МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«РОЗРОБКА ВЕБЗАСТОСУНКУ МОНІТОРИНГУ ЦІН НА ПРОДУКТИ»**

Виконав: студент 2 курсу, групи 8.1212-іпз-1
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми інженерія програмного забезпечення
(назва освітньої програми)
В.О. Калашник
(ініціали та прізвище)

Керівник завідувач кафедри програмної інженерії, доцент, к.ф.-м.н. Лісняк А.О.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент доцент кафедри комп'ютерних наук, доцент, к.т.н. Решевська К.С.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти магістр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2023 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Калашнику Володимиру Олександровичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка вебзастосунку моніторингу цін на продукти

керівник роботи Лісняк Андрій Олександрович, к.ф.-м.н., доцент
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 01 » травня 2023 року № 642-с

2. Строк подання студентом роботи 27.11.2023 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.
3. Розробка вебзастосунку моніторингу цін на продукти.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 03.05.2023 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	15.05.2023	
2.	Збір вихідних даних.	05.06.2023	
3.	Обробка методичних та теоретичних	23.06.2023	
	джерел.		
4.	Розробка першого та другого розділу.	28.08.2023	
5.	Розробка третього розділу.	30.10.2023	
6.	Оформлення та нормоконтроль	20.11.2023	
	кваліфікаційної роботи магістра.		
7.	Захист кваліфікаційної роботи.	14.12.2023	

Студент _____ В.О. Калашник
(підпис) (ініціали та прізвище)

Керівник роботи _____ А.О. Лісняк
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота магістра «Розробка вебзастосунку моніторингу цін на продукти»: 62 с., 35 рис., 1 табл., 19 джерел, 3 додатки.

АДМІН-ПАНЕЛЬ, ВЕБСКРАПІНГ, МОНІТОРИНГ ЦІН, LANDO, LARAVEL, LARAVEL ORCHID, MYSQL, PM2, PHP, PYTHON, RABBITMQ, SCRAPY.

Об’єкт дослідження – системи моніторингу цін на продукти в електронній комерції.

Предмет дослідження – методи збору та аналізу цінової інформації з вебсайтів різних роздрібних магазинів.

Мета роботи: розробка та впровадження ефективної системи моніторингу цін на продукти, яка надає актуальну та історичну інформацію про ціни.

Метод дослідження – комплексний аналіз сучасних методів скрапінгу, порівняння та узагальнення різних технологій веброзробки, практичне тестування та реалізація вебскраперів і вебінтерфейсу.

У магістерській кваліфікаційній роботі було досліджено проблематику актуальності та надійності інформації про ціни в електронній комерції. Було розроблено та впроваджено систему, що автоматизує збір цінової інформації з вебсайтів, використовуючи ефективні техніки скрапінгу. Система забезпечує користувачів інструментами для відстеження змін цін, включаючи доступ до історичних даних, що важливо для аналізу цінових тенденцій.

Процес розробки включав створення адміністративної панелі на Laravel для управління даними та моніторингу, а також реалізацію скраперів на Scrapy для ефективного збору інформації.

Результати цієї роботи вносять важливий вклад у сферу моніторингу цін у електронній комерції, пропонуючи новий інструмент для збору та аналізу цінових даних, що може бути корисним для споживачів та бізнес-власників.

SUMMARY

Master’s qualifying paper «Development of a Web Application of Products Price Monitoring»: 62 pages, 35 figures, 1 table, 19 references, 3 supplements.

ADMIN PANEL, LANDO, LARAVEL, LARAVEL ORCHID, MYSQL, PHP, PM2, PRICE MONITORING, PYTHON, RABBITMQ, SCRAPY, WEB SCRAPING.

The object of the study are systems for monitoring product prices in e-commerce.

The subject of research are methods of collecting and analyzing price information from websites of various retail stores.

The aim of the study is to develop and implement an effective system for monitoring product prices, providing current and historical price information.

The methods of research are comprehensive analysis of modern scraping methods, comparison and generalization of various web development technologies, practical testing and implementation of web scrapers and web interface.

In the Master’s qualification work, the issue of relevance and reliability of price information in e-commerce was investigated. A system that automates the collection of price information from websites using effective scraping techniques was developed and implemented. The system provides users with tools for tracking price changes, including access to historical data, which is important for analyzing price trends.

The development process included the creation of a Laravel-based administrative panel for data management and monitoring, as well as the implementation of scrapers on Scrapy for efficient information gathering.

The results of this work make a significant contribution to the field of price monitoring in e-commerce, offering a new tool for collecting and analyzing price data, which can be useful for consumers and business owners.

ЗМІСТ

ВСТУП

У сучасному світі, де електронна комерція стає невід'ємною частиною нашого повсякденного життя, актуальність моніторингу цін на продукти виявляється надзвичайно високою. Споживачі та бізнес-власники в пошуку оптимальних умов покупки та ціноутворення шукають інструменти, які надають точну та актуальну інформацію про динаміку змін цін.

Існуючі рішення в цій сфері часто опираються на дані, які надають самі магазини, що може призводити до неоднозначності та недостовірності інформації. Більшість з них обмежені лише поточними ціновими пропозиціями, а відсутність можливості перегляду історичних даних позбавляє користувачів засобів для глибокого аналізу та прийняття обґрунтованих рішень.

Цей дипломний проєкт спрямований на розробку вебзастосунку моніторингу цін на продукти, який не тільки надає користувачам можливість додавати та відслідковувати ціни на обрані товари, але й забезпечує доступ до повноцінної історичної динаміки зміни цін. Пропонований підхід базується на використанні вебадмінки на Laravel для зручної взаємодії користувачів і скраперів на Scrapy для систематичного збору даних безпосередньо з вебсайтів магазинів.

У цьому контексті, дипломний проєкт вирішує проблему відсутності повноцінних інструментів для моніторингу та аналізу цін на продукти, роблячи акцент на надійності та доступності історичних даних. Очікується, що розроблений вебзастосунок надасть користувачам нові можливості для ефективного вибору та планування покупок.

Основними задачами цього дослідження є:

- створення скраперів для автоматизованого збору цінової інформації з вебсайтів різноманітних роздрібних магазинів;
- розробка інтерфейсу адміністративної панелі для зручного управління та моніторингу цін на продукти;

– реалізація системи моніторингу для виявлення та реєстрації змін у цінах на продукти, надаючи користувачам доступ до історичної динаміки змін цін.

1 ОГЛЯД ТЕХНОЛОГІЙ

1.1 Цілі і задачі скрапінгу

Вебскрапери – це складні інструменти, розроблені для автоматичного збору інформації з вебсайтів. Вони відіграють важливу роль у різноманітних сферах, оптимізуючи процес збору даних. Перелік деяких задач, які допомагає вирішити вебскрапер:

- автоматизований збір даних – основною функцією вебскраперів є автоматизація збору даних із різних вебсайтів, ця автоматизація має вирішальне значення для ефективного вилучення великомасштабних даних;
- підвищення доступності даних – збираючи інформацію з різноманітних вебджерел, вебскрапери роблять дані більш доступними, це особливо важливо в епоху цифрових технологій, коли величезна кількість інформації поширена в інтернеті;
- спрощення обробки інформації – скрапери не лише збирають дані, але й допомагають упорядковувати та структурувати їх, це спрощує процес аналізу та використання даних для різних цілей;
- збереження часу та ресурсів – автоматизація, яку забезпечують вебскрапери, економить значну кількість часу та ресурсів, які в іншому випадку були б витрачені на збір даних вручну;
- полегшення всебічного огляду даних – агрегуючи дані з кількох джерел, скрапери дають змогу отримати більш повний огляд інформації, доступної щодо певної теми чи сектора;
- впровадження стратегій, керованих даними – в епоху, коли прийняття рішень на основі даних є ключовим, вебскрапери забезпечують вхідні дані, необхідні для стратегічного планування та виконання в різних сферах бізнесу;

– адаптація до різноманітних потреб у даних, вебскрапери – це універсальні інструменти, які можна налаштувати відповідно до конкретних вимог до даних, що робить їх можливими до застосування в широкому діапазоні контекстів.

По суті, вебскрапери є невід'ємною частиною сучасної цифрової екосистеми, пропонуючи автоматизовані, ефективні та універсальні засоби збору та використання вебінформації.

1.2 Класифікація скраперів

Класифікація скраперів включає різні типи інструментів, призначених для збору даних з вебсторінок. Два популярних приклади цього – Puppeteer і Scrapy, які використовуються для вебскрапінгу, але в різних контекстах і з різними підходами.

1.2.1 Puppeteer

Puppeteer – це бібліотека Node.js, яка надає можливість керувати браузером Chrome або Chromium через програмний інтерфейс [1]. Розроблена командою Google, вона зосереджується на автоматизації браузерних завдань, які особливо корисні для розробників вебсторінок та тестувальників. Одна з ключових особливостей Puppeteer полягає в тому, що вона дозволяє рендерити динамічний JavaScript-контент, що робить її ідеальною для скрапінгу динамічно генерованих вебсторінок, які використовують багато клієнтського JavaScript.

З Puppeteer користувачі можуть програмно відправляти запити, імітувати взаємодії з вебсторінкою, такі як кліки по посиланнях, заповнення форм, прокрутка сторінок тощо. Це робить його могутнім інструментом для автоматизації тестування вебдодатків, створення скріншотів вебсторінок або

PDF-файлів для звітів та аналізу. Puppeteer також використовується для моніторингу змін на вебсторінках, що може бути корисно для виявлення оновлень або змін в контенті.

Оскільки Puppeteer використовує повноцінний браузер для виконання своїх завдань, він може бути більш ресурсомістким, ніж інші інструменти, що працюють на рівні HTTP-запитів. Однак це також означає, що Puppeteer може взаємодіяти з складними вебдодатками, які залежать від великої кількості клієнтського JavaScript, тим самим надаючи більш глибоке розуміння того, як користувачі сприймають вебсторінку.

Завдяки своїй гнучкості та потужності, Puppeteer став популярним вибором серед розробників для різних завдань пов’язаних з веббраузером, включаючи автоматизований скрапінг вебданих, автоматизацію тестування вебдодатків, а також генерацію зображень вебсторінок. Його можливість працювати з реальним браузером і виконувати складні сценарії взаємодії з користувацьким інтерфейсом робить його незамінним у сучасному наборі інструментів розробника.

1.2.2 Scrapy

Scrapy є потужним фреймворком на Python, призначеним для вебскрапінгу та збору даних з інтернету. Розроблений для зручності та ефективності, Scrapy особливо корисний для збору великих обсягів даних із різноманітних вебджерел. Він підходить для широкого спектру задач від простого скрапінгу сторінок до складних проєктів краулінгу вебсайтів.

Основна перевага Scrapy полягає в його асинхронній архітектурі, яка дозволяє обробляти велику кількість запитів паралельно, забезпечуючи високу продуктивність та ефективність [2]. Це робить його ідеальним для проєктів, де необхідно швидко збирати дані з великої кількості вебсторінок. Scrapy також має потужні можливості для обробки і фільтрації зібраних даних, що дозволяє

користувачам ефективно витягувати корисну інформацію.

Фреймворк пропонує гнучку систему плагінів, яка дозволяє розширити його функціональність за допомогою додаткових модулів. Це дозволяє розробникам легко інтегрувати Scrapy з іншими інструментами та сервісами, наприклад, з базами даних, системами черг повідомлень або інструментами для обробки даних.

Однією з ключових особливостей Scrapy є його потужний механізм селекторів, який дозволяє точно визначати, які дані потрібно зібрати з вебсторінок. Використовуючи XPath або CSS селектори, користувачі можуть легко вказувати елементи вебсторінок, з яких потрібно витягувати дані. Ця функціональність робить Scrapy особливо корисним для задач, де точність витягування даних має вирішальне значення.

Фреймворк дає дуже потужні можливості для паралельної обробки запитів, що значно відрізняє його від багатьох інших інструментів у цій галузі. В основі паралельності Scrapy лежить його асинхронна архітектура, заснована на Twisted, асинхронному мережевому фреймворку. Ця особливість дозволяє Scrapy виконувати одночасно велику кількість HTTP-запитів, значно збільшуючи продуктивність і швидкість скрапінгу.

Завдяки асинхронності, Scrapy може одночасно обробляти запити до різних вебсторінок, не чекаючи завершення кожного запиту перед початком наступного. Це особливо ефективно при скрапінгу великих сайтів або при роботі з декількома вебресурсами одночасно, оскільки зменшує загальний час, необхідний для збору даних.

Scrapy також добре підходить для великих проєктів завдяки своїй масштабованості та гнучкості управління проєктами. Він підтримує різні стратегії краулінгу, що дозволяє розробникам налаштовувати поведінку скрапера відповідно до конкретних вимог проєкту. Також важливо відзначити, що Scrapy має активну спільноту та широкий набір документації, що робить його доступним для нових користувачів та забезпечує підтримку у вирішенні можливих проблем або при розробці складних проєктів.

Scrapy не тільки дозволяє ефективно збирати дані, але й включає можливості для їхньої подальшої обробки і зберігання (див. рис. 1.1 [3]). Це означає, що дані можна очищати, трансформувати та зберігати у різних форматах та базах даних без необхідності використання додаткових інструментів. Така інтеграція робить Scrapy вельми зручним інструментом для повного циклу обробки даних, від збору до зберігання.

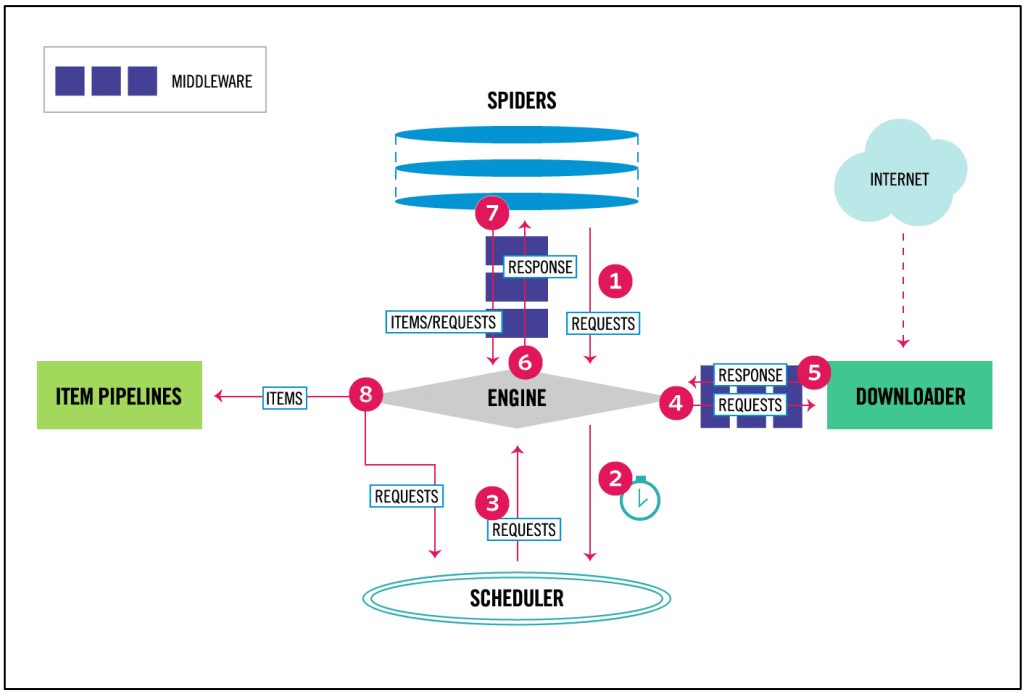

Рисунок 1.1 – Архітектура роботи Scrapy

Крім того, Scrapy має вбудовані засоби для управління помилками та обмеження доступу, що дозволяє розробникам краще контролювати процес скрапінгу та забезпечувати його стабільність. Це особливо важливо при роботі з великою кількістю даних або коли скрапер повинен працювати в умовах, що постійно змінюються.

У підсумку, Scrapy стоїть як високоефективний інструмент для вебскрапінгу, що поєднує в собі масштабованість, гнучкість та потужність. Його здатність обробляти складні завдання по збору даних та широкий спектр можливостей роблять його вибором номер один для багатьох професіоналів у сфері обробки даних та веброзробки.

1.2.3 Порівняння інструментів для скрапінгу

Для вибору інструменту, який буде використовуватися при розробці була підготовлена порівняльна таблиця (див. табл. 1.1), яка допоможе визначити, який саме фреймворк слід використовувати для вирішення поставленої задачі.

Таблиця 1.1 – Порівняння Puppeteer і Scrapy

Характеристика	Puppeteer	Scrapy
Мова програмування	JavaScript (Node.js)	Python
Основне призначення	Контроль браузера для рендерингу JavaScript, автоматизація та скрапінг вебсторінок.	Вебскрапінг, збір даних, краулінг сайтів.
Підхід до рендерингу	Повноцінний браузер (Chromium/Chrome).	Асинхронні HTTP-запити (не рендерить JavaScript на стороні клієнта).
Обробка JavaScript	Повна підтримка (рендерить динамічний контент).	Відсутня.
Масштабованість	Підходить для складних задач з динамічним вмістом, але може бути ресурсомістким.	Висока, оптимальна для великомасштабних проєктів з використанням паралельних запитів.
Управління користувацьким інтерфейсом	Повне (кліки, прокрутка, заповнення форм).	Обмежене (без прямої взаємодії з користувацьким інтерфейсом).

Продовження табл. 1.1

Характеристика	Puppeteer	Scrapy
Збір даних	Вручну налаштовувані сценарії для специфічних вебсторінок.	Використання селекторів CSS/XPath для збору даних.
Продуктивність	Залежить від складності вебсторінок, може бути повільнішим через повноцінний рендеринг.	Висока, особливо при використанні асинхронних запитів.
Легкість використання	Вимагає розуміння JavaScript та взаємодії з браузером.	Простий у використанні для тих, хто знайомий з Python, має велику спільноту.

1.3 Функціональні і технічні вимоги до вебзастосунку

Функціональні вимоги до вебзастосунку для моніторингу цін на продукти поєднують в собі важливі аспекти управління, гнучкості та адаптивності.

Інтерфейс адміністратора є ключовим компонентом системи. Він повинен забезпечувати легкість у додаванні, видаленні та редагуванні URL-адрес різних роздрібних магазинів. Цей інтерфейс має бути інтуїтивно зрозумілим, щоб адміністратор міг швидко асоціювати кожен URL з конкретним продуктом чи групою продуктів. Крім того, система має забезпечувати можливість легкого додавання нових користувачів, що дозволить делегувати обов’язки з моніторингу цін іншим членам команди.

Функціональність моніторингу цін є важливою частиною системи. Вебскрапери, інтегровані в систему, повинні ефективно збирати дані про ціни з вебсторінок. Вони повинні вміти ідентифікувати інформацію про ціни, наявність

товарів та спеціальні пропозиції, при цьому важливо бути достатньо гнучкими для адаптації до різних форматів вебсторінок. Скрапери мають бути спроєктовані таким чином, щоб легко адаптуватися до змін у структурі вебсторінок, що дозволить зберігати актуальність і точність зібраних даних. Це особливо важливо, оскільки вебсайти роздрібних магазинів часто оновлюють свій дизайн та структуру.

Щодо адаптивності системи, вона має бути спроєктована таким чином, щоб легко інтегрувати нові магазини. Це означає, що процес додавання нового магазину до системи моніторингу не повинен вимагати складних технічних змін чи великих втручань у код або базу даних. Такий підхід забезпечить масштабованість та гнучкість системи, дозволяючи швидко реагувати на ринкові зміни та розширювати перелік моніторингу без значного збільшення ресурсів чи часу, витраченого на технічну підтримку.

У підсумку, вебзастосунок для моніторингу цін на продукти має бути зручним у використанні, гнучким у технічному втіленні, та ефективним у зборі та аналізі даних, щоб забезпечити точний і своєчасний моніторинг цін у різноманітних роздрібних магазинах.

1.3.1 Вибір магазинів для скрапінгу

При виборі магазинів для скрапінгу в рамках розробки вебзастосунку для моніторингу цін, було вирішено зосередитись на двох великих українських роздрібних мережах: Allo (https://allo.ua) та Touch (https://touch.com.ua). Обидва магазини були обрані через їх значну присутність на ринку та широкий асортимент товарів, що робить їх важливими джерелами для моніторингу цін.

Особливістю цих магазинів є те, що вони віддають інформацію про ціни безпосередньо у HTML відповіді, що спрощує процес скрапінгу. Це означає, що для отримання даних про ціни не потрібно виконувати складні маніпуляції з динамічними запитами або JavaScript-рендерингом. Скраперам просто потрібно

парсити отриману HTML-структуру та витягувати з неї відповідну інформацію про ціни.

Для ефективного збору даних з цих сайтів, скрапери повинні бути спроєктовані таким чином, щоб вони могли розпізнавати та коректно обробляти HTML-структуру (див. рис. 1.2 і 1.3) цих конкретних вебсайтів. Важливо забезпечити, що скрапери можуть виявляти та адаптуватися до будь-яких змін у структурі HTML, які можуть відбуватися у майбутньому. Це забезпечить стабільність та надійність процесу моніторингу цін, незалежно від можливих оновлень на сайтах магазинів.

З урахуванням цих особливостей, скрапінг сайтів Allo та Touch стане важливою частиною процесу моніторингу, дозволяючи точно та ефективно відслідковувати зміни цін на різні товари, представлені в цих магазинах.

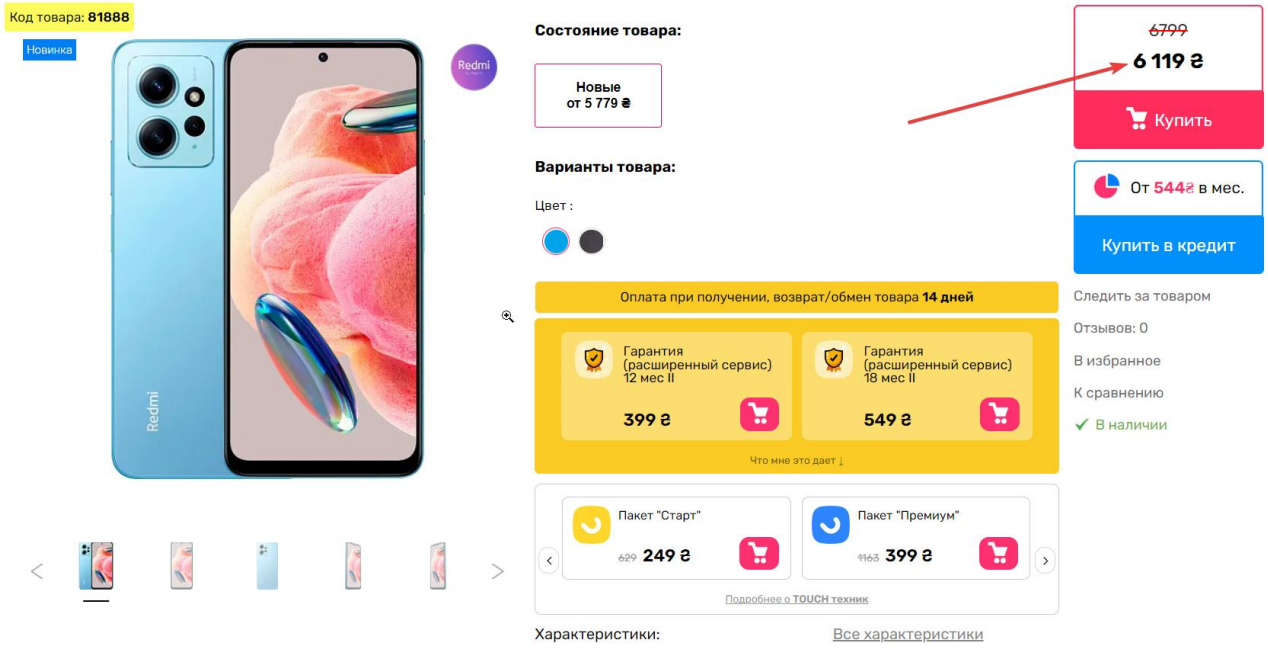

Рисунок 1.2 – Сторінка товару з магазина Allo

```
<meta itemprop="priceCurrency" content="UAH" />
<meta itemprop="price" content="6119" />
<link itemprop="availability" href="https://schema.org/InStock">
```

Рисунок 1.3 – Частина HTML-відповіді сторінки товару з магазина Allo (з інформацією про ціну)

1.3.2 Технічні інструменти для реалізації

Для реалізації вебзастосунку було обрано набір технічних інструментів, кожен з яких виконує ключову роль у розвитку та функціонуванні системи. Laravel Orchid використовується для розробки адміністративної панелі вебзастосунку. Цей інструмент, інтегрований у фреймворк Laravel, полегшує процес створення форм і інших елементів адмін-панелі, мінімізуючи необхідність написання HTML коду [4]. Він забезпечує ефективність управління інтерфейсом, збільшуючи швидкість розробки та гнучкість налаштувань.

Для вебскрапінгу було обрано Scrapy – високопродуктивний інструмент на Python, який ідеально підходить для ефективного збору даних з вебсайтів (див. пункт 1.2.2), зокрема з сайтів Allo та Touch. Слід зазначити, що Scrapy був обраний саме через те, що магазини віддають інформацію у HTML відповіді, а це означає, що немає ніякої необхідності у використанні більш складних інструментів для збору інформації, таких як Puppeteer (див. пункт 1.2.1), бо це буде створювати лише додаткове навантаження, як на сам сайт, з якого збираються дані, так і на клієнта (в даному випадку наш скрапер), який їх буде отримувати.

Для комунікації між вебзастосунком і скраперами обрано RabbitMQ, систему черг повідомлень, яка дозволяє асинхронно обробляти завдання скрапінгу. Використання RabbitMQ є ключовим для забезпечення ефективної комунікації та масштабування системи. RabbitMQ, як система управління чергами повідомлень, дозволяє організувати асинхронне виконання завдань між різними компонентами системи [5], зокрема між адміністративним інтерфейсом та вебскраперами. Ця можливість асинхронної обробки даних є особливо важливою у випадку, коли система потребує одночасного виконання великої кількості завдань скрапінгу, що може статися при збільшенні кількості моніторингових магазинів або продуктів.

RabbitMQ дозволяє ефективно розподіляти навантаження між серверами та скраперами, зменшуючи ризик перевантаження системи та забезпечуючи

стабільність її роботи. Це досягається завдяки чергам повідомлень, де завдання можуть бути відкладені або розподілені серед доступних ресурсів обробки.

Також RabbitMQ сприяє простоті масштабування системи. Зі збільшенням обсягів даних або з розширенням функціональності вебзастосунку, наприклад, при додаванні нових магазинів для моніторингу, система може легко адаптуватися без потреби в глобальному перепроєктуванні архітектури. RabbitMQ дозволяє додавати нові інстанси скраперів або обробників даних, які можуть бути інтегровані в систему без значних змін у основному коді або структурі.

У підсумку, RabbitMQ відіграє важливу роль у забезпеченні еластичності, надійності та масштабованості вебзастосунку для моніторингу цін. Цей інструмент допомагає управляти великими обсягами даних і високим навантаженням, забезпечуючи гладке та ефективне функціонування системи незалежно від її розміру чи складності.

В якості сервісу для зберігання даних буде використана MySQL – надійна та широко вживана система управління реляційними базами даних, яка підходить для зберігання великих обсягів інформації. MySQL забезпечує швидкий доступ до даних [6], ефективно організовуючи інформацію, отриману від скраперів, та полегшуючи доступ до історичних даних для аналізу та візуалізації.

Цей комплекс інструментів забезпечить створення ефективного, гнучкого та надійного інструменту, підвищуючи продуктивність та забезпечуючи точність зібраних даних.

1.4 Висновки до розділу 1

У цьому розділі було детально пропрацьовано основні аспекти розробки вебзастосунку для моніторингу цін на продукти. Особлива увага була приділена розгляду цілей та задач скрапінгу, які виявились ключовими для збору

актуальної інформації з вебсайтів. Аналіз і класифікація інструментів для скрапінгу, таких як Puppeteer та Scrapy, дозволив глибше зрозуміти потенціал та обмеження кожного з них, сприяючи обґрунтованому вибору інструментарію.

Особливо важливим став вибір технічних інструментів для реалізації проєкту, зокрема Laravel Orchid, RabbitMQ, MySQL та Scrapy. Цей вибір підкреслює зосередженість на ефективності, гнучкості та масштабованості рішення, що є критично важливим для успішної реалізації вебзастосунку. Слід підкреслити, що застосунок необхідно проєктувати таким чином, щоб його в подальшому було легко масштабувати і розширювати можливості, а саме – додавання нових магазинів не повинно потребувати значних змін у кодовій базі.

Були чітко сформульовані функціональні вимоги, яким повинен задовільняти фінальний варіант вебзастосунку для того, щоб вважати завдання успішно виконаним.

Вибір магазинів Allo та Touch як джерел для скрапінгу даних показує практичний підхід до визначення релевантних джерел інформації. Це дозволяє зосередитись на конкретних цільових ринках, забезпечуючи актуальність та точність зібраних даних. Такий підхід сприяє збільшенню ефективності збору даних та покращенню якості інформації, яка надається кінцевим користувачам вебзастосунку.

В результаті, цей розділ демонструє ретельну підготовку до створення вебзастосунку. Чітко сформульовані функціональні вимоги та вибір специфічних технічних рішень забезпечують міцну основу для ефективної реалізації проєкту та його подальшого розвитку.

2 ПРОЄКТУВАННЯ ДОДАТКУ

2.1 Аналіз вимог – визначення акторів

При проєктуванні застосунку, ключовим аспектом є визначення та аналіз акторів системи. Актори – це особи або компоненти системи, які безпосередньо взаємодіють з нею, виконуючи різні ролі та функції. Важливість цього аналізу полягає у забезпеченні чіткого розуміння потреб кожного актора, їх впливу на систему та вимог до її функціональності.

Визначення акторів допомагає в проєктуванні інтерфейсів, розробці функціональності та у формуванні загальної архітектури системи. Таким чином, цей аналіз є критичним для розуміння, як різні учасники будуть використовувати систему, та які основні вимоги мають бути враховані під час розробки. Нижче наведено таблицю, що описує ключових акторів вебзастосунку, їх основні обов’язки та способи взаємодії з системою.

Основними акторами, які будуть взаємодіяти з системою будуть:

- адміністратор системи;
- користувачі системи;
- вебскрапери;
- технічна підтримка.

Кожен з цих акторів відіграє унікальну роль у функціонуванні та управлінні системою.

Адміністратор системи несе відповідальність за загальне управління вебзастосунком. Його основні обов’язки включають управління URL-адресами роздрібних магазинів, додавання та видалення користувачів, а також налаштування параметрів скраперів. Адміністратор взаємодіє з системою через внесення змін до бази даних, оновлення конфігурацій, перевірку звітів про ефективність скраперів та управління правами доступу користувачів.

Користувачі системи, такі як аналітики чи менеджери з закупівель,

використовують систему для перегляду та аналізу зібраних даних. Вони відповідають за аналіз зібраних даних, виявлення тенденцій та змін цін на ринку, а також за підготовку звітів. Їхня взаємодія з системою включає перегляд даних в інтерфейсі системи, використання інструментів для генерації звітів та аналізу, а також внесення запитів на зміни або оновлення даних.

Вебскрапери є автоматизованими скриптами, які збирають інформацію з вебсторінок роздрібних магазинів. Вони забезпечують збір даних про ціни, наявність товарів та спеціальні пропозиції. Взаємодія скраперів з системою полягає в автоматичному скануванні визначених URL-адрес, визначенні змін у цінах та наявності товарів, і передачі цієї інформації у базу даних системи.

Технічна підтримка включає фахівців, які забезпечують стабільну роботу системи та вирішують будь-які технічні проблеми. Їхні обов’язки охоплюють підтримку роботи серверів, оновлення програмного забезпечення та вирішення проблем з безпекою. Вони забезпечують моніторинг стану серверів та інфраструктури, впровадження оновлень та патчів безпеки, реагують на технічні збої та запити на підтримку, забезпечуючи безперервну роботу системи та її компонентів.

2.2 Діаграма послідовності

Діаграма послідовності – це важливий інструмент у процесі розробки програмного забезпечення, який використовується для візуалізації послідовності процесів та взаємодій між різними компонентами системи [7]. Вона допомагає розробникам та аналітикам зрозуміти, як дані та запити переміщуються через систему, як вони обробляються та які дії виконуються відповідно до цих даних.

В контексті розробки вебзастосунку, діаграма послідовності показує, як взаємодіють ключові компоненти системи: адміністратор, панель адміністратора, база даних та вебскрапери (див. рис. 2.1).

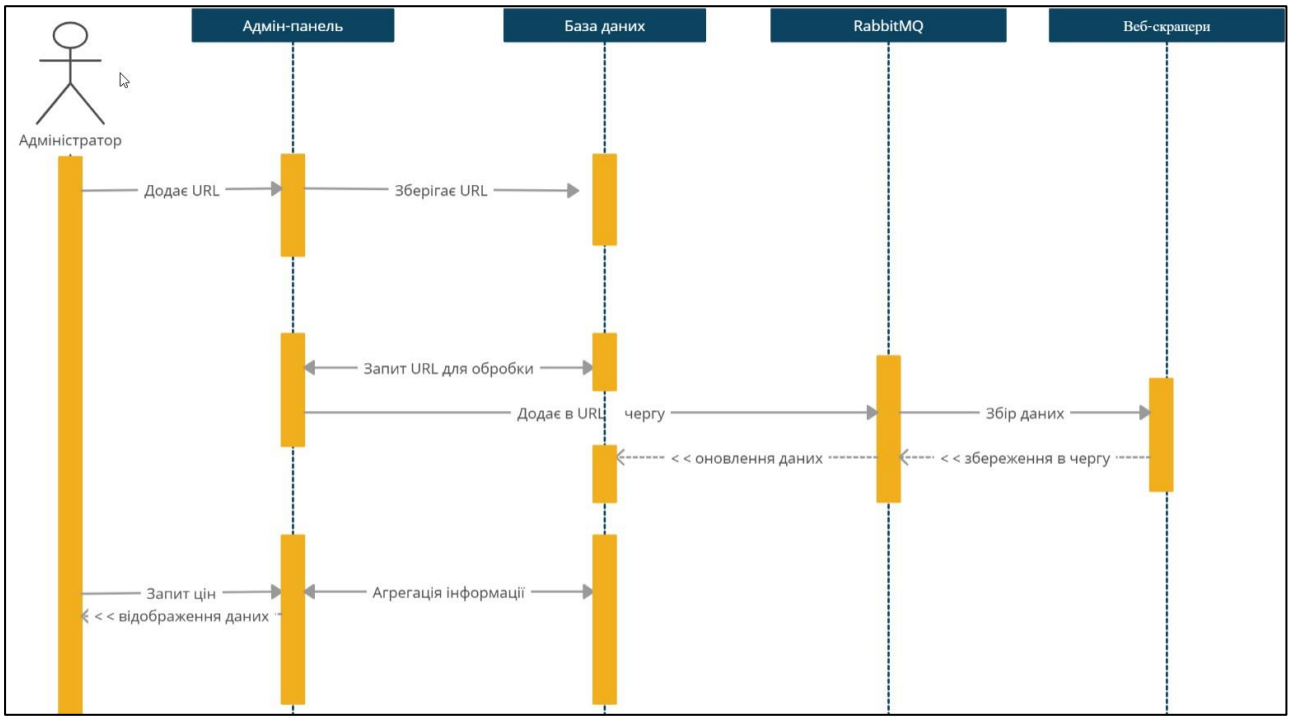

Рисунок 2.1 – Діаграма послідовності дій для адміністратора системи

Ця діаграма послідовності не тільки демонструє, як проходять дані та запити між різними частинами системи, але й допомагає ідентифікувати можливі проблеми або вузькі місця в потоці даних. Також вона є корисною для нових членів команди, що долучаються до проєкту, надаючи їм швидкий огляд архітектури та взаємодій всередині системи. Використовуючи діаграму послідовності, команда може ефективно планувати та розробляти систему, забезпечуючи, що всі компоненти працюють гладко і синхронно.

2.3 Мокапи вебзастосунку

Мокапи важливі для розробки вебзастосунків, адже вони допомагають візуалізувати структуру та дизайн інтерфейсу, перш ніж приступити до програмування. Вони є ключовим елементом у процесі проєктування, оскільки дозволяють розробникам та зацікавленим сторонам зрозуміти та обговорити структуру та функціональність застосунку, забезпечуючи візуальне представлення майбутнього продукту. Мокап – це візуальна модель інтерфейсу,

яка демонструє основні елементи та структуру, але не включає в себе деталізований дизайн чи функціональність.

Для підготовки мокапів платформи будемо використовувати готовий інструмент Wireframe. Wireframe.cc – це інтуїтивно зрозумілий вебсервіс для створення мокапів, який дозволяє користувачам швидко візуалізувати основну структуру вебсторінок або інтерфейсів додатків [8].

Так як, при плануванні, в якості основного інструменту для розробки адмін-інтерфейсу було обрано Laravel Orchid, то, при створенні мокапів, будемо опиратися на готові елементи, які пропонує ця бібліотека.

Для того, щоб зрозуміти, які саме елементи доступні ми можемо відвідати демо сторінку, яку пропонують розробники.

Сторінки авторизації і адміністрування юзерами одразу ж доступні після встановлення бібліотеки, тому для них ми не будемо створювати мокапи, але, наприклад, можемо одразу ж використати деякі елементи із сторінки для адміністрування юзерів на сторінці списку продуктів і детальному огляді продукту.

Використовуючи приклади вже готових сторінок створимо мокап для сторінки «Список продуктів» (див. рис. 2.2). На цій сторінці користувач зможе:

- переглянути коротку інформацію про всі продукти, які є у системі;
- користувач буде бачити поточну середню ціну продукта по всім магазинам, перелік магазинів з яких збирається інформація, а також дату останнього оновлення продукта;
- перейти на сторінку додавання нового продукту, натиснувши на відповідну кнопку зверху таблиці;
- знизу таблиці з продуктами будуть розташовані кнопки пагінації, які допоможуть користувачу працювати з великою кількістю продуктів, переходячи від одного списку до іншого;
- натискаючи на назву, або унікальний ідентифікатор товару (ID) користувач буде переходити на детальну сторінку огляду продукта.

Рисунок 2.2 – Сторінка списку продуктів

Наступним кроком в проєктуванні інтерфейсу платформи буде мокап для сторінки «Огляд продукту» (див. рис. 2.3) – однією з основних сторінок системи, бо саме на цій сторінці користувач буде мати можливість аналізувати результати скрапінгу по конкретному продукту, який його цікавить. Перелік основних функцій цієї сторінки:

– редагувати інформацію про товар – інпут, з можливістю змінити назву товару;
– дивитися динаміку зміни цін на продукт по різним магазинам за допомогою лінійних графіків;
– переходити на окрему сторінку для додавання або редагування URL продукту у різних магазинах;
– дивитися інформацію (у вигляді таблиці) про всі результати скрапінгу від різних магазинів з можливістю дивитися історичні дані, використовуючи кнопки пагінації;
– основною колонкою у таблиці буде зміна ціни продукту відносно попередньої сесії скрапінгу.

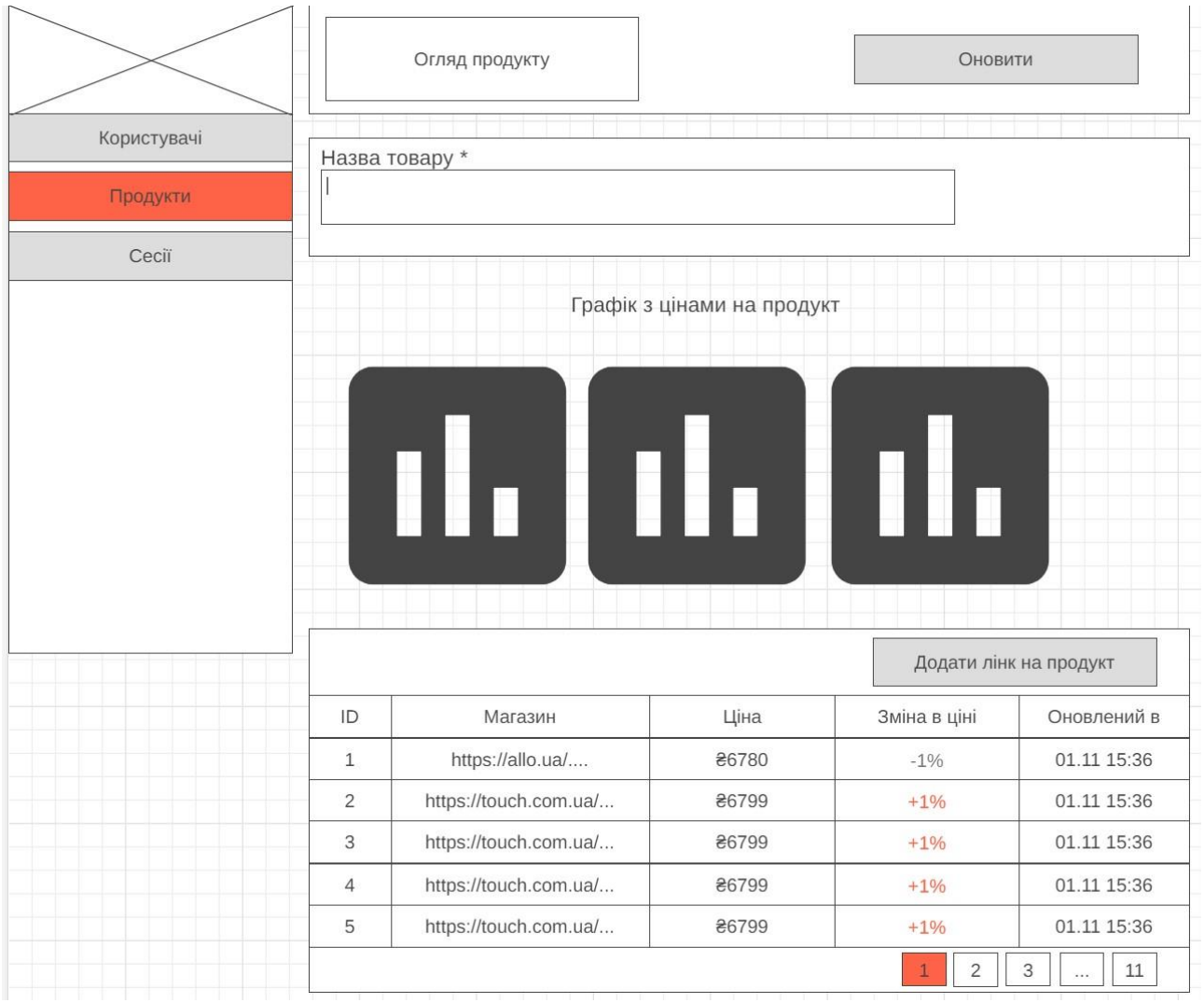

Рисунок 2.3 – Сторінка огляду продукту

Мокапи для інших сторінок окремо створювати не будемо, по причині того, що їх візуальний вигляд майже повністю буде таким самим, як на сторінках Список продуктів і Огляд продукту. Достатньо описати їх назву та функціональність:

- сторінка створення продукту;
- сторінка редагування/створення URL на продукт – додатково на цій сторінці користувач повинен мати можливість дивитися дані історичних сесій скрапінгу у вигляді таблиці;
- сторінка перегляду сесій скрапінгу;
- сторінка створення сесії скрапінгу.

2.4 Діаграма сутностей

Діаграми сутностей-зв’язків (Entity-Relationship Diagrams, ERD) є фундаментальним інструментом у теорії баз даних і системному аналізі. Вони використовуються для графічного представлення структури бази даних, демонструючи як сутності (тобто об’єкти або концепції, які мають зберігатися в базі даних) пов’язані між собою [9].

На діаграмі сутностей сутності зазвичай зображуються у вигляді прямокутників, а відносини між ними – у вигляді ліній, що з’єднують ці прямокутники. Кожна сутність має свої атрибути, які представляють характеристики або властивості цієї сутності. Атрибути часто зображуються у вигляді овалів, які з’єднуються з прямокутниками сутностей.

Основна ціль ERD – надати чітке та структуроване представлення даних, що спрощує проєктування баз даних. Завдяки візуальній природі, ERD є зручним інструментом для спілкування між розробниками, аналітиками, а також не-технічними сторонами, такими як менеджери проєктів або клієнти.

Так як вебзастосунок буде розроблятися за допомогою фреймворку Laravel, діаграми будуть показані у двох варіантах:

– у вигляді таблиць у базі даних;
– у вигляді моделей Laravel.

2.4.1 Таблиці бази даних

Для вебзастосунку основною метою якого є аналіз зберігаємих даних важливим аспектом є створення ефективної та структурованої бази даних. База даних служить основою для зберігання, організації та обробки інформації, необхідної для функціонування застосунку. В цьому контексті, наступні таблиці (див. рис. 2.4) є ключовими компонентами системи, які допомагають забезпечити зберігання та відслідковування різноманітних даних, від основної інформації про продукти до деталей сесій скрапінгу.

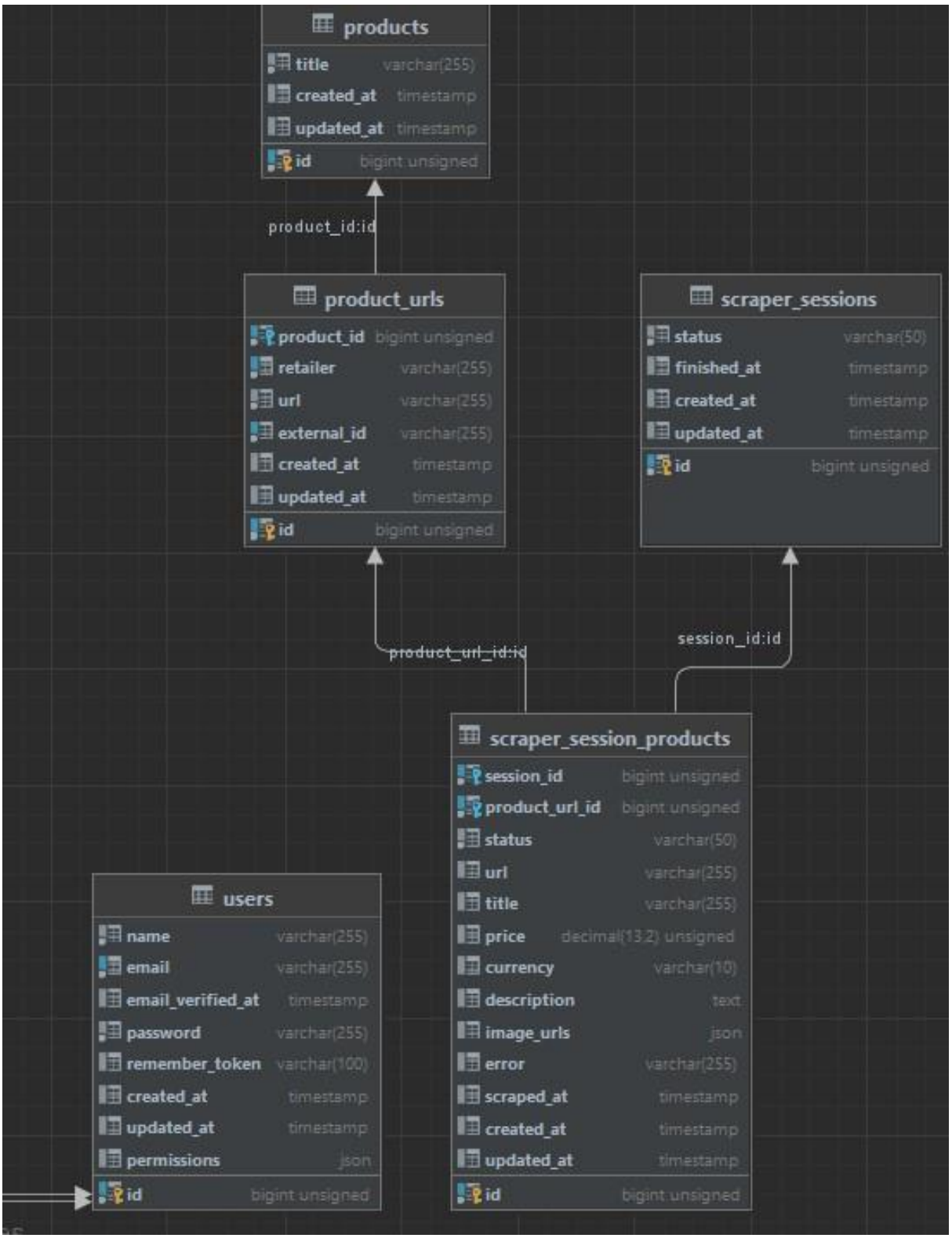

Рисунок 2.4 – Діаграма таблиць бази даних

Розберемо кожну з таблиць більш детальніше. В усіх таблицях, за замовчуванням, є поля для відстеження часу створення та останнього оновлення запису в базі даних.

Таблиця "products" призначена для зберігання інформації про продукти. Кожен продукт має унікальний ідентифікатор і назву. Ця таблиця є основою для каталогу продуктів у системі.

В таблиці "product_urls" зберігаються посилання на продукти, доступні в різних магазинах. Кожен запис містить унікальний ідентифікатор, ідентифікатор продукту, що забезпечує зв’язок з таблицею "products", назву роздрібного продавця, URL-адресу продукту, зовнішній унікальний ідентифікатор продукту у магазині з якого збираються дані.

Таблиця "scraper_sessions" використовується для відстеження сесій скрапінгу. Вона зберігає ідентифікатор сесії, статус, дату і час завершення. Ця таблиця дозволяє відслідковувати прогрес і статус процесів скрапінгу.

В таблиці "scraper_session_products" зберігається детальна інформація, отримана під час кожної сесії скрапінгу. Для кожного запису зберігаються ідентифікатор сесії, ідентифікатор URL продукту, статус скрапінгу, URL-адреса, назва, ціна, валюта, опис, URL зображень, будь-які помилки, які можливо виникали під час скрапінгу та час, коли ця інформація була зібрана. Ця таблиця є ключовою для відстеження та аналізу даних, зібраних під час скрапінгу.

Таблиця "users" використовується для зберігання інформації про користувачів системи.

Слід зазначити, що на вказаній діаграмі (див. рис. 2.4) і детальному описі не були розібрані усі таблиці, які є у системі, наприклад, таблиці які були створені при встановленні Laravel або Laravel Orchid не були включені в розбір (окрім таблиці “users”), по причині того, що вони не стосуються бізнес-логіки роботи застосунку.

2.4.2 Laravel моделі

Моделі в Laravel є ключовим компонентом архітектури цього вебфреймворку, який використовує паттерн MVC (Model-View-Controller) для

розробки вебдодатків. Модель у Laravel є представленням даних, які обробляються додатком, і відповідає за взаємодію з базою даних.

Важливою особливістю моделей у Laravel є використання Eloquent ORM (Object-Relational Mapping).

Eloquent ORM – це просунутий механізм у Laravel, який дозволяє розробникам працювати з об’єктами бази даних у більш інтуїтивний спосіб, використовуючи об’єктно-орієнтовані методи [10].

Замість написання “raw” SQL-коду, Eloquent дозволяє використовувати прості функції PHP для виконання запитів до бази даних, таким чином значно спрощуючи процес розробки.

Моделі в Laravel забезпечують легкий доступ до даних у базі, дозволяючи виконувати створення, читання, оновлення та видалення записів (CRUD-операції) без необхідності вручну керувати SQL-запитами.

Вони також дозволяють визначати відносини між різними таблицями бази даних, наприклад, один до одного, один до багатьох, багато до багатьох. Це полегшує управління зв’язками між даними і забезпечує більшу чистоту та організованість коду.

Крім цього, моделі Laravel можуть містити бізнес-логіку додатку, тобто вони не тільки представляють дані, але й містять правила та логіку, які визначають, як ці дані можуть бути створені, оновлені чи оброблені. Це дозволяє централізувати логіку обробки даних у одному місці, роблячи код більш модульним, читабельним та легшим для підтримки.

Використання моделей у Laravel значно спрощує процес розробки, забезпечуючи ефективність та чистоту коду, а також дозволяє розробникам зосереджуватися на бізнес-логіці додатку, замість того, щоб витрачати час на низькорівневі операції з базою даних.

На діаграмі моделей (див. рис. 2.5) можна ознайомитися з ключовими моделями Laravel, які будуть використовуватися в проєкті.

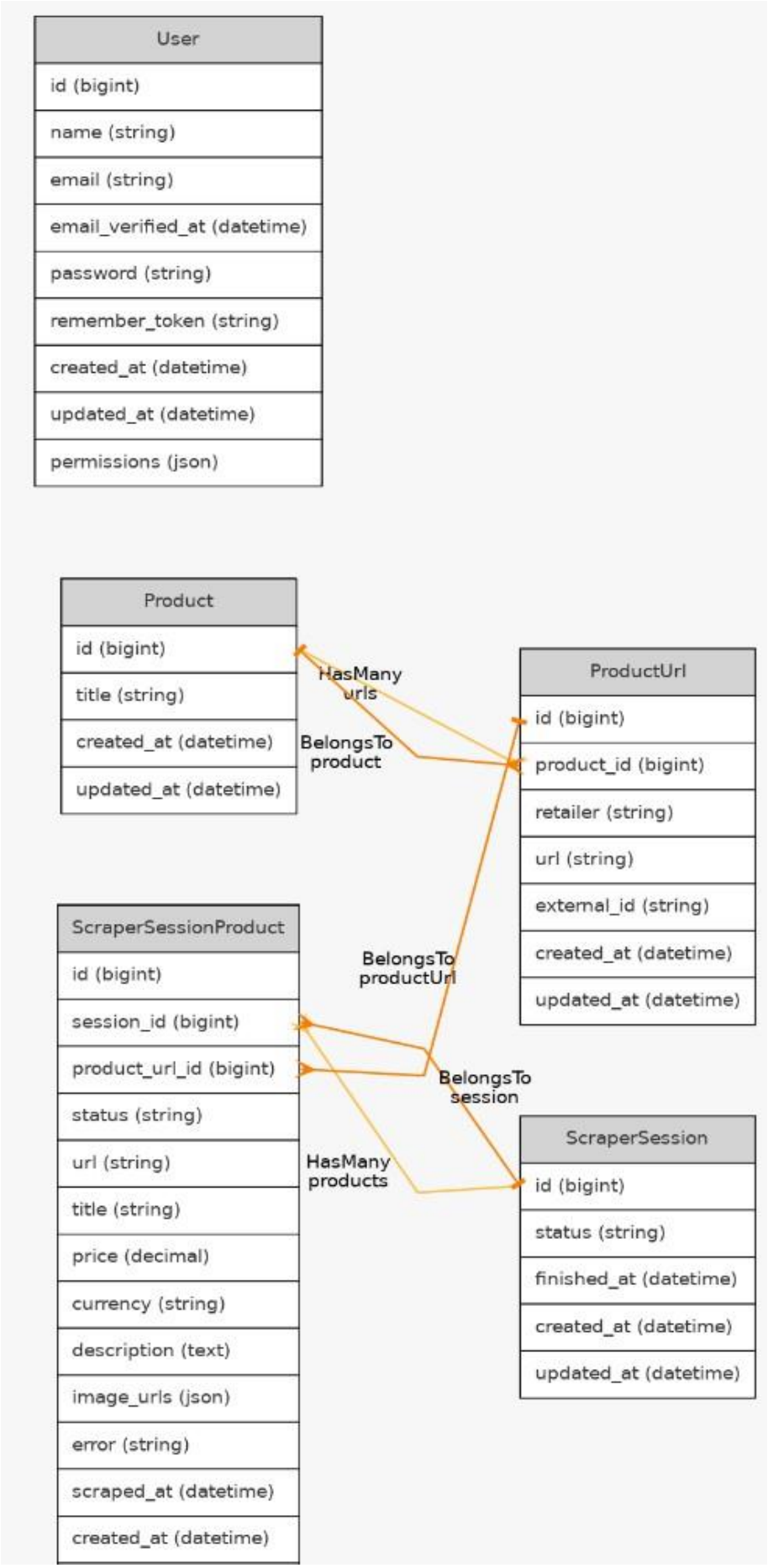

Рисунок 2.5 – Діаграма моделей Laravel

2.5 Діаграма розгортання

Діаграма розгортання є важливим інструментом в Уніфікованій Моделі Мови (UML), що використовується для візуалізації архітектури системи, зосереджуючись на її фізичній структурі. Ця діаграма включає в собі вузли, які представляють фізичні елементи системи, такі як сервери, комп'ютери та мережеві пристрої, а також артефакти, що є програмними компонентами розгорнутими на цих вузлах, наприклад бази даних, додатки, системні файли. Відносини між цими елементами показують, як компоненти взаємодіють між собою та з фізичними ресурсами [11].

На діаграмі розгортання вебзастосунку (див. рис. 2.6) можна побачити основні вузли і програмні компоненти, які на них будуть розгорнуті. Основний момент, на який слід звернути увагу – це те що, вебскрапери будуть повністю ізольовані від бази даних використовуючи RabbitMQ, і за рахунок цього їх можна буде легко масштабувати, як вертикально, так і горизонтально.

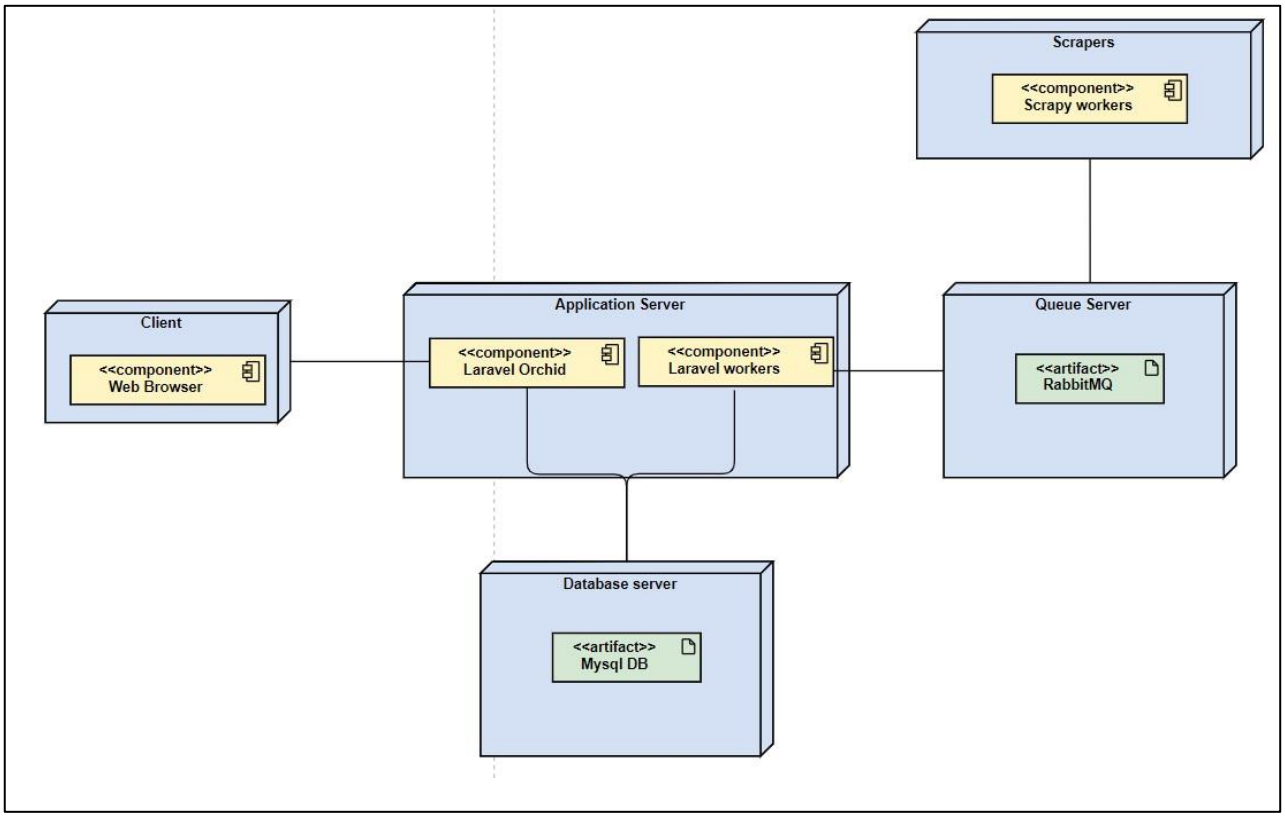

Рисунок 2.6 – Діаграма розгортання вебзастосунку

2.6 Висновки до розділу 2

У цьому розділі було проведено детальне проєктування вебзастосунку для моніторингу цін на продукти. Розгляд вимог до системи допоміг визначити ключових акторів та їх ролі у вебзастосунку, що забезпечило чітке розуміння потреб користувачів та взаємодій в системі.

Діаграма послідовності, що була розроблена, детально відображає процеси обробки та передачі даних між різними компонентами системи. Це важливо для визначення архітектури взаємодій та для розуміння робочих процесів у проєкті.

Мокапи, які були створені, надають візуальне представлення інтерфейсу. Це сприяє кращому розумінню зовнішнього вигляду та взаємодії з користувачем, допомагаючи в оптимізації користувацького досвіду.

Діаграма сутностей разом з детальним описом таблиць бази даних та моделей Laravel є ключовим елементом в плануванні структури даних. Це дозволить ефективно організувати зберігання та обробку інформації в системі, забезпечуючи надійність та швидкість доступу до даних.

Нарешті, діаграма розгортання сприяла чіткому визначенню розміщення компонентів системи в архітектурі обладнання та мережі, що є ключовим для забезпечення ефективності та надійності роботи вебзастосунку.

Загалом, процес проєктування вебзастосунку був здійснений із врахуванням всіх ключових аспектів розробки, що забезпечує міцний фундамент для наступного етапу – реалізації вебзастосунку.

3 РЕАЛІЗАЦІЯ ДОДАТКУ

3.1 Локальне оточення

Перед початком безпосереднього написання коду, критично важливим є підготовка і налаштування відповідного оточення розробки. Це забезпечує гладке впровадження та тестування різних компонентів проєкту, таких як адміністративна панель, база даних, вебскрапери і черги повідомлень.

У цьому контексті, Lando є ідеальним інструментом, який може спростити і прискорити процес налаштування розробницького оточення. Lando – це віртуалізоване оточення для розробки, що працює на основі Docker. Цей інструмент дозволяє легко і швидко налаштовувати складні середовища з усіма необхідними залежностями і службами, такими як MySQL для бази даних, Laravel Orchid для адміністративної панелі (включаючи вебсервер nginx для нього), і RabbitMQ для черги повідомлень.

Завдяки Lando, у програмістів з’являється можливість створити уніфіковане і консистентне середовище для розробки, яке дозволить уникнути проблем сумісності та забезпечить легкість перенесення вашого проєкту між різними робочими станціями [12]. Це особливо корисно в командній роботі, де всі розробники мають працювати у схожих умовах.

Користування Lando також сприяє ефективному управлінню проєктом, оскільки воно автоматизує багато рутинних задач налаштування інфраструктури, дозволяючи розробникам зосередитися на написанні коду та реалізації функціональності вебзастосунку.

Файл lando.yml є центральним елементом у конфігурації проєкту, при використанні Lando для налаштування оточення. Цей файл містить усі необхідні інструкції та конфігурації для створення та управління оточенням. Тож, після встановлення Lando на комп’ютер необхідно додати цей файл конфігурації, з сервісами, які необхідні в проєкті (див. рис. 3.1).

```
name: mt-admin
recipe: laravel
proxy:
  appserver_nginx:
    - monitoring-tool.lndo.site
  rabbitmq:
    - rabbit.monitoring-tool.lndo.site:15672
config:
  webroot: public
  php: '8.1'
  via: nginx
  database: 'mysql:8.0'
  cache: redis
services:
  database:
    portforward: 1999
  rabbitmq:
    type: compose
    services:
      image: "rabbitmq:3-management"
      hostname: "rabbit"
      environment:
        RABBITMQ_DEFAULT_USER: "user"
        RABBITMQ_DEFAULT_PASS: "pass"
      command: rabbitmq-server
      ports:
        - '1212:5672'
        - '15672'
```

Рисунок 3.1 – Файл lando.yml для розгортання локального оточення

Після виконання команди “lando start” у директорії, де знаходиться цей файл конфігурації (див. рис. 3.1) буде розгорнуто наступні сервіси:

– MySQL 8, для підключення використати порт 1999;
– Redis;
– Nginx;
– RabbitMQ та вебінтерфейс за адресою rabbit.monitoring-tool.lndo.site;
– PHP8.1-FPM сервер поєднаний з nginx, який доступен за адресою monitoring-tool.lndo.site.

Слід, зазначити, що усі домени *.lndo.site автоматично перенаправляються на локальну мережу – 127.0.0.1, але це буде працювати лише якщо на комп’ютері

є доступ до інтернету, а в іншому випадку необхідно вручну оновити файл з хостами, щоб перенаправляти усі запити для цього домену на локальну мережу.

Для вебскраперів Lando налаштовуватися не буде, по причині того, що в них немає залежностей (таких як база даних, вебсервер і т.д.), як у вебзастосунку. Тому для розробки скраперів буде використовуватися Python 3.12, який буде встановлений прямо на операційну систему комп'ютера, де ведеться розробка, або на сервер, на якому вебскрапери будуть працювати.

3.2 Розробка адміністративної панелі

Після встановлення Laravel і бібліотеки Laravel Orchid необхідно розробити сторінки для збереження і відображення даних згідно з мокапами, які були розроблені у попередньому розділі (див. розділ 2.3).

В попередньому розділі вже було сказано про те, що Laravel Orchid одразу ж надає користувачам можливість авторизуватися у адміністративній панелі (див. рис. 3.2) та додавати, або редагувати нових користувачів (див. рис. 3.3).

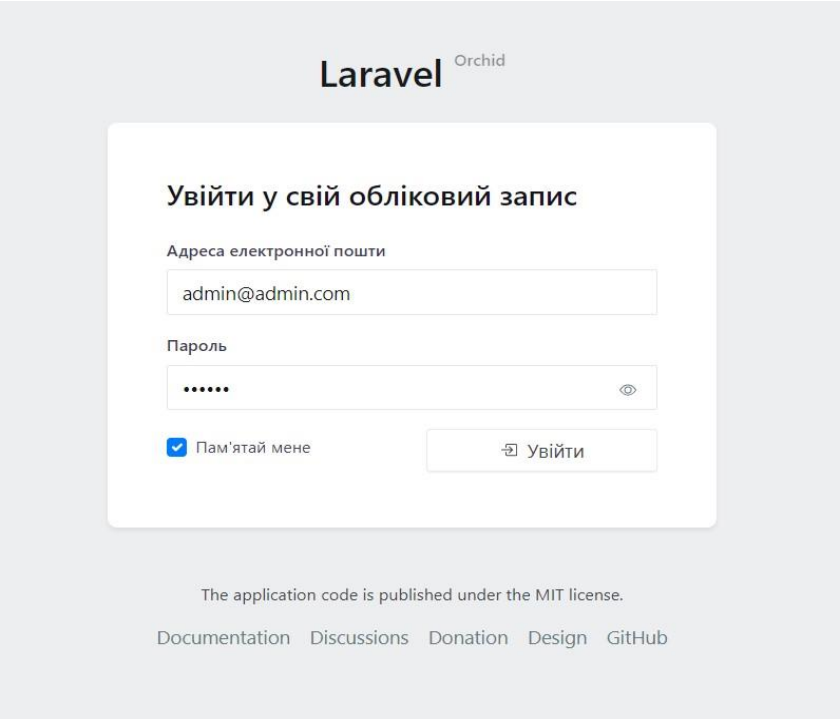

Рисунок 3.2 – Сторінка авторизації у систему

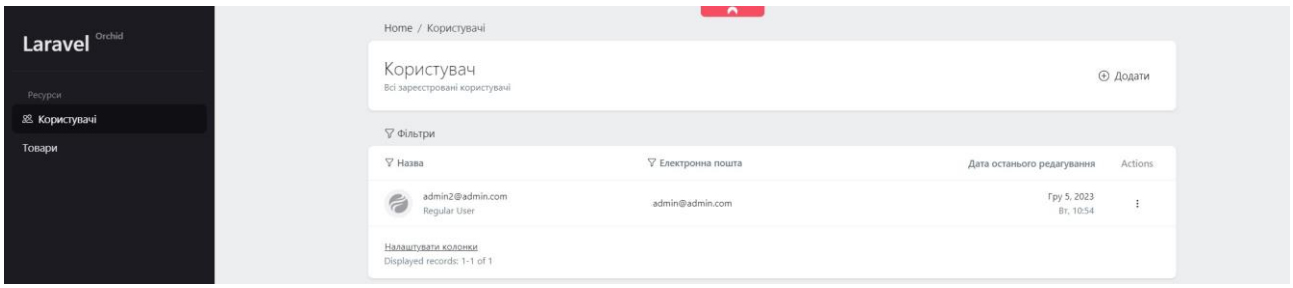

Рисунок 3.3 – Сторінка списку користувачів

Перед початком використання необхідно створити першого адміністратора з доступом в платформу, найпростішим способом зробити це – використати команду orchid:admin.

Бібліотека Laravel Orchid використовує концепцію "Screens" як ключову частину своєї архітектури для створення вебінтерфейсів. Screen у Laravel Orchid – це клас, який відповідає за логіку відображення і обробку даних на певній сторінці або розділі додатку [13]. Вони функціонують як контролери в MVC-парадигмі, але з деякими специфічними особливостями для Orchid.

Кожен Screen організований як окремий клас, що містить методи для визначення елементів інтерфейсу (наприклад, форм, таблиць), логіки обробки даних і переходів між сторінками. Ось основні аспекти Screens у Laravel Orchid:

– метод query(): цей метод використовується для запиту даних, які будуть відображені на Screen, це може включати завантаження даних з бази даних, виконання API-запитів тощо;
– метод layout(): тут визначаються компоненти інтерфейсу користувача, які будуть використані на Screen, Orchid пропонує широкий спектр вбудованих елементів інтерфейсу, таких як поля вводу, кнопки, таблиці, картки тощо;
– обробка дій: Screens можуть обробляти дії користувача, такі як натискання кнопок або відправлення форм, це робиться за допомогою методів, які можна визначити в класі Screen;
– маршрутизація: кожен Screen зазвичай асоційований з певним маршрутом у вебдодатку, що дозволяє легко керувати навігацією і доступом до різних

частин додатку;

– взаємодія з Моделями: Screens часто працюють безпосередньо з моделями Eloquent для відображення та обробки даних, що робить їх потужним інструментом для роботи з даними у Laravel;
– налаштування і розширення: Screens у Laravel Orchid можна налаштовувати та розширювати, додаючи власні компоненти або змінюючи поведінку стандартних компонентів.

Загалом, Screens у Laravel Orchid дозволяють розробникам створювати складні вебінтерфейси, використовуючи елегантний і структурований підхід.

Створимо Screen для перегляду і оновлення даних про продукт, за допомогою спеціальної команди (див. рис. 3.4).

```
lando artisan orchid:screen Product/ProductEditScreen
```

Рисунок 3.4 – Команда для створення Screen у Laravel Orchid

Наступним кроком буде імплементація усіх методів, які є у класі ProductEditScreen (див. рис. 3.5, 3.6).

```
public function commandBar(): iterable
    {
        return [
            Button::make(__('platform.create'))
                ->icon('save')
                ->method('createOrUpdate')
                ->canSee(!$this->product->exists),
            Button::make(__('platform.update'))
                ->icon('check-circle')
                ->method('createOrUpdate')
                ->canSee($this->product->exists),
        ];
    }
    public function layout(): iterable
    {
        return [
            Layout::rows([
                Input::make('product.title')
```

```
                ->title(__('platform.product_title'))
                ->required(),
        ]),
        Layout::rows([
            Button::make(__('platform.product_link_add'))
                ->style('margin-left: auto')
                ->icon('plus-lg')
                ->method('redirectToCreateProductUrlPage'),
        ])
            ->canSee($this->product->exists),
    ];
}
```

Рисунок 3.5 – Перелік action-кнопок класу ProductEditScreen

```
public function createOrUpdate(Request $request)
    {
        $request->validate([
            'product.title' => 'required|min:2|max:255',
        ]);
        $this->product
            ->fill($request->post('product'))
            ->save()
        ;
        Alert::success(__('platform.successfully_process'));

        return redirect()->route('platform.product.edit', $this->product);
    }
```

Рисунок 3.6 – Метод для збереження продукту в базу даних

Для того, щоб мати можливість перейти на сторінку редагування або створення продукту її необхідно додати до файлу routes/platform.php (див. рис. 3.7).

```
Route::screen('/product/{product?}', ProductEditScreen::class) >name('platform.product.edit');
```

Рисунок 3.7 – Роут для переходу на сторінку редагування продукту

У результаті роботи цього Screen ми отримуємо сторінку на якій користувач може:

– створювати/редагувати інформацію про продукт;
– переходити на сторінку для додавання URL магазинів к продукту.

Аналогічним способом необхідно створити і імплементувати наступні сторінки:

– ProductListScreen – сторінка для перегляду усіх продуктів у системі;
– ProductUrlEditScreen – сторінка для перегляду інформації по URL продукту;
– SessionListScreen – сторінка для перегляду усіх сесій в системі;
– SessionCreateScreen – сторінка для створення сесії скрапінгу.

Далі, після реалізації необхідних сторінок, на сторінку огляду продукту необхідно додати можливість передивлятися інформацію по зібраним даним усіх сесій, а також графік зміни ціни продукту у різних магазинах.

Для перегляду даних по останнім сесіям скрапінгу стосовно обраного продукту, спершу треба оновити метод query і додати туди вибірку з бази даних, після цього оновити метод layout і додати туди інформацію про ці дані у вигляді таблиці (див. рис. 3.8).

```
    Layout::table('data', [
        TD::make('id', __('platform.id')),
        TD::make('url', __('platform.shop'))
            ->render(
                fn (ScraperSessionProduct $sessionProduct) =>
HtmlBuilder::buildLink($sessionProduct->url, $sessionProduct->url)
            ),
        TD::make('price', __('platform.price'))
            ->render(fn (ScraperSessionProduct $sessionProduct) => sprintf('₴%s',
$sessionProduct->price)),
        TD::make('price_change', __('platform.price_change'))
            ->render(
                fn (ScraperSessionProduct $sessionProduct) =>
                    sprintf('%d%%', $sessionProduct->getPriceChange())
            ),
        TD::make('scraped_at', __('platform.updated_at'))
```

Рисунок 3.8 – Таблиця зібраних даних від вебскраперу

Для розробки графіку зміни цін буде використано готовий компонент від бібліотеки Laravel Orchid (див. рис. 3.9).

```
use Orchid\Screen\Layouts\Chart;
class ChartLine extends Chart
{
    protected $height = 300;
    protected $lineOptions = [
        'spline'     => 1,
        'regionFill' => 1,
        'hideDots'   => 0,
        'hideLine'   => 0,
        'heatline'   => 0,
        'dotSize'    => 3,
    ];
}
```

Рисунок 3.9 – Клас для відображення лінійних графіків

Оновимо наш метод layout, додаючи туди виклик створенного графіку (див. рис. 3.10). В якості першого параметра туди передається ключ від масиву, який повертає метод query. По цьому ключу Laravel Orchid буде очікувати дані у форматі масиву масивів з ключами: name, values, labels. У нашому випадку ключ буде називатися productPriceChanges. Ці дані повинні будуть запитуватися з бази даних, за допомогою моделей Laravel.

```
ChartLine::make('productPriceChanges', 'Графік зміни ціни на продукт у різних магазинах')
```

Рисунок 3.10 – Виклик компоненту для відображення графіків

Для тестування роботи доданих змін нам необхідно додати якісь дані у базу даних, в таблицю scraper_session_products, бо інакше графік і таблиця будуть пустими. Найпростіший спосіб зробити це – додати їх мануально, так як на даний момент вебскрапери ще не працюють.

Після того, як записи будуть додані у базу даних вже можна подивитися приклад, як буде виглядати ця сторінка (див. рис. 3.11).

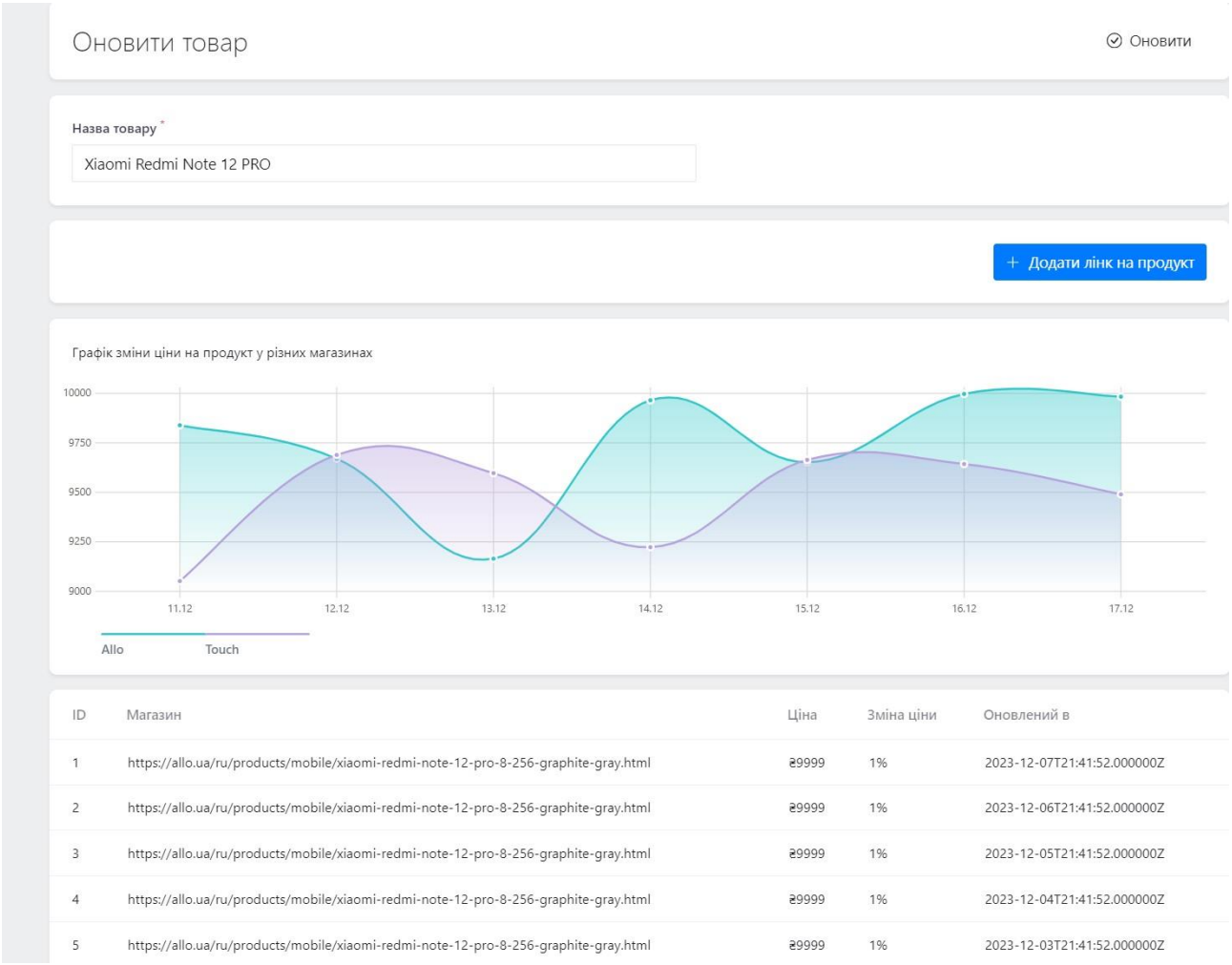

Рисунок 3.11 – Приклад сторінки Огляд продукту

3.3 Розробка Laravel workers

Після реалізації можливості додавання та редагування даних в нашій базі даних за допомогою адміністративної панелі необхідно розробити ряд сервісів, які будуть викликатися у бекграунді (тобто не напряму, при HTTP виклику). Ці сервіси будуть відповідальні за передачу посилань для скрапінгу, і збереження результатів скрапінгу у базу даних. Для цього будуть використані наступні інструменти:

- Laravel scheduler;
- Laravel jobs;
- Horizon;

– Pm2.

Laravel Scheduler є інструментом для планування періодичних завдань у Laravel-додатках, дозволяючи легко визначати і управляти завданнями [14]. Він буде використовуватися для періодичного запуску завдань Laravel, таких як запуск сесій, які були створені, та навпаки завершення сесій, для яких всі дані вже були зібрані.

Laravel Jobs дозволяє розробникам відкладати обробку тривалих або важких завдань, виконуючи їх у фоновому режимі, за допомогою черг [15]. В нашому випадку, логіка по перевірці статусу і додавання посилань на скрапінг в RabbitMQ чергу буде відбуватися саме використовуючи Laravel Jobs.

Horizon є інструментом для моніторингу і управління Laravel Queue системою, надаючи візуальний інтерфейс для відстеження статусу завдань, проведення звітності та діагностики проблем [16].

PM2 – це популярний менеджер процесів, розроблений на Node.js, який дозволяє легко управляти і масштабувати будь-які процеси, забезпечуючи такі функції як автоматичний перезапуск при виході з ладу, балансування навантаження та логування [17].

Почнемо з налаштування інструментів, які необхідні для запуску і моніторингу завдань у нашому додатку. Для встановлення пакету horizon необхідно послідовно виконати 2 команди, які наведені на рисунку 3.12.

```
lando composer require laravel/horizon
lando artisan horizon:install
```

Рисунок 3.12 – Встановлення пакету Laravel horizon

Для встановлення і налаштування PM2 слід оновити файл .lando.yml (див. рис. 3.13), а також створити pm2.config.cjs файл, в якому будуть описані інструкції – які саме процеси необхідно запускати і контролювати, у нашому випадку такими будуть: scheduler для періодичного запуску завдань, і horizon для запуску серверу воркерів.

```
…
  appserver:
    build_as_root:
      - curl -sL https://deb.nodesource.com/setup_14.x | bash -
      - apt-get install -y nodejs
      - npm install -g pm2
…
tooling:
  node:
    service: appserver
  npm:
    service: appserver
  pm2:
    service: appserver
events:
  post-start:
    - appserver:
        - pm2 start pm2.config.cjs
```

Рисунок 3.13 – Оновлений файл .lando.yml для налаштування PM2

Після перезапуску оточення, в контейнері вже буде встановлений PM2, а ми як, користувачі будемо мати можливість викликати pm2 за допомогою lando, використовуючи наступний синтаксис виклику: lando pm2 … , таким же чином будуть доступні node і npm. В додаток до цього, ми можемо перейти на сторінку monitoring-tool.lndo.site/horizon для відслідковування статусу наших воркерів (див. рис. 3.14).

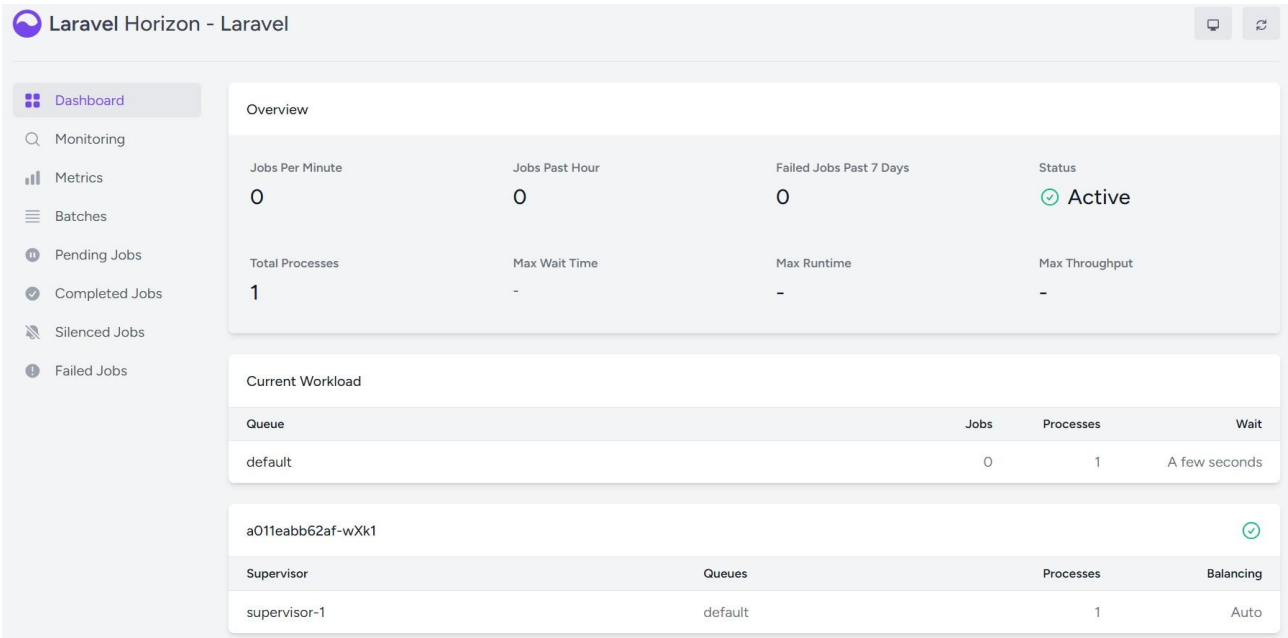

Рисунок 3.14 – Сторінка моніторингу черг Laravel (вебінтерфейс Horizon)

Далі, необхідно реалізувати 3 сервіси:

– SessionStatusCheckerJob – його завдання, перевіряти всі сесії, які не знаходяться у статусі finished, і робити певні дії, в залежності від статусу, для created сесії необхідно створити записи в таблиці scraper_session_products і перевести сесію в статус in_progress, а для in_progress сесій необхідно перевірити всі записи з таблиці з результатами сесії скрапінгу, і у випадку, коли всі вони finished необхідно перевести сесію у finished статус;
– SessionProductPusherJob – його завдання, додавати посилання для скрапінгу у чергу RabbitMQ (див. рис. 3.15), і переводити ScraperSessionProduct в статус in_progress;
– SessionProductSaver – його завдання, зберігати результати вебскрапінгу з черги RabbitMQ (див. рис. 3.16 і рис. 3.17) у базу даних, слід зазначити, що цей сервіс необхідно буде додати в конфігурацію pm2.config.cjs, так як він не буде викликатися за допомогою Laravel scheduler, а буде запускатися, як окремий демон процес для того, щоб зберігати дані від скрапінгу моментально, при отриманні.


```
{
  "id": 25,
  "url": "https://allo.ua/ua/products/mobile/xiaomi-redmi-note-12-pro-8-256-graphite-gray.html"
}
```


Рисунок 3.15 – Приклад повідомлення для вебскраперу


```
{
    "id": 25,
    "url": "https://allo.ua/ru/products/mobile/xiaomi-redmi-note-12-pro-8-256-graphite-gray.html",
    "title": "Xiaomi Redmi Note 12 Pro 8/256 Graphite Gray",
    "price": "9999.00",
    "currency": "UAH",
    "description": "<div>...</div>",
    "image_urls": ["https://i.allo.ua/media/catalog/...jpg"],
    "status": "success",
}
```


Рисунок 3.16 – Приклад повідомлення про успішний результат скрапінгу

```
{
   "id": 25,
   "status": "failed",
   "error": "page was not found (404)"
}
```

Рисунок 3.17 – Приклад повідомлення про неуспішний результат скрапінгу

3.4 Розробка вебскраперів

Якщо розглянути наш вебскрапер на високому рівні, він повинен вміти робити три речі:

– слухати чергу повідомлень з посиланнями і при отриманні нового повідомлення запускати процесс скрапінгу;
– скрапити отримане посилання і парсити HTML відповідь від магазину;
– зберігати отриманий результат у чергу RabbitMQ.

Реалізуємо кожен з пунктів, на прикладі магазину Allo.

Після генерації павуку (див. рис. 3.18), необхідно реалізувати метод, задача якого отримувати повідомлення з черги RabbitMQ і додавати запит на обробку посилання (див. рис. 3.19).

```
scrapy genspider allo allo.ua
```

Рисунок 3.18 – Команда для створення павука Scrapy

Scrapy Items є ключовим компонентом фреймворку Scrapy. Ці структури даних дозволяють зберігати інформацію, зібрану під час здійснення скрапінгу вебсайтів, у впорядкованому та структурованому форматі [18]. Вони дійсно схожі на словники в Python, але пропонують додаткові переваги, такі як визначення специфічних полів і типів даних, що можуть бути заздалегідь задані.

Це дозволяє розробникам більш ефективно управляти зібраними даними, оскільки поля Scrapy Items можуть бути задокументовані і типізовані,

забезпечуючи більшу чистоту і консистентність даних (див. рис. 3.20). Крім того, використання Scrapy Items сприяє легкій інтеграції із засобами для зберігання даних, такими як бази даних, оскільки вони забезпечують чітке визначення того, як дані мають бути структуровані.

```
def next_request(self):
        while True:
            try:
                stats = self.channel.queue_declare(queue=self.queues.get('allo_urls'), durable=True)
                if stats.method.message_count > 0:
                    meta, header_frame, data = self.channel.basic_get(queue=self.queues.get('allo_urls'))
                    message = json.loads(data)
                    url = message.get('url')
                    request = scrapy.Request(url=url, dont_filter=True, callback=self.parse,
                                             errback=self.errback,
                                             meta={
                                                'message': message,
                                                'delivery_tag': meta.delivery_tag,
                                             })
                    return request
            except ChannelClosed as ch_exc:
                self.rabbitmq_connect()
```

Рисунок 3.19 – Реалізація методу отримання повідомлення з черги RabbitMQ

```
class ProductItem(scrapy.Item):
    id = scrapy.Field()
    url = scrapy.Field()
    title = scrapy.Field()
    price = scrapy.Field()
    currency = scrapy.Field()
    description = scrapy.Field()
    image_urls = scrapy.Field()
```

Рисунок 3.20 – Scrapy Item для зберігання інформації про продукт

В цілому, Scrapy Items є важливим інструментом для розробників, які займаються вебскрапінгом, дозволяючи їм більш ефективно збирати, обробляти та управляти даними. Тому саме їх ми будемо використовувати для зберігання результатів роботи скраперу.

Для парсінгу сторінки магазину спочатку необхідно проаналізувати структуру HTML при запиті, обрати які поля нам необхідні (див. рис. 3.21), і реалізувати метод parse (див. рис. 3.22).

Метод parse в Scrapy є центральною частиною павука (spider) — основного компонента у Scrapy, відповідального за обробку відповідей, отриманих після запитів до вебсайтів. Цей метод автоматично викликається Scrapy, коли отримує відповідь від вебсайту, на який був здійснений запит.

Функція parse приймає один параметр: об'єкт відповіді (response), який містить всю інформацію, отриману від запиту, включаючи вміст сторінки, заголовки HTTP, URL-адресу та інші.

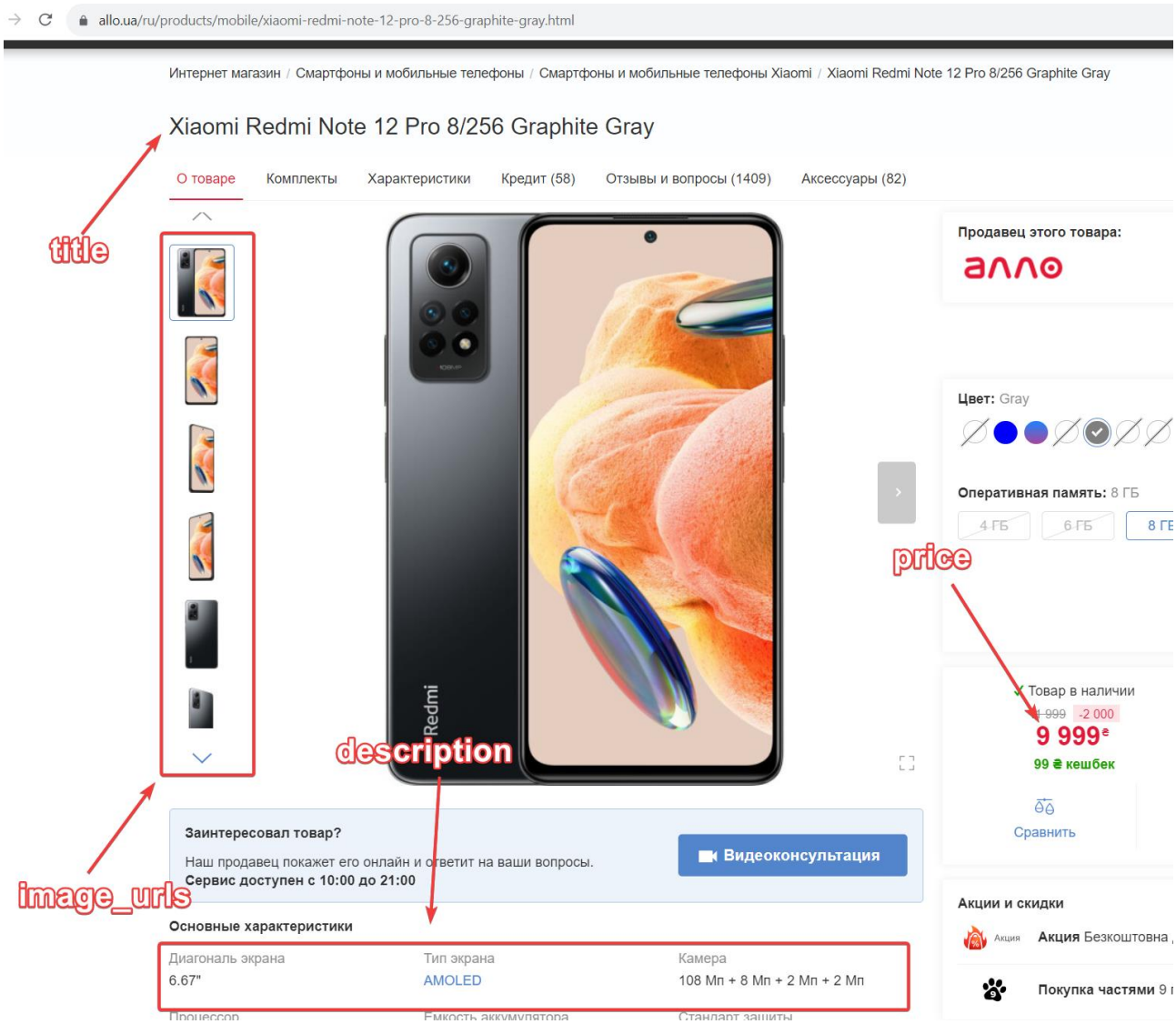

Рисунок 3.21 – Аналіз сторінки продукту магазину Allo

```
def parse(self, response):
        item = ProductItem()
        message = response.meta.message
        item['id'] = message['id']
        item['url'] = response.url
        item['title'] = response.xpath('//h1[@itemprop="name"]/text()').extract_first()
        item['price'] = response.xpath('//div[@id="p-trade-price"]/text()').extract_first()
        item['currency'] = 'UAH'
        item['description'] = response.xpath('//ul[@class="product-details__list"]').get()
        item['image_urls'] = response.xpath('//picture[@class="main-gallery__link"]/source/@srcset').getall()
        yield item
```

Рисунок 3.22 – Імплементація методу parse

І останнім кроком в реалізації скраперу буде збереження даних скрапінгу (див. рис. 3.23) у чергу RabbitMQ (див. рис. 3.24), для цього ми скористаємося функціоналом Scrapy Pipelines.

```
class ProductsPipeline(object):
    def process_item(self, item, spider):
        if isinstance(item, ProductItem):
            queue = "products_save"

            publish_item_to_rabbitmq(item, spider, queue)
        return item
```

Рисунок 3.23 – Scrapy pipeline для збереження продукту в чергу

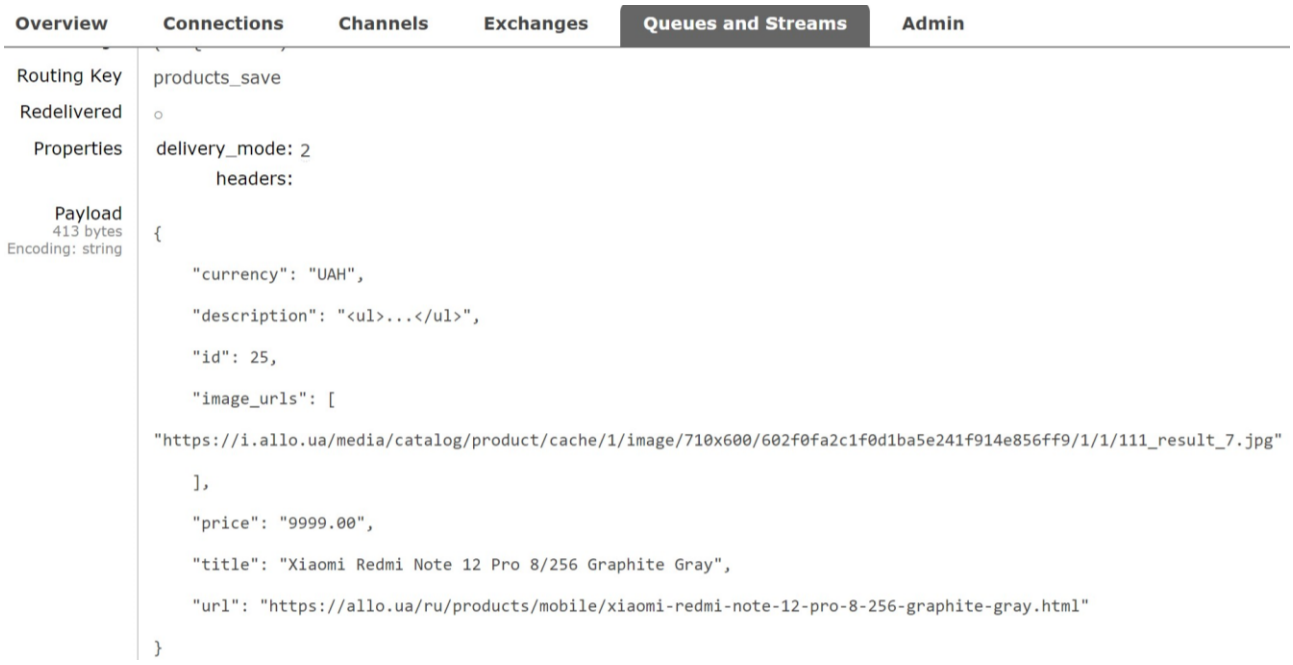

Рисунок 3.24 – Приклад повідомлення з даними про продукт в RabbitMQ

Scrapy Pipelines є ключовим компонентом фреймворку Scrapy, призначеним для обробки і зберігання зібраних даних. Коли Scrapy вилучає дані з вебсторінок, ці дані передаються через різні пайплайни перед тим, як їх можна буде зберегти або подальше використовувати [19].

Парсер для магазину Touch буде реалізований таким же чином, а саме:

– створення нового павука;
– читання даних з черги з посиланнями для цього магазину;
– збереження зібраних даних у чергу products_save.

3.5 Тестування і розгортання додатку

Для перевірки роботи застосунку у комплексі, а саме роботи адміністративної панелі і вебскраперів у зв’язці з брокером повідомлень RabbitMQ необхідно розгорнути проєкт в публічний доступ. В якості середовища для розгортання була обрана платформа AWS.

В AWS є усі необхідні інструменти, для розгортання проєкту (див. рис. 3.25) згідно з діаграмою розгортання, яка була розроблена на етапі проєктування додатку (див. підрозділ 2.5).

AWS (Amazon Web Services) є найбільшою і широко використовуваною хмарною платформою, яка надає широкий спектр хмарних сервісів, включаючи обчислювальні потужності, зберігання даних, бази даних, аналітику, машинне навчання, безпеку та багато іншого.

Після розгортання проєкту слід додати декілька продуктів у систему, запустити сесію скрапінгу і перевірити, коректність зібраних даних.

Дані, відповідно, були перевірені мануальним способом. Всі ціни, які були зібрані скраперами і відображені в адміністративній панелі (див. рис. 3.26) збіглися з даними зі сторінок магазинів.

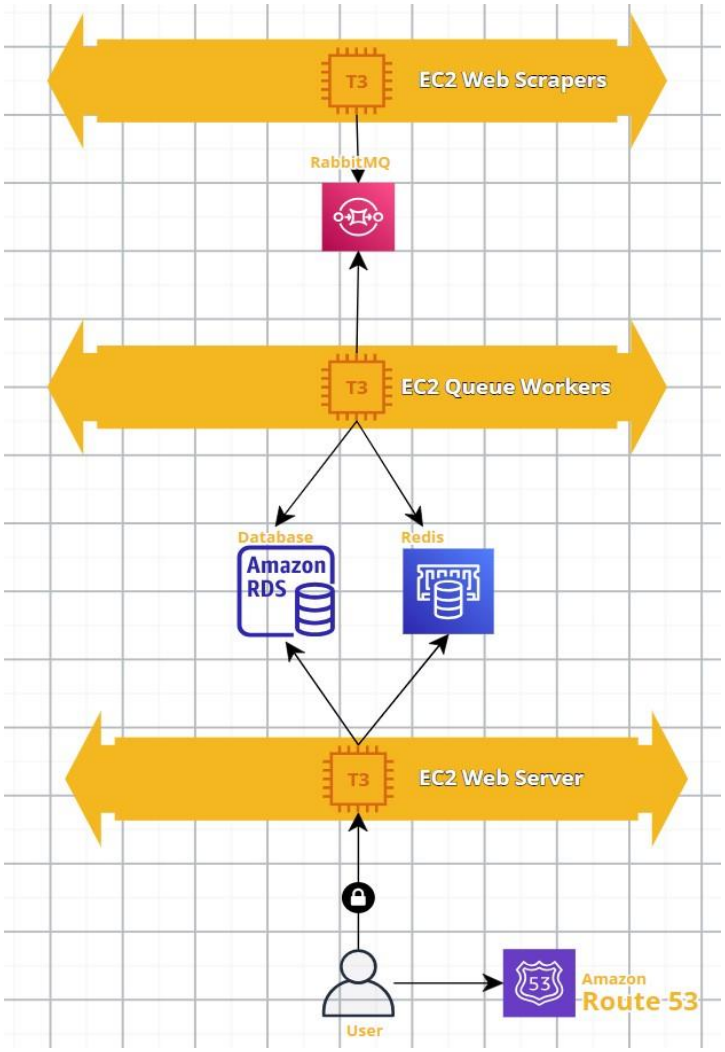

Рисунок 3.25 – Діаграма розгортання проєкту на AWS

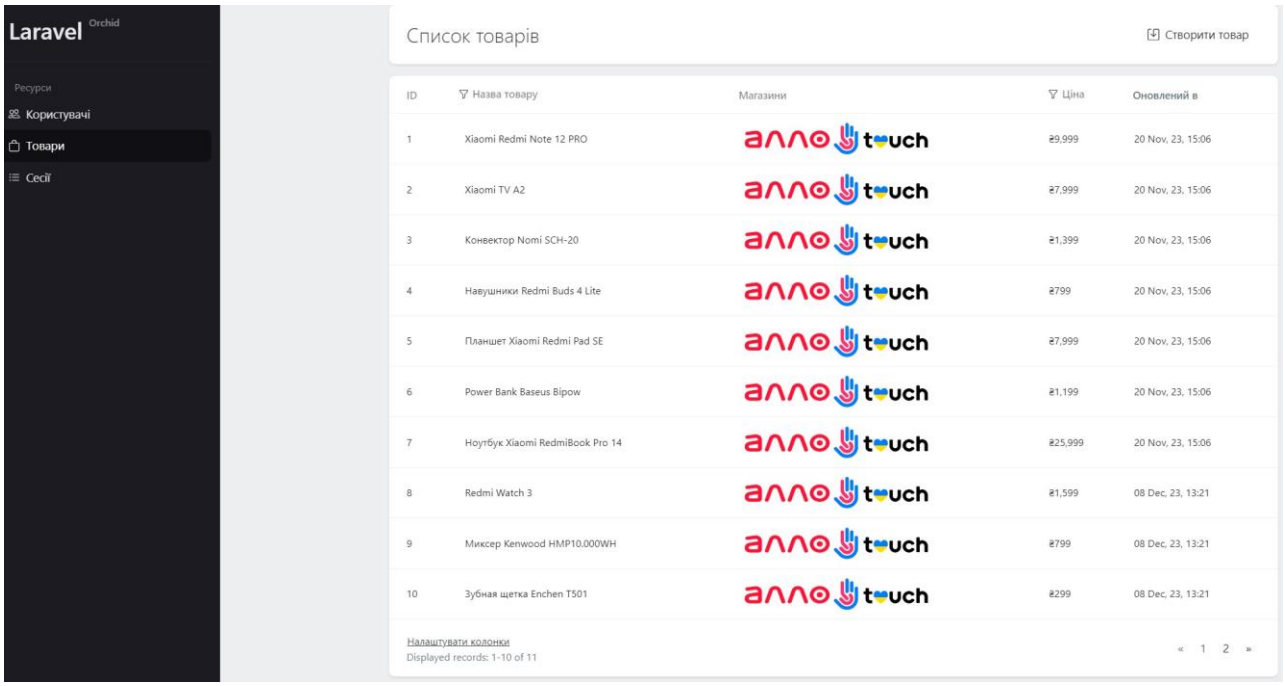

Рисунок 3.26 – Результати запуску сесії на інфраструктурі AWS

3.6 Висновки до розділу 3

У цьому розділі було детально описано реалізацію вебзастосунку для моніторингу цін на продукти. На етапі підготовки локального оточення було використано Lando, що забезпечило зручне та ефективне середовище для розробки. Під час розробки адміністративної панелі було створено інтуїтивно зрозумілий інтерфейс, що дозволяє легко управляти даними і моніторинговими процесами. Реалізація Laravel workers дала можливість ефективно обробляти задачі у фоновому режимі, особливо для збору даних з вебскраперів. Розробка вебскраперів була ключовою для збирання актуальної інформації про ціни з різних онлайн-магазинів. В кінцевому розділі, присвяченому тестуванню і розгортанню додатку, було виконана перевірка і розгортання всіх компонентів системи на платформі AWS, що забезпечило її надійність і готовність до використання. Кожен етап цього процесу сприяв створенню ефективного і функціонального рішення для моніторингу цін.

ВИСНОВКИ

Було проведено детальний огляд існуючих технологій та методів моніторингу цін у сфері електронної комерції. Під час аналізу було виявлено, що багато існуючих рішень не враховують необхідність надійного та всебічного моніторингу цін, включаючи історичну динаміку змін. Відповідно, в якості альтернативи була поставлена задача розробки власного вебзастосунку, що здатен задовольнити ці потреби.

У роботі було проведено аналіз вимог до системи, що включає ідентифікацію основних акторів, оцінку потреб користувачів, визначення ключових функціональних можливостей та технічних інструментів необхідних для успішної реалізації проєкту, а саме: Laravel Orchid для розробки вебінтерфейсу і Scrapy для розробки вебскраперів. Це дозволило розробити детальні діаграми послідовностей, розгортання та мокапи вебінтерфейсу, а також сформулювати структуру бази даних і моделей Laravel, які стануть основою для вебзастосунку.

Важливою частиною роботи стала розробка вебскраперів на основі Scrapy, які забезпечують автоматизований збір цінової інформації з вебсайтів роздрібних магазинів, таких як Allo та Touch.

Підсумковий етап проєкту включав тестування та розгортання додатку, що дозволило переконатися в його надійності та ефективності. В результаті було створено функціональний та користувач-орієнтований вебзастосунок, який надає повноцінні інструменти для моніторингу та аналізу цін на продукти, включаючи доступ до історичних даних змін цін. Цей проєкт вирішує проблему відсутності ефективних інструментів для моніторингу цін, забезпечуючи користувачам потужний засіб для інформованого вибору та планування покупок.

ПЕРЕЛІК ПОСИЛАНЬ

1. Puppeteer documentation – Getting Started. *Pptr.dev*. URL: https://pptr.dev/ (дата звернення: 06.07.2023).
2. Scrapy at a glance (general overview of the tool). *Scrapy.org*. URL: https://docs.scrapy.org/en/latest/intro/overview.html (дата звернення: 08.07.2023).
3. Architecture overview. *Scrapy.org*. URL: https://docs.scrapy.org/en/latest/topics/architecture.html (дата звернення: 14.07.2023).
4. Introduction to Laravel Orchid. *Orchid*. URL: https://orchid.software/en/docs/ (дата звернення: 20.07.2023).
5. OSS RabbitMQ Features. *RabbitMQ*. URL: https://www.rabbitmq.com/#features (дата звернення: 22.07.2023).
6. The Main Features of MySQL. *MySQL.com*. URL: https://dev.mysql.com/doc/refman/8.0/en/features.html (дата звернення 25.07.2023).
7. Al-Fedaghi S. UML Sequence Diagram: An Alternative Model. *Kuwait University*. URL: https://arxiv.org/abs/2105.15152 (дата звернення: 28.08.2023).
8. Drawing. *Wireframe.cc*. URL: https://wireframe.cc/docs/#-drawing (дата звернення: 01.09.2023).
9. Al-Fedaghi S. Conceptual Data Modeling: Entity-Relationship Models as Thinging Machines. *Kuwait University*. URL: https://arxiv.org/abs/2109.14717 (дата звернення: 05.09.2023).
10. Eloquent: Getting Started. *Laravel*. URL: https://laravel.com/docs/10.x/eloquent (дата звернення: 09.09.2023).
11. Kachmar Y. UML Deployment Diagram in modern word. *Medium*. URL: https://medium.com/@kachmarani/uml-deployment-diagram-in-modern-word-

caee0a2ecaa3 (дата звернення 14.09.2023).

12. How does it work? *Lando Documentation*. URL: https://docs.lando.dev/getting-started/what-it-do.html (дата звернення: 01.11.2023).
13. Screens. *Orchid*. URL: https://orchid.software/en/docs/screens/ (дата звернення: 03.11.2023).
14. Task Scheduling. *Laravel*. URL: https://laravel.com/docs/10.x/scheduling (дата звернення: 05.11.2023).
15. Queues. *Laravel*. URL: https://laravel.com/docs/10.x/queues (дата звернення: 05.11.2023).
16. Laravel Horizon. *Laravel*. URL: https://laravel.com/docs/10.x/horizon (дата звернення: 06.11.2023).
17. PM2 Process Management Quick Start. *PM2*. URL: https://pm2.keymetrics.io/docs/usage/quick-start/ (дата звернення: 07.11.2023).
18. Items. *Scrapy.org*. URL: https://docs.scrapy.org/en/latest/topics/items.html (дата звернення: 09.11.2023).
19. Item Pipeline. *Scrapy.org*. URL: https://docs.scrapy.org/en/latest/topics/item-pipeline.html (дата звернення: 10.11.2023).

ДОДАТОК А

Файл .lando.yml для конфігурації локального оточення

```
name: mt-admin
recipe: laravel
proxy:
  appserver_nginx:
    - monitoring-tool.lndo.site
  rabbitmq:
    - rabbit.monitoring-tool.lndo.site:15672
config:
  webroot: public
  php: '8.1'
  via: nginx
  database: 'mysql:8.0'
  cache: redis
services:
  database:
    portforward: 1999
  appserver:
    build_as_root:
      - curl -sL https://deb.nodesource.com/setup_14.x | bash -
      - apt-get install -y nodejs
      - npm install -g pm2
  rabbitmq:
    type: compose
    services:
      image: "rabbitmq:3-management"
      hostname: "rabbit"
```

```
    environment:
      RABBITMQ_DEFAULT_USER: "user"
      RABBITMQ_DEFAULT_PASS: "pass"
    command: rabbitmq-server
    ports:
      - '1212:5672'
      - '15672'
tooling:
  node:
    service: appserver
  npm:
    service: appserver
  pm2:
    service: appserver
events:
  post-start:
    - appserver:
        - pm2 start pm2.config.cjs
```

ДОДАТОК Б

Перелік роутів платформи

```
<?php
use App\Orchid\Screens\PlatformScreen;
use App\Orchid\Screens\Product\ProductEditScreen;
use App\Orchid\Screens\Product\ProductListScreen;
use App\Orchid\Screens\Product\ProductUrlEditScreen;
use App\Orchid\Screens\Session\SessionCreateScreen;
use App\Orchid\Screens\Session\SessionListScreen;
use App\Orchid\Screens\User\UserEditScreen;
use App\Orchid\Screens\User\UserListScreen;
use App\Orchid\Screens\User\UserProfileScreen;
use Illuminate\Support\Facades\Route;
Route::screen('/main', PlatformScreen::class)->name('platform.main');
Route::screen('profile', UserProfileScreen::class)->name('platform.profile');
Route::screen('users/{user}/edit', UserEditScreen::class)
->name('platform.systems.users.edit');
Route::screen('users/create',UserEditScreen::class)
->name('platform.systems.users.create');
Route::screen('users', UserListScreen::class)->name('platform.systems.users');
Route::screen('/products', ProductListScreen::class)->name('platform.products.list');
Route::screen('/product/{product?}',ProductEditScreen::class)
->name('platform.product.edit');
Route::screen('/product/{product}/url/{productUrl?}', ProductUrlEditScreen::class)
->name('platform.product_url.edit');
Route::screen('/sessions', SessionListScreen::class)->name('platform.sessions.list');
Route::screen('/sessions/create', SessionCreateScreen::class)
->name('platform.sessions.create');
```

ДОДАТОК В

Скрапер магазина Allo

```
import scrapy
import json
import time
import os
import sys
import traceback
from pika.exceptions import ChannelClosed
from configparser import ConfigParser
from mt_scrapers.spiders.ProductsBaseSpider import ProductsBaseSpider
from scrapy.spidermiddlewares.httperror import HttpError
from twisted.internet.error import DNSLookupError
from twisted.internet.error import TimeoutError, TCPTimedOutError
class AlloSpider(ProductsBaseSpider):
    name = 'allo'
    channel = None
    threads = None
    def __init__(self, threads=1, *args, **kwargs):
        super(AlloSpider, self).__init__(*args, **kwargs)
        self.queues = dict(self.config.items('QUEUES'))
        self.threads = int(threads)
        self.rabbitmq_connect()

    def rabbitmq_connect(self):
        self.channel = RabbitMQ.get_channel()
        self.channel.queue_declare(queue=self.queues.get('allo_urls'), durable=True)
    def next_request(self):
```

```
        while True:
            try:
                stats = self.channel.queue_declare(queue=self.queues.get('allo_urls'),
durable=True)
                if stats.method.message_count > 0:
                    meta, header_frame, data =
self.channel.basic_get(queue=self.queues.get('allo_urls'))
                    if not data:
                        continue
                    message = json.loads(data)
                    url = message.get('url')
                    request = scrapy.Request(url=url, dont_filter=True, callback=self.parse,
                                             errback=self.errback,
                                             meta={'message': message,'delivery_tag'
meta.delivery_tag})
                    return request
                else:
                    self.logger.info("QUEUE IS EMPTY. SLEEP")
                    time.sleep(10)
            except ChannelClosed as ch_exc:
                self.logger.error('Sleeping for 1 seconds...')
                self.logger.error('*' * 70)
                self.logger.error('Reconnecting...')
                self.rabbitmq_connect()
    def parse(self, response):
        self.logger.info('Crawled: ' + str(response.url))
        message = response.meta.get('message')
        item = ProductItem()
        item['id'] = message['id']
        item['url'] = response.url
```

```
        item['title'] = response.xpath('//h1[@itemprop="name"]/text()').extract_first()
        item['price'] = response.xpath('//div[@id="p-trade-price"]/text()').extract_first()
        item['currency'] = 'UAH'
        item['description'] = response.xpath('//ul[@class="product-details__list"]').get()
        item['image_urls'] = response.xpath('//picture[@class="main-
gallery__link"]/source/@srcset').getall()
        yield item
        self.channel.basic_ack(delivery_tag=response.meta.get('delivery_tag'))
        yield self.next_request()
    def errback(self, failure):
        error_repr = repr(failure)
        self.logger.error(error_repr)
        request = failure.request
        message = request.meta.get("message")
        if failure.check(HttpError):
            code = failure.value.response.status
            response = failure.value.response
            self.logger.error('HttpError on {} with code {}'.format(response.url,
str(code)))
        elif failure.check(DNSLookupError):
            code = 665
            self.logger.error('DNSLookupError on %s', request.url)
        elif failure.check(TimeoutError, TCPTimedOutError):
            code = 765
            self.logger.error('TimeoutError on %s', request.url)
        yield ProductFailedItem(
            id=message["id"],
            error=error_repr
        )
        self.channel.basic_ack(delivery_tag=request.meta.get('delivery_tag')
```