**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ**

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ КОМП’ЮТЕРНИХ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ПРИКЛАДНОЇ ІНФОРМАТИКИ

ДОПУСТИТИ ДО ЗАХИСТУ

Завідувач випускової кафедри

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_В.П. Гамаюн

«\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_2021 р.

**ДИПЛОМНА РОБОТА**

**(ПОЯСНЮВАЛЬНА ЗАПИСКА)**

**ВИПУСНИКА ОСВІТНЬОГО СТУПЕНЯ МАГІСТРА**

**ЗА ОСВІТНЬО-ПРОФЕСІЙНОЮ ПРОГРАМОЮ**

**«ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ ПРОЕКТУВАННЯ»**

**Тема:** **«Програмне забезпечення для збору та аналізу даних з веб сайтів»**

**Виконавець:** Петров Андрій Васильович

**Керівник:** к.т.н. доц. Сінько Юрій Іванович

**Нормоконтролер:**  к.т.н. доц. Боровик Володимир Миколайович

Київ 2021

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ | | | | |
|  | | | | |
| Навчально-науковийінститут | | | *комп’ютерних інформаційних технологій* | |
| Кафедра | *Прикладної інформатики* | | | |
| Спеціальність | | *122 «Комп’ютерні науки»* | | |
| Освітньо-професійна програма: | | | | *«Інформаційні технології проектування»* |

**ЗАТВЕРДЖУЮ**:

Завідувач кафедри ПІ

\_\_\_\_\_\_\_ проф. В.П. Гамаюн

(підпис)

«\_\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_2021 р.

ЗАВДАННЯ

на виконання дипломної роботи

|  |
| --- |
| Петрова Андрія Васильовича |
| (ПІБ випускника ) |

1. Тема роботи «Програмне забезпечення для збору та аналізу даних з веб сайтів» затверджена наказом ректора від « » 2021 р. №

2. Термін виконання роботи: з 04.10.2021 по 24.12.2021

3. Вихідні дані до роботи: розробка програмного забезпечення для збору даних

4. Зміст пояснювальної записки:

1. Аналіз програмного забезпечення для збору інформації та принципи його роботи.
2. Вимоги до програмного забезпечення.
3. Проектування програмного забезпечення.
4. Технічні рішення.
5. Прототип програмного забезпечення для збору інформації.

5. Перелік обов’язкового ілюстративного матеріалу:

1. Актуальність та практична цінність.
2. Задачі дипломного проекту;
3. Основні функції ПЗ для збору інформації.
4. Архітектура ПЗ.
5. Структура ПЗ.
6. Діаграма класів.
7. Обрані технології.
8. Інтерфейс користувача. Головне вікно.
9. Інтерфейс користувача. Вікно перегляду даних.
10. Інтерфейс користувача. Вікно експорту даних.
11. Висновки.

6. Календарний план-графік

|  |  |  |  |
| --- | --- | --- | --- |
| № з/п | Завдання | Термін виконання | Підпис керівника |
| 1. | Ознайомлення з постановкою задачі та вивчення літератури  Написання 1 розділу, представлення керівнику | 04.10.21 – 18.10.21 |  |
| 2. | Написання 2 розділу, представлення керівнику | 18.10.21 - 01.11.21 |  |
| 3. | Написання 3 розділу, представлення керівнику | 01.11.21 – 15.11.21 |  |
| 4 | Написання 4 та 5 розділів, представлення керівнику | 15.11.21 – 07.12.21 |  |
| 5. | Загальне редагування та друк пояснювальної записки, графічного матеріалу | 07.12.21 – 13.12.21 |  |
| 6. | Проходження нормоконтролю, перепліт пояснювальної записки. | 14.12.21 – 17.12.21 |  |
| 7. | Розробка тексту доповіді. Оформлення графічного матеріалу для презентації | 17.12.21 – 20.12.21 |  |

|  |  |
| --- | --- |
| 7. Дата видачі завдання | 04.10.2021 |

|  |  |  |
| --- | --- | --- |
| Керівник дипломної роботи |  | Сінько Ю.І. |
|  | (підпис керівника) | |
| Завдання прийняв до виконання |  | Петров А.В. |
|  | (підпис випускника) | |

**РЕФЕРАТ**

Пояснювальна записка до дипломної роботи «Програмне забезпечення для збору та аналізу даних з веб сайтів»: 81 с., 35 рис., 6 табл., 12 інформаційних джерел.

**Об’єкт дослідження** – програмне забезпечення для збору та аналізу даних з веб сайтів.

**Мета роботи** – розроблення програмного забезпечення для збору інформації з використанням існуючих технологій на мові Java.

Результати магістерської роботи рекомендується використовувати під час проведення наукових досліджень та в практичній діяльності фахівців із інформаційних технологій проектування.

ЗБІР ДАНИХ, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, ВЕБ САЙТ, АНАЛІЗ ДАНИХ.

**ЗМІСТ**

[ПЕРЕЛІК ПРИЙНЯТИХ СКОРОЧЕНЬ 7](#_Toc89645893)

[ВСТУП 8](#_Toc89645894)

[РОЗДІЛ 1 АНАЛІЗ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ЗБОРУ ІНФОРМАЦІЇ ТА ПРИНЦИПИ ЙОГО РОБОТИ 9](#_Toc89645895)

[1.1. Застосування програмного забезпечення для збору інформації 9](#_Toc89645896)

[1.2. Переваги та недоліки парсингу 10](#_Toc89645897)

[1.3. Типи парсерів 11](#_Toc89645898)

[1.3.1. Хмарні парсери 11](#_Toc89645899)

[1.3.2. Десктопні парсери 12](#_Toc89645900)

[1.3.3. Браузерні розширення та плагіни для Excel 12](#_Toc89645901)

[1.3.4. Google Таблиці 12](#_Toc89645902)

[1.4. Види парсерів по сферам застосування 14](#_Toc89645903)

[1.4.1. Для організаторів спільних покупок 14](#_Toc89645904)

[1.4.2. Парсери цін конкурентів 15](#_Toc89645905)

[1.4.3. Парсери для швидкого наповнення сайтів 15](#_Toc89645906)

[1.4.4. Парсери для SEO-фахівців 15](#_Toc89645907)

[1.5. Основні проблеми при створенні власного програмного забезпечення для збору інформації 16](#_Toc89645908)

[1.5.1. Складна та змінна структура веб-сторінок 16](#_Toc89645909)

[1.5.2. Блокування IP 16](#_Toc89645910)

[1.5.3. Динамічний контент 17](#_Toc89645911)

[1.5.4. Пастки Honeypot 17](#_Toc89645912)

[1.5.5. Повільна/нестабільна швидкість завантаження 18](#_Toc89645913)

[1.5.6. Вимога входу 18](#_Toc89645914)

[1.5.7. Вилучення даних у режимі реального часу 18](#_Toc89645915)

[1.5.8. Перевірка за допомогою CAPTCHA 18](#_Toc89645916)

[1.6. Методи та загальний алгоритм роботи 19](#_Toc89645917)

[Висновки 22](#_Toc89645918)

[РОЗДІЛ 2 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 24](#_Toc89645919)

[2.1. Вимоги до апаратної платформи 24](#_Toc89645920)

[2.2. Вимоги до операційної системи 25](#_Toc89645921)

[2.3. Вимоги до швидкодії парсера 26](#_Toc89645922)

[2.4. Вимоги до надійності 26](#_Toc89645923)

[2.5. Вимоги до безпеки 27](#_Toc89645924)

[2.6. Вимоги до інтерфейсу 28](#_Toc89645925)

[2.7. Функціональні вимоги 29](#_Toc89645926)

[2.8. Вимоги до структури документів, що будуть формуватися 30](#_Toc89645927)

[2.9. Вимоги, що пов’язані з існуючими державними та міжнародними стандартами і законами 31](#_Toc89645928)

[Висновки 32](#_Toc89645929)

[РОЗДІЛ 3 ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 34](#_Toc89645930)

[3.1. Архітектура програмного забезпечення 34](#_Toc89645931)

[3.2. Діаграми взаємодії 37](#_Toc89645932)

[3.3. Діаграми класів 41](#_Toc89645933)

[3.4. Реляційна модель бази даних 51](#_Toc89645934)

[Висновки 52](#_Toc89645935)

[РОЗДІЛ 4 ТЕХНІЧНІ РІШЕННЯ 54](#_Toc89645936)

[4.1. Вибір мови програмування 54](#_Toc89645937)

[4.2. Вибір фреймворку для каркасу ПЗ 55](#_Toc89645938)

[4.3. Вибір інструменту для інтерфейсу 58](#_Toc89645939)

[4.4. Вибір бази даних 59](#_Toc89645940)

[4.5. Інші інструменти та бібліотеки 60](#_Toc89645941)

[4.6. Інструмент для автоматичного збирання та пакування проекту 63](#_Toc89645942)

[Висновки 64](#_Toc89645943)

[РОЗДІЛ 5 ПРОТОТИП ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ЗБОРУ ІНФОРМАЦІЇ 66](#_Toc89645944)

[5.1. Конфігурація проекту 66](#_Toc89645945)

[5.1.1. Інтеграція програмних інструментів у проект 66](#_Toc89645946)

[5.1.2. Налаштування підключення до БД 67](#_Toc89645947)

[5.1.3. Налаштування типу застосунку 69](#_Toc89645948)

[5.2. Прототип інтерфейсу додатка 69](#_Toc89645949)

[5.2.1. Створення інтерфейсу 69](#_Toc89645950)

[5.2.2. Головне вікно додатку 70](#_Toc89645951)

[5.2.3. Вікно експорту 74](#_Toc89645952)

[5.2.4. Діалогові вікна 75](#_Toc89645953)

[Висновки 75](#_Toc89645954)

[ВИСНОВКИ 77](#_Toc89645955)

[СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ 81](#_Toc89645956)

# **ПЕРЕЛІК ПРИЙНЯТИХ СКОРОЧЕНЬ**

ПЗ – програмне забезпечення

ОС – операційна система

ООП – об’єктно-орієнтоване програмування

БД – база даних

XML – Extensible Markup Language

HTML – HyperText Markup Language

DDoS – Distributed Denial of Service

HTTP – Hypertext Transfer Protocol

JVM – Java Virtual Machine

ORM – Object-Relational mapping

REST – representational state transfer

SQL – Structured Query Language

JSON – JavaScript Object Notation

XSS – Cross-site scripting

POJO – Plain Old Java Object

# **ВСТУП**

Кожна людина хоча б один раз за своє життя шукала та збирала якусь інформацію в інтернеті. Можливо це відбувалося з метою вибору якогось товару або для перевірки достовірності даних на різних веб ресурсах. Проте не всі задумуються про автоматизацію такого збору. В деяких випадках це може з економити багато нашого часу та полегшить аналіз інформації.

Збір даних з метою подальшої аналітики або парсинг – це метод індексування даних з наступним перетворенням їх в інший тип даних або формат.

Даний метод дозволяє інформацію або файл в одному форматі перетворити в форму простішу для сприйняття, яку після цього можна використовувати в певних цілях. Наприклад, за допомогою парсингу HTML-файл можна легко трансформувати в «чистий» текст і таким чином зробити його максимально зручним для читання. Або ж перетворити в JSON і зробити зручним для використання у програмах та скриптах.

В даній роботі поняття парсинг розглядається в більш вузькій області, а саме збір даних з веб-сайтів. Тому йому можна дати точніше визначення. Отже, парсинг – це процес та метод обробки даних отриманих з веб-сторінок. Даний процес включає в себе аналіз тексту, екстракцію звідти необхідної інформації і її перетворення в заздалегідь зазначений формат, який можна використовувати відповідно до поставлених цілей. Завдяки парсингу можна знаходити на сторінках невеликі клаптики корисної інформації і в автоматичному режимі їх звідти витягувати, щоб потім перевикористати.

# **РОЗДІЛ 1 АНАЛІЗ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ЗБОРУ ІНФОРМАЦІЇ ТА ПРИНЦИПИ ЙОГО РОБОТИ**

Аналіз програмного забезпечення для збору інформації та принципи його роботи

9

15

*Літ.*

*Керівник.*

Сінько Ю.І.

*Розроб.*

Петров А.В.

НАУ 16 12 52 000 ПІ

Кафедра ПІ

*Лист*

*Листів*

ТП-215М – 6.050101

*Н-кнтр. Боровик* В.М.

## **Застосування програмного забезпечення для збору інформації**

Програмне забезпечення для збору та аналізу інформації або парсер – це програма, сервіс або скрипт, який збирає дані з зазначених веб-ресурсів, аналізує їх і повертає в потрібному форматі.

Парсери можуть виконувати різні корисні завдання:

* аналіз цінових діапазонів товарів. Це є актуальне завдання для інтернет-магазинів. Наприклад, за допомогою парсингу можна відслідковувати ціни конкурентів по певним товарам через певні проміжки часу. Також можна його використовувати для актуалізації цін на власному сайті відповідно до цін постачальника, розміщених на його веб-ресурсі;
* екстракція інформації про товари, а саме назви, артикули, описи, відгуки і тому подібне. Наприклад, якщо у постачальника є веб-сайт з каталогом товарів, але немає змоги імпортувати їх для власного онлайн магазину, то можна просто витягнути (спрарсити) всі потрібні товари, а не додавати їх вручну, що економить велику кількість часу;
* збір метаданих – SEO-фахівці можуть аналізувати вміст тегів title, description та інші метадані;
* аналіз та перевірка сайту на помилки. Це завдання буде корисним для тестувальників. Таким чином можна знаходити сторінки з помилками, наприклад з 404 – Not Found статус кодом, з надлишковими переспрямуваннями на інші сторінки, непрацюючі посилання і тому подібне.

Також варто зауважити, що є ще так званий сірий парсинг. Основна відмінність від білого полягає у скачуванні контенту конкурентів або копіювання цілих веб-сайтів.

Також сюди можна віднести збір контактних даних з агрегаторів і сервісів з метою здійснення спам-розсилок і дзвінків. В даній роботі мова йде тільки про дозволений, як з законодавчої так і з етичної точки зору, білий парсінг.

## **Переваги та недоліки парсингу**

В автоматичного збору інформації є багато переваг, особливо якщо порівнювати з ручним збором даних. Основні з них наведено нижче:

* програма працює самостійно, тому не потрібно витрачати час на пошук і сортування даних. При цьому збір даних відбувається значно швидше, ніж це робить людина. Працювати програма може також довше за людину та відповідно з меншою кількістю помилок. При правильному налаштуванні програмного забезпечення його швидкість може бути в 1000 разів більшою;
* можна передати таку кількість вхідних параметрів, скільки буде потрібно для ідеального налаштування необхідного контенту у його роботі, без сміття, помилок, надлишкової та нерелеватної інформації з невідповідних сторінок;
* на відміну від людини, парсер не допускатиме помилок через втому або неуважність;
* програмне забезпечення для парсингу може повертати знайдені дані в зручному для користувача форматі або форматі необхідному для їх подальшої обробки;
* розподілене навантаження на сайт. Це означає, що парсер, ненароком не виведе з ладу чужий ресурс через надмірне навантаження, а його власник не звинуватить в незаконній DDoS-атаці.

Тому немає ніякого сенсу збирати інформацію руками, коли можна довірити це завдання програмному забезпеченню.

Головний недолік парсерів полягає в тому, що ними не завжди вдається скористатися. Наприклад, коли власники чужих сайтів запобігають автоматичному збору інформації з веб-сторінок. Є відразу кілька методів перешкоджанню роботи парсерів такі, як блокування за IP-адресою та за допомогою налаштувань для пошукових роботів. Всі вони є досить ефективними проти парсингу.

До недоліків методу можна віднести і те, що конкуренти теж можуть використовувати його. Щоб захистити сайт від парсинга, доведеться вдатися до однієї з технік:

* або заблокувати запити, вказавши відповідні параметри в robots.txt;
* або налаштувати каптчу. Навчити програмне забезпечення їх вирішувати занадто витратно.

Але всі методи захисту можна обійти, тому, швидше за все, доведеться з цим явищем миритися.

## **Типи парсерів**

### **Хмарні парсери**

Хмарні парсери – це парсери, які працюють в хмарному середовищі. Основна перевага хмарних парсерів – не потрібно нічого завантажувати і встановлювати на комп'ютер. Вся робота проводиться «в хмарі». Все що необхідно – це завантажити результати роботи алгоритмів. В таких парсерів може бути веб-інтерфейс і / або API, що дозволить налашувати його частоту використання або інтегрувати у інше програмне забезпечення [1].

Прикладом хмарних парсерів можуть бути:

* Import.io;
* Mozenda;
* Octoparce;
* ParseHub;
* Xmldatafeed;
* Діггернаут;
* Catalogloader.

Кожний з сервісів, які наведені вище, включають безкоштовну free-версію. Цим можна скористатися для тестування сервісу. Однак вона має певні обмеження за обсягом парсинга даних, або за часом користування. Тому її вистачить лише для того, щоб оцінити базові можливості і ознайомитися з базовим функціоналом.

### **Десктопні парсери**

Десктопні парсери на відміну від хмарних вимагають їх завантаження на локальне апаратне забезпечення. Більшість десктопних парсерів розроблені під Windows – на macOS їх необхідно запускати з віртуальних машин. Також деякі парсери мають портативні версії – можна запускати з флешки або зовнішнього накопичувача.

Популярні десктопні парсери:

* ParserOK;
* Datacol;
* Screaming Frog, ComparseR, Netpeak Spider.

### **Браузерні розширення та плагіни для Excel**

Також існують багато браузерних розширень, які збирають необхідні дані з вихідного коду сторінок і дозволяють зберігати їх в зручному форматі (наприклад, в XML або XLSX).

Парсери-розширення – це хороший варіант, якщо вам потрібно збирати невеликі обсяги даних (з одного або декількох сторінок). Ось популярні розширення для Google Chrome:

* Parsers;
* Scraper;
* Data Scraper;
* Kimono.

Пасери можуть бути у вигляді плагінів для Microsoft Excel. Прикладом такого є ParserOK. У своїх алгоритмах вони використовують макроси, а результати збору даних відразу вивантажуються в файли з форматами XLS або CSV.

### **Google Таблиці**

Google таблиці дають можливість збирати будь-яку інформацію з веб-сайтів безкоштовно за допомогою двох нескладних формул: IMPORTXML і IMPORTHTML.

IMPORTXML – це функція, яка використовує мову запитів XPath і дозволяє аналізувати довільні дані з XML-фідів, HTML-сторінок та інших джерел. Ось так виглядає функція:

IMPORTXML (<Посилання>; <XPath>)

Функція приймає два параметри:

* посилання на сторінку або фід, з якого потрібно отримати дані;
* XPath-запит (спеціальний запит, який вказує, який саме елемент з даними потрібно обробити).

Перевагою використання цієї формули є те, що для її застосуання не обов'язково знати синтаксис XPath-запитів. Щоб отримати XPath-запит для елемента з даними, потрібно відкрити інструменти розробника в браузері, натиснути правою кнопкою миші на потрібному елементі та вибрати: Копіювати/Копіювати XPath, як зображено на рисунку 1.1.

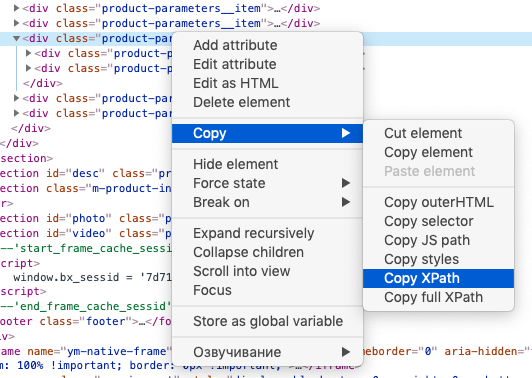


Рис. 1.1. Копіювання XPath потрібного елементу

За допомогою IMPORTXML можна збирати практично будь-які дані з html-сторінок: заголовки, описи, мета-теги, ціни і тому подібне. Даний метод має один значний недолік – неможливість витягнути дані з динамічних елементів, тобто тих, які створюються за допомогою скриптів в процесі формування веб-сторінки.

Приклад застосування формули:

IMPORTXML ("https://site.com/catalog"; "// a / @ href")

Функція IMPORTHTML має менше можливостей, ніж попередня. З її допомогою можна зібрати дані з таблиць або списків на сторінці. Ось приклад функції IMPORTHTML:

IMPORTHTML (<Посилання>; <Тег елементу>; <Порядковий номер>)

Функція приймає три параметри:

* посилання на сторінку, з якої необхідно зібрати дані;
* параметр елемента, який містить потрібні дані. Якщо хочете зібрати інформацію з таблиці, вкажіть «table», для парсинга списків – параметр «list»;
* число – порядковий номер елемента в коді сторінки.

Приклад застосування функції:

IMPORTHTML ("https: // https: //site.com/catalog/sweets"; "table"; 4)

## **Види парсерів по сферам застосування**

Парсери можуть мати досить широку область застосування. Нижче наведено основні.

### **Для організаторів спільних покупок**

Є спеціалізовані парсери для організаторів спільних покупок. Їх встановлюють на свої сайти виробники товарів (наприклад, одягу або взуття). І будь-хто може прямо на сайті скористатися парсером і вивантажити весь асортимент.

Чим зручні ці парсери:

* інтуїтивно зрозумілий інтерфейс;
* можливість вивантажувати окремі товари, розділи або весь каталог;
* можна вивантажувати дані в зручному форматі;

Популярні парсери для СП:

* SPparser.ru;
* Турбо.Парсер;
* PARSER.PLUS;
* Q-Parser.

### **Парсери цін конкурентів**

Інструменти для інтернет-магазинів, які хочуть регулярно відслідковувати ціни конкурентів на аналогічні товари. За допомогою таких парсерів можна вказати посилання на ресурси конкурентів, зіставляти їх ціни з вашими і коригувати при необхідності.

Ось три таких інструменти:

* Marketparser;
* Xmldatafeed;
* ALL RIVAL.

### **Парсери для швидкого наповнення сайтів**

Сервіси, які використовують такі парсери збирають необхідну інформацію, наприклад назви товарів, описи, ціни, зображення та інші дані з «сайтів-донорів». Потім дані вивантажуються в файли або відразу завантажуються на необхідний сайт. Такий підхід може використовуватися для тестування новоствореного сайту. В порівнянні з ручною роботою він працює на порядок швидше та економить багато часу.

У подібних парсерах можна автоматично додавати власну інформацію або обробляти поточну. Наприклад, можна додавати певне число до ціни товару (якщо дані були зібрані з сайту постачальника, який пропонує оптові ціни). Також можна налаштовувати автоматичний збір або оновлення даних по розкладу.

Приклади таких сервісів:

* Catalogloader;
* Xmldatafeed;
* Діггернаут.

### **Парсери для SEO-фахівців**

Вузько- або багатофункціональні програми, створені спеціально під рішення задач SEO-фахівців – це окрема категорія парсерів. Вони призначені для спрощення комплексного аналізу оптимізації сайту. З їх допомогою можна:

* аналізувати вміст robots.txt і sitemap.xml;
* перевіряти наявність title і description на сторінках сайту, аналізувати їх довжину, збирати заголовки всіх рівнів (h1 – h6);
* перевіряти статус коди відповіді сторінок;
* збирати і візуалізувати структуру сайту;
* перевіряти наявність описів зображень (атрибут alt);
* аналізувати внутрішні переадресування і зовнішні посилання;
* знаходити непрацюючі посилання;
* і багато іншого.

## **Основні проблеми при створенні власного програмного забезпечення для збору інформації**

### **Складна та змінна структура веб-сторінок**

Більшість веб-сторінок базуються на HTML (мова розмітки гіпертексту) [2]. Розробники веб-сторінок можуть мати власні стандарти для оформлення сторінок, тому структури веб-сторінок можуть мати суттєві відмінності (іноді навіть в межах одного веб-сайту). Якщо потрібно зібрати дані з кількох веб-сайтів, то скоріше за все необхідно буде створювати окремий парсер для кожного веб-сайта.

Крім того, веб-сайти періодично оновлюють вміст сторінок, щоб покращити user experience або додати нові функції, що може призвести до структурних змін на веб-сторінці. Оскільки парсери налаштовані відповідно до певної структури сторінки, вони не працюватимуть для оновленої сторінки. Іноді навіть незначна зміна цільового веб-сайту вимагає повторного налаштування парсера.

Повністю вирішити цю проблему неможливо. Проте можливо мінімізувати переналаштування парсера після таких змін. Для цього необхідно правильно визначити методи збору інформації, яка може бути доступною в декількох місцях.

### **Блокування IP**

Блокування IP – це поширений метод, який запобігає парсерам отримувати доступ до даних веб-сайту по певній IP-адресі. Зазвичай це відбувається, коли веб-сайт виявляє велику кількість запитів з однієї і тієї ж IP-адреси. Веб-сайт або перманентно блокує IP (в такому випадку необхідна її заміна), або обмежує доступ на певний період часу (в такій ситуації процес збору інформації буде відновлений після зняття блокування IP-адреси).

Для вирішення даної проблеми можна скористатися IP-проксі сервісами, які частіше є платними, або використовувати безкоштовні IP-проксі, які можуть бути знайденими на просторах інтернету. Існує багато сервісів IP-проксі, таких як Luminati, які можна інтегрувати з автоматичними парсерами. Ще одним методом боротьби з такою проблемою – це обмежити кількість запитів до веб-сайту, які робить парсер до природньої, тобто такої, яку може робити людина без використання будь-якого програмного забезпечення. Наприклад, один запит через кожні дві-три секунди.

### **Динамічний контент**

Багато веб-сайтів використовують динамічний контент, який завантажується не одразу при переході на веб-сторінку, а при настанні певної умови. Частіше за все для цього використовується AJAX. Прикладами є ледаче завантаження зображень, нескінченна прокрутка та відображення додаткової інформації при натисканні кнопки. Це робить використання сайту зручнішим для користувачів, але не для парсерів.

В даної проблеми є декілька можливих вирішень, наприклад, здійснення додаткових запитів до веб-сайтів або застосування selenium, який використовується для автоматизованої роботи в браузері, в розробці програмного забезпечення.

### **Пастки Honeypot**

Honeypot – це пастка, яку власник веб-сайту ставить на сторінку, щоб зловити програмне забезпечення для збору даних. Пастки можуть бути непомітними для людей, але небезпечними для роботи парсерів. Як тільки парсер потрапляє у пастку, веб-сайт може використовувати отриману інформацію (наприклад, його IP-адресу), щоб заблокувати його.

### **Повільна/нестабільна швидкість завантаження**

Веб-сайти можуть повільно реагувати або навіть не завантажуватися, при занадто великій кількості запитів, які надходять до нього. Це не проблема, коли люди переглядають сайт, оскільки їм просто потрібно перезавантажити веб-сторінку і чекати, поки веб-сайт відновиться. Але процес парсингу може бути перерваним, оскільки він не знає, як впоратися з такою надзвичайною ситуацією.

Для вирішення даної проблеми необхідно імплементувати механізм повторів запитів на такі веб-сторінки. Це симуляція їх оновлення. Також варто забезпечити такий алгоритм роботи програми, при якому після невдалої N спроби, процес парсингу не переривався.

### **Вимога входу**

Деякі захищені дані можуть вимагати авторизації до системи. Після того, як користувач буде зареєстрований та авторизований, до запитів будуть додані cookie, значення яких розпізнає веб-сайт, тому він знатиме, що особа щойно увійшла в систему. Тому потрібно звертати увагу що файли cookie надісилаються разом із запитами, під час роботи з такими веб-сайтами.

### **Вилучення даних у режимі реального часу**

Вилучення даних в режимі реального часу має важливе значення для порівняння цін, відстеження наявності тощо. Дані можуть змінюватися в один момент і можуть призвести до величезного приросту капіталу для бізнесу. Парсер повинен постійно контролювати веб-сайти та збирати дані. Незважаючи на це, він все ще має деяку затримку, оскільки запит та доставка даних потребують часу. Крім того, отримання великої кількості даних у режимі реального часу також є великою проблемою.

### **Перевірка за допомогою CAPTCHA**

CAPTCHA – це повністю автоматизований публічний тест Тьюринга для розпізнавання комп’ютерів та людей окремо, який часто використовується для розпізнавання програмного забезпечення, яке зстосовується для збору інформації або автоматизованих дій, шляхом відображення зображень чи логічних проблем, які людям легко вирішити, а програмному забезпеченню – ні. Приклад зображений на рисунку 1.2.

Для вирішення можна інтегрувати розв'язувач CAPTCHA в парсер. Це забезпечить безперервний процес парсингу. Однак технології подолання CAPTCHA можуть дещо уповільнити цей процес.



Рис. 1.2. Приклад CAPTCHA

## **Методи та загальний алгоритм роботи**

Існують два основні методи парсинга: низхідний і висхідний. Зазвичай вони відрізняються порядком, в якому веб-сторінки будуть оброблятися.

Зверху-вниз: при низхідному методі парсер перевіряє наявність зверху – з початкової сторінки і шукає відповідні елементи та посилання на ній. Таким чином, дерево сторінок розростається зверху вниз.

Знизу-вгору: висхідний парсер починає обробку знизу, з нижнього рівня дерева. Це робиться до тих пір, поки не буде досягнуто початкової сторінки.

Метод роботи – це не найважливіше. Добре спроектоване програмне забезпечення для збору інформації – висхідне чи низхідне – буде розрізняти, яка інформація, наприклад, в HTML є важливою та необхідною.

Варто відзначити, що парсер не прив'язаний до певного формату даних. Це просто інструмент, який перетворює одне представлення даних в інше. Як він його перетворює, залежить від поточних завдань.

Парсер працює наступним чином: він аналізує сторінку на наявність необхідної інформації, а потім витягує її, перетворивши в систематизовані дані.

Процес пошуку і вилучення знайденої інформації для парсера виглядає так:

1. Спочатку користувач вказує вхідні дані та параметри для парсинга на сайті. Це може бути як необхідна інформація, так і певне ключове слово для пошуку на сайті.
2. Потім вказує список сторінок або ресурсів, на яких потрібно здійснити пошук.
3. Після цього програма в автоматичному режимі проводить глибокий аналіз знайденого контенту і систематизує його.
4. В результаті користувач отримує звіт в заздалегідь обраному форматі.

Процес збору інформації за допомогою спеціалізованого програмного забезпечення описана лише в загальних рисах. Для кожного вона буде виглядати по-різному. Також на процес роботи з парсером впливають цілі, переслідувані користувачем.

Під глибоким аналізом мається на увазі пошук посилань на інші веб-сторінки для їх подальшого аналізу та розбору. Це може бути, напрклад, посилання на наступні пагінаційні сторінки або ж сторінки, які містять додаткову необхідну інформацію, наприклад опис або відгуки.

На даний момент є три основні види пагінації, які реалізуються на різних веб-сайтах. Серед них:

* безкінечна прокрутка (infinite scroll);
* наступна сторінка;
* будь-яка сторінка.

При безкінечній прокрутці інформація завантажується на сайт автоматично порціями у випадку коли на екрані користувача відображено від 90% до 100% інформації. Зазвичай при цьому самостійно генерується та відправляється на сервер ресурсу додатковий запит для її підвантаження, починаючи з останнього елементу, який переглядався, або ж всі ці дані можуть зберігатися у зашифрованому форматі в джерелі сторінки. Також користувач може бачити додаткові вказівки на інтерфейсі, яка вказують процес завантаження нових даних. Це зображено на рисунку 1.3, де в даному випадку це три крапки.

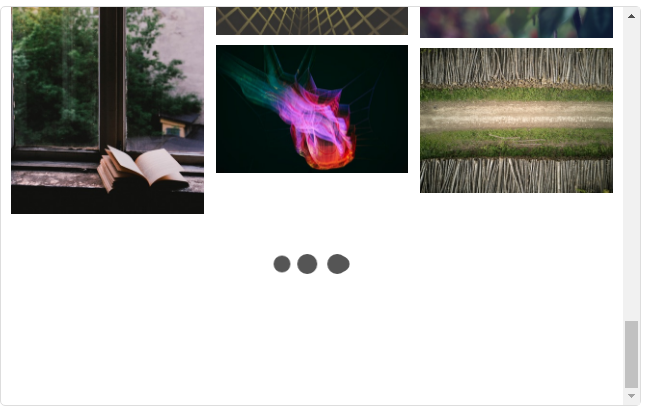


Рис. 1.3. Безкінечна прокрутка

Даний вид пагінації є найважчим у реалізації, оскільки для цього треба знайти, звідки підвантажуються нові дані, що не в усіх випадках є просто.

Другий тип пагінації, а саме наступна сторінка, дозволяє отримати посилання лише на наступну сторінку. Це є набагато простіше від попереднього типу. Зазвичай вона відображається за допомогою кнопок зі стрілками або з текстом, який означає наступну або попередню сторінку. Приклад інтерфейсу з такою пагінацію зображений на рисунку 1.4.

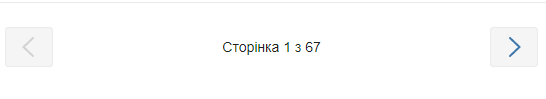


Рис. 1.4. Пагінація виду «наступна сторінка»

Третій тип дозволяє переходити на будь-яку сторінку. Це означає, що можна одразу згенерувати всі необхідні запити на наступні веб-сторінки на відміну від двох попередніх. Приклад інтерфейсу з таким видом пагінації зображений на рисунку 1.5.



Рис. 1.5. Пагінація виду «будь-яка сторінка»

При роботі з парсером весь процес будується навколо параметрів, що вводяться для пошуку і вилучення контенту. Залежно від того, з якою метою планується парсинг, будуть виникати тонкощі у визначенні вхідних параметрів.

## **Висновки**

В даному розділі було детально розглянуто класифікацію, принципи та алгоритми роботи програмного забезпечення для збору та аналізу інформації. Було розглянуто основні завдання, які можуть виконувати парсери. Серед них можна виділити такі:

* аналіз цінових діапазонів товарів;
* екстракція інформації про товари, а саме назви, артикули, описи, відгуки і тому подібне;
* збір метаданих;
* аналіз та перевірка функціонування сайту.

Дане програмне забезпечення має свої недоліки, однак переваги все ж таки перекривають їх. Також при створенні власного парсеру можна зіткнутися з певними перешкодами та проблемами. Серед них складна та змінна структура веб-сторінок, блокування по IP, динамічний контент, перевірка за допомогою CAPTCHA, вимога входу, повільна або нестабільна швидкість завантаження веб-сторінок, пастки honeypot, вилучення даних у режимі реального часу тощо.

Тому при проектуванні та розробці варто звертати увагу на наявність вище згаданих проблем та способи їх вирішення. Також варто пам’ятати, що метод роботи парсера – це не найголовніше, головне – це правильно його налаштувати та визначити необхідні вхідні параметри, які будуть забезпечувати виконання поставлених завдань.

Основні етапи роботи парсера:

1. Визначення вхідних даних та параметрів для парсинга.
2. Вибір списку сторінок або ресурсів, на яких потрібно здійснити пошук.
3. Проводення глибокого аналізу знайденого контенту і його систематизація.
4. Формування звіту на основі зібраної інформації в заздалегідь обраному форматі.

Глибокий аналіз – це пошук посилань на інші корисні джерела. Зазвичай це пагінаційні сторінки. Їх є три основні види:

* безкінечна прокрутка;
* наступна сторінка;
* будь-яка сторінка.

Найважчою для реалізації є перший тип, оскільки потрібно зрозуміти звідки дані підвантажуються.

Отже, використання програмного забезпечення для збору інформації може критично вплинути на виконання певних завдань, з економити велику кількість часу та людських ресурсів, запобігти помилкам, які виявляються за допомогою процеса парсингу або щонайменше допущених через людський фактор при ручному виконанні даної роботи.

# **РОЗДІЛ 2 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ**

Вимоги до програмного забезпечення

24

10

*Літ.*

*Керівник.*

Сінько Ю.І.

*Розроб.*

Петров А.В.

НАУ 16 12 52 000 ПІ

Кафедра ПІ

*Лист*

*Листів*

ТП-215М – 6.050101

*Н-кнтр. Боровик* В.М.

## **Вимоги до апаратної платформи**

Для нормального функціонування програмного забезпечення для збору даних необхідно мати комп’ютер середньої потужності, на якому воно буде запускатися. Запуск та виконання програми буде відбуватися локально, тобто не на віддаленому сервері або хмарному середовищі. За потреби можна використати віддалений сервер, на якому буде розміщена БД. Вона необхідна для збереження зібраних даних.

Апаратна платформа має залишатися актуальною протягом процесу створення програмного забезпечення для збору даних. До платформ, які задовольняють такі вимоги відносяться Intel-64, SPARC, IA-64 та AMD-64.

AMD-64 – апаратна платформа з 64-бітною архітектурою мікропроцесора і відповідний набір інструкцій, яку створила AMD. Intel-64 – це апаратна платформа з 64-бітною архітектурою мікропроцесора, яку створила Intel. Забезпечені вище платформи можуть проводити обчислювання різної бітності одночасно без втрати ресурсів та ефективності, наприклад 32-/64-бітне. Прикладами таких процесорів є , AMD Opteron/FX з архітектурою Bulldozer і Piledriver, EPYC, чотирьох ядерний AMD Athlon ІІ, Ryzen, Threadripper. Core i5, Core i7, Core i9 та Xeon є аналогами таких процесорів в Intel.

Sun Microsystems створила апаратну платформу SPARC. Процесори цього виробника повинні бути V8 версії або вище.

IA-64 належить до 64-бітних апаратних платформ, у якій використана та реалізоана ідея паралельного виконання команд. Вона розроблена співробітництвом компаній Hewlett-Packard та Intel. Такий процесор може сам обирати із всього списку команди, до яких буде застосовано паралелізм.

Прикладами таких процесорів є Itanium 9500 (Poulson), Itanium 9300 (Tukwila) та Itanium 9700 (Kittson). На даний момент останній згаданий процесор є неактуальним, бо знятий з виробництва.

## **Вимоги до операційної системи**

Оскільки програмне забезпечення для збору даних буде створюватися за допомогою крос-платформної мови програмування, то воно є здатним до виконання на всіх можливих ОС, які здатні до встановлення на апаратні платформи, які наводилися у пункті 2.2.

До підходящих ОС можна віднести такі:

* Windows 7;
* Windows 8;
* Windows 10;
* Linux;
* Solaris;
* OpenSolaris;
* FreeBSD;
* OpenBSD;
* NetBSD;
* SunOS.

Підходить будь-який дистрибутив Linux, на який можна встановити віртуальну машину, який буде компілювати код та виконувати його. Необхідно звернути увагу на те, що процесори Intel сьомого покоління і вище, Core i9 та найбільш нові AMD Ryzen не підтримують Windows 7 та 8 версій.

Програмне забезпечення буде орієнтуватися на роботу в ОС Windows та Linux, оскільки вони є найбільш широко використовуваними, а остання ще й безплатною та вільно розповсюджуваною ОС. Запуск додатку на інших ОС може спровокувати розбіжності в управлінні ресурсами пам’яті, які можуть впливати на роботу парсера в цілому.

## **2.3. Вимоги до швидкодії парсера**

Серед вимог до швидкодії парсера можна виділити декілька показників [3]:

* пропускна здатність ПЗ;
* час очікування відповіді;
* час виконання процесу збору інформації з веб сайту.

Пропускна здатність – це кількість даних, оброблюваних програмою за якусь одиницю часу. Кешування даних може бути застосованим при перенавантажені серверу користувачами. Це зможе пришвидшити витягнення інформації з БД.

Час очікування відповіді від системи – це час, який програмне забезпечення витрачає на оброблення вхідних даних та повернення результату користувачу. Даний показник повинен бути меншим за півтори секунди при максимальному навантаженні.

Час виконання процесу збору інформації з веб сайту – це середній час, за який парсер оброблює одну веб сторінку. Порахувати його можна за формулою 2.1, яка наведена нижче:



(2.1)

Де – це час обробки веб сторінок, n – це загальна кількість веб сторінок. Даний час повинен прямувати до нуля.

## **2.4. Вимоги до надійності**

Програмне забезпечення для збору та аналізу даних повинно бути безпечним з функціональної точки зору. Тобто програма має забезпечувати надійність окремих модулів парсера, функцій та команд. При цьому функціональні частини ПЗ мають володіти низькою інтенсивністю відмов [4].

Програма повинна забезпечувати такі високорівневі метрики надійності:

* стійкість – функціонування при помилках, збоях та відмовах;
* безвідмовність – функціонування в конкретному середовиші;
* коректованість – можливість легко знайти та виправити різного типу помилки та збої;
* захищеність від проникнення третіх осіб.

Метрика інтенсивності відмов має бути низькорівневою.

Надійне виконання функцій програми має бути впроваджене за допомогою:

* перевірці правильності вхідних даних, які надходять від користувача, або їх подальшій перевірці в ядрі програми;
* відновлення роботи з останнього збереженого тривкого стану в разі завершення програми в результаті певного збою;
* копіювання інформації з БД через певні інтервали часу з метою їх використання при незвичайній ситуації, в якій справжні дані стають недоступними для ПЗ.

Дванадцять годин – це максимальний термін відновлення парсеру після настання збою.

## **2.5. Вимоги до безпеки**

Веб-додатки більш уразливі за рахунок використання Інтернету та забезпечення багатьом користувачам одночасного доступу до них.

Основні загрози, які не повинні викликати проблем для парсера, включають:

* DDoS атаки;
* XSS атаки;
* SQL-injection;

DDoS – це атака, під час якої на сервер або в даному випадку на програмне забезпечення для збору даних, надсилається велика кількість запитів на виконання функціоналу. Таким чином навантаження стає максимальним протягом деякого періоду часу. Метою цієї атаки є відключення ПЗ. При цьому в користувачів повністю або частково відбирається можливість застосування ПЗ. Основними методами захисту є використання сервісів для запобігання DDoS-атак, а також використання спеціального обладнання та ПЗ для фільтрації трафіку.

XSS – це атака, яка впроваджує або реалізує шкідливий код на сторінки сайту. Коли ці веб-сторіни використовуються, відбувається його запуск та передача даних користувача третій стороні. Щоб усунути цю загрози необхідно використовувати фреймворки або бібліотеки з встроєним валідатором на шкідливий код, який може спровокувати XSS атаку.

Ін’єкція SQL – це метод вставки частини запиту SQL у форму, яка застосовується для введення даних, щоб запустити змінений, некоректний та руйнуючий запит в базі діних. Ціллю цієї атаки є заволодівання даними третьою стороною або ру нування структури БД, видалення та руйнування цілісності даних. Щоб запобігти виконанню цієї атаки необхідно виконувати підставку лише провалідованих параметрів в запити SQL.

## **2.6. Вимоги до інтерфейсу**

Графічний інтерфейс користувача не є обов’язковим для такої програми як парсер. Оскільки всі отримані дані можна одразу завантажувати в файли необхідного формату, наприклад, JSON, DOCS, CSV або звичайний текстовий файл та інші. Проте для підвищення досвіду користування (user experience) необхідно реалізувати інтерфейс користувача. Інтерфейс програмного забезпечення для збору інформації повинен відповідати наступними вимогам [5]:

* адаптованість:

1. задовольняти вимоги користувачів у необхідних функцій;
2. забезпечувати користувачів різноманітні функціональні підказками та інструкції;
3. володіти зручним меню, в якому можна просто перемикатися з однієї функції на іншу;
4. надавати користувачам інформацію про виконання функцій, наприклад збір інформації.

* ясність:

1. доступні користувачеві функції повинні бути чіткими та виконувати лише одну одиницю роботу;
2. сервер повинен однозначно відповідати на запити користувача.

* дружність:

1. швидкий, простий і легкий для навчання користувачів;
2. не перевантажуватися зайвими функціями та деталями;
3. забезпечувати зручну взаємодію між користувачами та системою.

* гнучкість, тобто можливість зміни та налаштування конфігурації інтерфейсу в залежності від вимог користувача;
* продуктивність, тобто витрати часу повинні бути щонайменшими на виконання основних функцій;
* естетичність – інтерфейс повинен бути гарним та привабливим для користувачів кольоровою схемою, естетичністю вигляду його елементів (кнопок, вкладок, полів вводу/виводу), їх розміщення і дизайном в цілому.

## **2.7. Функціональні вимоги**

Весь функціонал парсера можна розділити на дві категорії:

* основні;
* другорядні.

Основні функції – це найважливіші функції програмного забезпечення для збору даних. Без них програма вважається не реалізованою до кінця тому, що сенс парсера зникає.

Другорядні функції розширюють функціонал, значно покращують ПЗ та роблять його зручнішим для застосування. Без другорядних функцій парсер вважається розробленим та готовим до експлуатації. Дані функції можуть бути дороблені та впроваджені під час експлуатації ПЗ.

Основні функції:

* запуск збору даних за допомогою посилання;
* запуск збору даних за допомогою ключового слова;
* вибір необхідних даних для збору;
* перегляд зібраних даних;
* порівняння даних з метою аналізу;
* завантаження даних у файл.

До допоміжних функцій відносяться:

* реєстрація облікового запису;
* вхід у власний обліковий запис;
* перегляд історії запусків збору даних.

## **2.8. Вимоги до структури документів, що будуть формуватися**

Протягом розробки програмного забезпечення для збору та аналізу даних з веб сайтів створюватимуться наступні документи [6]:

* специфікація, яка включає потрібні дані по ПЗ та його документації;
* відомість власників оригіналів – документ, що включає дані фізичних і юридичніих осіб, які мають оригінал ПЗ;
* код програми – документ, що включає повний код програми з потрібними зауваженнями та примітками;
* опис програми – документ, що містить опис логічних та фізичних рівнів ПЗ, список реалізованих функцій та опис того, як веде себе ПЗ при їх виконанні. Він має включати розділи з загальними відомостями, описом логічної структури ПЗ, потрібних технічних пристроїв, а також описомпочаткових та результуючих дані;
* програма і методика тестування, що включає вимоги до програми, на які треба звернути свою увагу при виконанні тестів, а також порядок і способи контролю;
* технічне завдання, що містить опис застосування та область використання ПЗ, незвичайні вимоги до нього, а також період для створення та типи тествувань;
* пояснювальна записка, що містить схеми та основні принципи роботи алгоритмів, що застосовувалися для створення ПЗ, а також включає пояснення про обрані та застосовувані технології.

Повинні формуватися наступні експлуатаційні документи:

* відомість експлуатаційних документів, що включає список експлуатаційних документів;
* формуляр – документ, що включає головні відомості та характеристики програмного забезпечння, а також відомості, які стосуються експлуатації програми;
* опис застосування, що включає пояснення про задачі, що будуть вирішуватися, область застосування та найменший список і налаштування апаратних пристроїв;
* інструкція розробника – знання для розробки програми;
* інструкція користувача – знання користувача для використання програми;
* опис мови – опис синтаксису та семантики обраної для створення ПЗ мови програмування;
* інструкція по технічному обслуговуванню – документ, що включає знання по використанню тестових та діагностичних програм для обслуговування пристроїв, на яких виконується програма.

## **2.9. Вимоги, що пов’язані з існуючими державними та міжнародними стандартами і законами**

Документація програма має бути сформована по стандарту ISO/IEC 6592:2000 або його офіційним українським аналогом ДСТУ 4302:2004. В ньому описані головні аспекти формування документації програмного забезпечення, ліміти, спосіб ведення документації, загальний вигляд характеристики інформаційних елементів та принципи їх детальної характеристики.

Створення програмного забезпечення для збору та аналізу інформації має відбуватися згідно стандартів IEEE, IEC та ISO, які спрямовані на стандартизацію ЖЦ ПЗ.

Група стандартів IEEE характеризує робіти по створенню та супроводу програмного забезпечення. В даних стандартах також охарактеризовано процеси, що вадповідають етапам ЖЦ.

Управління етапами ЖЦ має відбуватися згідно стандарту ISO/IEC/IEEE 12207-1:2017.

Застосування практик Agile та DevOps має відповідати ISO/IEC AWI TR 24586 та ISO/IEC AWI TR 24587 стандартам.

Застосування уніфікованої мови моделювання UML для проектування та створення ПЗ має відповідати ISO/IEC 19793:2015, ISO/IEC 19505-1:2018 та ISO/IEC 19505-2:2018 стандартам.

Оцінювання та керування якістю етапів ЖЦ мають відповідати групі стандартів ISO. При цьому повинна бути використана модель зрілості процесів CMMI, яка орієнтована на покращення процесів ЖЦ. Насамперед це стандарти серій 9000 та 10000:

* ISO 9000:2015 та український аналог ДСТУ ISO 9000:2007 – характеризує головні принципи керування якістю процесів створення ПЗ;
* ISO 9001:2015 та український аналог ДСТУ ISO 9001:2015 – включає сукупність вимог до систем куревання якістю;
* ISO 9004:2015 – характеризує використання системи керування якістю;
* ISO 19011:2018 – характеризує способи проведення аудиту в системах керування якістю.

Закони України на інтелектуальну власність не мають бути порушеними у процесі створення програмного забезпечення для збору та аналізу даних.

## **Висновки**

Програмне забезпечення для збору та аналізу даних буде створюватися за допомогою крос-платформної мови програмування. Тобто парсер зможе працювати на будь-якій апаратній платформі, на якій можливе встановлення та функціонування віртуальної машини. Апаратні платформи, які підходять під задані вимоги: AMD 64, Intel 64, SPARC та IA-64. Процесор локального комп’ютера користувача, на якому буде запускатися ПЗ, повинен володіти та забезпечувати середню потужність та ефективність для безперебійної та швидкої роботи програми при високих навантаженнях. Найбільш потужними серед таких є Intel Core i9 останнього покоління, AMD Ryzen та Itanium 9500.

Парсер може працювати на всіх ОС, які здатні функціонувати на вищезгаданих апаратних платформах. Розробка та тестування програми буде проводитися на Windows та Linux, оскільки вони є найбільш популярними ОС. На інших ОС можливі відмінності в роботі з пам’яттю, які можуть повпливати на функціонування парсера в цілому.

Програма повинна володіти високими показниками швидкодії. Час відгуку повинен залишатися на рівні двох секунд при високих навантаженнях.

Для парсера не мають бути небезпечними такі кібер атаки:

* DDoS атаки;
* XSS атаки;
* ін’єкція SQL;

Для цього необхідно вжити заходів, серед яких основними є перевірка введених даних, застосування пристроїв та сервісів для фільтрування трафіку.

Головними вимогами до графічного інтерфейсу користувача є зрозумілість, дружність, естетичність, адаптованість та гнучкість.

Функції парсера можна розподілити на дві категорії: головні та другорядні. Реалізація головних функцій є обов’язковою на відміну від другорядних. До головних відносяться: запуск збору даних за допомогою посилання, запуск збору даних за допомогою ключового слова, вибір необхідних даних, перегляд зібраних даних, порівняння даних з метою аналізу та завантаження даних у файл. Другорядні функції роблять користування програмою більш зручним. До них відносяться: реєстрація облікового запису, вхід у власний обліковий запис та перегляд історії запусків збору даних.

Також встановлені вимоги до структури документів, які відповідають стандартам ISO, IEC, IEEE. При створенні ПЗ не мають порушуватися закони України.

# **РОЗДІЛ 3 ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ**

Проектування програмного забезпечення

34

20

*Літ.*

*Керівник.*

Сінько Ю.І.

*Розроб.*

Петров А.В.

НАУ 16 12 52 000 ПІ

Кафедра ПІ

*Лист*

*Листів*

ТП-215М – 6.050101

*Н-кнтр. Боровик* В.М.

## **Архітектура програмного забезпечення**

Для програмного забезпечення, яке буде розроблятися, обрано багаторівневу архітектуру з поділом пакетів за функціями. Тобто в ідеальній ситуації кожен пакет з набором класів буде відображати та відповідати за певну функцію в програмному забезпеченні. Вона володіє критично важливими для даного ПЗ перевагами, а саме гнучкістю та простотою внесення змін на різні рівні, які їх потребують [7].

Пакети в даній архітектурі мають високу згуртованість і високу модульність, а також мінімальне зчеплення між пакетами. Предмети, які тісно взаємодіють один з одним, розміщуються поруч. Вони не розподілені по всьому додатку. Також варто відзначити, що в деяких випадках видалення функції може звестися до однієї операції – видалення каталогу, а додавання – до додавання нового каталогу. Операції видалення можна розглядати як хороший тест на максимальну модульність: елемент має максимальну модульність, лише якщо його можна видалити за одну операцію.

В даній архітектурі буде існувати щонайменше три логічних рівні або як їх ще називають шари (layer):

* рівень доступу до даних – відповідальний за виконання маніпуляцій з інформацією та її обробку, яка надходить до програмного середовища або вже збережена в БД. Вона представлена локальною або за потреби віддаленою базою даних;
* рівень бізнес-логіки, який містить основну логіка парсера для обробки даних або ядро програми;
* рівень інтерфейсу користувача, за допомогою якого користувач даного програмного забезпечення буде взаємодіяти з ним та отримувати результат виконання необхідних функцій.

Обрана архітектура є складнішою за схожу до неї дворівневу, проте володіє наступними перевагами:

* висока гнучкість;
* достатньо просте збільшення кількості функціоналу;
* підвищена безпека, яка проявляється завдяки розподілу програми на рівні. Для кожного шару можна окремо налаштувати заходи безпеки;
* висока ефективність, яка забезпечується змогою виконання рівнів паралельно при незалежності їх завдань.

Дана архітектура також відповідає наступним принципам:

* висока зчепленість коду (High Cohesion) – код, відповідальний за схожу функціональність є зосередженим в одному місці;
* низька зв'язаність коду (Low Coupling) – програмні елементи є максимально незалежними від інших програмних елементів;
* вказуй, а не питай (Tell, Don't Ask) – програмні елементи містять інформацію і функції для маніпулювання цими даними, при цьому для них немає потреби знати про інші програмні елементи, їхні функції та дані;
* не розмовляй з незнайомцями (Don't talk to strangers) – програмні елементи спрямовані на мінімізацію знання про інші програмні елементи для забезпечення більш стійкого коду.

Проте дана архітектура має також і певні недоліки:

* підвищена складність написання програми;
* підвищена складність в процесах розгортання та адміністрування;
* наявність високої швидкості мережі між сервером (наприклад з БД) та локальним комп'ютером.

Схема розподілу програмного забезпечення для збору даних на логічні рівні зображена на рисунку 3.1.

На рисунку 3.2 можна побачити фізичні рівні ПЗ. Input – відповідає за вхідні дані. UI (user interface) – це інтерфейс користувача, через який він взаємодіє з додатком. Parser – центральний рівень програмного забезпечення для збору даних. Саме тут відбувається збір даних. Request відповідає за надсилання запитів до веб-ресурсів, з яких буде відбуватися збір. Export представляє функціонал, який виконує форматування та обробку даних відповідно до необхідного типу. Product представляє дані та функції зберігання і отримання їх з БД.



Рис. 3.1. Логічні рівні ПЗ

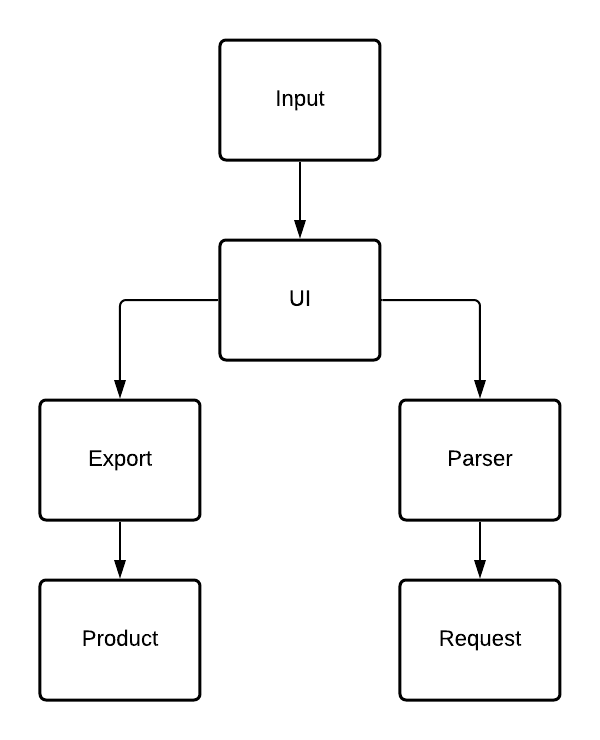


Рис. 3.2. Фізичні рівні ПЗ

## **Діаграми взаємодії**

Для представлення функціонування ПЗ, яке розробляється, використано уніфіковану мову моделювання UML [8]. На рисунку 3.3 зображена діаграма варіантів використання (use case). На ній можна побачити як користувач даного програмного забезпечення може використовувати його. Для початку користувачу необхідно ввести початкові дані для запуску збору інформації. Потім він може обрати режим, в якому вони будуть збиратися: за допомогою посилання або за пошуковим словом. Після завершення даної операції він може переглянути ці дані. Далі користувач може порівнювати дані і таким чином аналізувати їх. В кінці він може експортувати збережену інформацію до файлу. Також користувач може закрити програму.

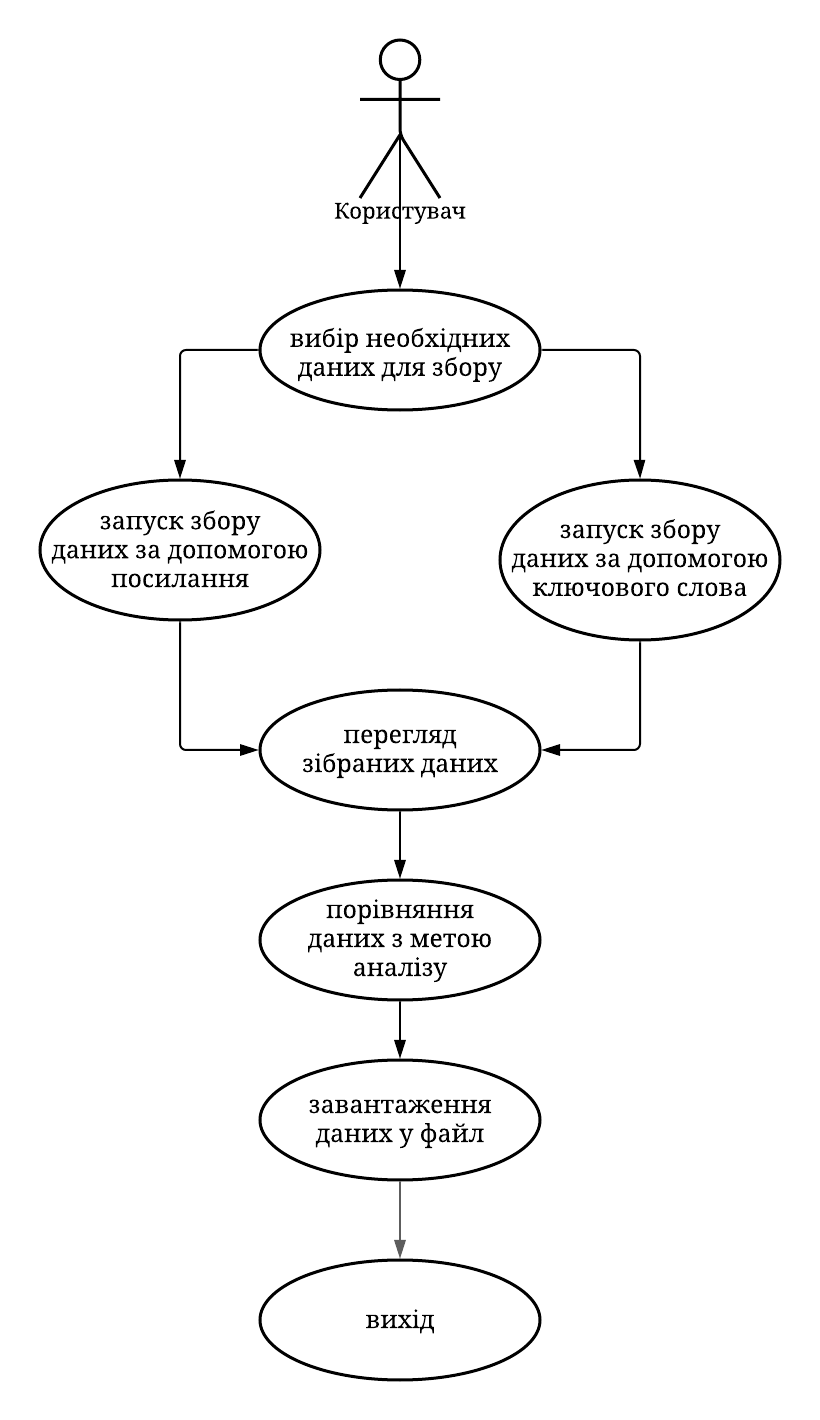


Рис. 3.3. Діаграма варіантів використання

На рисунку 3.4 зображено діаграму розгортання. З цієї діаграми можна зрозуміти, що для використання програмного забезпечення для збору та аналізу даних користувач повинен із свого пристрою за допомогою клавіатури, монітору та інших периферійних пристроїв таких, як мишка та за потреби гарнітура взаємодіяти з графічним інтерфейсом користувача. Парсер в свою чергу має зв’язок з БД за допомогою з’єднання TCP/IP. БД представлена локальною або віддаленою Postgre SQL.

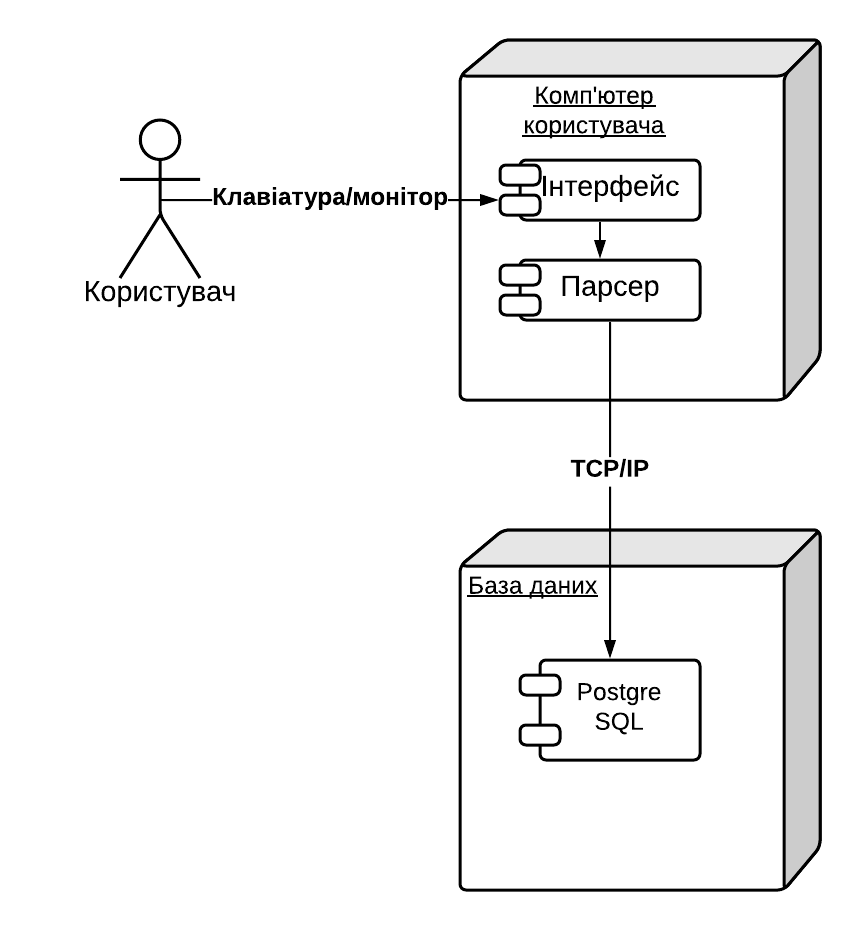


Рис. 3.4. Діаграма розгортання ПЗ

Наступні діаграми описують функціонування ПЗ та взаємодію між користувачем та різними компонентами ПЗ.

На рисунку 3.5 зображено діаграму послідовності для функції збору даних з веб сайтів. Для цього користувачу необхідно спочатку ввести початкові дані в залежності від його потреб та обрати, з якого сайту буде відбуватися збір. Ці дані будуть перевірені. Після цього відбудеться ініціалізація парсера та розпочнеться збір даних. В кінці вони будуть збережені до БД, а користувачу буде відображене повідомлення про успішність та завершення збору даних.

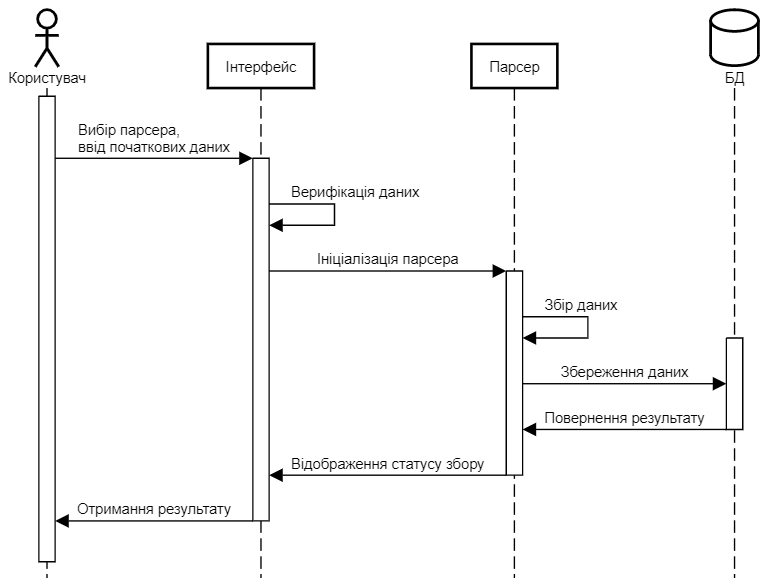


Рис. 3.5. Діаграма послідовності для функції збору даних

На рисунку 3.6 зображено діаграму такого ж типу для функції перегляду зібраних даних.

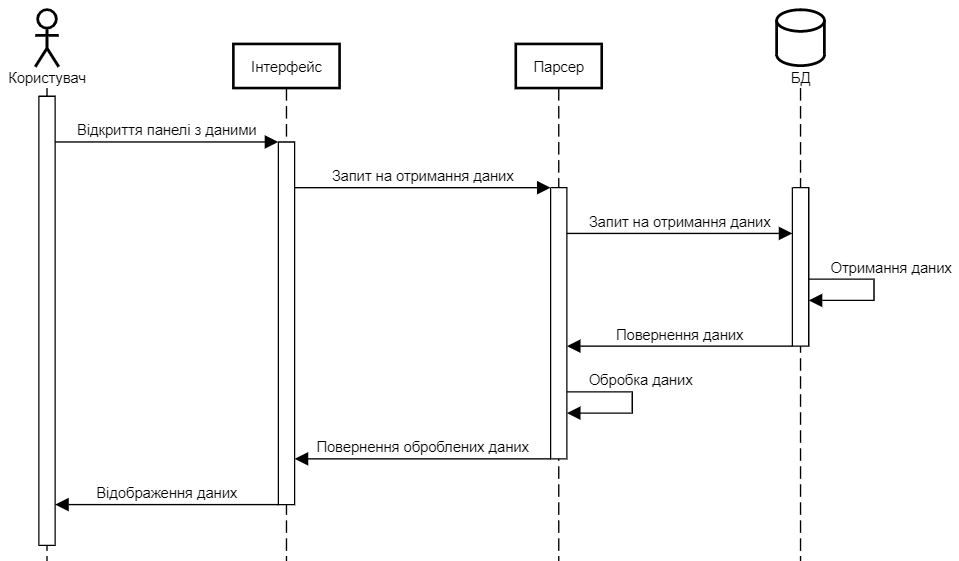


Рис. 3.6. Діаграма послідовності для функції перегляду даних

Для цього користувач повинен відкрити панель з даними. Це автоматично згенерує та відправить запит на їх отримання. Після їх вивантаження з БД вони будуть оброблені та внесені у таблицю з даними, яку побачить користувач.

На рисунку 3.7 зображена діаграма послідовності для функції оновлення зібраних даних в таблиці. Дана функція зазвичай застосовується для завантаження нових даних у таблицю. Для застосування цієї функції користувачу необхідно натиснути на кнопку оновлення даних на графічному інтерфейсі користувача. Після натискання відбудеться очищення таблиці та відправиться запит на отримання даних з БД. Тоді успішного їх вивантаження та обробки вони будуть внесені у таблицю. Після цього користувач може побачити оновлені дані в таблиці.

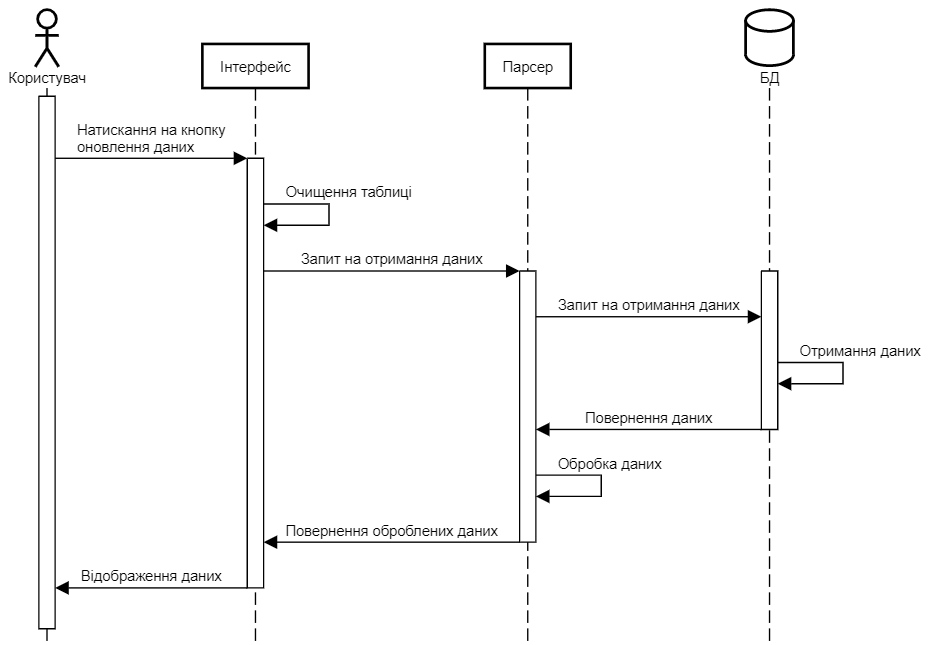


Рис. 3.7. Діаграма послідовності для оновлення даних

На рисунку 3.8 зображено діаграму послідовності для функції експорту даних у файл. Для ініціації початку роботи цієї функції користувачу необхідно натиснути на відповідну кнопку, яка розміщена на інтерфейсі. Після цього він побачить нове згенероване вікно інтерфейсу, яке буде пропонувати ввести необхідні дані, наприклад, ім’я файлу та тип файлу. Тоді дані будуть відформатовані згідно обраного формату та записані у новостворений файл. Користувач одразу після цього отримає повідомлення про успішність операції.

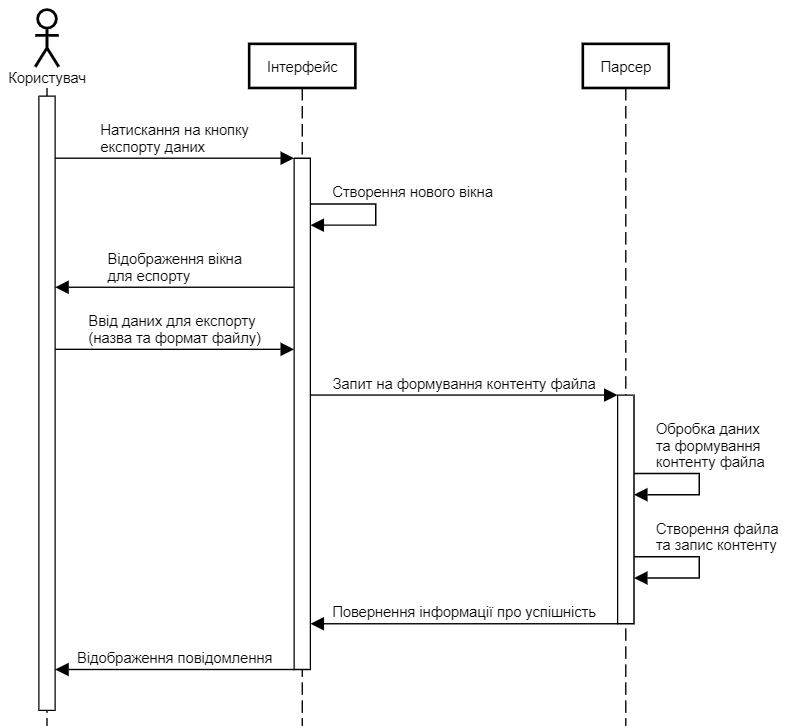


Рис. 3.8. Діаграма послідовності для функції експорту даних у файл

## **Діаграми класів**

Проект представлений сукупністю Java класів, які містять код. Класи в свою чергу розподілені по різним пакетам в залежності від функцій, які вони виконують. Кожний пакет відповідає одному з фізичних рівнів зображених на рисунку 3.2. Тобто вся програма складається з таких основних пакетів:

* input;
* ui;
* parser;
* request;
* export;
* product.

Їхні загальні функції та призначення були описані раніше.

На рисунку 3.9 зображено діаграму класів для пакету input.

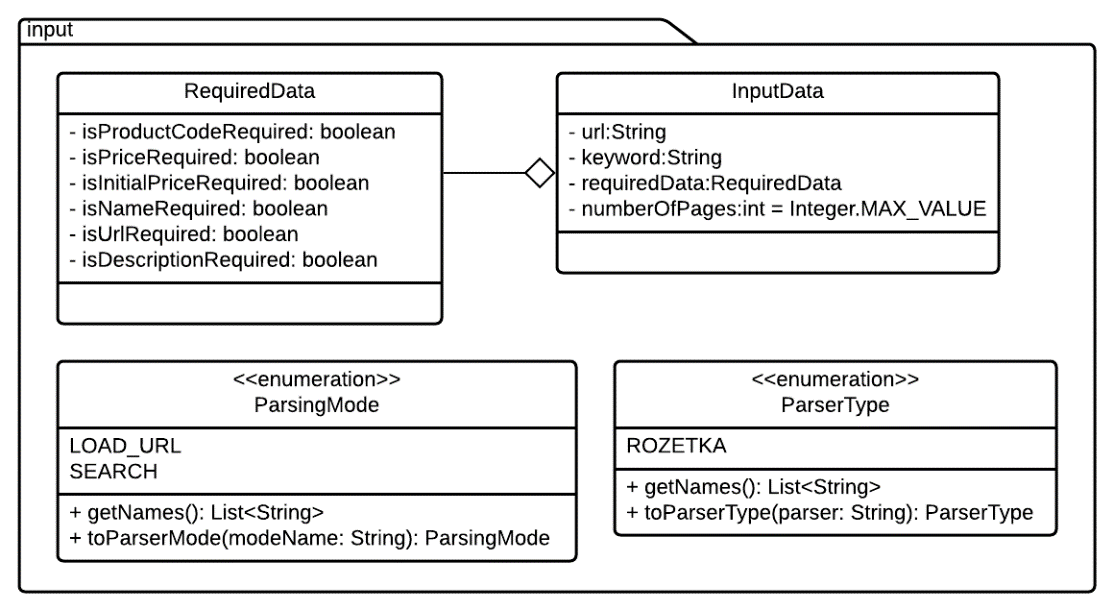


Рис. 3.9. Діаграма класів для пакету input

Даний пакет, як було зазначено раніше, відповідає за початкові дані. В ньому містяться такі програмні елементи як RequiredData, InputData, ParsingMode та ParserType.

RequiredData – клас, який містить дані щодо інформації, яку необхідно збирати з веб-сайту. Містить поля, які приймають булеве значення. Якщо значення дорівнює true, то інформацію повинна бути зібрана, інакше її треба упустити. Наприклад, якщо поле isPriceRequired має значення true, ціну потрібно збирати.

InputData – початкові дані, які вводяться користувачем, необхідні для ініціалізації та початку роботи парсера. Має наступні поля:

* url – посилання, з якого розпочнеться збір інформації;
* keyword – пошукове слово, за яким будуть знайдені необхідні для збору веб-сторінки;
* requiredData – екземпляр класу RequiredData;
* numberOfPages – кількість навігаційних сторінок, які треба обробити. За змовчуванням будуть оброблені всі сторінки.

Варто зазначити, що одночасно url та keyword не можуть мати значення, а лише одне з цих полів.

ParsingMode – перерахування (enumeration), яке містить всі можливі режими, в яких може працювати дане ПЗ. Планується реалізувати щонайменше два режими:

* LOAD\_URL – за посиланням;
* SEARCH – за пошуковим словом.

ParserType – тип парсера, за яким можна легко визначити на який веб-сайт ПЗ повинно робити запити для отримання інформації.

На рисунку 3.10 зображено діаграму класів для пакету ui.

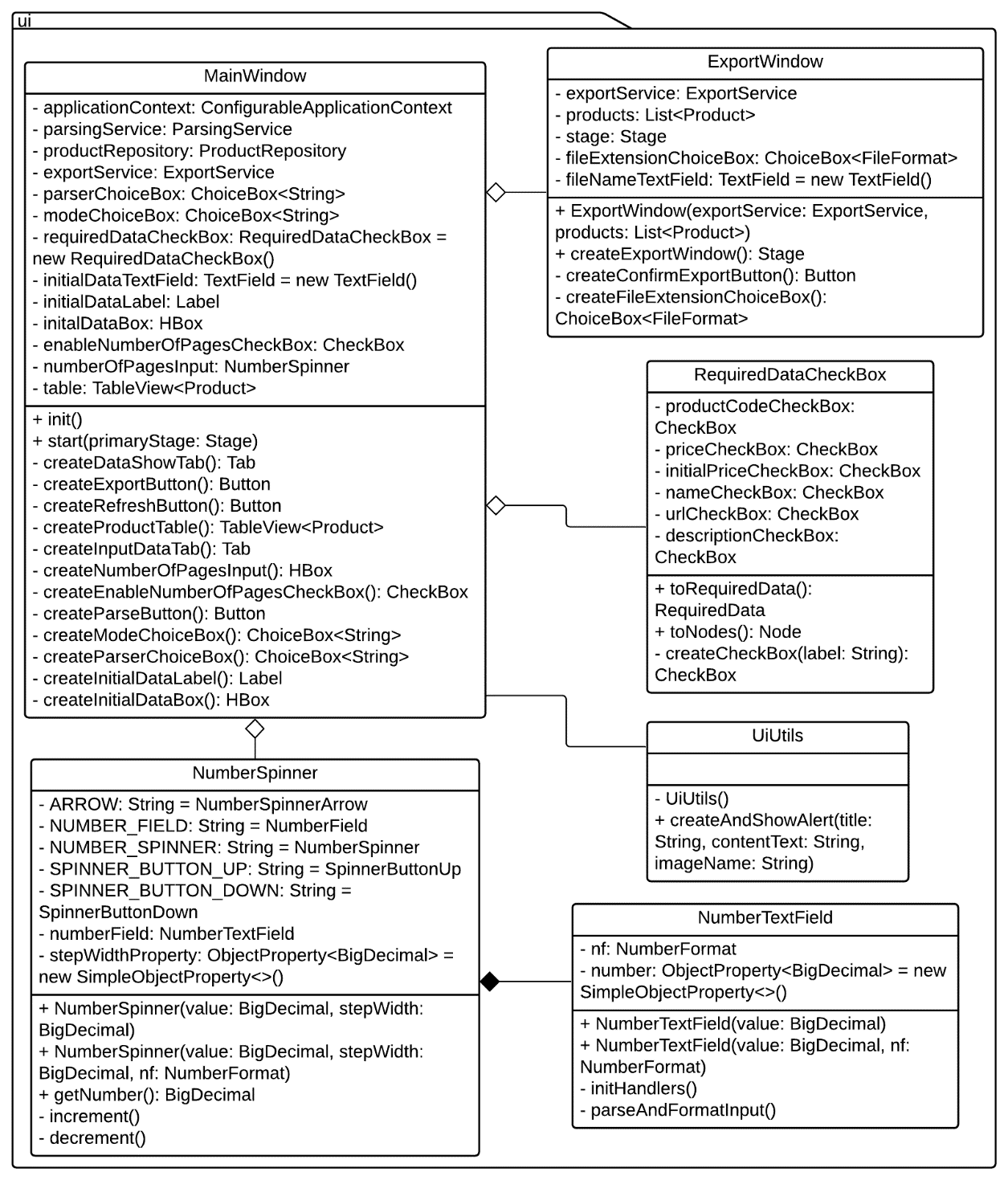


Рис. 3.10. Діаграма класів для пакету ui

Пакет відповідає за створення графічного інтерфейсу користувача. Він включає класи MainWindow, ExportWindow, NumberSpinner, NumberTextField, RequiredDataCheckBox та UiUtils.

MainWindow – відповідає за створення та функціонування основного вікна додатка. Клас має багато полів, серед яких основними є:

* applicationContext – бере участь у ініціалізації додатку;
* parsingService, за допомогою якого відбувається ініціалізація парсера тапочаток збір інформації;
* productRepository, за допомогою якого відбувається отримання даних з БД.

Також реалізує наступні функції:

* init – ініціалізація додатка;
* start – створення основного вікна графічного інтерфейсу користувача;
* createDataShowTab – створення панелі для перегляду даних;
* createInputDataTab – створення панелі для введення початкових даних.

ExportWindow – відповідає за створення та функціонування вікна, яке з’являється після натискання на кнопку «Експорт».

NumberSpinner – представляє елемент графічного інтерфейсу користувача, за допомогою якого можна вводити лише числові дані та зібльшувати/зменшувати введені дані за допомогою стрілок (numpad up/down) або шляхом натискання на стрілки, які відображаються на екрані;

NumberTextField – елемент графічного інтерфейсу, який представляє текстове поле призначене для введення лише числових даних. Використовується для реалізації NumberSpinner;

RequiredDataCheckBox – клас, який є контейнером для групи елементів графічного інтерфейсу, які відповідають за вибір необхідних даних для збору з веб-сайту;

UiUtils – клас-утиліта, який містить часто застосовувані функції, які використовуються для формування інтерфейсу. В даному випадку містить лише одну функцію для створення діалогового вікна, за допомогою якого користувач отримує відповідь про успішність або неуспішність завершення виконання певної функції.

На рисунку 3.11 зображена діаграма класів для пакету request. Він призначений для відправлення та обробки запитів на ресурси, що належать третім особам. В ньому виділено такі програмні класи: HttpMethod, Request та RequestExecutor.

HttpMethod – це перерахування для визначення метода запиту на веб-сторінку. Зазвичай для отримання даних необхідно лише два методи: POST та GET. HTTP запит з POST методом направлений для відправлення даних на сервер, з GET методом – на отримання ресурсу із сервера. Інші методи такі, як PUT, PATCH, DELETE та інші частіше застосовуються для зміни ресурсу, до якого вони застосовуються.

Request – клас, який є представленням HTTP запиту та містить всю ключову інформацію для його здійснення. Містить наступні важливі поля:

* url – посилання, за яким буде здійснюватися запит;
* httpMethod – метод запиту;
* headers – заголовки, які відправляються разом з запитом та дозволяють відправляти та отримувати додаткові дані.

RequestExecutor – клас-сервіс, який реалізує відправлення HTTP запитів на зазначені веб-сторінки.

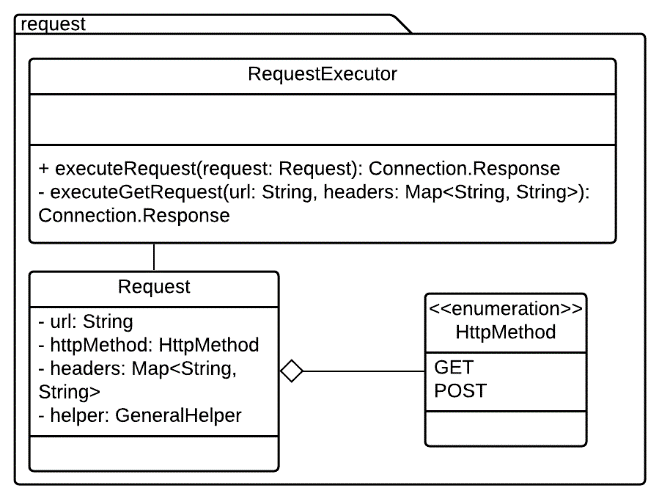


Рис. 3.11. Діаграма класів для пакету request

Діаграма класів для пакету product зображена на рисунку 3.12. Даний пакет містить всього два програмні елементи: Product та ProductRepository.

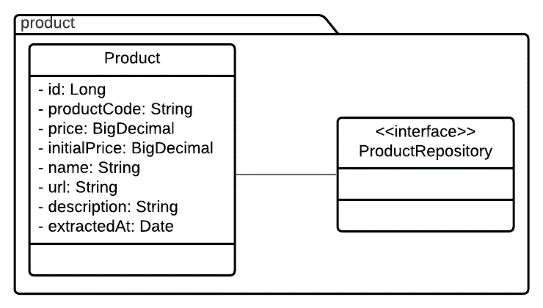
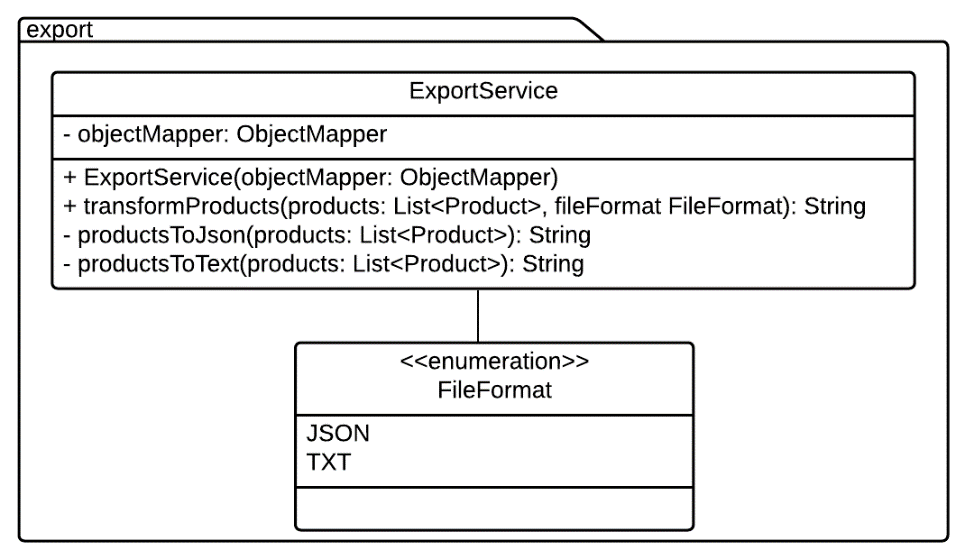


Рис. 3.12. Діаграма класів для пакету product

Product – представляє зібрані дані готові для зберігання або обробки, в даному випадку – це продукти.

ProductRepository – програмний інтерфейс, який відповідає за взаємодію програми з БД, а саме з SQL таблицями, які стосуються сутності продукту. Даний інтерфейс не має жодних функцій тому, що він повинен наслідувати вже реалізований програмний інтерфейс, який міститься в одному з фрейморків або бібліотек призначених для цього.

На рисунку 3.13 зображено діаграму класів для пакету export.

  
Рис. 3.13. Діаграма класів для пакету export

Він також містить у собі всього два програмних компонента: FileFormat та ExportService.

FileFormat – це перерахування, яке представляє формат файлу, в який буде відбуватися експорт. На даний момент тут знаходиться лише два найпоширеніші формати файлів: JSON та TXT;

ExportService – клас-сервіс, який відповідає за перетворення зібраних даних у контент файлу з обраним форматом.

На рисунку 3.14 зображено діаграму класів для пакету parser. Даний пакет, окрім єдиного класу ParsingService, також містить два підпакети: abstractparser та rozetka.

На даній діаграмі вони виглядають пустими, проте це не так. Для більшої зручності та детального опису вони будуть зображені на окремих діаграмах.

Вкладений пакет abstractparser – це одна з основних частин програми тому, що від нього залежить наскільки просто буде імплементувати парсери для різних веб-сайтів. Він є абстракцією реалізації будь-якого парсера.

Підпакет rozetka реалізує програмні компоненти розміщені в abstractparser та власне виконує збір даних з відомого продуктового веб-сайту Rozetka. За потреби можна з легкістю створити інші підпакети, не вносячи зміни у вже існуючий код та реалізації парсерів, які будуть відповідати за збір інформації з інших веб-сайтів.

Клас ParsingService є початковою точкою, з якої починається процес збору інформації. Він містить всього один публічний метод parse, який відповідає за ініціалізацію необхідного парсера та за запуск процесу збору інформації на основі обраного режиму та початкових даних для цього парсера.

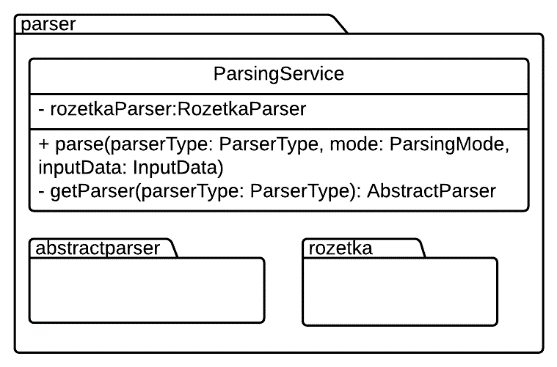


Рис. 3.14. Діаграма класів для пакету parser

На рисунку 3.15 зображено діаграму класів для пакету abstractparser, який знаходиться у пакеті parser.

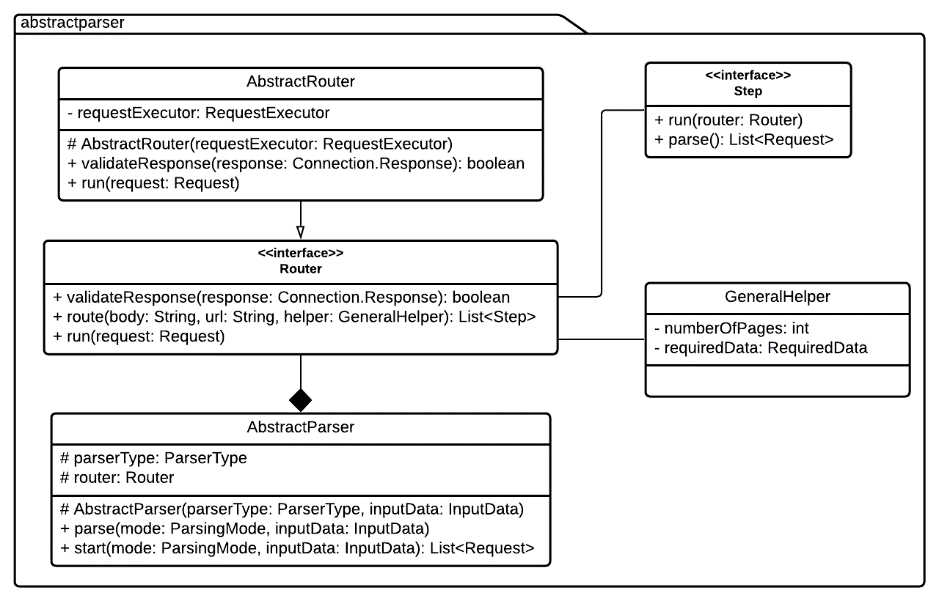


Рис. 3.15. Діаграма класів для пакету abstractparser

Даний пакет містить такі програмні компоненти: AbstractParser, Router, AbstractRouter, Step та GeneralHelper.

AbstractParser – абстрактний клас, який відповідає за формування першого запиту до веб-сайта або веб-сторінки. З першого погляду ця функція може здатися надлишковою, проте це вкрай необхідно для збору інформації за пошуковим словом. В цьому випадку спочатку потрібно з формувати посилання на веб-сторінку. Іноді це потребує кодування спеціальних символів та виконання іншої програмної логіки. Також цей клас відповідає за делегацію виконання цього запиту Router’у.

Router – програмний інтерфейс, який об’являє функції для делегації виконання HTTP запиту до RequestExecutor та створення відповідного Step’у на основі отриманої відповіді з вмістом сторінки. Тобто реалізація даного інтерфейсу переспрямовує відповідь з веб-сайту до конкретного її обробника. Також містить функцію валідації та верифікації отриманої відповіді.

AbstractRouter – це часткова реалізація інтерфейсу описаного вище створена для полегшення реалізації конкретного парсера для збору інформації.

Step – це програмний інтерфейс, який об’являє функції розпізнавання типу веб-сторінки та збору з неї даних. При цьому ці дані можуть бути не кінцевими, а проміжними, наприклад посилання на подальші веб-сторінки, вміст яких потребує обробки.

GeneralHelper – допоміжний клас, який використовується для передачі додаткових даних, між різними обробниками вмісту веб-сторінок, які впливають на їх реалізацію.

На рисунку 3.16 зображено діаграму класів для пакету rozetka. Як вже було зазначено раніше це реалізація парсера для сайту Rozetka. Даний пакет містить такі класи: RozetkaParser, RozetkaRouter, RozetkaNavigationStep, RozetkaSearchStep та RozetkaProductStep.

RozetkaParser – це реалізація класу AbstractParser. Він відповідає за формування початкового запиту на веб-сторінку сайту Rozetka;

RozetkaRouter – це реалізація класу AbstractRouter. Він відповідає за переспрямування відповіді на запити до конкретного обробника, якими є RozetkaNavigationStep, RozetkaSearchStep та RozetkaProductStep;

RozetkaNavigationStep – реалізація інтерфейсу Step. Відповідає за обробку навігаційних веб-сторінок, на яких містяться посилання на декілька продуктів. Також даний клас формує запити на подальші навігаційні сторінки з продуктами або як їх ще називають пагінаційні сторінки;

RozetkaSearchStep – реалізація інтерфейсу Step. Відповідає за обробку веб-сторінок, які є результатом пошуку за ключовим словом. Такі сторінки за інтерфейсом схожі до навігаційних сторінок, проте вони в даному випадку мають зовсім інший вміст. Також він відповідає за формування запитів на подальші сторінки з результатами пошуку, якщо вони існують;

RozetkaProductStep – реалізація інтерфейсу Step. Відповідає за обробку продуктових веб-сторінок, на яких можна знайти детальну інформацію про продукт.

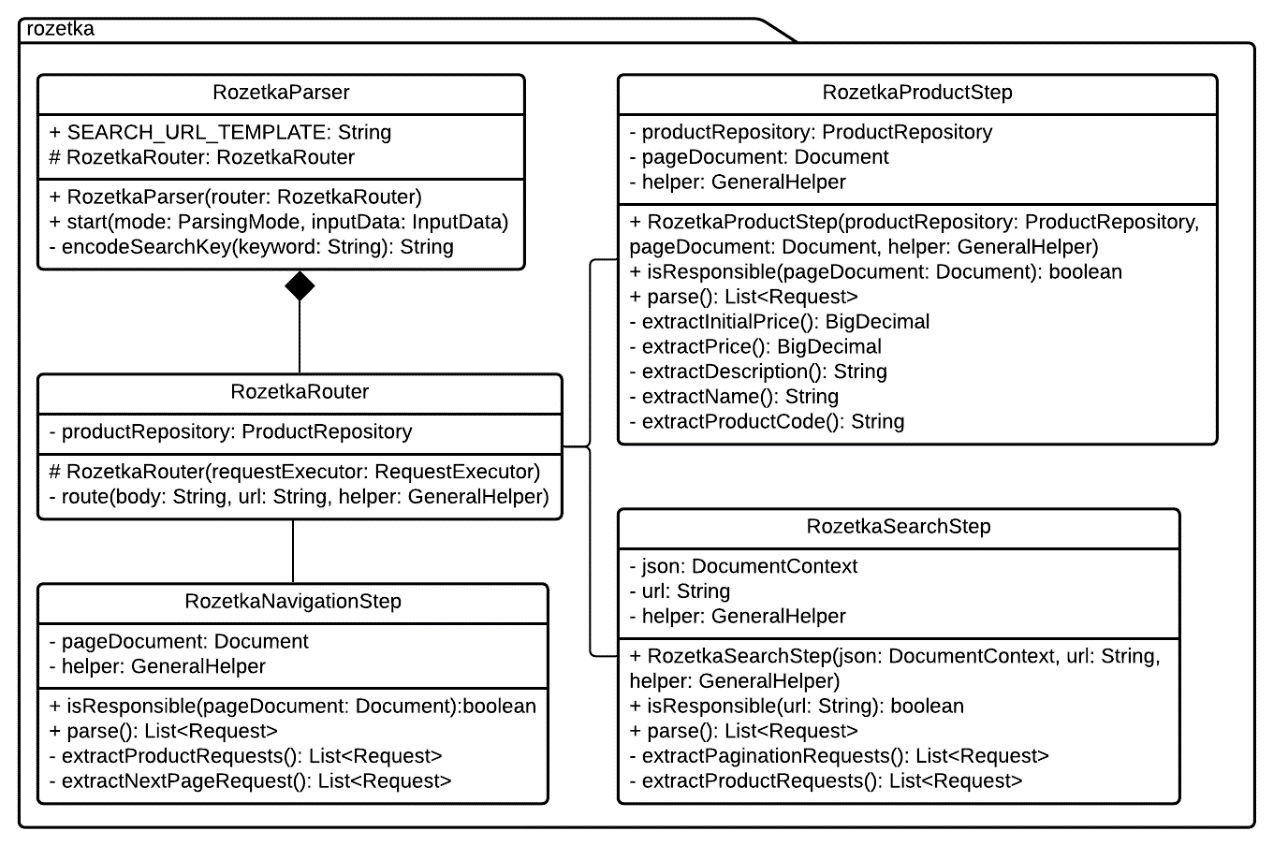


Рис. 3.16. Діаграма класів для пакету rozetka

На рисунку 3.17 зображена загальна діаграма класів. На ній можна побачити зв’язки між класами, які знаходяться у різних пакетах і таким чином вияснити як вони взаємодіють один з одним.

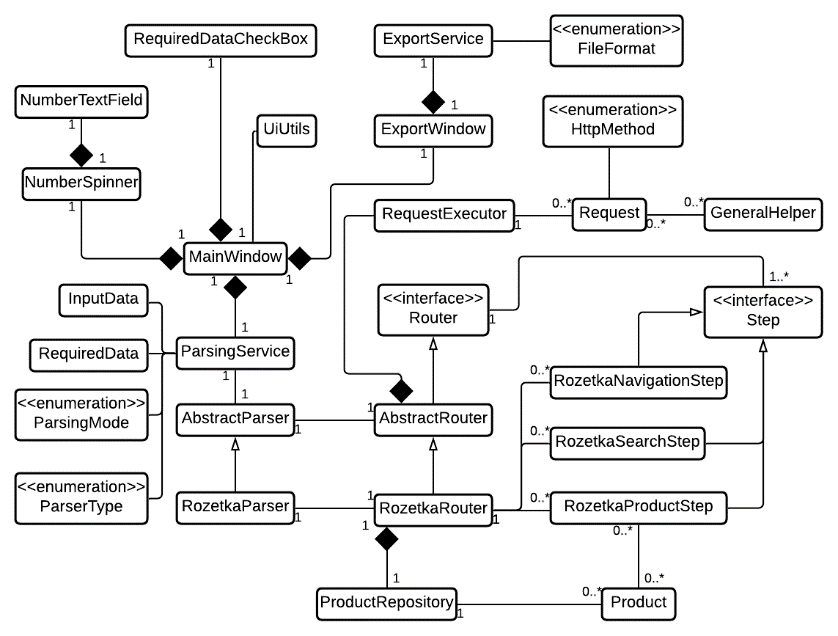


Рис. 3.17. Загальна діаграма класів

## **Реляційна модель бази даних**

В реляційній моделі БД виділено одну сутність, яка відображає дані, які збиралися з веб-сайта, або продукти – PRODUCTS [9].

Дана сутність містить такі поля:

* ідентифікатор (ID);
* опис продукту (DESCRIPTION);
* дата збереження до БД (EXTRACTED\_AT);
* початкова ціна продукту (ціна без знижки) (INITIAL\_PRICE);
* назва продукту (NAME);
* ціна продукту (із знижкою якщо вона є) (PRICE);
* унікальний код продукту (PRODUCT\_CODE);
* посилання на веб-сторінку, з якої був виконаний збір (URL).

В таблиці 3.1 можна побачити детальну інформацію про дану сутність, а саме формат, розмір полів та обмеження, що їх стосуються.

Таблиця 3.1.

Детальна інформація про сутність PRODUCTS

|  |  |  |  |
| --- | --- | --- | --- |
| **Поле** | **Ключ** | **Розмір** | **Формат** |
| ID | PK | - | BIGINT |
| DESCRIPTION | - | 1000 | VARCHAR |
| EXTRACTED\_AT | - | - | TIMESTAMP |
| INITIAL\_PRICE | - | 19 | DECIMAL |
| NAME | - | 255 | VARCHAR |
| PRICE | - | 19 | DECIMAL |
| PRODUCT\_CODE | - | 255 | VARCHAR |
| URL | - | 255 | VARCHAR |

## **Висновки**

Архітектура програмного забезпечення має велике значення. Для розроблюваного додатку було обрано архітектуру package by feature (пакет за функцією) з графічним інтерфейсом користувача. Згідно неї кожний пакет відповідає та виконує певну функцію. Дана архітектура володіє суттєвими перевагами на відміну від її аналогів. Серед них можна виділити низьку зв’язність програмних елементів, особливо між різними пакетами, високу гнучкість, просте нарощування та розширення функціоналу, а також висока зв’язність коду, що відповідає за певну функцію. Додавання або видалення функцій зводиться до додавання або видалення пакета з програмними елементами, який за неї відповідає. Це є досить зручно не тільки для тестування коду, але й для розробки тому, що в таким чином передбачена можливість додання декількох парсерів.

ПЗ для збору та аналізу даних поділене на логічні та фізичні рівні. Програма має такі логічні рівні:

* рівень доступу до БД;
* рівень бізнес-логіки;
* рівень представлення.

Також існують такі фізичні рівні, кожний з яких представлений окремим пакетом з класами:

* input;
* ui;
* parser;
* product;
* export;
* request.

Деякі фізичні рівні співпадають з логічними. Наприклад, рівень представлення та ui, які відповідають за графічний інтерфейс користувача.

Для проектування програмного забезпечення використано уніфіковану мову моделювання UML. Розроблено діаграму прецедентів, або варіантів використання, за допомогою якої можна побачити як користувач може взаємодіяти з ПЗ; діаграму розгортання, на якій можна побачити обчислювальні вузли протягом роботи програми, а також компоненти, які знаходяться на цих вузлах; діаграми послідовностей для кожної з функцій, які показують як користувач та різні програмні компоненти взаємодіють між собою; діаграми класів для кожного рівня, які показують вміст та функції програмних компонентів, та одну загальну, яка показує зв’язки між різними рівнями.

Також наведено сутності в БД та складено таблицю з форматами полів та обмеженнями. Спроектовано лише одну сутність, яка представляє зібрані дані – PRODUCTS. Вона міститиме наступні поля:

* id – унікальний ідентифікатор;
* description – опис продукту;
* extracted\_at – дата збереження даних;
* initial\_price – початкова ціна продукту;
* name – назва продукту;
* price – ціна продукту;
* product\_code – унікальний код продукту;
* url – посилання на продукт.

# **РОЗДІЛ 4 ТЕХНІЧНІ РІШЕННЯ**

Технічні рішення

54

12

*Літ.*

*Керівник.*

Сінько Ю.І.

*Розроб.*

Петров А.В.

НАУ 16 12 52 000 ПІ

Кафедра ПІ

*Лист*

*Листів*

ТП-215М – 6.050101

*Н-кнтр. Боровик* В.М.

## **Вибір мови програмування**

Програмне забезпечення повинно бути розробленим на крос-платформній мові програмування. Це дозволить збільшити кількість апаратних платформ, на яких зможе запускатися ПЗ, а також полегшить процес переходу з однієї апаратної платформи на іншу в разі потреби. Більшість сучасних високо рівневих мов програмування мають таку властивість. Крос-платформність можна поділити на дві категорії в залежності від рівня:

* на рівні мов програмування;
* на рівні середовища виконання.

На обох рівнях програмний код не залежить від платформи, на якій він буде виконуватися.

В першому випадку це досягається за рахунок різних трансляторів та компіляторів, кожен з яких призначений для конкретної платформи. Серед таких мов можна виділити: C та C++.

В другому випадку це досягається за рахунок інтерпретаторів та віртуальних машин, що часто дає можливість запускати виконавчі файли на різних платформах без їх перекомпіляції перед запуском. До таких мов програмування належать Java, C# та Python. Всі вищезгадані мови окрім С є об’єктно-орієнтованими.

Оскільки при проектуванні був розрахунок на використання принципів ООП, то С не підходить для реалізації даного ПЗ.

Для замірів продуктивності був проведений експеримент, в якому відбувалося перемноження матриць розміром 2048 на 2048. Вони були заповнені випадковими значеннями від 1 до 10.

Після відпрацювання програм отримано результати наведені в таблиці 4.1.

Таблиця 4.1.

Результати замірів швидкодії

|  |  |  |  |
| --- | --- | --- | --- |
| **Ітерація** | **С++** | **Java** | **Python** |
| 1 | 51 | 85.7 | 2872.5 |
| 2 | 50.22 | 82.4 | 2681.4 |
| 3 | 50.3 | 83.06 | 2921.6 |
| 4 | 49.4 | 86.4 | 2945.3 |
| 5 | 48.5 | 85.9 | 2685.1 |
| Середній час (секунди) | 49.88 | 84.69 | 2821.18 |

Всі заміри проводилися у секундах, на одинаковій ОС та апаратній платформі з незмінними обсягами оперативної пам’яті та без будь-яких обмежень для компіляторів даних мов програмування. З отриманих результатів можна зробити висновок, що Python є значно повільнішим від Java та С++, а останній в свою чергу є трішки продуктивнішим від Java. З огляду на легкість написання програм, простоту синтаксису та ефективність Java є золотою серединою в порівнянні між складним, але ефективним С++, та простим, проте більш повільним Python.

Щодо C#, то ця мова досить схожа з Java, але більш пристосована для ОС Windows на відміну від Java. Оскільки необхідно забезпечити виконання програми у різних операційних системах, то Java найкраще підходить для цієї цілі в порівнянні з вищезгаданими мовами програмування. Для реалізації буде використано Java з версією 11 або вище [10].

## **Вибір фреймворку для каркасу ПЗ**

Java пропонує широкий вибір різних фреймворків та різноманітних за функціоналом бібліотек. Перш за все необхідно вибрати основний фреймворк, за допомогою якого можна створити каркас ПЗ. Серед найбільш популярних фреймворків можна виділити Spring Boot, Griffon, PicoContainer та SupernautFx.

PicoContainer – це легкий фреймворк, який реалізує Dependency Injection патерн. Він виступає контейнером інверсії контролю. Проте він нажаль не може запропонувати іншого корисного функціоналу.

Griffon – це ефективний фреймворк, який в першу чергу призначений для написання та оформлення інтерфейсу в настільних програмах. Він добре взаємодіє з популярними бібліотеками для створення інтерфейсу і полегшує цей процес. Але за допомогою даного фреймворку не вдасться підвищити ефективність ПЗ або зменшити кількість використовуваних ресурсів. Також він так же як і PicoContainer не допоможе в роботі з БД.

SupernautFx – це фреймворк, який за своїм функціоналом поєднує можливості двох попередніх за винятком взаємодії з популярними бібліотеками для створення інтерфейсу. Він призначений для написання додатків з інтерфейсом, в основі якого закладена лише бібліотека JavaFx.

Spring Boot – це досить обширний фреймворк призначений для написання як веб-додатків, так і настільних програм. Він сумісний з більшістю бібліотек та фреймворків для інтерфейсу. Порівнюючи з вище описаними інструментами Spring Boot дає можливість також полегшити роботу та написання коду пов’язаного із збереженням даних в БД та їх отриманням. Фреймворк поділений на декілька модулів, що робить зручним його використання тому, що можна підключити до проекту лише необхідні модулі. Це також збереже ресурси при роботі програмного забезпечення. Він значно спрощує налаштування додатку та дає вибір як це можна зробити, наприклад XML-файли, анотації або конфігурація за допомогою коду. Тестування з даним фреймворком також спрощується тому, що він дає змогу підставляти тестові дані, замість реальних. Тобто Spring Boot допоможе не тільки створити каркас програмного забезпечення, але й полегшить створення програмного коду, зробить його чистішим та зрозумілішим. На рисунку 4.1 зображені модулі, які входять до складу Spring Framework [11].

Для створення каркасу будуть застосовані такі модулі Spring Boot версії 2.3.11:

* Spring Core Container;
* Spring ORM;
* Spring Instrumentation;
* Spring Test.

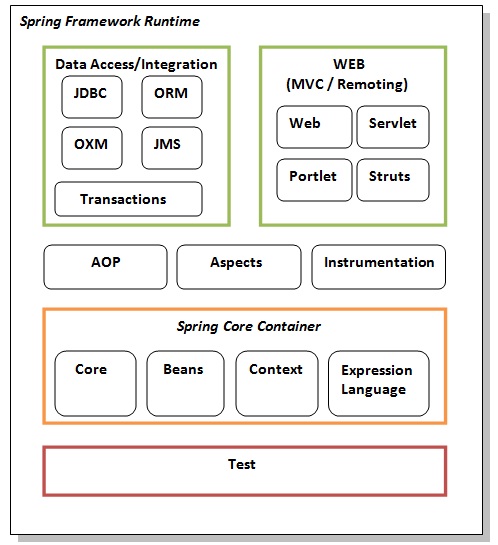


Рис. 4.1. Модулі Spring Framework

Spring Core Container є ядром Spring Framework. Контейнер забезпечує створення об’єктів, їх з’єднання, налаштування, а також відповідає за управління їх повним життєвим циклом від створення до знищення. Контейнер Spring використовує DI для керування компонентами, які складають програму. Ці об’єкти називаються Spring Beans.

Контейнер отримує команди на рахунок того, які об’єкти треба створити, налаштувати та з’єднати, з наданих метаданих конфігурації. Метадані конфігурації можуть бути представлені XML, анотаціями Java або кодом Java. Контейнер Spring IoC використовує класи Java POJO і метадані конфігурації для створення повністю налаштованої та виконуваної системи або програми.

Spring ORM – це модуль, який відповідає за роботу з рівнем доступу до даних, тобто з БД.

Spring Instrumentation містить класи необхідні для підтримки інших модулів та для завантаження класів у програму.

Spring Test використовується для написання юніт тестів та їх запуску.

## **Вибір інструменту для інтерфейсу**

Серед популярних інструментів для створення графічного інтерфейсу користувача можна виділити AWT (Abstract Window Toolkit), Swing, SWT (Standard Widget Toolkit) та JavaFx.

AWT – це перший GUI фреймворк включений в Java 1.1. Він використовує рідні елементи інтерфейсу. Тобто вони залишаються постійно такими ж незалежно від того, де програма запускається. Проте для різних середовищ їх може бути недостатньо, не говорячи про те, що розмітка інтерфейсу може значно відрізнятися в залежності від середовища.

Swing – альтернативний інструмент, який був включений у Java 1.2. Він використовує форми AWT, проте відображення елементів інтерфейсу бере на себе. За допомогою такого підходу інтерфейс виглядає однаково в різних середовищах, але це сповільнює його роботу.

SWT – це фреймворк, який подібно до AWT, використовує рідні елементи інтерфейсу. Основний недолік – це невідповідність принципу «одного разу створений, працює всюди». Тобто він нездатний працювати на всіх платформах та ОС, на яких може працювати Java.

JavaFx – найновіший серед вище описаних інструментів. По ідеології він нагадує Swing, а саме використовує нерідні елементи інтерфейсу. Проте в його реалізації усунутий важливий недолік – повільність. Даний інструмент є значно швидшим. Також він дає вибір у налаштуванні інтерфейсу. Це можна зробити програмним кодом, за допомогою XML конфігурації або використовувати обидва варіанти одразу. Також він забезпечує змогу використовувати CSS, що робить створення інтерфейсу ще простішим [12].

Очевидно, що серед всіх наведених інструментів JavaFx є найкращим варіантом для реалізації графічного інтерфейсу користувача, тому буде використано цей інструмент з версією 11.0.2.

## **Вибір бази даних**

Для зберігання даних необхідна реляційна БД. Можливі два основні варіанти:

* використання віддаленої БД;
* використання локальної БД.

Для першого варіанту варто розглянути такі БД, як PostgreSQL та MySQL. Ці дві БД працюють у режимі клієнт-сервер. На відміну від PostgreSQL, MySQL набагато легше адмініструвати, але її продуктивність не збалансована.

Для локальних БД підходять такі, як H2 або SQLite. Наведені бази даних є досить простими та безплатними, а також вони є крос-платформними. Їхньою особливістю є те, що всі таблиці, індекси та інші компоненти зберігаються в єдиному файлі на жорсткому диску комп’ютера, з якого запускалася програма.

SQLite – це спрощена реляційна БД. Основна її відмінність є те, що вона працює не окремим процесом, а вбудовується в процес, в якому працює програма. Це відбувається завдяки компіляції програми разом з бібліотекою, яку надає ця БД. Такий підхід зменшує витрати апаратних ресурсів, а також час, протягом якого надходить відповідь від БД до кінцевого користувача. Єдиним недоліком для цієї БД є те, що у Spring Boot відсутнє автоматичне налаштування для неї на відміну від Н2.

Особливістю Н2 є те, що вона написана повністю на Java, а отже і повністю сумісна з нею. Також на відміну від SQLite тут реалізовано два режими робити:

* вбудований – тобто процес БД та процес, в якому запускається програма є одним і тим же процесом;
* клієнт-сервер, тобто процес БД – це зовсім інший процес, до якого звертається програма для отримання даних.

За результатами тестувань на продуктивність, які наведені в таблиці 4.2, Н2 є близько в 50 разів швидша за SQLite, в 4 рази швидша, ніж MySQL та несуттєво продуктивніша за PostgreSQL.

Таблиця 4.2.

Тестування на продуктивність

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Тестовий випадок** | **Одиниці виміру** | **H2** | **PostgreSQL** | **MySQL** |
| Замір1: Ініціалізація | мс | 20993 | 27794 | 107723 |
| Замір1: Транзакції | мс | 16549 | 23113 | 65036 |
| Замір1: Пам’ять | MB | 12 | 1 | 4 |
| Замір2: Ініціалізація | мс | 26785 | 32369 | 115398 |
| Замір2: Транзакції | мс | 898 | 818 | 1794 |
| Замір2: Пам’ять | MB | 16 | 2 | 5 |
| Замір3: Ініціалізація | мс | 18266 | 24547 | 70531 |
| Замір3: Транзакції | мс | 6569 | 8916 | 19150 |
| Замір3: Пам’ять | MB | 17 | 2 | 7 |
| Виконані операції | - | 2222032 | 2222032 | 2222032 |
| Загальний час | мс | 179460 | 237392 | 763113 |
| Операцій за секунду | оп/с | 12381 | 9360 | 2911 |

Оскільки одним користувачем для одного запущеного процесу програми буде використовуватися єдине підключення до БД, то найкраще підходить база даних Н2. При цьому неважливо у якому режимі вона буде використовуватися.

## **Інші інструменти та бібліотеки**

Для реалізації всіх необхідних функцій та написання ефективного програмного коду необхідно застосувати деякі інші бібліотеки. Серед них можна назвати такі:

* Jsoup;
* Jayway JsonPath;
* Apache Commons;
* Jackson;
* Project Lombok.

Jsoup – це бібліотека для роботи з HTML. Даний інструмент реалізує зручний АРІ для отримання URL-адрес, витягування та обробка даних, використовуючи найкращі HTML5 DOM методи та CSS селектори. Jsoup проводить аналіз HTML подібно до сучасних веб-браузерів:

* збирає та аналізує HTML отриманого за допомогою URL-адреси, файла або тексту;
* проводить пошук та збір даних, використовуючи обходу документа або CSS селекторів;
* маніпулює HTML елементами, атрибутами текстом;
* очищає вміст записаний користувачем порівнюючи з надійним, щоб запобігти проведення XSS атак;
* виводить HTML.

Ця бібліотека призначена для роботи з усіма різновидами HTML, включаючи навіть недійсний набір різних тегів, та створює зрозуміле дерево тегів HTML. Також вона реалізує безліч методів отримання необхідних даних: за атрибутами, за класами, за ідентифіркатором, за назвою тега та багато інших.

Jayway JsonPath – це бібліотека призначена для роботи з JSON. Вона необхідна тому, що деяка інформація на веб-сторінці знаходиться саме у форматі JSON, наприклад у скриптах або прихованих тегах. Також вся відповідь від сервера може складатися з інформації у цьому форматі. За допомогою цієї бібліотеки можна проводити валідацію JSON контенту та витягувати дані з нього, вказуючи абсолютний або відносний шлях, у різні типи даних.

Apache Commons – це бібліотека, яка містить різні утилітні класи та методи для роботи з основними класами в Java. Вона зробить код більш чистим, зрозумілим та менш громіздким. Також вона допоможе обробляти витягнуті дані з веб-сторінок.

Jackson – це бібліотека призначена також для роботи JSON. Проте її функції відрізняються від Jayway JsonPath. Вона вміє перетворювати екземпляри POJO (plain old java object) на текст у форматі JSON і навпаки. Це відбувається одним з трьох способів:

* потоковий API, тобто обробка JSON відбувається як окремі події;
* модель дерева – весь контент представляється у вигляді дерева за допомогою вузлів JsonNode;
* зв’язування даних – перетворення об’єктів за допомогою їхніх аксесорів (гетерів та сетерів), за допомогою яких відбувається доступ до полів об’єкта, а також анотацій.

В даної бібліотеки є багато аналогів, наприклад Gson, JsonUnit, Json-IO та інші, проте вона є однією з найбільш функціональних і в той же момент залишається простою та зрозумілою.

Project Lombok додає функціональність в Java класи за рахунок зміни їхнього вмісту перед початком процесу компіляції. В основі цього закладено патерн Proxy. Ця бібліотека допомагає позбутися такого шаблонного коду:

* конструктори за змовчуванням або ж пусті конструктори без будь-яких параметрів;
* конструктори зі всіма параметрами, які ініціалізують всі поля об’єкта;
* конструктори з обов’язковими парметрами. Дані параметри мають модифікатор final;
* аксесори – гетери та сетери, які використовуються для доступу до полів об’єкта;
* методи для порівняння об’єктів – equals та hashCode;
* метод для перетворення об’єкта класу в рядок – toString.

Також за допомогою цієї бібліотеки можна опустити код для створення логера, за допомогою якого буде здійснюватися логування подій у програмі та реалізувати патерн Builder за допомогою однієї анотації. Це робить його читання набагато простішим, а код – чистішим, що є дуже важливо для процесу додавання нових функцій або зміни вже існуючих, при яких часто дускається забруднення коду, шляхом не видалення коду, який не використовується.

У таблиці 4.3 представлені версії застосованих бібліотек.

Таблиця 4.3.

Версії застосованих бібліотек

|  |  |
| --- | --- |
| **Назва** | **Версія** |
| Jsoup | 1.14.3 |
| Jayway JsonPath | 2.5.0 |
| Apache Commons | 3.12.0 |
| Jackson | 2.13.0 |
| Project Lombok | 1.18.22 |

## **Інструмент для автоматичного збирання та пакування проекту**

Для того, щоб полегшити процес інтеграції всіх необхідних фреймворків та бібліотек у проект, а також для спрощення процесу пакування у один файл (в даному випадку JAR) потрібно використати один з інструментів для автоматичного збирання та пакування. Серед таких варто виділити Ant, Maven та Gradle.

Ant є найстаршим інструментом з усіх перерахованих. Його конфігураційні файли за форматом є XML. Він є процедурним, на відміну від інших інструментів, які є декларативними. Тому для нього потрібно чітко прописувати що робити та коли, а також задавати порядок, в якому все це буде виконуватися, інакше можливе виникнення помилок та збоїв. По функціоналу він є найбіднішим і призначений лише для збору проекту на відміну від двох наступних, які також є інструментами для керування проекту.

Maven досить популярний інструмент, який використовує файли POM для впровадження бібліотек та налаштування збирання проекту. Формат цього файлу – це XML як і у випадку з Ant. Це є недоліком, оскільки його вміст часто повторюється. Наприклад, щоб додати певний модуль необхідно застосувати щонайменше чотири тега: dependency, groupId, artifactId та version, не згадуючи про те, що ще треба їх закрити. Також файли Maven потребують компіляції, що робить процес збору та пакування довшим.

Gradle є найновішим, однак не таким розповсюдженим, як Maven. На відміну від двох попередніх його конфігураційні файли можуть бути написані намовах програмування Groovy або Kotlin, що робить його більш функціональним та простішим у реалізаціях ідей, щодо збирання та менеджменту проекта, проте більш складнішим у розумінні. Варто зазначити, що ці мови програмування є досить близькими до Java. Також Gradle є більш продуктивним, більш оптимізованим та швидшим в порівнянні з його аналогами. Отже, для написання даного ПЗ буде використано саме Gradle.

## **Висновки**

Для написання програмного коду ПЗ для збору та аналізу інформації обрано крос-платформну мову програмування Java. Код створений за допомогою цієї мови може запускатися на будь-яких платформах, на яких можна встановити JVM. Також він не потребує перекомпіляції при запуску.

Каркас ПЗ розроблятиметься за допомогою Spring Framework. Це є досить обширний фреймворк, проте добре розділений на різні модулі, тому можна використати лише необхідні модулі. В порівнянні з своїми аналогами він є найбільш функціональним та практичним, а також простим у налаштуванні. Використано такі його модулі:

* Spring Core Container;
* Spring ORM;
* Spring Instrumentation;
* Spring Test.

Для інтерфейсу було розглянуто такі інструменти: AWT, SWT, Swing та JavaFx. JavaFx є найновішим серед них та пропонує декілька методів конфігурації. Також він є найбільш ефективним та швидким.

База даних може бути як віддаленою так і локальною, але повинна бути реляційною. Було розглянуто такі БД, як MySQL, PostgreSQL, SQLite та H2. H2 є БД, яка може працювати у двох режимах: вбудованому та клієнт-сервер. В порівнянні з іншими БД вона є найбільш продуктивнішою, тому саме вона буде використовуватися.

Для створення програми також будуть використані наступні бібліотеки:

* Jayway JsonPath;
* Jackson;
* Apache Commons;
* Jsoup;
* Project Lombok.

Для інтеграції цих всіх бібліотек в один проект, а також процесу пакування необхідно застосувати інструмент призначений для цього. Серед таких інструментів було розглянуто Ant, Maven та Gradle. Оскільки Gradle пропонує найбільш зручні способи конфігурації, а саме за допомогою програмного коду Groovy та Kotlin, а також є найшвидшим та найбільш оптимізованим в порівнянні з його аналогами, то саме він буде використовуватися для досягнення цієї цілі.

# **РОЗДІЛ 5 ПРОТОТИП ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ЗБОРУ ІНФОРМАЦІЇ**

Прототип програмного забезпечення для збору інформації

66

11

*Літ.*

*Керівник.*

Сінько Ю.І.

*Розроб.*

Петров А.В.

НАУ 16 12 52 000 ПІ

Кафедра ПІ

*Лист*

*Листів*

ТП-215М – 6.050101

*Н-кнтр. Боровик* В.М.

## **Конфігурація проекту**

### **Інтеграція програмних інструментів у проект**

Передусім необхідно створити та задекларувати всі необхідні фреймворки та бібліотеки, які будуть використовуватися під час створення програмного забезпечення, у конфігураційному файлі Gradle – build.gradle. Цей файл буде створюватися за допомогою скриптів на мові Groovy. На рисунку 5.1 зображено скрипт з декларацією бібліотек та фреймворків.



Рис. 5.1. Скрипт з декларацією бібліотек та фреймворків

Проте цієї декларації є недостатньо. Для правильної роботи програмного забезпечення необхідно підключити плагіни у скрипті з назвою plugins та налаштувати версії JavaFx та Jackson у скриптах з навами javafx та versions відповідно. Це показано на рисунку 5.2.



Рис. 5.2. Підключення плагінів та налаштування версій

### **Налаштування підключення до БД**

Можна підмітити, що одним з компонентів є com.h2database:h2:1.4.200. Це є драйвер обраної БД. Наступним кроком буде налаштування з’єднання до БД H2. Spring Boot забезпечує можливістю зробити це легко за допомогою спеціального файлу з назвою application.properties або application.yaml. В залежності від назви він буде мати формат properties або yaml відповідно.

Properties містить рядки, кожен з яких представляє пару типу ключ-значення. YAML – це формат файлів, вміст яких має деревоподібну структуру. В даній конфігурації використовується перший варіант. Для налаштування підключення необхідно визначити такі властивості як:

* URL-адреса;
* назва класу драйвера БД;
* ім’я користувача;
* пароль;
* діалект SQL запитів;
* режим при запуску ПЗ.

Цим властивостям відповідають ключі та значення наведені в таблиці 5.1.

Таблиця 5.1.

Властивості для підключення БД

|  |  |  |
| --- | --- | --- |
| **Назва** | **Ключ** | **Значення** |
| URL-адреса | spring.datasource.url | jdbc:h2:~/parserdb |
| Назва класу драйвера БД | spring.datasource.driverClassName | org.h2.Driver |
| Ім’я користувача | spring.datasource.username | admin |
| Пароль | spring.datasource.password | admin |
| Діалект SQL запитів | spring.jpa.database-platform | org.hibernate.dialect.H2Dialect |
| Режим при запуску ПЗ | spring.jpa.hibernate.ddl-auto | update |

До URL-адреси також варто додати ще два параметри:

* DB\_CLOSE\_ON\_EXIT із значенням FALSE, який буде забезпечувати продовженню роботи БД при не активності ПЗ;
* AUTO\_SERVER із значенням TRUE, який дозволить використовувати БД у режимі клієнт-сервер.

Режим при запуску ПЗ із значенням update забезпечить створення схеми БД на основі Java класів та її оновлення при потребі. Загальний вигляд цих налаштувань зображений на рисунку 5.3.

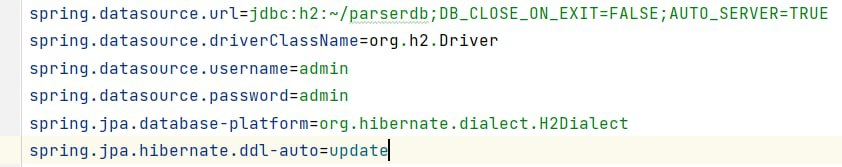


Рис. 5.3. Налаштування підключення до БД

### **Налаштування типу застосунку**

За змовчуванням програма розроблена за допомогою фреймворка Spring Boot – це веб-додаток. Оскільки спроектоване програмне забезпечення є настільне (десктопне) та запускається локально з комп’ютера користувача, то його тип потрібно змінити. Це можна зробити двома способами: програмним та за допомогою спеціальної властивості, яку треба додати до файлу application.properties, який було описано раніше.

Для першого варіанту потрібно для екземпляру класу SpringApplication, який представляє контейнер додатку, змінити значення поля webApplicationType за допомогою публічного сетера на WebApplicationType.NONE. Це буде мати наступний вигляд:

SpringApplication myApp = new SpringApplication(AppConfig.class);

myApp.setWebApplicationType(WebApplicationType.NONE);

Для другого варіанту все набагато простіше. Ця властивість має ключ вигляду spring.main.web-application-type. Вона повинна мати значення none.

## **Прототип інтерфейсу додатка**

### **Створення інтерфейсу**

Клас головного вінка MainWindow повинен наслідуватися від абстрактного класу Application та реалізувати метод start, який власне і буде відповідати за відображення графічного інтерфейсу користувача. Його код можна побачити на рисунку 5.4.

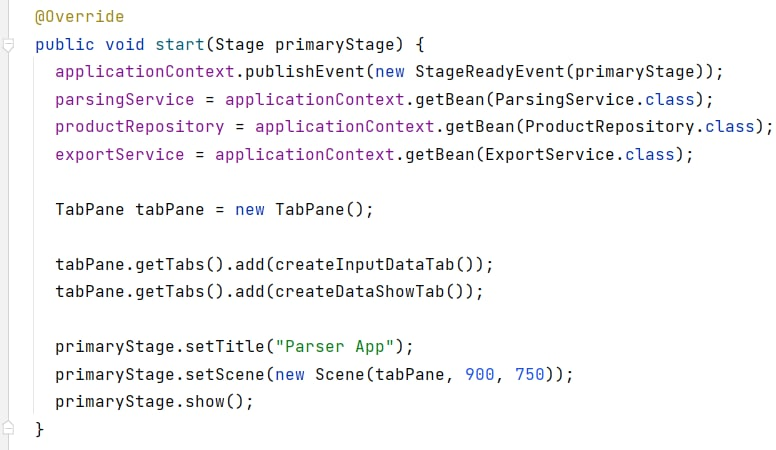


Рис. 5.4. Метод start класу MainWindow

Цей метод можна розділити на дві частини ініціалізація полів, які є Spring beans, тобто класи, створенням та знищенням екземплярів яких займається цей фреймворк, та формування вікна. TabPane – це панель з вкладками. Всього на ній буде розміщуватися дві вкладки:

* для введення початкових даних;
* для перегляду зібраних даних.

Методи createInputDataTab та createShowDataTab відповідають за їх створення відповідно.

Також в цьому методі проставляється назва вікна за допомогою метода setTitle, а також його розміри за допомогою метода setScene. Тобто початкові розміри будуть 900х750. Останній рядок (метод show) відповідає за відображення вікна інтерфейсу користувачу.

### **Головне вікно додатку**

На рисунку 5.5 зображено вигляд головного вінка додатку.

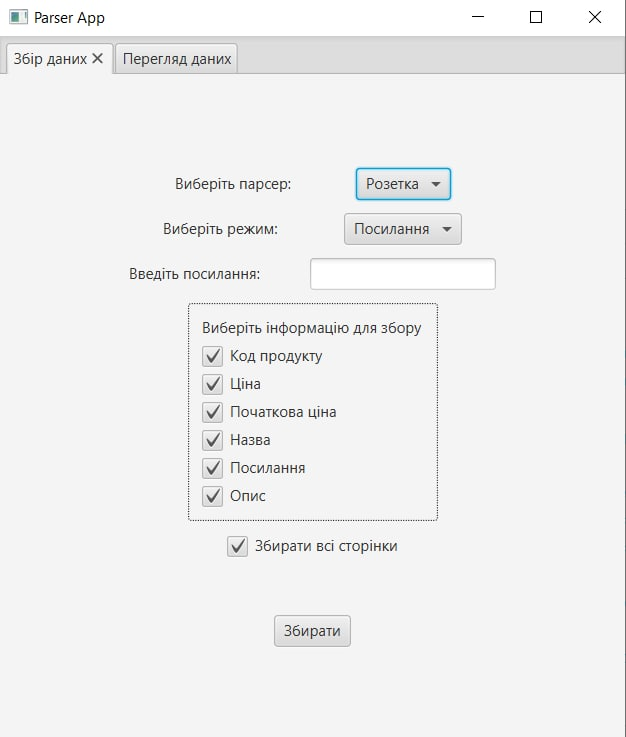


Рис. 5.5. Головне вікно додатку

На першій вкладці панелі меню «Збір даних» розташовані наступні елементи:

1. Список парсерів, кожний з яких відповідає певному сайту. На даний момент вибраний парсер – «Розетка».
2. Список режимів, в яких може працювати парсер. Всього їх два – «Посилання» та «Пошук». Вигляд розгорнутого списка можна побачити на рисунку 5.6.

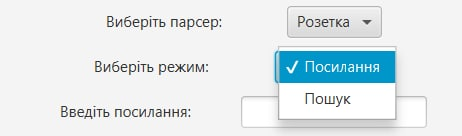


Рис. 5.6. Розгорнутий список режимів роботи парсерів

1. Текстове поле для вводу посилання, якщо режим – «Посилання» або текстове поле для вводу пошукового поле для режиму «Пошук». Вигляд інтерфейсу для останнього режиму зображений на рисунку 5.7.

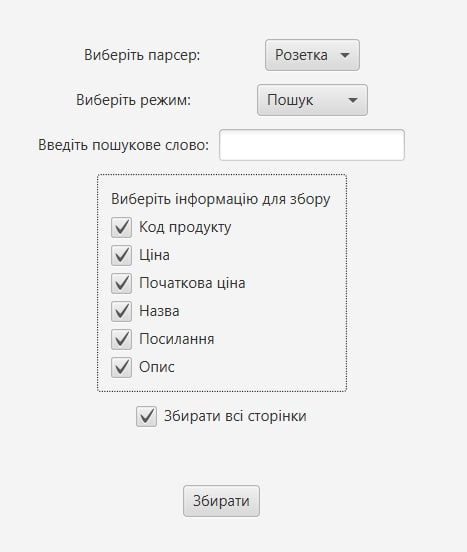


Рис. 5.7. Інтерфейс з обраним режимом «Пошук»

1. Група елементів для вибору необхідної інформації, збір якої буде проводитися. Серед такої інформації є:

* Код продукту;
* Ціна;
* Початкова ціна;
* Назва;
* Посилання;
* Опис.

В даному випадку користувач може зняти відмітки з непотрібної інформації.

1. Вибір для збору зі всіх сторінок. В разі зняття відмітки з даного елемента, користувачу необхідно буде ввести кількість сторінок, з яких буде проводитися збір. При цьому всі наступні сторінки будуть проігноровані. Вигляд такого інтерфейсу зображений на рисунку 5.8.

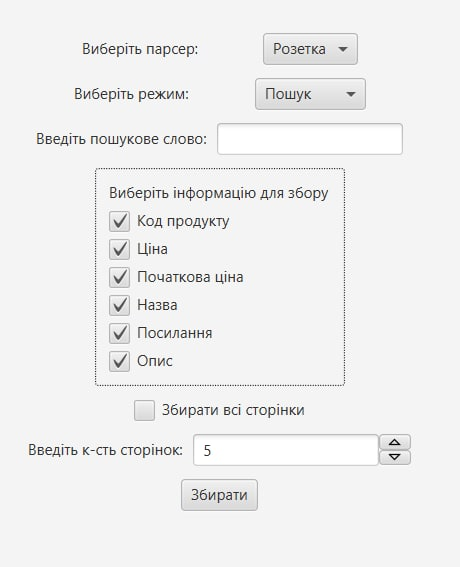


Рис. 5.8. Вигляд інтерфейсу

1. Кнопка «Збирати», при натисканні якої відбувається збір даних з введеними вхідними даними.

Всі елементи даної вкладки розміщені по центру вікна.

Також тут можна перейти на другу вкладку – «Перегляд даних». Її вміст зображено на рисунку 5.9.

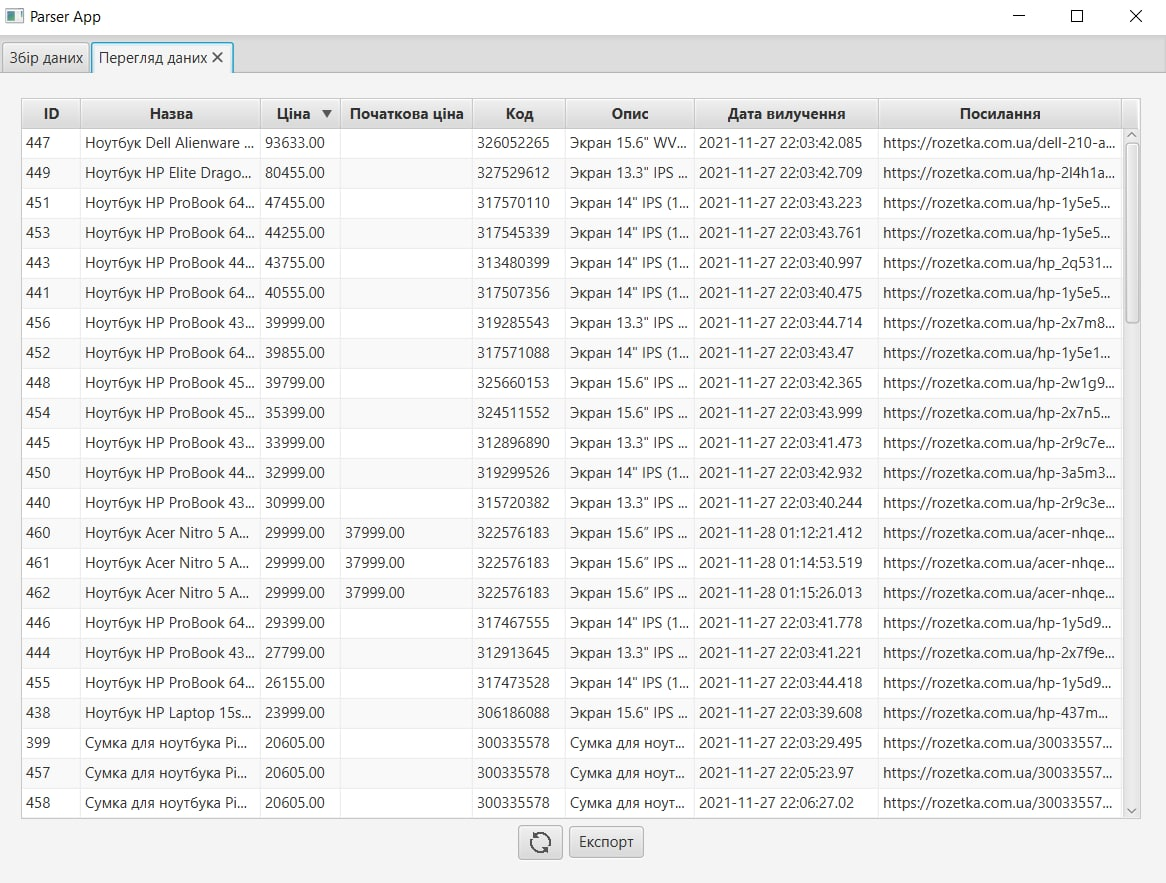


Рис. 5.9. Зображення вкладки «Перегляд даних»

На цій вкладці головним елементом є таблиця з даними. В таблиці є прокрутки по горизонталі та вертикалі. Також є можливість звуження або розширення стовпців. Це є досить зручно для перегляду даних. Інформація в таблиці може бути відсортованою за зростання або спаданням, якщо це число, або за алфавітним порядком. Невеликий чорний трикутник направлений вверх або вниз показує, що сортування відбулося за ним по зростанню або спаданню відповідно. В даному випадку таблицю відсортовано за ціною товару за спадання. З цієї вкладки є можливість повернутися до попередньої. Внизу вікна розміщена панель з кнопками. На даний момент на ній всього дві кнопки. Назви та функції цих кнопок описані в таблиці 5.2.

Таблиця 5.2.

Кнопки на вкладці перегляду даних

|  |  |  |
| --- | --- | --- |
| **Назва кнопки** | **Вигляд** | **Функція** |
| Оновлення |  | Оновлення даних у таблиці. Зазвичай це потрібно після завершення збору нових даних |
| Експорт |  | Ініціює процес експорту даних в файли, шляхом створення нового вікна |

### **Вікно експорту**

Після натискання на кнопку експорту відкривається нове вікно зображене на рисунку 5.10.

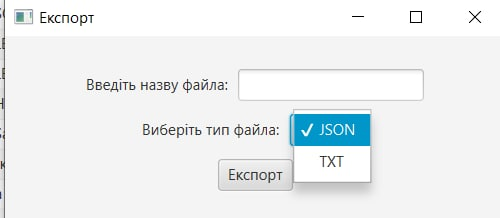


Рис. 5.10. Вигляд вікна експорту

В ньому присутні такі елементи інтерфейсу:

* текстове поле для вводу назви файлу;
* список для вибору формату файлу, в який буде відбуватися експорт. На даний момент доступні лише два формати: JSON та TXT;
* кнопка «Експорт» для підтвердження операції.

### **Діалогові вікна**

Для інформування користувача про успішність операцій використовуються діалогові вікна. На рисунку 5.11 зображене вікно з повідомленням про успішність збору інформації.

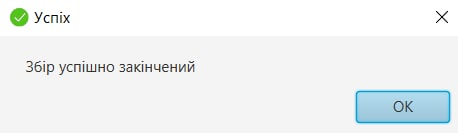


Рис. 5.11. Діалогове вікно з повідомленням про успіх

На рисунку 5.12 зображене вікно з повідомленням про помилку, а саме про неправильні початкові дані.

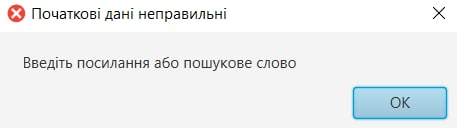


Рис. 5.12. Діалогове вікно з повідомленням про помилку

## **Висновки**

В даному розділі було описано створення програмного забезпечення для збору та аналізу даних. Перш за все необхідно задекларувати всі використовувані бібліотеки та фреймворки у файлі build.gradle для підключення їх до проекту. Варто зауважити, що JavaFx та Jackson потребують вказання версій за допомогою іншого способу. Також потрібно задекларувати плагіни, які допоможуть при розробці програмного забезпечення.

Другим кроком є налаштування підключення до БД. Це можна зробити двома способами. В даному випадку обрано спосіб конфігурації через властивості, які забезпечує Spring Boot, у файлі application.properties.

Третім кроком є налаштування типу застосунку. Оскільки Spring Boot призначений перш за все для створення веб-додатків, а при проектуванні планувалося розробити настільний додаток, то тип додатку треба змінити. Це можна зробити декількома способами. Запропоновано це зробити за допомогою властивості у вже відомому файлі application.properties.

Головне вікно ПЗ для збору та аналізу даних запускається за допомогою метода start. У ньому створюється панель, в якій міститься дві вкладки. Перша призначена для введення початкових даних та початку процесу збору інформації. Деякі елементи інтерфейсу залежать від обраних даних користувача та змінюються динамічно. Друга – для перегляду даних у вигляді таблиці. Вона є досить зручною з можливостями вертикальної та горизонтальної прокрутки, зміни ширини колонок, а також сортуванням даних за значеннями стовпців.

При натисканні на кнопку експорту з’являється нове вікно. В ньому користувач може обрати та ввести додаткові налаштування такі, як назва файлу, а також його формат. Це вікно можна закрити або воно закриється самостійно при підтвердженні операції.

Для інформування користувача про помилки або успішність чи не успішність завершення певної операції використані діалогові вікна. Загалом вони дуже схожі за винятком невеликої іконки, яка знаходиться в лівому верхньому куті. Це робить інтерфейс більш зрозумілим для користувача.

# **ВИСНОВКИ**

Збір інформації відіграє велику роль для існування не тільки великих компаній, а й для простих людей. Якщо правильно його використовувати, то це може зняти навантаження з людини, тим самим звести витрати часу та інших ресурсів до мінімуму. Багато людей навіть побудували свій бізнес на цьому. Наприклад відома європейська компанія Web Data Solutions досить успішно збирає дані з веб-сайтів та продає їх користувачам з різних куточків світу.

Процес збору даних з метою її структуризації та аналізу називається парсингом. Програми, які реалізують та виконують цей процес – парсери. Парсери можуть створюватися для кардинально різних цілей. Ось деякі з них:

* для слідкування за ціновим діапазоном на різні товари, які є на сайті конкурентів;
* збір інформації про продукти: назви, числові коди продуктів, середня оцінка покупців, відгуки і тому подібне. Такий збір може виконуватися також з різною метою, наприклад для наповнення свого сайту або для своїх власних цілей;
* збір метаданих для аналізу вмісту тегів та інші метадані;
* аналіз та перевірка сайту на помилки, що зберігає час та полегшує роботу тестувальників під час проходу по дереві сторінок сайту.

Виходячи з різних цілей можна виділити основні групи або види парсерів. Серед них є:

* парсери для організаторів спільних покупок;
* парсери для аналізу цін конкурентів;
* парсери для наповнення сайтів;
* парсери для SEO-фахівців.

За типами та методами реалізації парсери поділяють на:

* хмарні;
* десктопні;
* браузерні розширення;
* плагіни в exel.

Програмне забезпечення для збору даних може не тільки принести користь, але й при неправильному застосуванні шкоду. Наприклад воно може розпочати DDoS атаку на ресурс, з якого відбувається збір, або значно сповільняти його роботу. Це в свою чергу призводить до бажання власників сайтів захищати їх за допомогою різних методів та сервісів. Тому це слід враховувати при розробці такого програмного забезпечення. Перепони при роботі парсера можуть бути такі:

* складна структура веб-сторінок;
* блокування за ІР-адресою;
* динамічний або часто змінюваний контент ресурсу;
* обов’язкова авторизація на веб-сайті;
* повільна робота ресурсу;
* пастки різного роду і особливо Honeypot;
* captcha.

Всі ці потенційні проблеми можуть бути усунуті, проте деякі вирішення можуть дорого коштувати в прямому сенсі. Наприклад, якщо відбувається блокування за адресою, то необхідно володіти достатньою кількістю цих адрес аби заміняти їх ще до блокування. Шифри captcha можуть бути різними. Деякі з них легко обходити, проте інші вимагають застосування спеціальних платних сервісів. Варто зауважити, що більшість вказаних проблем важко передбачити заздалегідь, так як отримання вмісту веб-сторінок відбувається програмним шляхом, а не звичним для користувачів відкриттям в браузері. Тому одне натискання кнопки може перетворитися у окремий HTTP запит, який повертає у відповідь кардинально інший результат. Ці проблеми слід вирішувати по мірі їх надходження інакше можна реалізувати багато надлишкової логіки, яка буде навпаки перешкоджати роботі програмного забезпечення.

Парсер може бути реалізованим двома методами: висхідний або низхідний. ПЗ для збору даних з висхідним методом, спочатку аналізує всі веб-сторінки, посилання на які містяться в поточній сторінці, а в кінці проводить парсинг початкової сторінки, з низхідним – спочатку аналізує поточну сторінку, а потім ресурси, посилання на які було знайдено.

Загальний алгоритм роботи парсера виглядає наступним чином:

1. Користувач вводить вхідні дані та параметри для парсинга на сайті.
2. Користувач вказує список сторінок або ресурсів, на яких потрібно здійснити збір та аналіз.
3. Програмне забезпечення в автоматичному режимі проводить глибокий аналіз зібраного вмісту і структуризує його.
4. Користувач отримує результат в попередньо обраному форматі.

В даній роботі було спроектовано та розроблено десктопний парсер для збору інформації про товари. В процесі проектування було визначено вимоги до апаратної платформи, ОС, швидкодії, надійності, безпеки, інтерфейсу, а також складено функціональні вимоги. Виділено такі основні функції:

* запуск збору даних за допомогою посилання;
* запуск збору даних за допомогою ключового слова;
* вибір необхідних даних для збору;
* перегляд зібраних даних;
* порівняння даних з метою аналізу;
* завантаження даних у файл.

В якості архітектури обрано package by feature з графічним інтерфейсом користувача. Програмне забезпечення було поділено на логічні та фізичні рівні. Це також відображено на відповідних діаграмах. Логічні рівні включають:

* рівень доступу до БД;
* рівень бізнес-логіки;
* рівень представлення.

Кожний фізичний рівень відповідає певній сукупності класів, які об’єднані в пакети. Серед них виділено такі:

* ui;
* input data;
* export;
* parser;
* request;
* product.

Проектування відбувалося за допомогою діаграм UML та текстового пояснення до них. Було створено такі діаграми:

* прецедентів;
* розгортання;
* послідовності (для кожної функції);
* класів.

Також було складено реляційну модель БД.

Java – це мова програмування, на якій дане програмне забезпечення створювалося. Вона є золотою серединою між простотою та швидкодією. Також це є крос-платформна мова програмування, що дозволить встановлювати та запускати додаток на різних апаратних платформах та ОС. В якості каркасу обрано Spring Boot, який має досить широкий функціонал та хорошу модульність. Графічний інтерфейс користувача створювався за допомогою JavaFx, який серед аналогів є найновішим та більш функціональним. H2 – це база даних, в якій буде зберігатися зібрана інформація. Серед інших БД вона є найпродуктивнішою, про що свідчать проведені заміри.

Для написання застосунку використано ряд інших бібліотек. Вони полегшіть написання програмного коду та допоможуть зберігати його зрозумілим. Це є: Project Lombok, Jsoup, Jayway JsonPath, Apache Commons та Jackson.

Всі ці компоненти будуть інтегровані в один проект за допомогою інструменту для автоматичного збирання та пакування. Gradle є фаворитом серед таких, тому саме він використаний для досягнення цієї цілі.

Також в даній роботі наведено конфігурацію такого застосунку та розроблено прототип, який успішно пройшов тестування. Інтерфейс ПЗ має два вікна: головне та додаткове – для функції експорту. Деякі елементи інтерфейсу змінюються відповідно з взаємодією користувача. Головне вікно має панель вкладок з двома елементами: для введення початкових даних та для перегляду даних. Для інформування про виникнення помилок та успішність виконання різних операцій використано діалогові вікна.

# **СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ**

1. Шипин А. И. 30+ парсеров для сбора данных с любого сайта [Електронний ресурс] / Александр Игоревич Шипин. – 2020. – Режим доступу до ресурсу: https://habr.com/ru/company/click/blog/494020/.
2. 9 Web Scraping Challenges You Should Know [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: https://www.octoparse.com/blog/9-web-scraping-challenges.
3. Wiegers K. Software Requirements / Karl Wiegers., 2003. – 544 с. – (Microsoft Press).
4. Грицюк Ю. І. Аналіз вимог до програмного забезпечення / Юрій Іванович Грицюк., 2018. – 456 с.
5. Wood L. User Interface Design: Bridging the Gap from User Requirements to Design / Larry Wood., 1997. – 320 с. – (CRC Press).
6. Barker T. Writing Software Documentation: A Task-Oriented Approach / Thomas Barker., 2002. – 468 с. – (Longman Pub Group).
7. Martin R. Clean Architecture: A Craftsman's Guide to Software Structure and Design / Robert Martin., 2017. – 432 с. – (Pearson). – (1st edition).
8. Martin R. UML for Java Programmers / Robert Martin., 2003. – 274 с. – (Prentice Hall).
9. C. J. Date. Database Design and Relational Theory / C. J. Date., 2012. – 278 с. – (O'Reilly Media). – (1st edition).
10. Mitchell R. Instant Web Scraping with Java / Ryan Mitchell., 2013. – 72 с.
11. Spring Boot [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: https://spring.io/projects/spring-boot.
12. Jenkov J. JavaFX Tutorial [Електронний ресурс] / Jakob Jenkov. – 2020. – Режим доступу до ресурсу: http://tutorials.jenkov.com/javafx/index.html.