МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА

Факультет прикладної математики та інформатики
(повне найменування назва факультету)

кафедра інформаційних систем
(повна назва кафедри)

ДИПЛОМНА РОБОТА

РОЗРОБКА СИСТЕМИ ПОШУКУ ОПТИМАЛЬНИХ ЦІН НА ТОВАРИ З ВИКОРИСТАННЯМ АЛГОРИТМІВ ПАРСИНГУ

Виконав(ла): студент(ка) групи ПМі-44
спеціальності 122 – комп'ютерні науки
(шифр і назва спеціальності)

Клочковський А.О
(підпис) (прізвище та ініціали)

Керівник доцент Вовк В.Д
(підпис) (прізвище та ініціали)

Рецензент
(підпис) (прізвище та ініціали)

2023

ЗМІСТ

ВСТУП

Не можна заперечувати, що онлайн-покупки сьогодні є не просто актуальними, а являють собою невід'ємну частину нашого життя. Хоча при щоденному використанні ми і не звертаємо уваги на те, яку еволюцію вони пройшли та як непомітно стали нашою буденністю, їх розвиток лише набирає оберти. Клієнти все прискіпливіше обирають сайти, яким би вони надали перевагу для оформлення замовлення, враховуючи найрізноманітніші фактори такі як: вигляд інтерфейсу, розташування магазину, наявність мобільного додатку, тощо - але головним критерієм вибору завжди є ціна.

Навряд, хтось стане заперечувати важливість ціни і, з розумінням цього, нині стали популярні сервіси, що надають користувачам можливість порівняти вартість товарів на різних сайтах. В свою чергу вже вони беруть участь у перегонах за інтерес користувача.

Критичним у цьому змаганні є асортимент товарів представлений на платформах. Та за для того, щоб не відставати від конкурентів, вони вимушені регулярно відстежувати сотні та навіть тисячі позицій різних товарів. Крім того, специфікації кожного лоту можуть суттєво відрізнятись одна від одної, тому модератори таких платформ можуть годинами додавати, або змінювати характеристики відповідних товарів. Такий процес може відбуватись різними шляхами: додавання конкретних товарів вручну, надсилання запитів до API інтернет-магазинів, тощо.

Поглянувши з іншого боку, користувачі теж не завжди виграють використовуючи відповідні платформи, адже переважна більшість з них, працюють за моделлю рекламної інтеграції, тобто сервіс розміщує у себе товари при умові плати з боку продавця. Певною мірою це може не відповідати очікуванням користувача, оскільки продавці самі вирішують чи опубліковувати конкретний товар на майданчику і чи оформлювати партнерство з ним взагалі. Відповідно, будуть просуватися товари, в яких

продавець бачить інтерес, а деякі можуть і зовсім ігноруватись, або товари з умовного магазину, що йде другим за результатами пошукового рушія, можуть просто не бути представленні на платформі, адже продавець відмовився платити за їх публікацію. Така система зручна для магазинів і майданчиків, але певною мірою не є клієнтоорієнтованою.

Взявши до уваги, проблеми описані вище, у мене виникла ідея цього проекту. Вона полягала у створенні застосунку, який надає перевагу клієнтам та без публікації окремих лотів магазинами, оглядати їх асортимент та порівнювати ціни на представлені ними товари і в свою чергу, зекономити час, та полегшити роботу модераторів сайту.

ПОСТАНОВКА ЗАДАЧІ

Головним етапом у роботі над цим проєктом стало формування чіткого бачення цілісного продукту та визначення пунктів, що змогли б задовольнити його проблематику. Сформульована задача – це створення системи, що надає користувачу простий та швидкий доступ, до свого функціоналу і сама є достатньо цілісною і автоматизованою, потребуючи при цьому мінімального втручання для функціонування.

Фізична проблематика

1. Необхідно вирішити, як буде відбуватись відбір магазинів, адже потрібно взяти конкретний та релевантний діапазон. Для прикладу, виключати магазини з-за кордону, або майданчики, що пропонують вживані товари, тощо.
2. Потрібно знайти головну інформацію про конкретні товари, які є в наявності на сайті, так шукаючи кавомашину нас не цікавлять ноутбуки, чи навушники.
3. Адміністратори системи не мають проводити постійне втручання у систему задля поповнення асортименту, або підтримки її працездатності.

Проблематика реалізації

1. Платформа реалізації.
2. Швидкодія.
3. Механіка пошуку магазинів.
4. Зчитування інформації з сайтів.
5. Алгоритм пошуку.
6. Подання знайденої інформації користувачу у зручному та зрозумілому вигляді.
7. Взаємодія адміністрації з системою.

МЕТОДИКА ВИРІШЕННЯ

1. Обрані технології

Я обрав веб-застосунок як основу реалізації своєї ідеї, адже для сфери онлайн-покупок цей варіант залишається найбільш зручним та розповсюдженим.

Відштовхуючись від потреби у швидкій та стабільній платформі із уніфікованим інтерфейсом, я зупинився на ASP.NET Core MVC.

ASP.NET Core MVC (Model-View-Controller) є фреймворком для розробки веб-додатків, реалізованим на базі ASP.NET Core. Цей фреймворк дозволяє побудувати веб-додатки з використанням шаблону проектування Model-View-Controller, що дозволяє розділити логіку додатку на три основні компоненти: модель (Model), представлення (View) та контролер (Controller). Кожен з цих компонентів виконує певні функції: Модель (Model): Відповідає за обробку даних, бізнес-логіку та взаємодію з базою даних. Це можуть бути класи, структури або сервіси, що представляють дані та функції, пов'язані з обробкою даних. Представлення (View): Відповідає за відображення даних користувачу. Це можуть бути HTML-файли, які містять розмітку та код для відображення даних користувачу. Представлення отримують дані з моделі та генерують відповідний HTML-код. Контролер (Controller): Відповідає за обробку запитів від користувача, взаємодію з моделлю та відправку відповіді до відповідного представлення. Контролери приймають запити від користувача, виконують необхідну логіку та передають дані до моделі, а потім вибирають відповідне представлення для відображення результату. ASP.NET Core MVC надає багато вбудованих можливостей, таких як маршрутизація, фільтри, аутентифікація, авторизація, підтримка RESTful API та багато іншого. Крім того, він є легким у використанні, оскільки побудований на базі ASP.NET Core, який підтримує роботу на різних платформах (Windows, macOS, Linux). ASP.NET Core MVC є популярним фреймворком для розробки веб-додатків, особливо при створенні додатків, що використовують шаблон Model-View-Controller для розділення логіки та представлення.

Razor Pages є частиною фреймворка ASP.NET Core і надають простий та ефективний спосіб розробки веб-сторінок з кодом серверної частини. Вони базуються на концепції Model-View-Controller (MVC), але з фокусом на простоту і зрозумілість. Основна відмінність між Razor Pages та MVC полягає у способі організації логіки. У Razor Pages розділення логіки між моделлю (Model) та представленням (View) відбувається на рівні окремих сторінок, тоді як у MVC це відбувається на рівні контролерів та представлень. Це дозволяє швидше створювати прості веб-сторінки з меншою кількістю коду. Razor-розмітка, тобто можна використовувати Razor-синтаксис для розмітки сторінок, що дозволяє вставляти динамічні дані, виконувати цикли та умови, викликати методи, тощо.

Microsoft SQL Server є однією з провідних реляційних систем управління базами даних (СУБД), розроблених компанією Microsoft. Він надає надійне та потужне середовище для зберігання, керування та взаємодії з даними у веб-додатках та корпоративних системах. Основні характеристики Microsoft SQL Server:

1. Реляційна модель: SQL Server базується на реляційній моделі даних, що дозволяє організовувати дані у вигляді таблиць зі зв'язками між ними. Це забезпечує структуровану і консистентну організацію даних.

2. Мова запитів SQL: SQL Server використовує мову структурованих запитів SQL (Structured Query Language) для взаємодії з базою даних. SQL дозволяє створювати, змінювати та видаляти дані, виконувати складні запити, агрегувати дані та багато іншого.

3. Масштабованість та продуктивність: SQL Server може обробляти великі обсяги даних та забезпечувати високу продуктивність завдяки оптимізації запитів, індексам, кешуванню даних та іншим оптимізаціям. Він підтримує різні режими роботи, включаючи розподілені системи та кластери, для забезпечення масштабованості та високої доступності.

4. Розширені можливості: SQL Server надає широкий спектр функціональності, такої як транзакційна безпека, резервне копіювання та

відновлення, реплікація, шифрування даних, інтеграція з .NET Framework та інші. Він також підтримує різні типи даних, включаючи текстові, числові, дати, графічні, JSON та багато інших.

5. Інструменти управління: Microsoft SQL Server постачається з набором потужних інструментів для управління базою даних, таких як SQL Server Management Studio (SSMS) і SQL Server Data Tools (SSDT). Ці інструменти дозволяють розробникам та адміністраторам легко керувати базами даних, виконувати запити, створювати схему бази даних, налагоджувати та моніторити процеси.

Entity Framework (EF) є фреймворком для доступу до даних у .NET-платформі. Він надає швидкий і простий спосіб взаємодії з базами даних, дозволяючи розробникам працювати з даними у вигляді об'єктів і класів замість прямих запитів SQL. Основні характеристики Entity Framework:

1. Об'єктно-орієнтоване моделювання: Entity Framework дозволяє визначати модель даних у вигляді класів, що відповідають таблицям бази даних. Кожне поле таблиці представляється як властивість класу, а зв'язки між таблицями виражаються за допомогою властивостей навігації. Це дає змогу працювати з даними у вигляді об'єктів, що спрощує розробку і підтримку додатків.

2. ORM (Object-Relational Mapping): EF виконує ORM, що означає, що він автоматично виконує відображення між об'єктно-орієнтованою моделлю даних і структурою бази даних. Він дозволяє здійснювати операції створення, зчитування, оновлення та видалення (CRUD) з об'єктами моделі без прямого написання складних запитів SQL.

3. Підтримка різних баз даних: Entity Framework підтримує різні типи баз даних, включаючи Microsoft SQL Server, MySQL, PostgreSQL, Oracle та багато інших. Він забезпечує абстракцію від конкретного постачальника бази даних, що дозволяє легко перемикатись між різними системами управління базами даних. Лінива завантаження

4. (Lazy Loading): Entity Framework підтримує ліниве завантаження, що означає, що дані з бази даних завантажуються лише тоді, коли вони потрібні. Це дозволяє економити ресурси і забезпечує більш ефективну роботу з великими обсягами даних.

5. Міграції бази даних: EF надає підтримку для автоматичного створення та оновлення структури бази даних на основі змін в моделі. Це дозволяє зберігати базу даних синхронізованою зі змінами в програмному коді без необхідності вручного написання скриптів міграцій.

Selenium WebDriver - це потужний фреймворк для автоматизації веб-додатків. Він дозволяє розробникам тестувати веб-додатки, взаємодіяти з ними та перевіряти їх функціональність шляхом програмного керування браузером. Основні риси і можливості Selenium WebDriver:

1. Багатобраузерна підтримка: Selenium WebDriver підтримує різні веб-браузери, такі як Chrome, Firefox, Safari, Edge, Opera та інші. Це дозволяє розробникам тестувати веб-додатки на різних платформах та переконуватися в їх сумісності з різними браузерами.

2. Мови програмування: Selenium WebDriver підтримує кілька мов програмування, таких як Java, C#, Python, Ruby та інші.

3. Взаємодія з елементами веб-сторінки: За допомогою Selenium WebDriver можна здійснювати різні дії з елементами веб-сторінки, такі як натискання на кнопки, заповнення форм, вибір елементів зі списку, перевірка тексту та багато іншого. Це дозволяє розробникам імітувати взаємодію користувача з веб-додатком.

4. Очікування та перевірки: Selenium WebDriver надає можливість встановлювати очікування на елементи веб-сторінки, наприклад, очікування на відображення елемента або завершення завантаження сторінки. Він також дозволяє здійснювати різні перевірки, щоб переконатися, що веб-додаток працює правильно.

5. Інтеграція з фреймворками тестування: Selenium WebDriver можна поєднувати з різними фреймворками тестування, такими як JUnit, TestNG,

NUnit, PyTest та іншими. Це дозволяє розробникам створювати структуровані та розширювані автоматизовані тести.

6. Підтримка розширених сценаріїв: Selenium WebDriver дозволяє створювати складні сценарії тестування, такі як робота з фреймами, робота з діалоговими вікнами, обробка випадкових подій та інші. Це робить його універсальним інструментом для автоматизації різноманітних веб-додатків. Selenium WebDriver є одним з найпопулярніших інструментів для автоматизації веб-тестування, оскільки він дозволяє розробникам швидко і ефективно перевіряти функціональність веб-додатків та забезпечує багатофункціональність і гнучкість у роботі з браузером.

XPath та TruePath. XPath (XML Path Language) є мовою запитів для вибору елементів у XML-документах, включаючи HTML. XPath використовується для локації та навігації по структурі документа, щоб знайти конкретні елементи або групи елементів на основі їх властивостей або шляху до них. Основні поняття в XPath:

Елементи - це вузли в XML-документі, які мають теги. У випадку HTML документа, це відповідає різним HTML-тегам, таким як , <p>, <a>, тощо.

Атрибути - це додаткові властивості елементів, які можуть містити додаткову інформацію про них. Наприклад, у HTML елементу <a href="https://www.example.com">, href є атрибутом.

Шлях в XPath використовується для вибору конкретного вузла або групи вузлів в структурі документа. Наприклад, //div вибирає всі елементи в документі, а /html/body/div вибирає елемент вкладений в <body>, який, з свого боку, є вкладеним в <html>.

Умови в XPath використовуються для фільтрації вузлів на основі певних критеріїв. Наприклад, //div[@class="container"] вибирає всі елементи , які мають атрибут class зі значенням "container".

XPath можна використовувати для вибору елементів, виконання пошуку, отримання значень атрибутів, навігації по батьківським, дочірнім і сусіднім

елементам тощо. Він є потужним інструментом для взаємодії з HTML-документами та отримання даних з них.

TruePath - це розширення для браузера, яке надає інструменти для роботи з XPath на веб-сторінках, інтерфейс програми зображено на рис. 1 і рис. 2. Коли користувач клацає правою кнопкою миші на будь-якому елементі веб-сторінки, розширення відображає весь XPath цього елемента у вигляді пункту меню. TruePath допомагає створювати динамічний, відносний, унікальний та індексний XPath. Воно також може генерувати XPath на основі текстового значення. З використанням TruePath можна швидко створювати сценарії автоматизації для веб-сторінок і значно зменшити зусилля, необхідні для створення таких сценаріїв. Загалом, TruePath полегшує розробку та тестування веб-додатків, надаючи зручні інструменти для роботи з XPath і автоматизації дій на веб-сторінках.

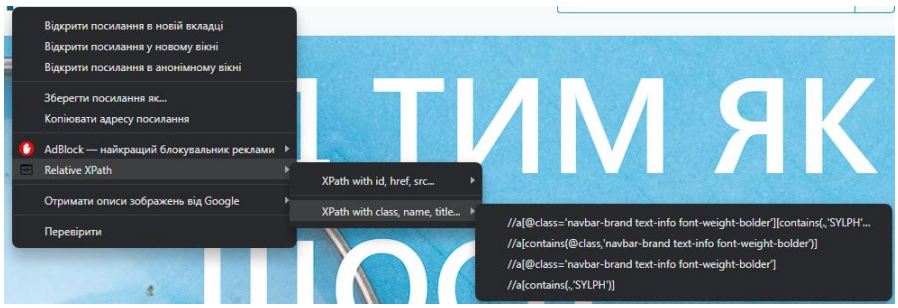


Рисунок 1. Варіанти подані TruePath

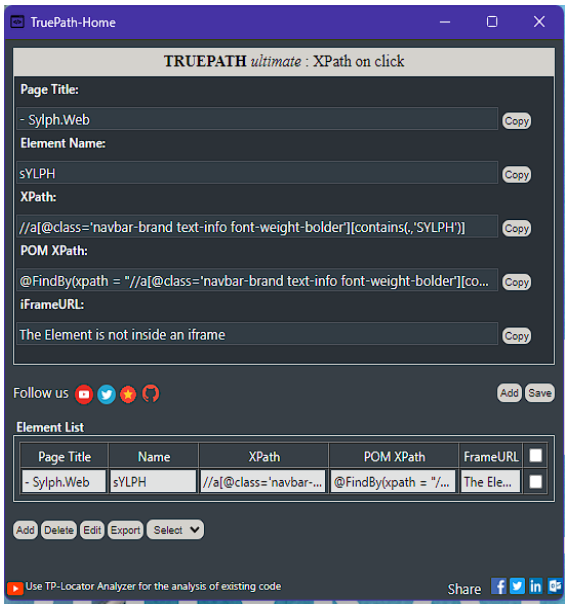


Рисунок 2. Інтерфейс розширення TruePath

2. Пошук магазинів

При реалізації пошуку магазинів було вирішено звернутись до пошукового рушія Google та опрацьовувати отримані результати. Для цього, я використав можливості парсингу Selemium WebDriver.

Алгоритм пошуку магазинів

1. Відвідувач сайту використовує поле пошуку застосунку і його запит опрацьовується сервісом.

2. На стороні сервера, сервіс використовує WebDriver, який дозволяє відкривати конкретне посилання, тому достатньо буде звернутись за посиланням https://www.google.com/search?q= та додати «купити» + назву необхідного нам товару. Це може бути як загальне слово “ноутбук”, так і конкретна модель. Отримаємо веб-елемент – нашу сторінку із результатами пошуку.

3. Наступним кроком, буде взяття посилань на потрібні магазини. За допомогою WebDriver можна звертатись до елементів за назвою HTML класу. У нашому випадку, посилання-результати містяться у елементах одного класу - «yuRUbf». Посилання з дописами “Реклама” – проігноровані, адже вони не завжди містять потрібну наразі інформацію. Тепер ми маємо список посилань магазинів.

4. Передбачимо перехід на наступні сторінки з результатами пошуку. Наразі обмежимось максимумом у 10 сторінок, адже у реальному часі, навіть така кількість сторінок буде опрацьовуватись відносно довго. Ми можемо регулювати кількість сторінок результатів пошукового рушія, відштовхуючись від діапазону сайтів, який ми хочемо опрацювати, а також часом, який ми готові витратити на їх парсинг.

5. Залишається зробити фільтрацію за українськими сайтами, а конкретніше за доменом «.ua».

6. Результати фільтрації записуємо у список сайтів, по яким в подальшому буде відбуватись пошук.

3. Пошук товарів

Відстеження бажаних лотів неможливо було реалізувати аналогічно алгоритму, який був використаний при пошуку магазинів. На це є дві головні причини: унікальність HTML-розмітки кожного магазину, а також ймовірність існування двох видів сторінок у одного сайту – одинарної та сітки. Ця проблема була вирішена створенням бази даних у яку будуть вноситись фрейми зі стрічками коду XPath, через які ми і доступаємось до необхідних елементів.

Модель сайту, зображену на рис. 3, яку ми зберігаємо у базі даних, має такі властивості:

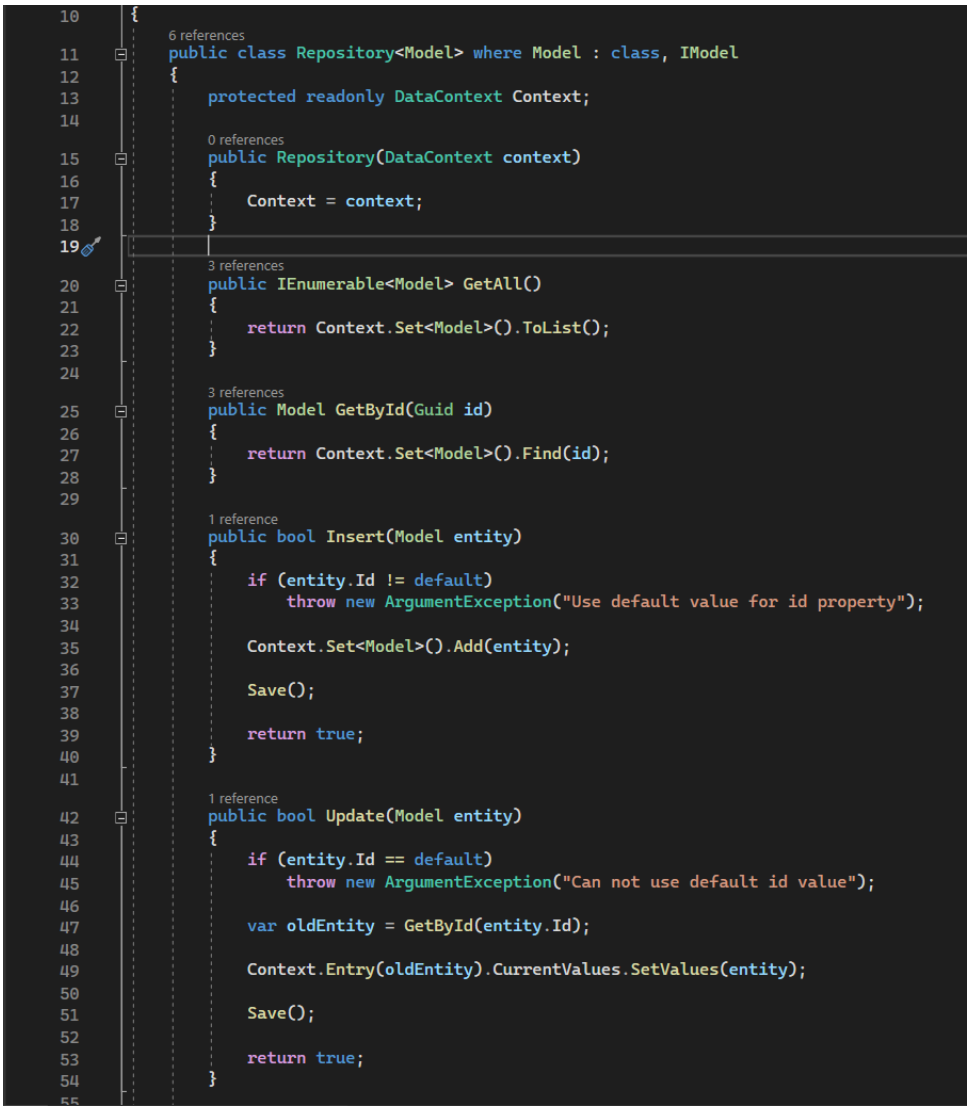
1. Id – унікальний ідентифікатор.
2. Url – адреса сайту.
3. Option – змінна, значення якої відповідає якого типу є конкретна модель, одинарна чи сітка.
4. ContentPathOption – збергіє значення XPath, за яким ми доступаємось до HTML-тегу, що містить у собі сам товар.
5. NamePathOption - збергіє значення XPath, за яким ми доступаємось до HTML-тегу, що містить у собі назву товару.
6. ImagePathOption - збергіє значення XPath, за яким ми доступаємось до HTML-тегу, що містить у собі зображення товару.
7. PricePathOption - збергіє значення XPath, за яким ми доступаємось до HTML-тегу, що містить у собі ціну товару.
8. NotAvailableFilter – зберігає унікальне для конкретного сайту слово, або вираз, що сигналізує про відсутність товару, наприклад, “Немає в наявності”, “Сповістити про наявність”.
9. NextPagePathOption – зберігає значення для елементу, через який відбувається перехід до наступної сторінки, в разі парсингу сітки товарів.

```
public class Site : IModel
{

    public Guid Id { get; set; }
    14 references
    public string Url { get; set; }
    21 references
    public int Option { get; set; }
    17 references
    public string ContentPathOption { get; set; }
    14 references
    public string NamePathOption { get; set; }
    18 references
    public string ImagePathOption { get; set; }
    14 references
    public string PricePathOption { get; set; }
    12 references
    public string NotAvailableFilter { get; set; }
    9 references
    public string NextPagePathOption { get; set; }
}
}
```

Рисунок 3. Модель фрейма сайту

Доступ до бази даних, було реалізовано патерном Repository рис. 4.1 та рис. 4.2, він надає абстракцію між бізнес-логікою застосунку і джерелом даних, дозволяючи легко взаємодіяти з даними без прив'язки до конкретного джерела даних або реалізації доступу до них. Основна ідея патерна Repository полягає в тому, що весь доступ до даних здійснюється через спеціальний об'єкт - репозиторій. Репозиторій виступає як проміжний шар між бізнес-логікою та джерелом даних. Він надає єдину точку доступу до даних для всіх операцій збереження, отримання, оновлення та видалення. Якщо ми хочемо, щоб він опрацьовував модель даних “Site”, необхідно, щоб вона наслідувалась від уніфікованого інтерфейсу, в моєму випадку це інтерфейс IModel, який зобов’язує реалізувати властивість Id рис. 4.3.



```
{
    6 references
    public class Repository<Model> where Model : class, IModel
    {
        protected readonly DataContext Context;

        0 references
        public Repository(DataContext context)
        {
            Context = context;
        }

        3 references
        public IEnumerable<Model> GetAll()
        {
            return Context.Set<Model>().ToList();
        }

        3 references
        public Model GetById(Guid id)
        {
            return Context.Set<Model>().Find(id);
        }

        1 reference
        public bool Insert(Model entity)
        {
            if (entity.Id != default)
                throw new ArgumentException("Use default value for id property");

            Context.Set<Model>().Add(entity);

            Save();

            return true;
        }

        1 reference
        public bool Update(Model entity)
        {
            if (entity.Id == default)
                throw new ArgumentException("Can not use default id value");

            var oldEntity = GetById(entity.Id);

            Context.Entry(oldEntity).CurrentValues.SetValues(entity);

            Save();

            return true;
        }
```

```
        1 reference
        public bool Delete(Guid id)
        {
            var model = GetById(id);
            Context.Set<Model>().Remove(model);

            Save();

            return true;
        }

        3 references
        public void Save()
        {
            Context.SaveChanges();
        }
    }
}
```

Рис. 4.1- 4.2. Клас-реалізація патерну Repository

```
2 references
public interface IModel
{
    10 references
    public Guid Id { get; set; }
}
```

Рис. 4.3. Інтерфейс IModel

Загальна структура застосунку складається із двох проєктів Sylph.Data і Sylph.Web рис. 5

У Sylph.Data реалізовано структури даних, які зберігаються у базі даних, а також репозиторій і DataContext, який працює як посередник між даними бази даних та застосунком.

У Sylph.Web реалізовано, контролери, які опрацьовують запити на відповідних сторінках, а також сервіси для пошуку та фільтрації товарів, і додавання нових, або зміни старих фреймів.

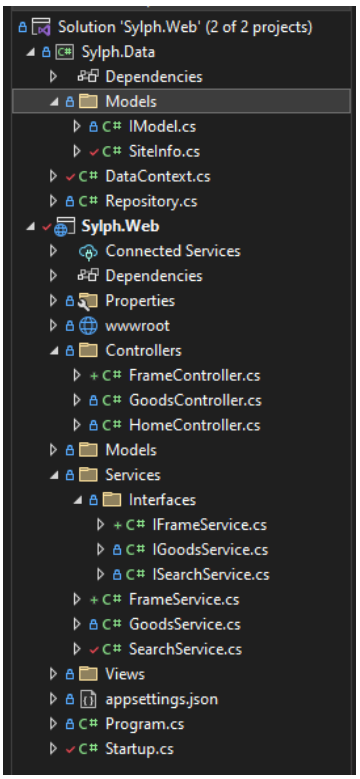


Рис. 5. Структура програми

Алгоритм пошуку товарів

1. Для кожного сайту, який є у списку, ми обрізаємо назву до домену та відкидаємо «https://», так-от, щоб умовний «https://sylph.ua/search» став просто «sylph» та беремо з бази даних, ті значення, що співпадають. Тепер ми маємо список сайтів і інформацію, як доступатись до їх елементів у різних випадках.
2. За допомогою WebDriver переходимо вже на сайт магазину та відповідно до значення ContentPathOption, рахуємо кількість елементів, що мають збіг(1 або більше відповідно).
3. При наявності більше ніж одного товару на сторінці, можна звернути увагу, що вони формують сітку, яка містить в собі елементи із класами, що повторюються. Розуміючи це можна утворити із них колекцію та проводити ітерацію по ним, для цього надаємо їм індекс і тепер звертання до елементів буде відбуватись у вигляді: XPath + індекс.
4. Перевіряємо чи в тексті тегу з XPath ContentPathOption є слово відповідного NotAvailableFilter, якщо так – відкидаємо цей товар.
5. Назву товару, шукаємо за XPath, причому назва має містити пошуковий запит, або ключові слова на які його було розбито, інакше – відкидаємо.
6. Необхідно дістати картинки товару, тут важливо пам’ятати, що зображення можуть завантажуватись не одразу, або лише якщо вони поміщаються у простір екрану користувача, тому необхідно час від час прогортати екран WebDriver. Для цього виконуємо скрипт «window.scrollBy()» і за XPath беремо посилання на зображення.
7. За XPath шукаємо елемент із ціною пропозиції, та зчитуємо значення перед знаком «₴», або «грн».
8. Зберігаємо посилання на сайт для конкретного товару.
9. Робимо перевірку, чи у товару немає пустих полів, інакше – відкидаємо.
10. Записуємо результат.

11. У разі, якщо сторінка по якій відбувався пошук, містила декілька товарів, за тегом NextPagePathOption, буде знайдено елемент, через який можна перейти на наступну сторінку з товарами інтернет-магазина.

В результаті отримаємо список товарів зі всіх переглянутих магазинів, що задовольняють пошуковий запит.

4. Результати пошуку та їх фільтрація

Перед представленням результату користувачу, необхідно посортувати товари таким чином, щоб пропозиції із однаковими назвами були в одній секції та їх можна було з легкістю порівняти. Для цього я написав метод FilterByName. У цьому методі відбувається фільтрація списку елементів (items) за назвою.

Створюється порожній список filterResult, який буде містити результати фільтрації. Для кожного елемента item у списку items виконується наступне:

1. Отримується назва елемента (name).
2. Перевіряється, чи немає вже елемента з такими самими критеріями фільтрації в filterResult. Якщо немає, то створюється новий об'єкт FilterResult зі збереженою назвою (name) та порожним списком елементів Items, додається поточний item до списку Items, а потім цей новий FilterResult додається до filterResult.
3. Якщо вже є елемент з такими самими критеріями фільтрації в filterResult, то поточний item додається до відповідного FilterResult.

В кінці методу повертається список filterResult, який містить відфільтровані елементи згруповані за критеріями фільтрації.

Основна логіка методу полягає в тому, щоб групувати елементи за їх назвою, використовуючи різні критерії порівняння. Якщо для поточного елемента не знайдено жодного існуючого FilterResult зі співпадаючими критеріями фільтрації, створюється новий FilterResult. Якщо ж вже є існуючий FilterResult зі співпадаючими критеріями, то елемент додається до цього FilterResult.

5. Додавання нових фреймів

Для додавання нових фреймів у базу даних, необхідно спочатку дістати кожне значення XPath з самого інтернет-магазину. Для цього скористаємось TruePath і візьмемо значення з поля XPath рис. 6 для кожної з необхідних властивостей.

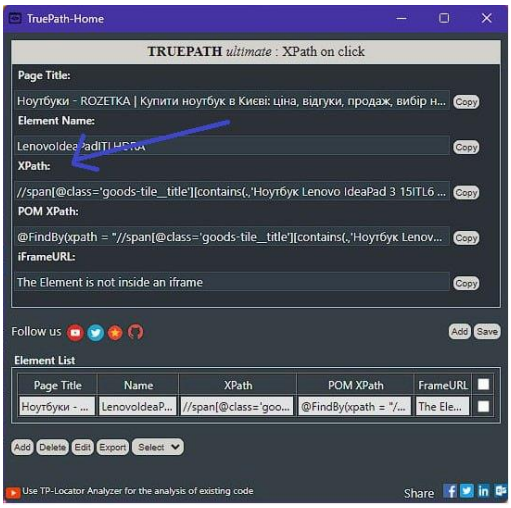


Рис. 6. Процес видобування XPath нового елементу

Після чого надсилаємо Post-запит, який занесе фрейм сайту в базу даних. Схожа операція відбувається також при зміні вже існуючих фреймів.

ДЕМОНСТРАЦІЯ РОБОТИ ПРОГРАМИ

Головна сторінка веб-застосунку «SYLPH» рис. 7

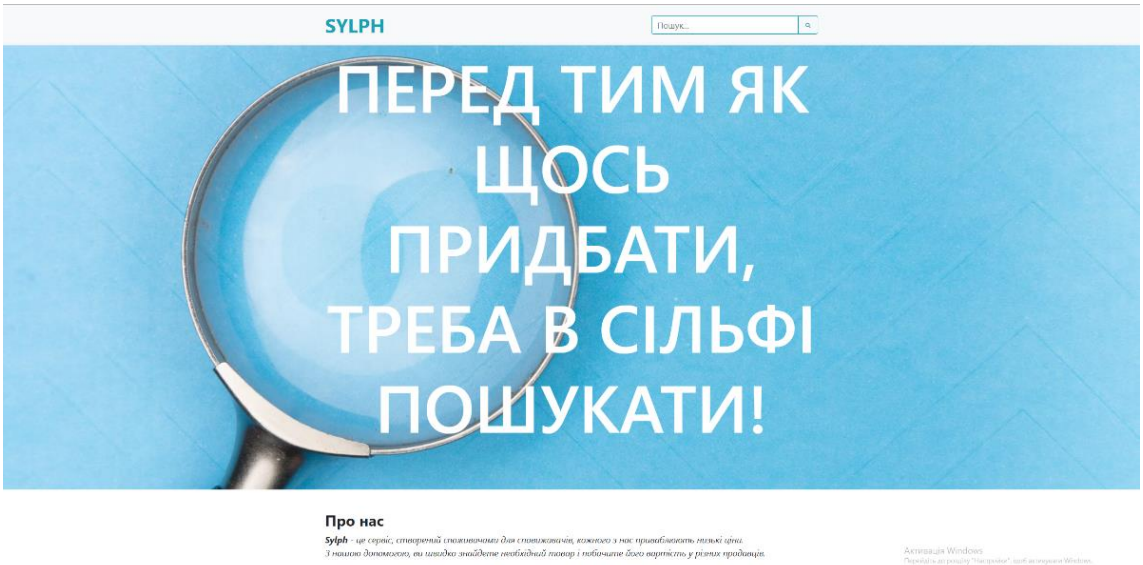


Рис. 7. Головна сторінка

Вводимо пошуковий запит в поле «кавомашина» та починаємо пошук рис. 8

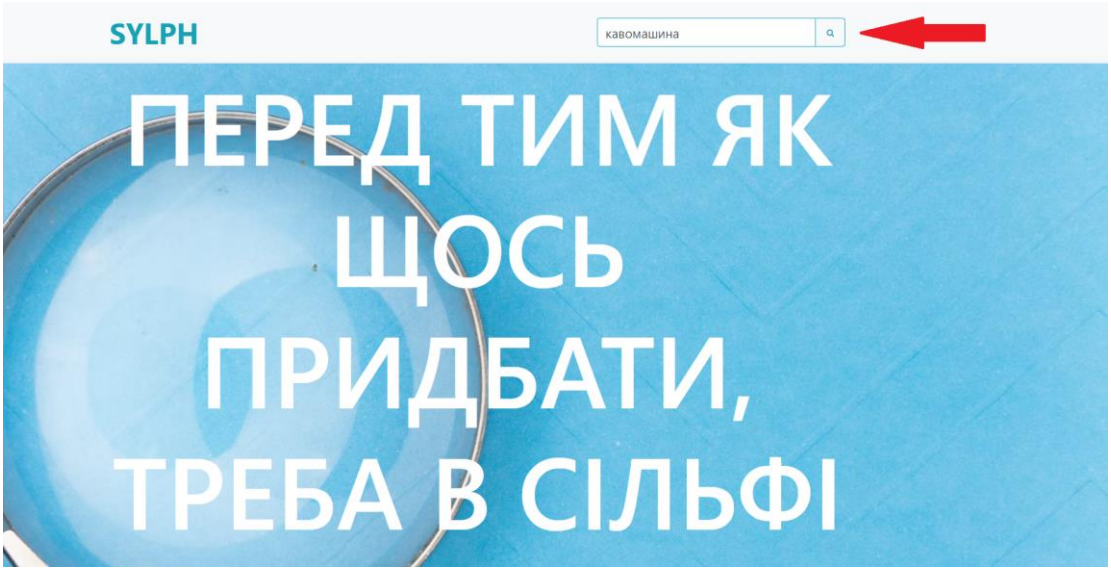


Рис. 8. Введення пошукувого запиту

Відбувається пошук у Google рис. 9. Бачимо, що адреса співпадає з очікуваною.

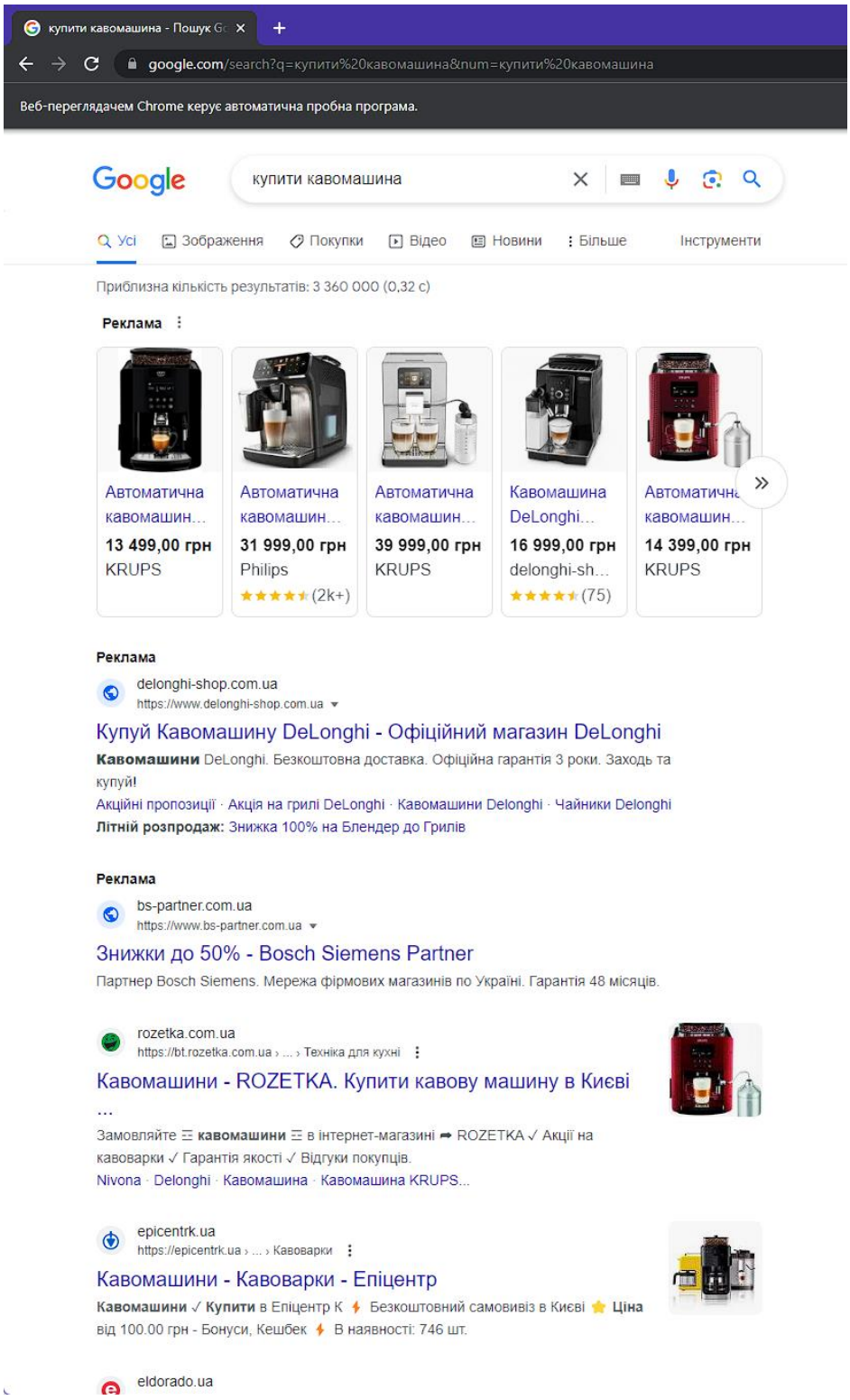


Рис. 9. WebDriver переглядає результати пошуку Google

Також обробляються наступні сторінки з результатами пошукового рушія рис. 10

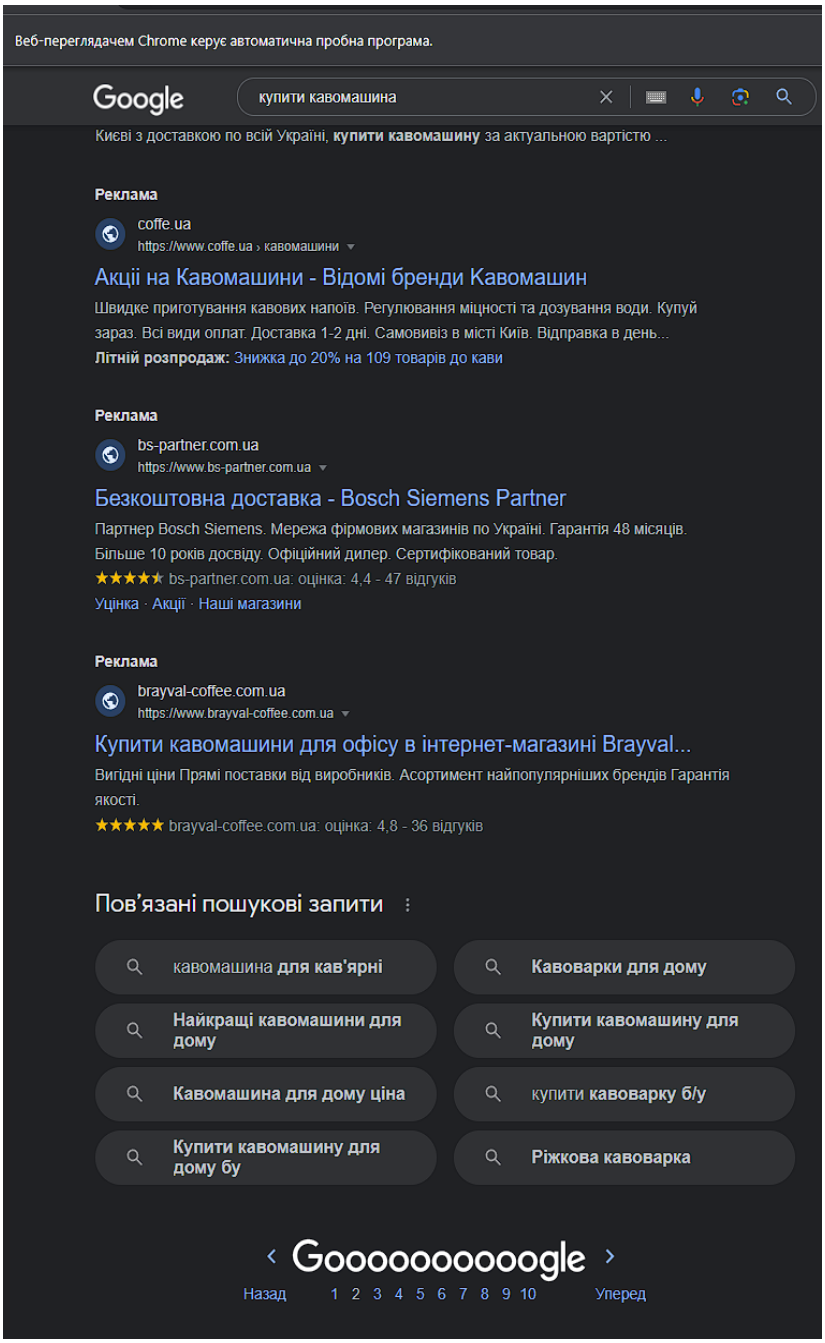


Рис. 10. Перехід Selenium WebDriver на наступні сторінки пошукових результатів

Після обробки результатів Google, маємо переходимо до зчитування даних з інтернет-магазинів рис. 11

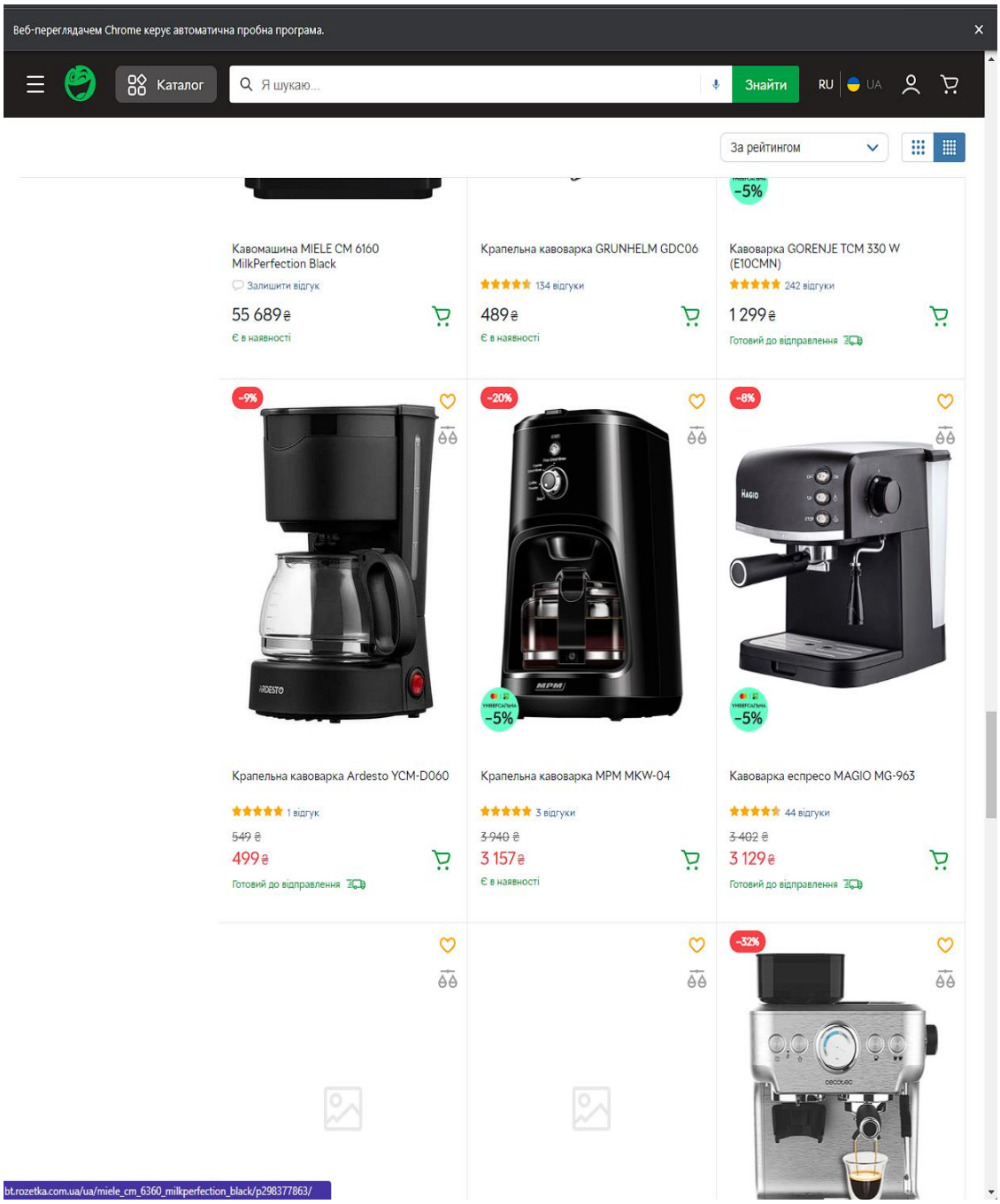


Рис. 11. Пошук Selenium WebDriver по інтернет-магазинам

Результати пошуку рис. 12

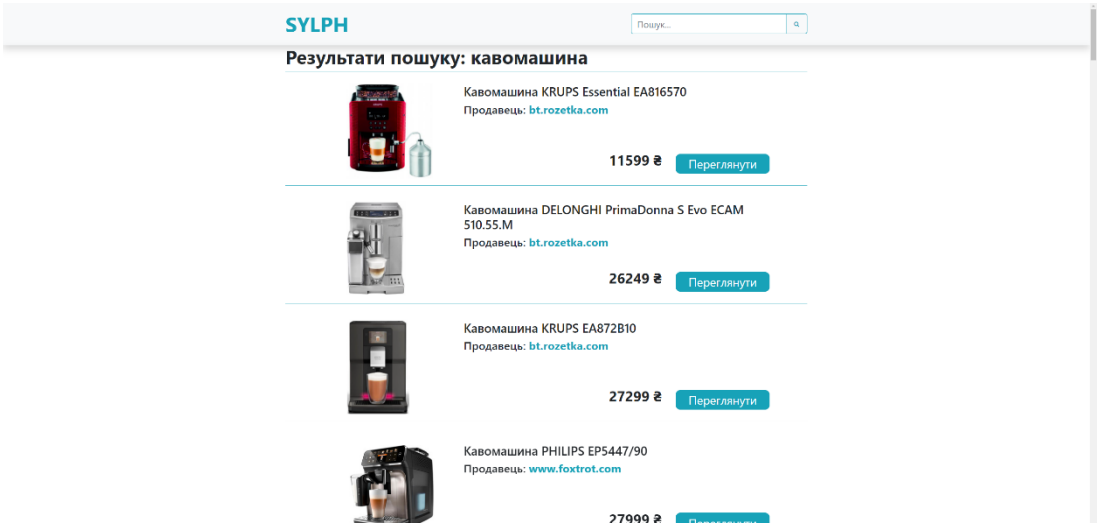


Рис. 12. Вивід результатів

Карусель товарів з різних магазинів рис. 13.1 та рис. 13.2. При наявності декількох результатів пошуку, які задовольняють умовам фільтрації, вони будуть виведені у один блок. Цей блок-карусель буде автоматично перегортатись з певним інтервалом, або ж користувач може вручну переключити його, натиснувши на один з країв блоку.

Звернемо увагу на представлені товари на рисунку 13.1, а конкретно другий та третій. Помітно, що на рисунку 13.2 на цих же позиціях знаходяться ці ж товари, але від інших продавців та з іншою ціною.

Отже, пошук та фільтрація працюють, як заплановано.

Рис. 13.1. Початкова карусель

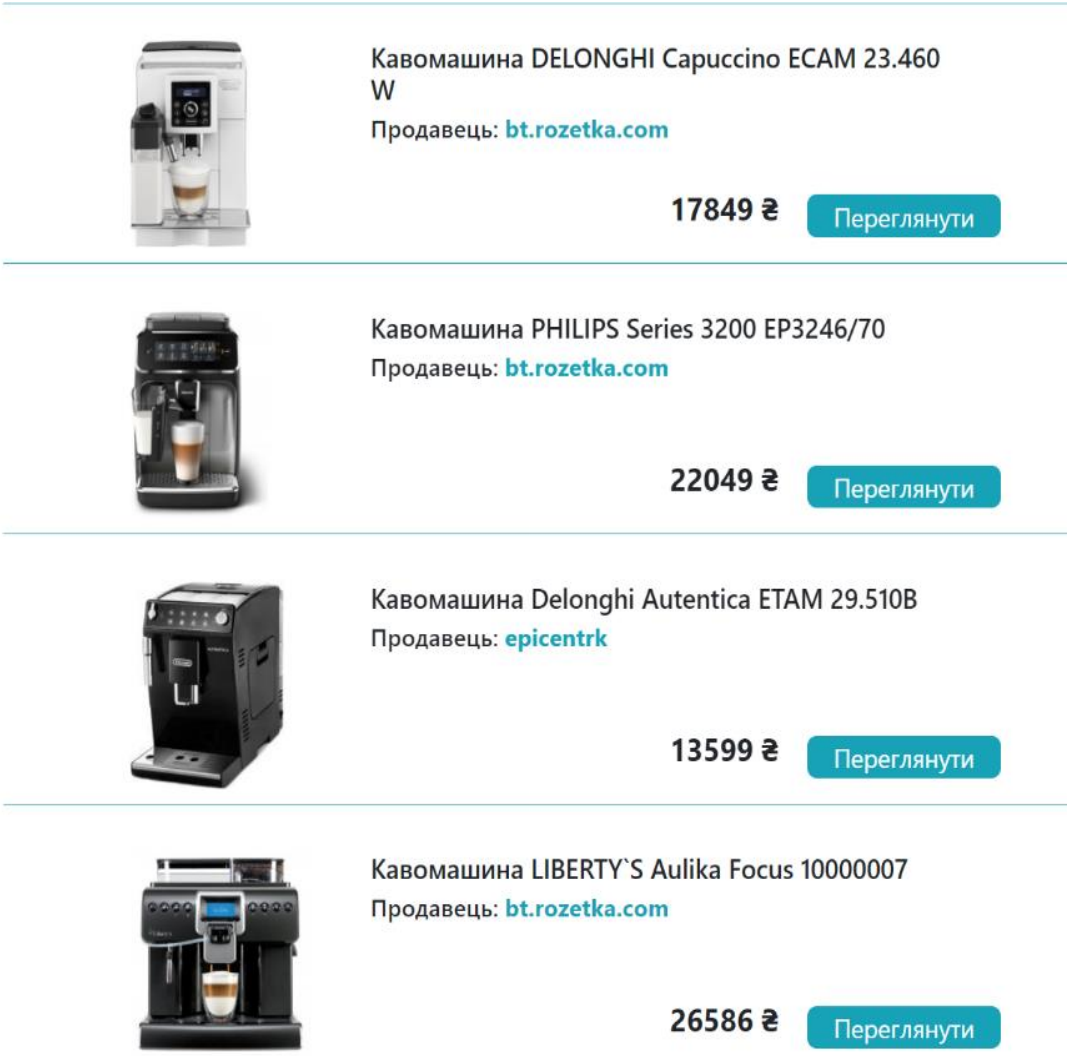


Рис. 13.2. Змінена карусель

Екран додавання нових фреймів рис. 14 Натиснувши “+”, можна створити новий фрейм.

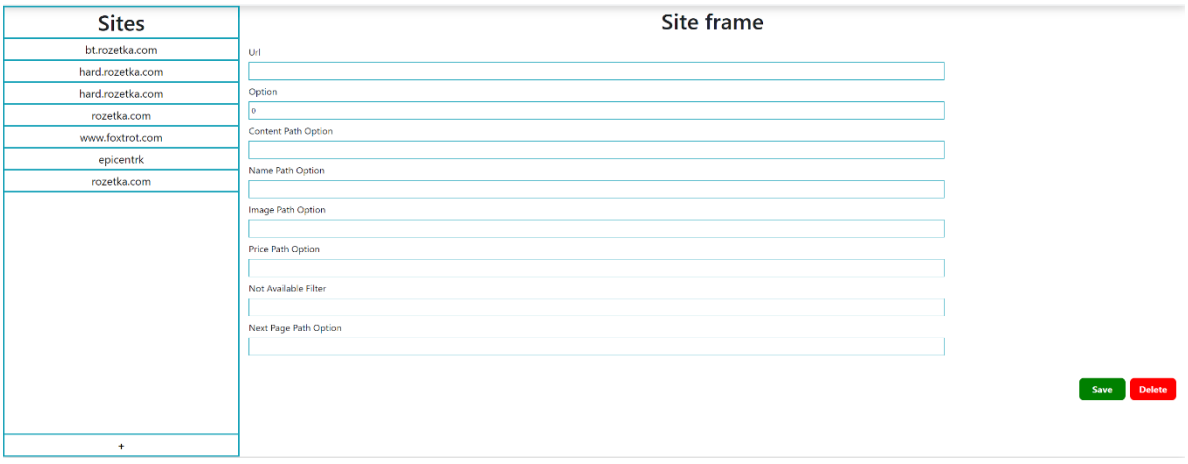


Рис. 14. Екран додавання нових фреймів

Зміна існуючих фреймів рис 15.1 - 15.3. Натиснувши на назву сайту на панелі зліва, можна перейти до режиму редагування її фрейма.

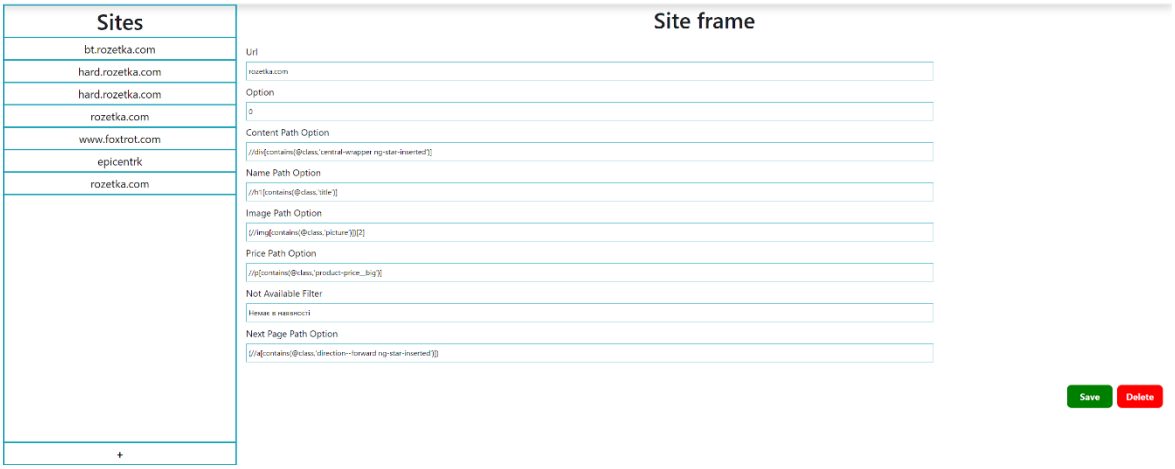


Рис. 15.1. Збираємо існуючий фрейм

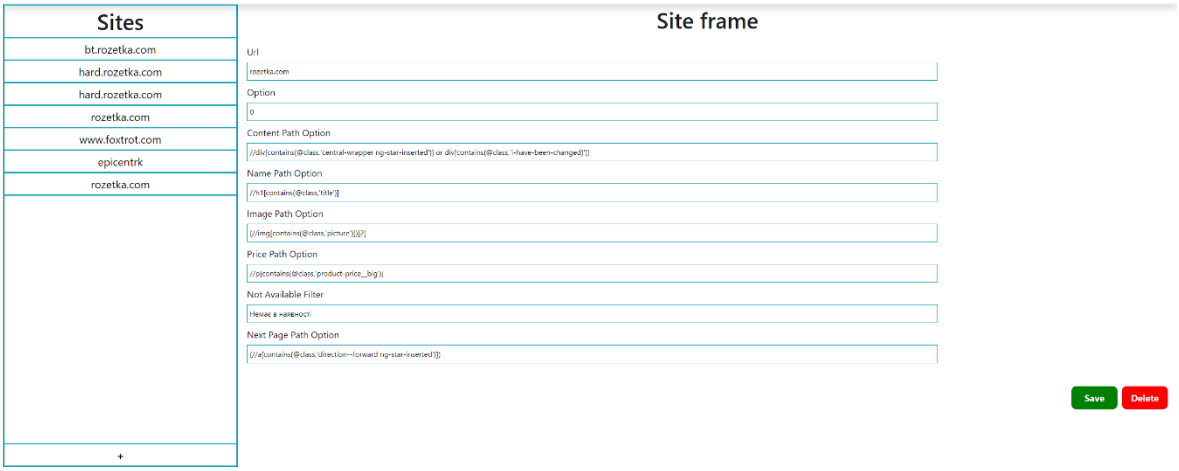


Рис. 15.2. Змінюємо і зберігаємо фрейм

688C8FEA-8BCD-4628-A298-08DB64E9312A	rozetka.com	0	//div[contains(@class,'central-wrapper ng-star-inserted')] or div[contains(@class,'i-have-been-changed')]

Рис. 15.3. Запис зміненого фрейма у базі даних

ВИСНОВКИ

Результатом виконання дипломної роботи став сервіс для порівняння цін на товари запропонованих у інтернет-магазинах, який в основі своєї роботи має закладений алгоритм парсингу.

На мою думку, це дослідження пройшло успішно і мені вдалось досягти мети - я створив алгоритм, який може виконувати пошук товарів за назвами найрізноманітніших товарів. При цьому, варто зауважити, що виконуючи парсинг в реальному часі(одразу після запиту), велику роль грає діапазон сайтів і їх сторінок, адже чим більшу кількість джерел ми задаємо, тим довше буде виконуватись пошук і навпаки, тому в цьому питанні необхідно витримати баланс. Вирішенням такої дилеми може стати використання бази даних для збереження пошукових запитів, але розміри сховища мають бути достатньо великими і записи мають регулярно оновлюватись. Інакшим вирішенням цієї проблеми, може стати виклик багатьох екземплярів WebDriver одночасно, але при такому навантажені машині, на якій буде запускатись сервіс необхідно мати великий запас швидкої оперативної пам’яті.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Steve Smith. Overview of ASP.NET Core MVC. https://docs.microsoft.com/uk-ua/aspnet/core/mvc/overview?view=aspnetcore-6.0

2. Daniel Roth, Rick Anderson & Shaun Luttin. Overview to ASP.NET Core. https://docs.microsoft.com/uk-ua/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-6.0

3. Himanshu Sheth. Scrapping Dynamic Web Pages Using Selenium And C#. https://www.lambdatest.com/blog/scraping-dynamic-web-pages/

4. Andy Feng. Grab and Parse Data using Selenium WebDriver Part 1 – Selenium Fundamentals. https://www.codeproject.com/Articles/1053815/Grab-and-Parse-Data-using-Selenium-WebDriver-Part

5. MVC Mozilla Developer https://developer.mozilla.org/en-US/docs/Glossary/MVC

6. Entity Framework Core documentation https://docs.microsoft.com/uk-ua/ef/core/

7. MS SQL Server documentation https://docs.microsoft.com/uk-ua/sql/?view=sql-server-ver16

8. Complete Guide For Using XPath In Selenium With Examples https://www.lambdatest.com/blog/complete-guide-for-using-xpath-in-selenium-with-examples/