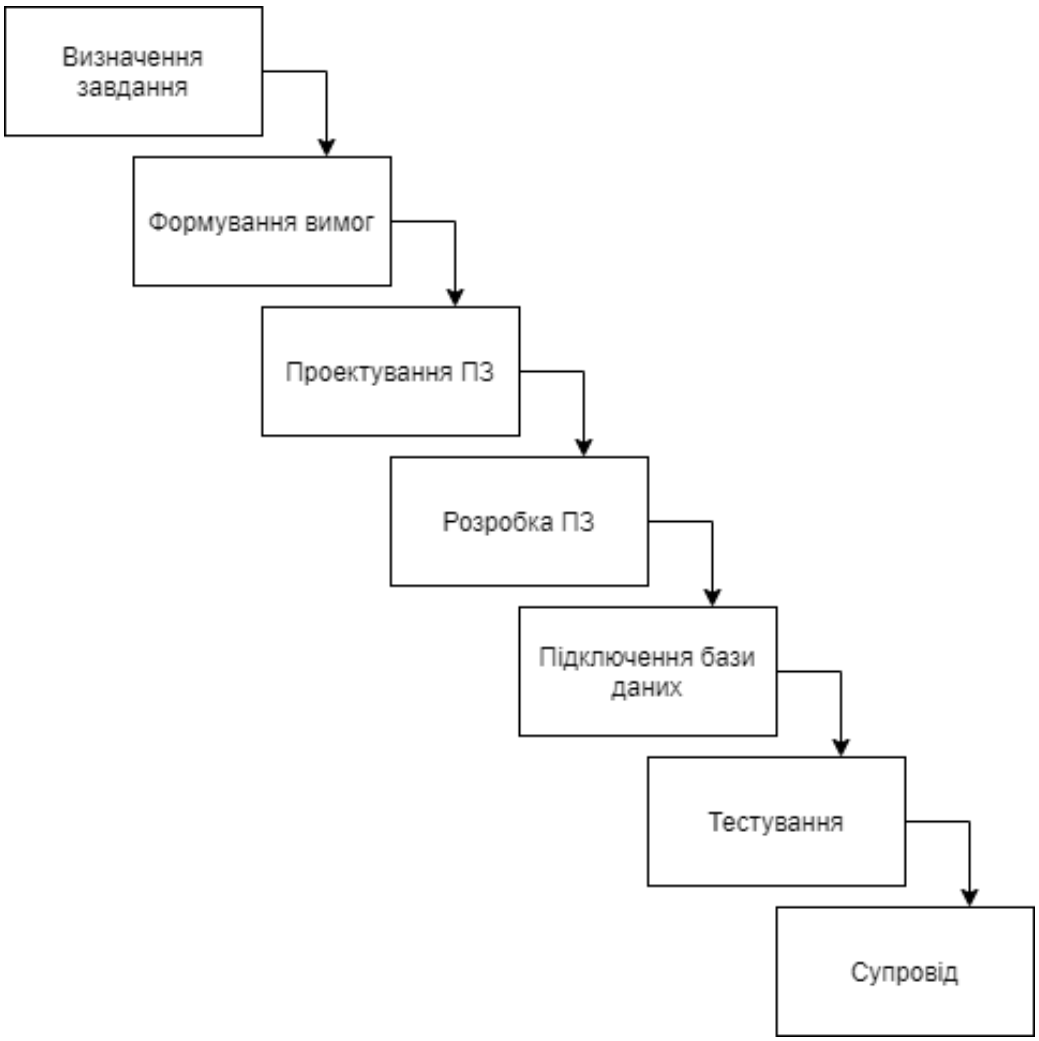

Рис. 2.1. Модель життєвого циклу розробленого програмного забезпечення

2.4. Загальний опис проекту

Програмне забезпечення здійснення аналізу та пошуку оптимального варіанту закупівель в інтернеті було реалізоване як ПЗ робочого столу. Воно має простий інтуїтивний інтерфейс, який дозволить із першого запуску швидко розібратись та зручно користуватись ним.

Для роботи із розробленим програмним забезпеченням потрібне підключення до інтернету, адже ПЗ в реальному часу моніторить ціни на товари.

Розроблене програмне забезпечення працює наступним чином:

1. Користувач вводить URL посилання на товар.

2. Програмне забезпечення визначає із посилання, чи даний інтернет-магазин підтримується (у випадку, якщо ні – користувачу виводиться відвоідне повідомлення).
3. Із інтернет-магазину зчитується назва та ціна товару.
4. Дані про товар (унікальний ідентифікатор, назва, URL посилання, актуальна ціна, магазин, дата зміни ціни) записуються в базу даних.
5. Користувачу виводиться повідомлення, що товар був успішно доданий до моніторингу.
6. Дані про товар відображаються на користувацькому інтерфейсі.
7. Коли ціна на товар в інтернет-магазині падає, користувачу виводиться відповідне повідомлення.
8. Двічі нажавши на товар, можна отримати інформації про нього, а саме: ціну на товар у декількох підтримуваних інтернет-магазинах, а також історію змін ціни з початку відслідковування товару.
9. У разі необхідності, можна видалити товар із моніторингу.

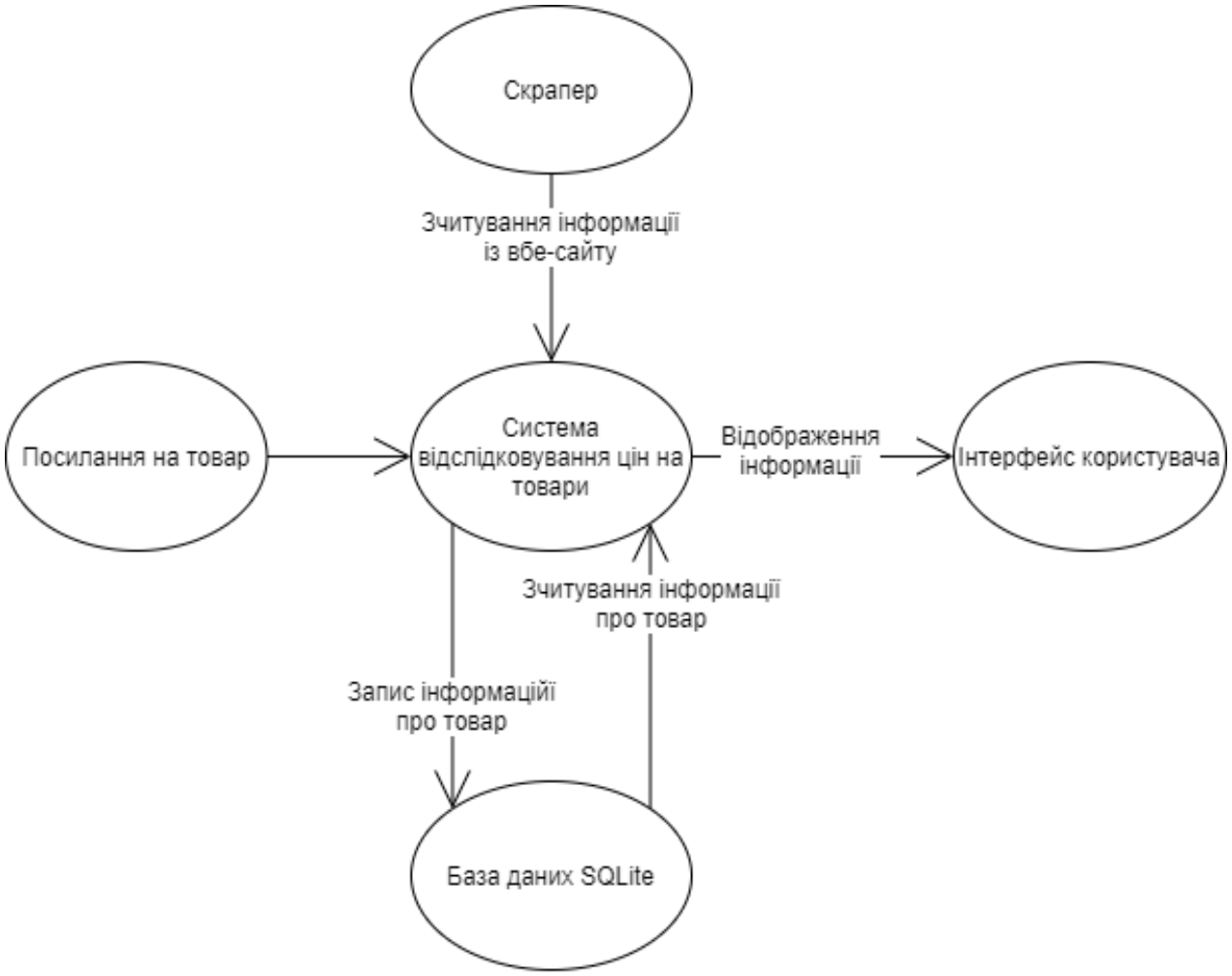

Рис.2.2. Структура ПЗ здійснення аналізу та пошуку оптимального варіанту закупівель в інтернеті

2.5. Архітектура програмного забезпечення

Для зручності система була розділена на декілька модулів, кожен з яких відповідає за окремий функціонал.

Модуль scrapper зберігає в собі клас Scrapper який реалізує основну логіку зчитування даних із інтернет-магазинів. У даному класі знаходиться функціонал, який визначає чи підтримується даний інтернет магазин програмним забезпеченням, методи для вичитування даних про товар, а також XPath шляхи до елементів, по яким зчитуються дані.

Модуль database зберігає клас Database у якому знаходяться методи для

роботи із базою даних, а саме: записування товару в базу, оновлення товару, отримання товарів, видалення товару, а також методи для стоверння підключення до бази даних та створення бази даних із товарами, якщо вона відсутня.

Модуль main_window зберігає класи MainWindow, ProductPage та CustomDialog, він є вхідною точкою для старту роботи програмного забезпечення. У класі MainWindow із використанням PyQT реалізований графічний користувацький інтерфейс, а також методи для моніторингу та збереження інформації про товари за допомогою модулів scrapper та database. Клас CustomDialog є реалізацією всіх вікон-сповіщень, які отримує користувач.

Також в проекті знаходиться файл-ресурс, який задає кольори розроленого програмного забезпечення, а також файл-іконка головного вінка.

База даних записується у тимчасовий файл database.db, тож у випадку його видалення програма створить БД заново при наступному запуску.

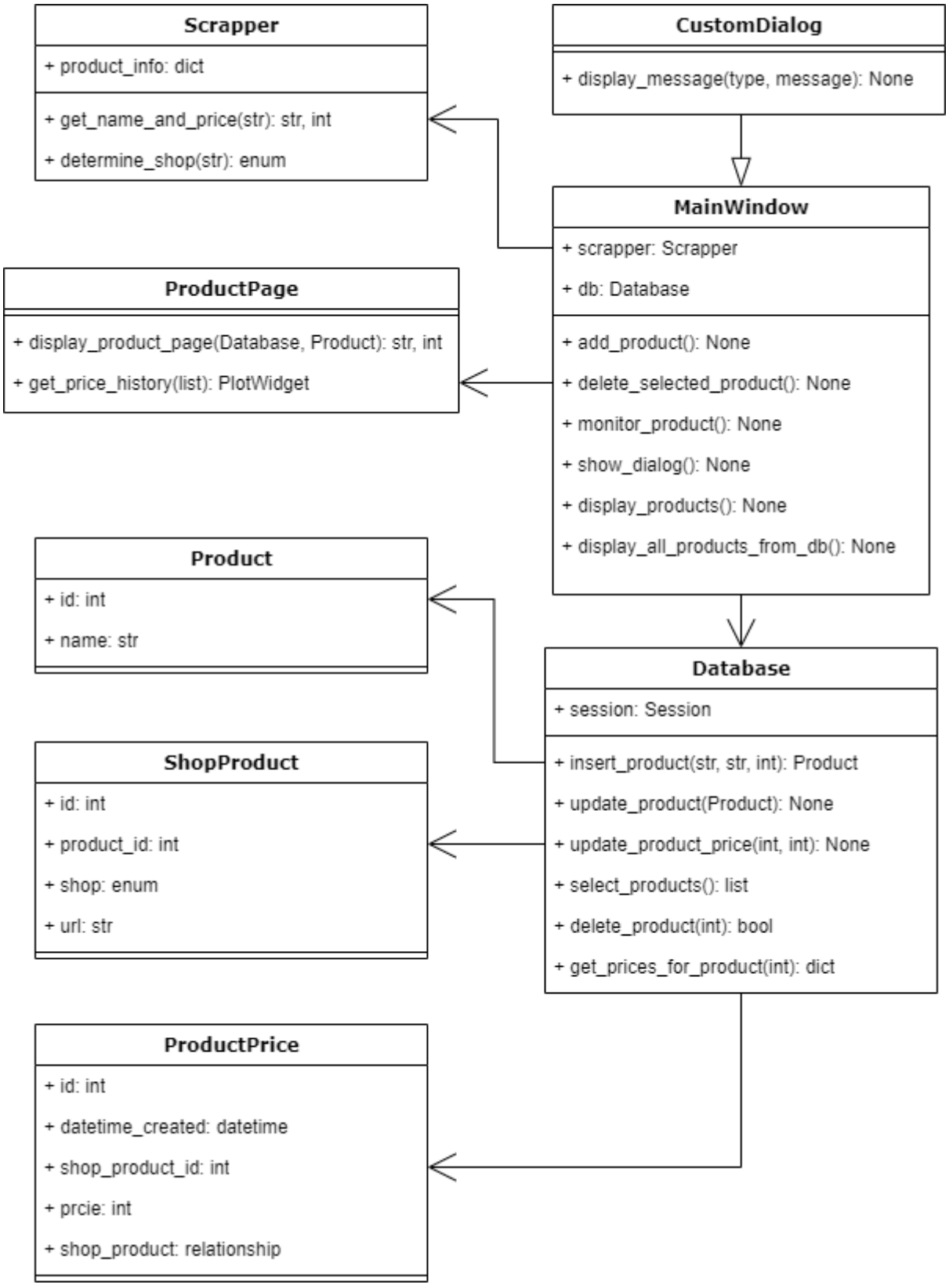

Рис. 2.3. Діаграма класів

2.6. Структура бази даних

Структура бази даних складається із трьох таблиць та одного Enum типу.

Таблиця Product використовується для зберігання основної сутності – товару, і має два поля: id(типу int, primary key) та name(типу string).

Таблиця ShopsProduct використовується для класифікації товарів по різним магазинам, це надає можливість відслідковувати ціни на один і той самий товар на різних веб-сайтах. Складається із таких полів: id(типу int, primary key), product_id (типу int, foreign key що посилається на id в таблиці Product), shop(типу enum Shops) та url(типу string).

Таблиця ProductPrice використовується для зберігання цін на товари у різних інтернет-магазинах та відслідковування історії зміни цін. Складається із таких полів: id(типу int, primary key), shop_product_id (типу int, foreign key що посилається на id в таблиці ShopsProduct), datetime_created(типу datetime) та price(типу int).

Вищезазначений тип enum Shops не присутній в БД фізично, адже в SQLite не існує типу ENUM і на справді в полі shop таблиці ShopsProduct зберігаються стрічкові значення, але за допомогою sqlalchemy можна описати потрібний enum в коді програми і зручно користуватися ним, як звичайним переліченням.

Така структура бази даних дозволяє писати зручні та ефективні SQL запити із максимальною оптимізацією швидкості доступу до даних в БД.

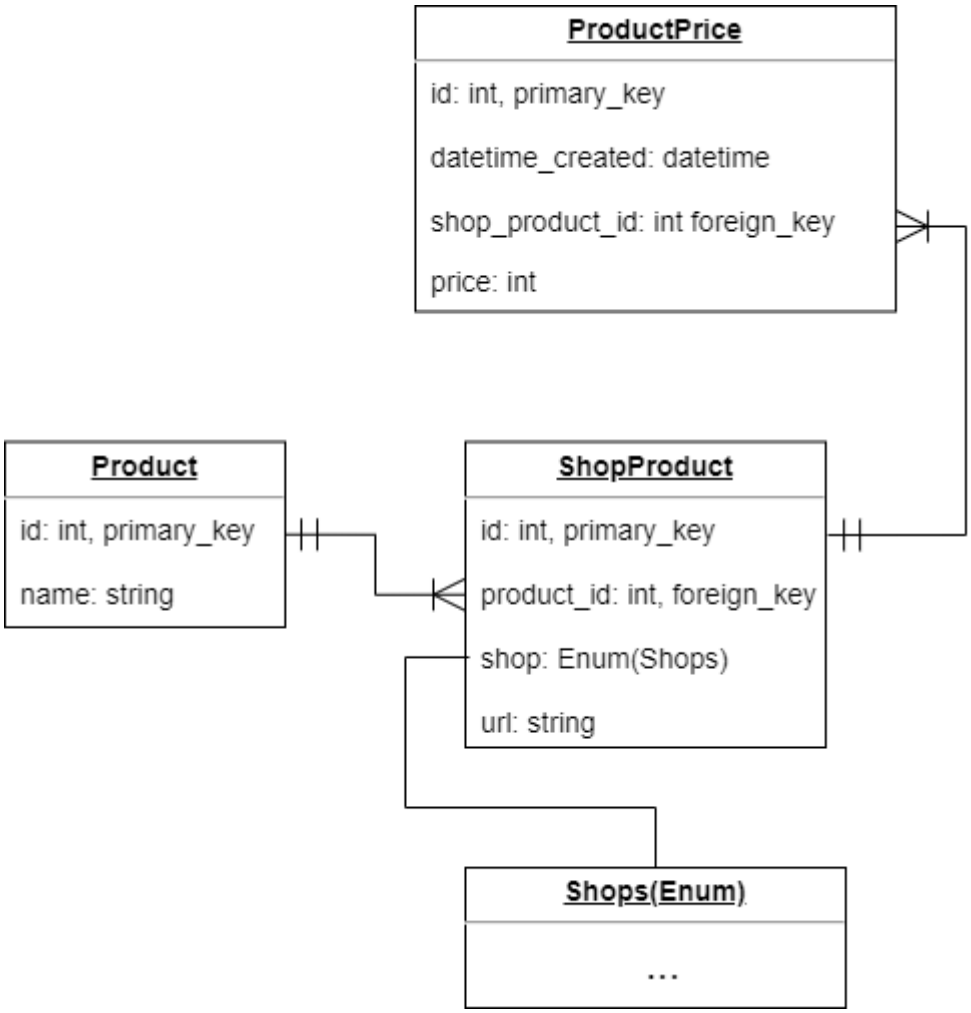

Рис.2.4. Діаграма БД

2.7. Інтеграція в іншому ПЗ

Окрім використання через користувацький інтерфейс, розроблену систему можна інтегрувати в іншому програмному забезпеченні написаному на мові програмування Python. Для цього потрібно встановити розроблене ПЗ через pip (package installer for python).

Перейшовши в директорію із файлом price_tracker-0.0.1-py3-none-any.whl У командному рядку потібно ввести:

pip install price_tracker-0.0.1-py3-none-any.whl

Встановлення самого файлу проходить майже моментально, адже файл із розробленим ПЗ не займає багато пам’яті, але встановлення залежностей може зайняти деякий час.

```
Processing d:\simple_pyside_base\price_tracker-0.0.1-py3-none-any.whl
Installing collected packages: price-tracker
Successfully installed price-tracker-0.0.1
```

Рис. 2.5. Процес встановлення розробленого ПЗ

.whl – це формат файлів для інсталювання у Python.

Після встановлення можна імпортувати модулі та методи із основного модулю price_tracker наприклад:

from price_tracker import scrapper

from price_tracker.scrapper import get_name_and_price

from price_tracker.scrapper import determine_shop

Дана інтеграція буде працювати лише із програмним забезпеченням написаним на Python 3.6 і вище.

2.8. Опис аналогів та порівняння їх із розробленим ПЗ

2.8.1. CamelCamelCamel

CamelCamelCamel - це один з найпопулярніших способів отримання знижок та відстеження цін на продукти Amazon. Його можна встановити як розширення для браузера (Chrome і Firefox), щоб отримати доступ до функцій сайту, не виходячи з Amazon, або скопіювати посилання з Amazon та використати його на веб-сайті відслідковувача цін. CamelCamelCamel показує, як ціна товару зросла або знизилася з часом, як змінилися ціни через Amazon безпосередньо та коливання у сторонніх продавців [14].

Якщо зареєструватись у безкоштовному акаунті, то можна налаштувати сповіщення, щоб сайт повідомляв електронною поштою, коли ціна товару опуститься нижче певної суми. Також даний відслідковувач надає можливість

побачити попередні історичні максимуми та мінімуми для складання картини, коли ціна знизиться наступного разу.

2.8.2. Honey

Розширення Honey, яке доступне в Chrome, порівнює ціни від цілого ряду роздрібних продавців, включаючи Amazon. Він дозволяє створити список розсилки, який повідомляє вас про зниження цін на товар у вашому списку. Також Honey знаходить промо-коди та автоматично застосовує їх до вашої картки під час покупки на веб-сайті продавця. Honey доступний у Chrome, Safari, Firefox, Edge та Opera.

Користувачі Honey можуть також збирати Honey Gold, відсоток від онлайн-покупок, здійснених у понад 4500 магазинах. Бонуси можна обміняти на подарункові картки в магазинах, таких як Amazon, Groupon, Macy та інші.

2.8.3. SlickDeals

SlickDeals добре відомий тим, що пропонує знижки, купони та інші угоди з усієї мережі в Інтернеті, таких їх веб-відстежувач підходить для перегляду цін на будь-який товар в Amazon і на деяких інших популярних торгових сайтів, таких як Newegg, Gamestop, Chwey, Home Depot та інші.

Сервіс не надає детальної історії цін, як деякі інші в списку, але він відстежує ціну товару і надішле вам електронний лист, якщо ціна опуститься нижче межі, яка була встановлена. Також можна в будь-який час повернутися до відстежувача цін, увійти у свій обліковий запис і побачити всі ваші відстежувані товари на одному екрані.

2.8.4. Wikibuy

Wikibuy порівнює ціни від інших продавців, коли купується товар в Amazon. Розширення сповістить про те, що товар, який шукається, дешевший десь в іншому місці, і пропонує підсумок історії цін, орієнтовний час доставки та загальну ціну, включаючи податок та доставку. Якщо можна застосувати

купон, він також додасть його. Коли ціна на продукт, який переглядали, знизиться, Wikibuy повідомить про це. Розширення доступне для Chrome, Firefox, Edge та Safari, а додаток iOS дозволяє сканувати штрих-коди на шукані продукти для порівняння цін зі свого телефону.

Під час покупки можна заробляти кредити на деяких веб-сайтах, таких як Walmart та eBay, з якими співпрацює Wikibuy. Потім можна обіняти кредити на подарункові картки або вносити їх на покупки через сайт Wikibuy.

2.8.5. Earny

Поряд із відстеженням цін та сповіщеннями про їх зниження, Earny дає повідомляє, чи можливі зниження цін для певного товару, який ви переглядаєте. Він також сканує електронні листи, для пошуку раніше придбані продукції, щоб відстежувати товари від Amazon та інших інтернет-магазинів, таких як Walmart, Nike тощо.

Однією з перваг Earny є те, що можна отримати повернення грошей за вже придбані товари, якщо його ціна знизиться. Покупець отримає відшкодування різниці за те, що вже купили, якщо Earny побачить, що його ціна знизилася.

Зазвичай ця послуга не є абсолютно безкоштовною. Це 5 доларів на місяць або 40 доларів США за рік. І якщо користувач отримає сповіщення про зниження ціни на щось, що вже купив, Earny бере 25% гонорару і повертаэ решту.

Також є можливість завантажити додаток для Android та iOS або додати його у свій браузер Chrome, при цьому додатки будуть безкоштовними, якщо не потрібно мати доступ до преміум-функцій.

2.8.6. CheapShark

У той час як більшість інших засобів фокусуються на таких продуктах, як електроніка, одяг, побутові товари та техніка, CheapShark позиціонує себе як

варіант для геймерів, які хочуть заощадити гроші на відеоіграх. CheapShark поєднує в собі всі засоби, щоб отримати знижки на ігри в одному місці - один легкодоступний каталог та базу даних поточних цін, із сайтів, включно Steam, GoG, Green Man Gaming, GameStop, Amazon та інше. CheapShark обмежений у відеоіграх, але він дозволяє шукати по назві ігри не просто, щоб побачити, скільки коштуватиме придбання, але чи взагалі можна її придбати - і який продавець має її в наявності. Також можна фільтрувати за іграми, які зараз продаються у вибраних магазинах.

Коли йде пошук конкретної гри, CheapShark повідомляє, якою була найдешевша ціна, коли гра потрапила до цієї ціни, а також дає можливість придбати її у роздрібного продавця, або зареєструватися на повідомлення про ціну, якщо вона знову знизиться.

2.8.7. Порівняння розробленого ПЗ

На відміну від розробленого ПЗ більшість аналогів фокусується лише на відслідковуванні цін на Amazon або на інтернет-магазинах у Сполучених Штатах. Отримане у ході даної курсової роботи програмне забезпечення не прив’язане до локалізації, чи країни інтернет-магазину.

У ході роботи було виявлено, що для Українського ринку майже не існує автоматизованих засобів для відслідковування цін на товари, тому легке додавання підтримки нових магазинів та моніторинг товарів у вітчизняних продавців робить розроблене ПЗ унікальним та ефективним засобом для відстеження зміни ціни.

Недоліком, на даний момент, розробленого ПЗ у порівнянні з аналогами є те, що воно розроблене лише як програмне забезпечення робочого столу, коли інші відслідковувачі мають і розширення для браузерів, і веб-сайти, і мобільні застосунки. Але вище наведені засоби розроблялись цілими командами або компаніями, тому вони мали більше ресурсів для реалізації своїх ідей на різних

платформах. Уподальшому розроблену систему можна перевести і на інші платформи.

2.9. Інструкція користувачу програми

Інтерфейс користувача має наступний вигляд:

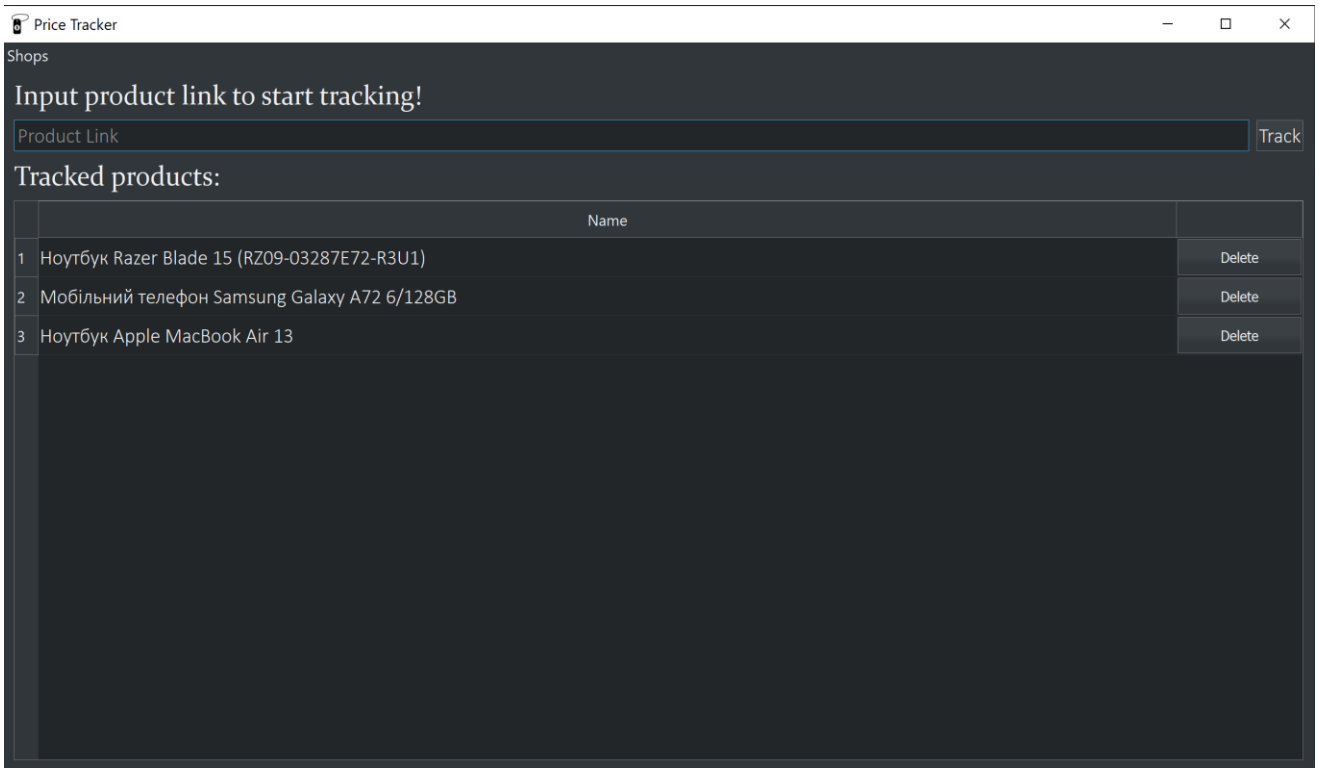

Рис.2.6. Інтерфейс користувача

Список доступних на даний момент інтернет магазинів можна переглянути у вкладці Shops

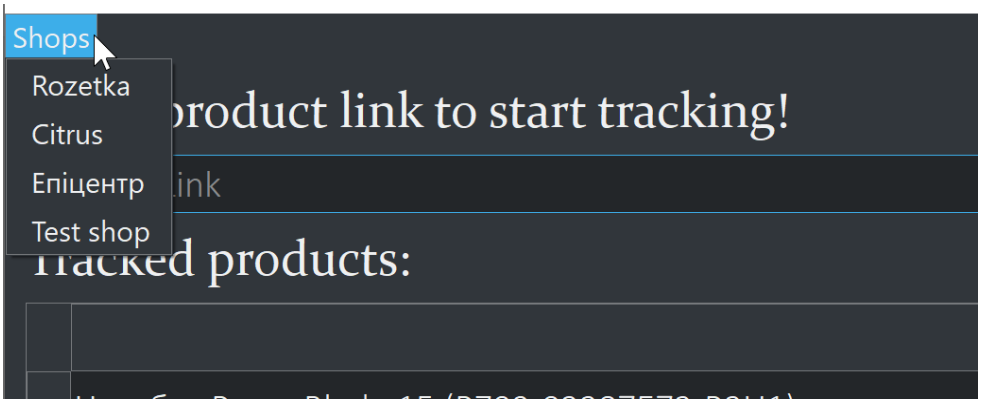

Рис.2.7. Список доступних інтернет магазинів

Для того, щоб почати відслідковувати ціну товару потрібно виконати наступні дії:

1) Заповнити поле Product Link - посилання на товар скопйоване із браузера;
2) Натистути кнопку Track.

Після введення даних про товар, програмне забезпечення зчитує актуальну ціну та назву товару із інтернет магазину та відкриває вікно із інформацією, що даний товар був доданий до моніторингу.

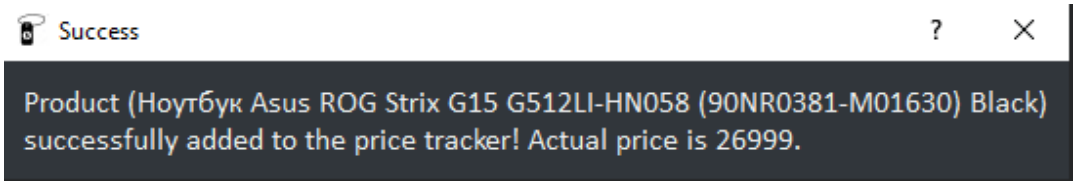

Рис.2.8. Повідомлення про успішне додавання товару до моніторингу

Натиснувши двічі на продукт можна отримати детальну інформацію про нього:

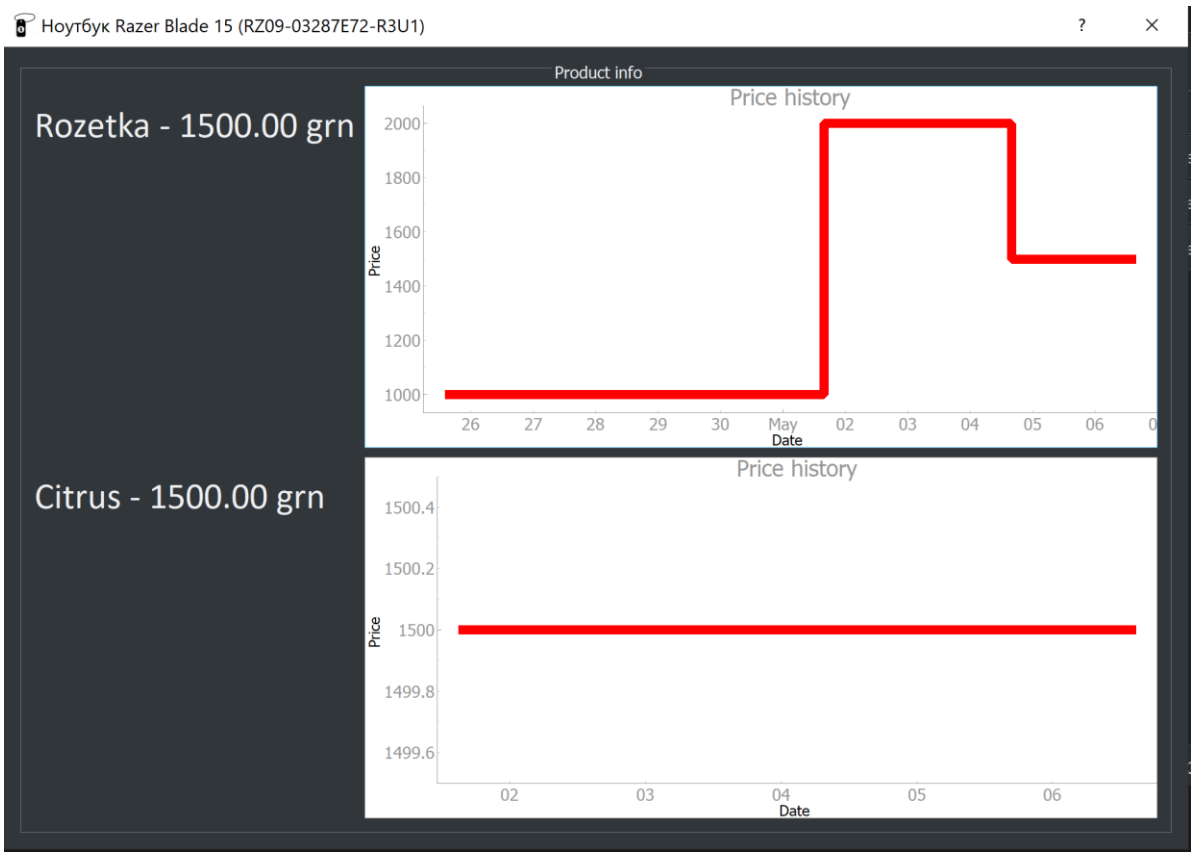

Рис. 2.9. Сторінка товару

В лівій частині вікна відображаються магазини та ціна на товар в даний момент. В правій частині відображаються графіки як ціна змінювалась починаючи із додавання товару в моніторинг до теперішнього моменту.

Графіки ціни не статичні, тобто їх можна рухати та збільшувати масштаб значення дати або ціни для більшої наочності:

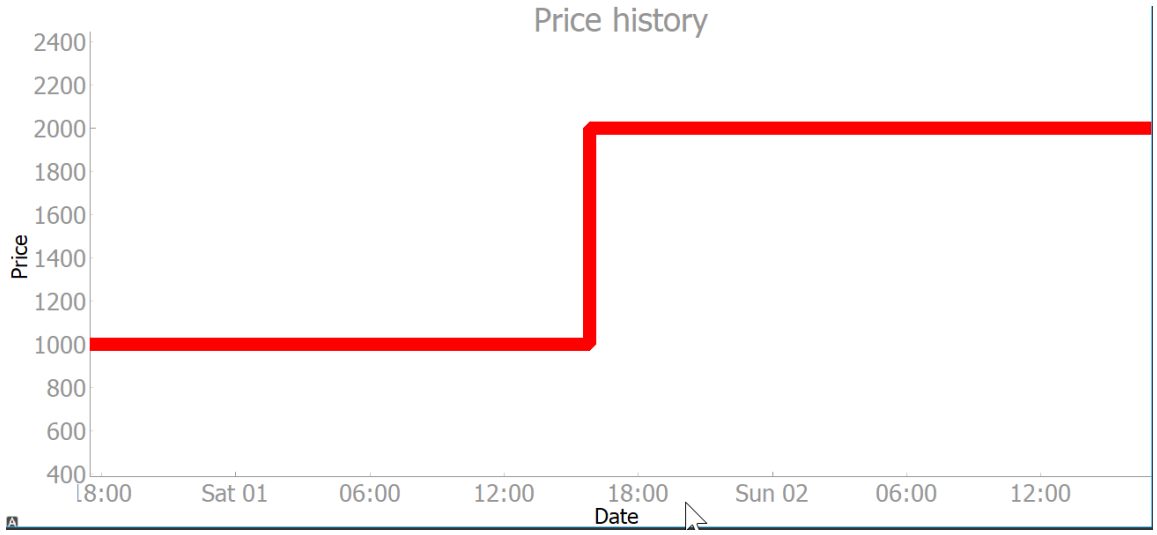

Рис.2.10. Графік зміни ціни

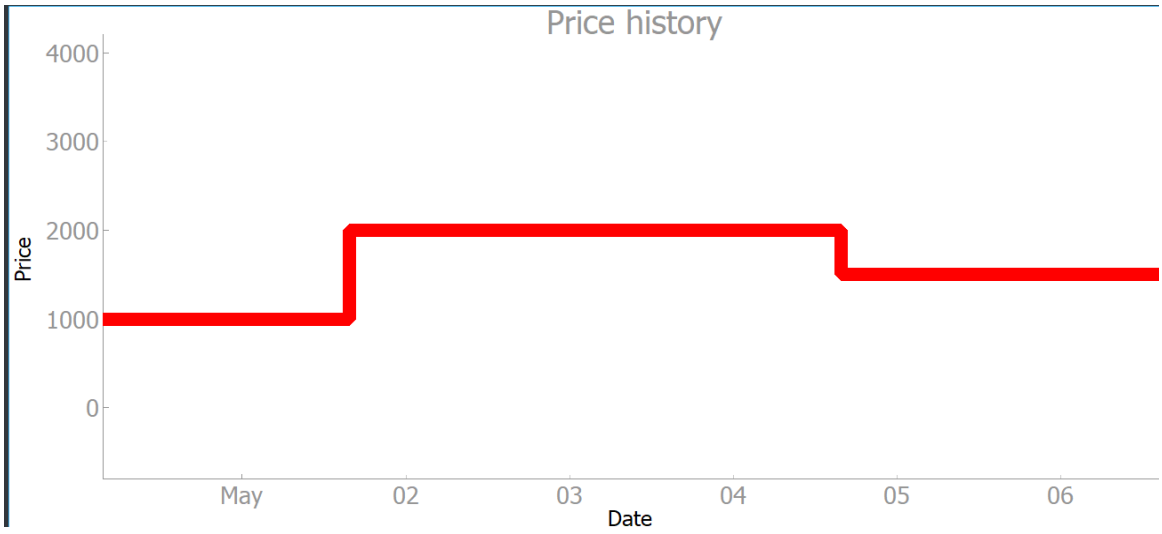

Рис. 2.11. Той самий графік ціни із зміненим масштабом

Коли ціна на товар знизиться, користувачу відкриється інформаційне вікно із повідомленням, що ціна змінилась.

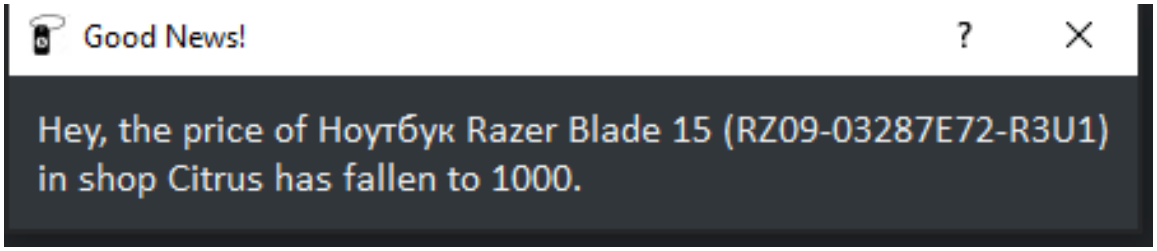

Рис.2.12. Повідомлення про зниження ціни

Якщо є потреба видалити товар із моніторингу, то це можна зробити натиснувши кнопку Delete біля потрібного товару.

2.10. Висновок до розділу

В результаті роботи сформовано та виконано функціональні та нефункціональні вимоги до системи, побудовано архітектуру системи та реалізовано програмне забезпечення здійснення аналізу та пошуку оптимального варіанту закупівель в інтернеті.

ВИСНОВОК

В результаті виконання роботи досліджено процес зчитування даних із веб-сайтів. Проаналізовано види, засоби та проблеми, які виникають під час реалізації програмного забезпечення здійснення аналізу та пошуку оптимального варіанту закупівель в інтернеті. Визначено, що для українського ринку майже не існує аналогів розробленого програмного забезпечення. Більшість існуючих засобів орієнтовані на інтернет-магазини Сполучених Штатів Америки. Також описано переваги та недоліки у порівнянні із схожими засобами.

Після досліджень сформовано та виконано функціональні та нефункціональні вимоги до програмного забезпечення, побудовано архітектуру та реалізовано ПЗ для здійснення аналізу та пошуку оптимального варіанту закупівель в інтернеті, яке дозволяє відслідковувати ціну на товари у різних інтернет-магазинах та може бути інтегроване у інше програмне забезпечення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Опис веб-скрапінгу [Електронний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/Web_scraping

2. Технології веб-скрапінгу [Електронний ресурс]. – Режим доступу: https://limeproxies.com/blog/top-10-web-scraping-techniques/

3. Способи обходу блокування веб-скрапінгу [Електронний ресурс]. – Режим доступу: https://tproger.ru/translations/web-scraping-without-getting-blocked/

4. Сучасні сервіси для веб-скрапінгу [Електронний ресурс]. – Режим доступу: https://tproger.ru/translations/luchshie-servisy-dlja-veb-skrapinga-dannyh-top-7/

5. Офіційна документація Python [Електронний ресурс]. – Режим доступу: https://docs.python.org/3/

6. Інструкції щодо використання SQLite [Електронний ресурс]. – Режим доступу: https://www.sqlitetutorial.net/sqlite-python/

7. Офіційна документація SQLite [Електронний ресурс]. – Режим доступу: https://docs.sqlalchemy.org/en/13/orm/

8. Офіційна документація orm sqlalchmey [Електронний ресурс]. – Режим доступу: https://sqlite.org/index.html

9. Офіційна документацій PyQT [Електронний ресурс]. – Режим доступу: - https://www.riverbankcomputing.com/static/Docs/PyQt5/

10.Розробка графічного інтерфейсу за допомогою Python [Електронний ресурс]. – Режим доступу: https://realpython.com/learning-paths/python-gui-programming/

11.Побудова графіків із PyQTgrapgh [Електронний ресурс]. – Режим доступу: https://www.mfitzp.com/tutorials/plotting-pyqtgraph/

12.Побудова графіків із часовими проміжками на pyqtgrapgh [Електронний ресурс]. – Режим доступу: https://www.mfitzp.com/tutorials/plotting-pyqtgraph/

13. Офіційна документація pyqtgrapgh [Електронний ресурс]. – Режим доступу: https://pyqtgraph.readthedocs.io/en/latest/index.html

14.Засоби для відстежування цін [Електронний ресурс]. – Режим доступу: https://lifehacker.com/five-best-price-tracking-tools-1692745053

ДОДАТКИ

Додаток А – програмна реалізація основних модулів

Модуль main_window:

```
import os
import time
import datetime as dt

from PyQt5.QtWidgets import QApplication
from PyQt5.QtWidgets import QWidget
from PyQt5.QtWidgets import QLabel, QAction
from PyQt5.QtWidgets import QMainWindow, QGroupBox, QFormLayout
from PyQt5.QtWidgets import QTableWidget, QTableWidgetItem, QHeaderView
from PyQt5.QtWidgets import QHBoxLayout, QVBoxLayout, QGridLayout
from PyQt5.QtWidgets import QSizePolicy, QAbstractScrollArea
from PyQt5.QtWidgets import QLineEdit
from PyQt5.QtWidgets import QPushButton
from PyQt5.QtWidgets import QDialog, QDialogButtonBox

from PyQt5 import QtCore
from PyQt5.QtCore import QTimer
from PyQt5.QtCore import QSize

from PyQt5.QtGui import QIntValidator
from PyQt5.QtGui import QFont
from PyQt5.QtGui import QStandardItemModel
from PyQt5.QtGui import QIcon
from PyQt5.QtCore import QFile, QTextStream
from sqlalchemy import func
import pyqtgraph as pg

from gui import breeze_resources

from backend import database
from backend.database import Database, Product
from backend.scrapper import Scrapper


class MainWindow(QMainWindow):
    def __init__(self, parent=None):

        super().__init__(parent)
        self.scrapper = Scrapper()
        self.db = Database()

        self._create_icon()
        self._create_shops_list()
```

```
        self.setWindowTitle("Price Tracker")
        # self.resize(2600, 1400)
        self.resize(1400, 800)
        self.main_lt = QVBoxLayout()
        self.input_product_lt = QHBoxLayout()

        self._create_input_header()
        self._creat_url_input()
        self._create_track_btn()
        self.main_lt.addLayout(self.input_product_lt)

        self._create_products_header()
        self._create_products()
        self.products = self.db.select_products()
        self.monitor_product()
        self.display_all_products_from_db()

        self.main_widget = QWidget()
        self.main_widget.setLayout(self.main_lt)
        self.setCentralWidget(self.main_widget)

        self.timer = QTimer(self)
        self.timer.start(15000)  # 15 seconds
        self.timer.timeout.connect(self.monitor_product)

    def add_product(self):
        url = self.url_input.text()
        if not url:
            self.show_dialog("Warning", "Product URL shouldn't be blank!")
        else:
            try:
                name, price = self.scrapper.get_name_and_price(url)
                # id = len(self.products) + 1
                product = self.db.insert_product(name, url, price)
                self.display_product(product)
                self.products.append(product)
                self.show_dialog("Success", f"Product ({name})\nsuccessfully added
to the price tracker! Actual price is {price}.")
            except Exception as ex:
                self.show_dialog("Error", str(ex))

    def delete_selected_product(self):
        delete_btn = self.sender()
        if delete_btn:
            try:
                row = self.products_tbl.indexAt(delete_btn.pos()).row()
                self.db.delete_product(self.products[row].id)
```

```
                self.products.pop(row)
                self.products_tbl.removeRow(row)
            except Exception as ex:
                self.show_dialog("Error", str(ex))

    def monitor_product(self):
        for product in self.products:
            try:
                product_and_prices = self.db.get_prices_for_product(product.id)
                for shop_product_id, prices in product_and_prices.items():
                    last_price = prices[-1]
                    _, price = self.scrapper.get_name_and_price(last_price.shop_pro
duct.url)
                    if price < last_price.price:
                        shop_name = last_price.shop_product.shop.name.capitalize()
                        self.show_dialog("Good News!", f"Hey, the price of {product
.name}\nin shop {shop_name} has fallen to {price}.")
                        self.db.update_product_price(shop_product_id, price)
                    elif price > last_price.price:
                        self.db.update_product_price(shop_product_id, price)
            except Exception as ex:
                self.show_dialog("Error", str(ex))

    def show_dialog(self, type, text):
        dlg = CustomDialog(self)
        dlg.display_message(type, text)
        dlg.exec_()

    def display_product(self, product):
        rowPosition = self.products_tbl.rowCount()
        self.products_tbl.insertRow(rowPosition)
        self.products_tbl.setItem(rowPosition, 0, self._create_readonly_item(produc
t.name))
        self.delete_row_btn = QPushButton("Delete")
        self.delete_row_btn.clicked.connect(self.delete_selected_product)
        self.products_tbl.setCellWidget(rowPosition, 1, self.delete_row_btn)

    def display_all_products_from_db(self):
        for product in self.products:
            self.display_product(product)

    def _create_readonly_item(self, title):
        item = QTableWidgetItem(title)
        item.setFlags(item.flags() ^ QtCore.Qt.ItemIsEditable)
        return item

    def _create_icon(self):
```

```
        scriptDir = os.path.dirname(os.path.realpath(__file__))
        app_icon = QIcon()
        app_icon.addFile(os.path.join(scriptDir, '256x256.png'), QSize(256, 256))
        self.setWindowIcon(app_icon)

    def _create_shops_list(self):
        menubar = self.menuBar()
        available_shops = menubar.addMenu("Shops")
        rozetka = QAction("Rozetka", self)
        citrus = QAction("Citrus", self)
        epicentr = QAction("Епіцентр", self)
        test = QAction("Test shop", self)
        available_shops.addAction(rozetka)
        available_shops.addAction(citrus)
        available_shops.addAction(epicentr)
        available_shops.addAction(test)

    def _create_input_header(self):
        self.intput_header_lbl = QLabel("Input product link to start tracking!")
        self.intput_header_lbl.setFont(QFont("Constantia", 16))
        self.main_lt.addWidget(self.intput_header_lbl)

    def _creat_url_input(self):
        self.url_input = QLineEdit()
        sizePolicy = QSizePolicy(QSizePolicy.Expanding, QSizePolicy.Fixed)
        sizePolicy.setHorizontalStretch(30)
        self.url_input.setSizePolicy(sizePolicy)
        self.url_input.setAutoFillBackground(False)
        self.url_input.setClearButtonEnabled(True)
        self.url_input.setPlaceholderText("Product Link")
        self.url_input.setFont(QFont("Calibri Light", 12))
        self.input_product_lt.addWidget(self.url_input)

    # Deprecated
    def _create_exp_price_input(self):
        self.exp_price_input = QLineEdit()
        self.exp_price_input.setClearButtonEnabled(True)
        onlyInt = QIntValidator()
        self.exp_price_input.setValidator(onlyInt)
        self.exp_price_input.setPlaceholderText("Expected Price")
        self.exp_price_input.setFont(QFont("Calibri Light", 12))
        self.input_product_lt.addWidget(self.exp_price_input)

    def _create_track_btn(self):
        self.track_btn = QPushButton("Track",)
        self.track_btn.clicked.connect(self.add_product)
        self.track_btn.setFont(QFont("Calibri Light", 12))
```

```
        self.input_product_lt.addWidget(self.track_btn)

    def _create_products_header(self):
        self.products_header_lbl = QLabel("Tracked products:")
        self.products_header_lbl.setFont(QFont("Constantia", 16))
        self.main_lt.addWidget(self.products_header_lbl)

    def _create_products(self):
        self.products_tbl = QTableWidget(self)
        self.products_tbl.setColumnCount(2)
        item = QTableWidgetItem("Name")
        self.products_tbl.setHorizontalHeaderItem(0, item, )
        item = QTableWidgetItem("")
        self.products_tbl.setHorizontalHeaderItem(1, item, )
        header = self.products_tbl.horizontalHeader()
        header.setSectionResizeMode(0, QHeaderView.Stretch)
        header.setFont(QFont("Calibri", 14))
        self.products_tbl.setFont(QFont("Calibri Light", 12))
        self.products_tbl.doubleClicked.connect(self._on_click)
        self.main_lt.addWidget(self.products_tbl)

    @QtCore.pyqtSlot(QtCore.QModelIndex)
    def _on_click(self, index):
        row = index.row()
        dlg = ProductPage(self)
        dlg.display_product_page(self.db, self.products[row])
        dlg.exec_()
        print(row)

    def resizeEvent(self, event):
        return super(MainWindow, self).resizeEvent(event)

class ProductPage(QDialog):
    def __init__(self, *args, **kwargs):
        super(ProductPage, self).__init__(*args, **kwargs)
        self.resize(1600, 1000)

    def display_product_page(self, db, product):
        self.setWindowTitle(product.name)
        formGroupBox = QGroupBox("Product info")
        layout = QFormLayout()

        products_and_prices = db.get_prices_for_product(product.id)
        for prices in products_and_prices.values():
            last_price = prices[-1]
            shop_name = last_price.shop_product.shop.name.capitalize()
            price_in_shop = QLabel(f"{shop_name} - {last_price.price}.00 grn")
```

```
            price_in_shop.setFont(QFont("Calibri", 18))
            layout.addRow(price_in_shop, self.get_price_history(prices))

        self.setLayout(layout)

        formGroupBox.setLayout(layout)
        mainLayout = QVBoxLayout()
        mainLayout.addWidget(formGroupBox)
        self.setLayout(mainLayout)

    def get_price_history(self, shop_prices):
        graph_widget = pg.PlotWidget(axisItems={'bottom': pg.DateAxisItem()})
        graph_widget.setTitle("Price history")
        styles = {'color':'w', 'font-size':'24px'}
        graph_widget.setLabel('left', 'Price', **styles)
        graph_widget.setLabel('bottom', 'Date', **styles)
        dates = []
        prices = []
        for i, price_obj in enumerate(shop_prices):
            formated_date = self.format_date(price_obj.datetime_created)
            dates.append(formated_date)
            if i > 0:
                dates.append(formated_date)
                prices.append(shop_prices[i-1].price)
            prices.append(price_obj.price)

        dates.append(self.format_date(dt.datetime.now()))
        prices.append(prices[-1])

        pen = pg.mkPen(color=(255, 0, 0), width=15, style=QtCore.Qt.SolidLine)
        line = pg.PlotCurveItem(clear=True, pen=pen)
        line.setData(dates, prices)
        graph_widget.addItem(line)
        graph_widget.setBackground('w')

        return graph_widget

    def format_date(self, datetime):
        return int(time.mktime(datetime.timetuple()))


class CustomDialog(QDialog):
    def __init__(self, *args, **kwargs):
        super(CustomDialog, self).__init__(*args, **kwargs)
        self.display_message_lbl = QLabel("")
        self.layout = QVBoxLayout()
        self.layout.addWidget(self.display_message_lbl)
        self.setLayout(self.layout)
```

```
    def display_message(self, type, message):
        self.setWindowTitle(type)
        self.display_message_lbl.setText(message)
        self.display_message_lbl.setFont(QFont("Calibri", 12))

if __name__ == "__main__":
    import sys
    app = QApplication(sys.argv)
    file = QFile(":/dark.qss")
    file.open(QFile.ReadOnly | QFile.Text)
    stream = QTextStream(file)
    app.setStyleSheet(stream.readAll())
    ui = MainWindow()
    ui.show()
    sys.exit(app.exec_())
```

Модуль database:

```
import enum
import sqlite3
import datetime
from os import path
from sqlite3 import Error

from sqlalchemy import orm
from sqlalchemy import Column, Integer, String, DateTime, Enum, ForeignKey, Table
from sqlalchemy.orm import relationship, backref, sessionmaker
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy import create_engine

Base = declarative_base()

PATH_TO_DB = path.join(path.dirname(__file__), "..", "res", "database.db")


class Product(Base):
    __tablename__ = 'product'
    id = Column(Integer, primary_key=True)
    name = Column(String)


class Shops(enum.Enum):
    rozetka = 0
    citrus = 1
    epicentr = 2
    test = 3
```

```
class ShopProduct(Base):
    __tablename__ = 'shop_product'
    id = Column(Integer, primary_key=True)
    product_id = Column(Integer, ForeignKey("product.id"))
    shop = Column(Enum(Shops), default=Shops.test)
    url = Column(String)


class ProductPrice(Base):
    __tablename__ = 'product_price'
    id = Column(Integer, primary_key=True)
    datetime_created = Column(DateTime, default=datetime.datetime.now)
    shop_product_id = Column(Integer, ForeignKey("shop_product.id"))
    price = Column(Integer)
    shop_product = orm.relationship('ShopProduct', foreign_keys=[shop_product_id])



class Database:
    def __init__(self):
        engine = create_engine(f'sqlite:///{PATH_TO_DB}', echo=True)
        Base.metadata.create_all(engine)
        Session = sessionmaker(bind=engine)
        self.session = Session()

    def insert_product(self, name, url, price):
        """
        Insert new product into products table.

        :param product: Product to insert in DB.
        :type name: Product
        :return: product id
        """
        shop = self._determine_shop(url)

        product = Product(name=name)
        self.session.add(product)
        self.session.commit()

        shop_product = ShopProduct(product_id=product.id, url=url, shop=shop)
        self.session.add(shop_product)
        self.session.commit()

        self.session.add(ProductPrice(shop_product_id=shop_product.id, price=price)
)
        self.session.commit()
        return product
```

```
    def update_product(self, product):
        """
        Update expected price of the product.

        :param product: product id
        :type product: Product
        :return: product id
        """
        self.session.add(product)
        self.session.commit()

    def update_product_price(self, shop_product_id, price):
        """
        Update expected price of the product.

        :param product: product id
        :type product: Product
        :return: product id
        """
        self.session.add(ProductPrice(shop_product_id=shop_product_id, price=price)
)
        self.session.commit()

    def select_products(self):
        """
        Select all products in DB.

        :returns: products
        :rtype: list
        """
        products = self.session.query(Product).all()
        return products

    def delete_product(self, product_id):
        """
        Delete product with all related data from DM by id.

        :param id: id of thd product in DB
        :return: if product was deleted
        """
        shop_products = self.session.query(ShopProduct).filter(ShopProduct.product_
id == product_id)
        shop_product_ids = [shop_product.id for shop_product in shop_products]
        self.session.query(ProductPrice).filter(ProductPrice.shop_product_id.in_(sh
op_product_ids)).delete(synchronize_session=False)
        shop_products.delete(synchronize_session=False)
```

```
        self.session.query(Product).filter(Product.id == product_id).delete(synchro
nize_session=False)
        self.session.commit()
        return True

    def get_prices_for_product(self, id):
        product_in_shops = self.session.query(ShopProduct).filter(
            ShopProduct.product_id == id
        ).all()
        product_ids = [product_.id for product_ in product_in_shops]
        product_prices = self.session.query(ProductPrice).filter(
            ProductPrice.shop_product_id.in_(product_ids)
        ).all()
        products_and_prices = dict.fromkeys(product_ids, None)
        for price in product_prices:
            if not products_and_prices[price.shop_product_id]:
                products_and_prices[price.shop_product_id] = [price]
            else:
                products_and_prices[price.shop_product_id].append(price)

        return products_and_prices


    def _determine_shop(self, url):
        for key in Shops:
            if key.name in url:
                return key
        else:
            raise Exception(f"The shop with url {url} is not supported!"
                            "Try one from the supported shops list.")
```

Модуль scrapper:

```
import re
from enum import Enum

from backend.database import Shops

import requests
from lxml import html
from googlesearch import search


class Scrapper:
    def __init__(self):
        self.product_info = {
            Shops.rozetka: {"price": "\"price\"",
                            "name": "//h1[@class = 'product__title']"},
```

```
            Shops.citrus: {"price": "//b[@class='buy-section__new-price']",
                           "name": "//h1[@class = 'product__title']"},
            Shops.epicentr: {"price": "//div[@class = 'p-price__main']",
                             "name": "//h1[@class = 'p-header__title nc']"},
            Shops.test: {"price": "//p[@id='price']",
                         "name": "//p[@id='name']"},
        }

    def get_name_and_price(self, url):
        if url.startswith("D:"):
            with open(url, "r") as f:
                page = f.read()
                tree = html.fromstring(page)
                page_text = page
        else:
            page = requests.get(url)
            tree = html.fromstring(page.content)
            page_text = page.text

        # with open("result.html", "w") as f:
        #     f.write(page.text.encode('utf-8').decode('ascii', 'ignore'))

        shop = self._determine_shop(url)
        encoded_name = self._xpath(tree, page_text, self.product_info[shop]["name"]
)
        name = self._format_name(encoded_name)
        str_price = self._xpath(tree, page_text, self.product_info[shop]["price"])
        price = self._format_price(str_price)
        return name, price

    def _determine_shop(self, url):
        for key in self.product_info.keys():
            if key.name in url:
                return key
        else:
            raise ShopIsNotSupported(f"The shop with url {url} is not supported!"
                                     "Try one from the supported shops list.")

    def _format_price(self, price):
        price = price.replace("\xa0", "")
        price = re.sub("[^0-9]", "", price)
        return int(price)

    def _format_name(self, name):
        try:
            name = name.encode("l1")
            name = name.decode()
```

```
        except UnicodeEncodeError:
            pass
        name = name.strip()
        name = name.replace("\n", " ")
        return name

    def _xpath(self, tree, page_text, xpath):
        if xpath.startswith('//'):
            try:
                return tree.xpath(xpath)[0].text
            except IndexError:
                raise ElementNotFoundError("Failed to find element on the web page.
")
        else:
            index = page_text.find(xpath)
            if index != -1:
                start = index + 9
                end = page_text.find('.', start)
                return page_text[start:end]
            else:
                raise ElementNotFoundError("Failed to find element on the web page.
")

    def search(self, query):
        for result in search(query, tld="com.ua", num=10, stop=10, pause=2):
            print(result)



class ElementNotFoundError(Exception):
    """
    Raised when can`t find element on the page.
    """
    pass


class ShopIsNotSupported(Exception):
    """
    Raised when shop is not supported.
    """
    pass
```