МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА ПАРСЕРА ВЕБСТОРІНОК»**

Виконала: студентка 4 курсу, групи 6.1219-1пі
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми програмна інженерія
(назва освітньої програми)
М.І. Барнаш
(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. Матвіїшина Н.В.
(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 бакалавр

Спеціальність 121 інженерія програмного забезпечення

(шифр і назва)

Освітня програма програмна інженерія

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

_____ Лісняк А.О.
(підпис)

“ 07 ” 02 2023 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТЦІ

Барнаш Марії Іванівні

(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка парсера вебсторінок

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

затверджені наказом ЗНУ від « 26 » січня 2023 року № 102-с

2. Строк подання студентом роботи 07.06.2023 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 парсера вебсторінок.

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

6. Консультанти розділів роботи

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

7. Дата видачі завдання 07.02.2023 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.	08.02.2023	
2.	Збір вихідних даних.	13.02.2023	
3.	Обробка методичних та теоретичних	27.02.2023	
	джерел.		
4.	Розробка першого та другого розділу.	20.03.2023	
5.	Розробка третього розділу.	03.04.2023	
6.	Розробка четвертого розділу.	08.05.2023	
7.	Оформлення та нормоконтроль	01.06.2023	
	кваліфікаційної роботи.		
8.	Захист кваліфікаційної роботи.	21.06.2023	

Студент _____ М.І. Барнаш
(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 бакалавра «Розробка парсера вебсторінок»: 53 с., 17 рис., 1 табл., 14 джерел, 3 додатки.

ВЕБСТОРІНКИ, ЗБІР ДАНИХ, ПАРСЕР, СКРАПЕР.

Об’єкт дослідження – технології розробки парсерів вебсторінок.

Мета роботи: дослідити технології розробки парсерів вебсторінок та використати їх на практиці, розробивши власний парсер.

Метод дослідження – аналітичний, порівняльний, методи програмної інженерії.

Актуальність розробки парсерів вебсторінок зростає з кожним днем. Це пов’язано з тим, що інтернет-ресурси стають все більш важливими джерелами інформації. Багато компаній, дослідницьких установ і приватних осіб залежать від автоматизованого збору інформації з вебсторінок. Парсери дозволяють автоматизувати процес збору даних з вебсторінок, що значно зменшує час та зусилля, необхідні для здійснення цієї роботи вручну. Крім того, вони можуть бути використані для збору даних з вебсторінок з різних джерел, що дозволяє отримати цінну інформацію для аналізу та прийняття рішень. Одним з основних застосувань парсерів збір даних для аналізу конкурентів, моніторингу цін на товари, відстеження новин та багато іншого. Також парсери можуть бути використані для створення власних баз даних, які можуть бути використані для подальшої обробки даних. В цій роботі буде оглянуто існуючі технології та продукти для парсингу, спроєктовано, реалізовано та розгорнуто власний парсер, що буде мати змогу збирати дані з будь-який вебсторінок певного типу.

SUMMARY

Bachelor’s Qualifying Paper «Development of a Webpage Parse»: 53 pages, 17 figures, 1 table, 14 references, 3 supplements.

WEBPAGE, DATA COLLECTION, PARSER, SCRAPER.

The object of the study is technologies for developing web page parsers.

The aim of the study is to investigate technologies for developing web page parsers and apply them in practice by developing a custom parser.

The methods of research are analytical, comparative, software engineering methods.

The relevance of developing web page parsers is growing every day. This is due to the fact that internet resources are becoming increasingly important sources of information. Many companies, research institutions, and individuals rely on automated data collection from web pages. Parsers allow the automation of the process of collecting data from web pages, which significantly reduces the time and effort required to perform this task manually. In addition, they can be used to collect data from web pages from various sources, which allows for valuable information for analysis and decision making. One of the main applications of parsers is data collection for competitor analysis, price monitoring, news tracking, and much more. Parsers can also be used to create proprietary databases that can be used for further data processing. In this work existing technologies and products for parsing will be reviewed, a custom parser that will be able to collect data from any web pages of a certain type will be designed, implemented and deployed.

ЗМІСТ

ВСТУП

У сучасному світі інформаційні технології займають все більш вагому позицію в різних сферах людської діяльності, відповідно збільшується і вимога до кількості даних, які потрібно отримати для подальшого аналізу та використання. Саме тут використання парсера, що дозволяє збирати великі об’єми інформації з різних джерел в Інтернеті, стає надзвичайно корисним і важливим.

Метою кваліфікаційної роботи бакалавра є розробка парсера вебсторінок.

Задачі, які необхідно розв’язати для поставленої мети:

– проаналізувати існуючі технології парсингу вебсторінок;
– спроєктувати схему даних та розробити архітектуру додатка;
– реалізувати парсер та вебінтерфейс користувача;
– розгорнути систему;
– протестувати додаток.

Об’єкт дослідження – процес розробки парсера вебсторінок.

Предмет дослідження – сучасні технології парсингу вебсторінок.

Методи дослідження: методи програмної інженерії, системний аналіз.

Структурно робота складається з 4 розділів. У першому розділі аналізуються існуючі продукти та технології парсингу та ставиться задача. У другому розділі розробляється архітектура системи, будуються відповідні діаграми. У третьому розділі розповідається про технології, використані для реалізації та наводяться приклади коду. У четвертому розділі відбувається розгортання парсера.

1 ТЕОРЕТИЧНА ЧАСТИНА

1.1 Поняття парсингу

Розрізняють поняття парсингу та скрапінгу. Вебскрапінг – це процес вилучення даних з певної вебсторінки. Парсинг – це метод, за допомогою якого один тип даних перетворюється на інший. Таким чином, для того щоб отримати зі сторінки необхідні дані, необхідно спочатку провести скрапінг – зібрати дані у HTML, XML чи у будь якому іншому форматі, а потім парсинг – привести дані у необхідний вигляд – наприклад, у CSV чи JSON формат.

Ці поняття досить схожі за значенням і зазвичай використовуються як синоніми, тому для того, щоб не було повторень, в цій роботі вони будуть замінювати один одне.

Отже, головним об’єктом дослідження в данній роботі є вебскрапінг – збір даних будь-якими засобами, крім програмної взаємодії з API (або, очевидно, людини, що використовує браузер). Найчастіше це досягається шляхом написання автоматизованої програми, яка запитує вебсервер, запитує дані (зазвичай у вигляді HTML та інших файлів, що становлять вебсторінки), а потім розбирає ці дані, щоб витягнути необхідну інформацію [1].

Сфера застосування парсингу дуже широка, його можна використовувати, наприклад, для маркетингових досліджень, моніторингу цін на товари, створення баз даних, навчання моделей машинного навчання та багато іншого. Збір даних за допомогою парсера значно спрощує процес збору інформації та збільшує швидкість її обробки. У випадках, коли обсяги даних надто великі, вручну збирати інформацію стає майже неможливим завданням, і тоді скрапінг може стати єдиним варіантом для отримання потрібних даних.

Парсинг даних може бути корисним інструментом для збору інформації, однак варто пам’ятати, що це може бути незаконним. У багатьох країнах, в тому числі і в Україні, скрапінг може порушувати закони про авторське право

та захист персональних даних. Тому, перед початком збору даних необхідно впевнитися, що правила сайту та його власник це дозволяють. Загалом, парсинг даних може бути корисним інструментом, але слід дотримуватися законів та етичних правил, щоб уникнути порушень і проблем з власниками сайтів та даних.

1.2 Аналіз існуючих продуктів для парсингу вебсторінок

За останні роки з’явилося безліч різноманітних продуктів, які дозволяють отримувати дані з вебсторінок. Деякі з них є комерційними, а деякі є безкоштовними. Розглянемо три найпопулярніших додатки для парсингу вебсторінок: ParseHub, Octoparse та Import.io.

ParseHub є досить простим та ефективним інструментом для створення скраперів без програмування. Однак, як і в будь-якому інструменті, є свої обмеження та недоліки.

Один з найбільших недоліків ParseHub полягає у тому, що він не завжди добре працює з великими обсягами даних. Зокрема, якщо потрібно зібрати велику кількість даних з багатьох сторінок, то може виникнути проблема зі зациклюванням на певних сторінках, що може затримати або навіть призвести до збою скрапера. Також слід зазначити, що він не підтримує всі можливості, які доступні програмістам при написанні скраперів. Наприклад, він не завжди дозволяє зібрати дані з динамічних сторінок, де контент змінюється після завантаження сторінки.

Однак, ParseHub має кілька корисних функцій, таких як можливість імпортувати дані з URL або документа, використання регулярних виразів, збір даних з декількох сторінок, збереження результатів у різних форматах та інші.

Octoparse є потужним інструментом для автоматизованого парсингу вебсторінок, який надає більш широкі можливості порівняно з ParseHub. Цей інструмент дозволяє збирати дані з різних джерел, використовуючи різні

методи, такі як введення ключових слів або застосування регулярних виразів. Octoparse також має можливість автоматичного навчання (machine learning), що дозволяє працювати з більш складними сайтами.

Octoparse має кілька функцій, які роблять його привабливим для користувачів. Він може автоматично виконувати дії, такі як натискання кнопок, заповнення форм, перехід на наступну сторінку і т.д. Крім того, Octoparse дозволяє вам візуалізувати структуру вебсторінки та вибрати ті елементи, які вам потрібні для збору даних.

Однак, мінусом Octoparse є те, що для користування ним необхідні певні навички програмування, що може стати перешкодою для новачків. Крім того, Octoparse може бути менш ефективним для парсингу великих обсягів даних, тому що це може зайняти багато часу та ресурсів. Також, навіть з використанням Octoparse, можуть виникнути проблеми з парсингом деяких вебсторінок, особливо якщо ці сторінки мають складну структуру або захищені від парсингу захистом від ботів.

Import.io є сервісом, який дозволяє створювати власні API для збору даних зі складних джерел, таких як Flash або JavaScript. Цей сервіс має декілька переваг порівняно з іншими інструментами для парсингу. Наприклад, велика база даних API Import.io дозволяє користувачам знайти та використовувати потрібні API швидко та ефективно. Більше того, він забезпечує можливість встановлення спеціальних правил для обробки даних та експорту результатів у різноманітні формати.

Однак, Import.io не є ідеальним інструментом для швидкого збору великих обсягів даних. Завантаження великих масивів даних може займати дуже багато часу, що зменшує ефективність його використання. Крім того, використання Import.io вимагає певного рівня технічної компетентності, що може бути складним для користувачів, які не мають досвіду з програмуванням або API.

Незважаючи на це, Import.io залишається корисним інструментом для збору даних зі складних джерел та створення власних API. Він може

допомогти користувачам зібрати цінну інформацію та зменшити зусилля, які зазвичай пов’язані з парсингом складних вебсторінок.

Таким чином, незважаючи на великий вибір програм та сервісів, не всі з них можуть ефективно працювати з великими обсягами даних. Це обмеження зумовлене технічними особливостями деяких інструментів та сервісів. У зв’язку з цим і було вирішено зробити парсер, направлений на збір та обробку великих масивів інформації.

1.3 Сучасні технології парсингу вебсторінок

Існує багато технологій та інструментів для парсингу даних з вебсторінок, що дозволяють зручно та ефективно отримувати необхідну інформацію.

BeautifulSoup – це бібліотека для мови програмування Python, яка дозволяє швидко та легко аналізувати HTML-код вебсторінок та видобувати з нього потрібну інформацію. Додатково, бібліотека підтримує різні методи парсингу, такі як парсинг DOM-дерева, парсинг XML-файлів та інші.

BeautifulSoup дозволяє швидко та легко розбирати HTML-код та видобувати з нього потрібну інформацію, зокрема зі складних структур вебсторінок, таких як таблиці або форми. Крім того, бібліотека має вбудований парсер HTML-коду, що дозволяє швидко та ефективно обробляти великі обсяги даних. BeautifulSoup також має багато корисних функцій, які дозволяють здійснювати різні операції з даними, такі як фільтрування, пошук, заміна тощо.

Однак, є деякі недоліки BeautifulSoup, зокрема, бібліотека не підтримує автоматичну обробку JavaScript, які часто використовуються на сучасних вебсторінках. Також, парсинг великих обсягів даних може займати досить багато часу та ресурсів, тому у випадку потреби в обробці великих масивів даних може бути краще використовувати інші інструменти.

Scrapy є потужним фреймворком для парсингу вебсторінок, який дозволяє створювати складні скрапери з використанням Python. Він забезпечує швидкий та ефективний парсинг даних з вебсторінок, підтримує різні формати даних, такі як HTML, XML та JSON, та забезпечує зручну систему обробки та збереження даних.

Однією з головних переваг Scrapy є його асинхронний та багатопоточний підхід, що дозволяє швидко та ефективно обробляти великі обсяги даних. Крім того, Scrapy має багато різноманітних функцій, таких як автоматичне виявлення нових сторінок, обробка динамічного контенту, кешування даних та багато іншого.

Недоліком Scrapy може бути складність налаштування та використання для новачків. Крім того, у деяких випадках може виникати проблема з парсингом сторінок, що вимагають виконання JavaScript-коду, але цю проблему можна вирішити за допомогою додаткових бібліотек. Загалом, Scrapy є потужним та надійним інструментом для парсингу даних, який дозволяє з легкістю отримувати потрібну інформацію з вебсторінок.

Також значною популярністю користується Cheerio – це легка та швидка бібліотека на мові JavaScript. Вона дозволяє робити запити до HTML-сторінок, вибирати та маніпулювати елементами сторінки, а також отримувати значення атрибутів і тексту елементів. Бібліотека Cheerio працює на Node.js, що дозволяє легко використовувати її для створення вебскраперів. Через свою легкість та швидкість, Cheerio є дуже ефективним інструментом для парсингу даних з вебсторінок в обмеженому просторі часу.

За результатами огляду, Scrapy, BeautifulSoup і Cheerio є дуже популярними та ефективними інструментами для збору даних з вебсторінок. Кожен з цих інструментів має свої переваги та недоліки, і вибір між ними залежить від потреб користувача та типу проєкту. У таблиці 1.1 наведене порівняння цих бібліотек.

Таблиця 1.1 – Порівняння Cheerio, Scrapy та BeautifulSoup

Характеристика	**Cheerio**	**BeautifulSoup**	**Scrapy**
Мова	JavaScript	Python	Python
Швидкість	Швидка	Повільна	Повільна
Обробка JS	Не підтримується	Підтримується	Підтримується
Структура документів, що підтримується	HTML, XML	HTML, XML	HTML, XML
Селектори	CSS	CSS, XPath	CSS, XPath

Як можна побачити, у питанні швидкості роботи Cheerio є найшвидшим інструментом з трьох, а так як в цій роботі мова йде про великі обсяги інформації, то це і є вирішальним критерієм.

Але Cheerio має один важливий недолік. Ця бібліотека не може взаємодіяти зі сторінками, вміст яких генерується JavaScript, тому для цього необхідно використовувати додатковий інструмент, такий як Puppeteer. Puppeteer має широкий функціонал для автоматизованого тестування, збору даних та інших завдань. Він забезпечує зручні методи для взаємодії та контролю поведінки браузера, що дозволяє отримувати потрібні дані зі сторінок.

Таким чином, внаслідок аналізу сучасних технологій парсингу вебсторінок, було вирішено обрати Cheerio та Puppeteer.

1.4 Системи контейнеризації та оркестрації

Для розгортання парсеру буде використовуватися Docker та Kubernetes.

Docker – це відкрита платформа для розробки, доставки та запуску додатків. Docker дозволяє відокремити додатки від інфраструктури, щоб можна було швидко доставляти програмне забезпечення. З Docker можливо

керувати інфраструктурою так само, як керуються додатками. Використовуючи методології Docker для швидкої доставки, тестування та розгортання коду, можна значно скоротити затримку між написанням коду та його запуском в продакшн середовищі [2]. Основою Docker є Docker-двигун, програмний демон, який працює на одному хості і дозволяє створювати та керувати контейнерами [3].

Kubernetes (K8s) – це система з відкритим вихідним кодом для автоматичного розгортання, масштабування і управління контейнеризованими застосунками [4].

Найбільшою перевагою кластера Kubernetes є те, що він приховує роботу запуску контейнерів на кількох хостах за допомогою шару абстракції. Кластер Kubernetes – це "чорна скринька", яка виконує те, що ми їй кажемо, з автоматичним масштабуванням, переключенням на резервні сервери та оновленням до нових версій нашого додатку [5].

Використання Kubernetes для розгортання парсера може мати декілька переваг. По-перше, Kubernetes надає гнучкість та масштабованість. Можна налаштувати парсер так, щоб він автоматично масштабувався відповідно до потреб проєкту. Коли обсяги даних зростають, Kubernetes забезпечує можливість додавати нові ресурси та розподіляти навантаження між контейнерами. По-друге, Kubernetes забезпечує високу доступність та надійність. Він може автоматично відновлювати контейнери та забезпечувати неперервну роботу парсера навіть у разі виникнення помилок чи збоїв. Крім того, Kubernetes надає механізми для моніторингу та управління здоров’ям контейнерів, що дозволяє швидко виявляти та вирішувати проблеми. По-третє, Kubernetes забезпечує простоту розгортання та керування додатками. За допомогою конфігураційних файлів із специфікаціями контейнерів, можно легко налаштувати та розгорнути парсер на кластері Kubernetes.

Для роботи з Kubernetes ви використовуєте об’єкти API Kubernetes для того, щоб описати бажаний стан вашого кластера: які застосунки або інші робочі навантаження ви плануєте запускати, які образи контейнерів вони

використовують, кількість реплік, скільки ресурсів мережі та диску ви хочете виділити тощо. Ви задаєте бажаний стан, створюючи об’єкти в Kubernetes API, зазвичай через інтерфейс командного рядка kubectl. Ви також можете взаємодіяти із кластером, задавати або змінювати його бажаний стан безпосередньо через Kubernetes API [4].

Для розгортання розробленого парсеру будуть використовуватися такі абстракції Kubernetes, як Deployment та Job.

Deployment використовується для керування розгортанням та масштабуванням stateless додатків, тобто додатків, що не зберігають стан між запусками. Deployment дозволяє задати кількість реплік (копій) додатку, які мають бути запущені в кластері, та автоматично здійснювати масштабування, якщо потрібно. Також, Deployment дозволяє оновлювати версії додатку, забезпечуючи безперервну роботу застосунку в процесі релізу нових версій.

Job використовується для запуску та керування довгочасними процесами, що мають кінцеву точку. Наприклад, виконання скрипту парсингу на певних сторінках сайту. Job може бути запущений один раз або регулярно за допомогою планувальника. Якщо Job не вдалося виконати через помилку або з іншої причини, Kubernetes може автоматично перезапустити його з збереженням попереднього стану.

За допомогою Deployment та Job можна ефективно керувати розгортанням та масштабуванням додатків, а також керувати довгочасними процесами в кластері Kubernetes.

Також, для забезпечення комунікації між компонентами додатку, а також для доступу до додатку ззовні, необхідно використовувати Service.

Service – це абстракція Kubernetes, яка використовується для встановлення стійкої адресації для одного чи декількох реплік певного додатку. Коли створюється Service, Kubernetes створює віртуальну IP-адресу, яку можна використовувати для доступу до додатку. Це дозволяє іншим компонентам додатку взаємодіяти з ним, не звертаючись до конкретної репліки.

Service використовується разом з іншими абстракціями Kubernetes, такими як Deployment або StatefulSet, щоб забезпечити високу доступність та масштабованість додатку. Використання Service дозволяє легко змінювати розмір додатку, не змінюючи його конфігурацію, тому що адресація до додатку залишається незмінною.

Крім того, Service може бути налаштований для забезпечення балансування навантаження між різними репліками додатку. Це дозволяє зменшити завантаження на окремі репліки, збільшити швидкість відповіді додатку та підвищити його надійність.

1.5 Постановка задачі

В попередньому розділі було встановлено те, що існує проблема в наявності парсерів орієнтованих на великі масиви інформації. Так як метою проєкта повинно бути створення актуального та унікального продукта, то саме такий парсер і буде розроблюватися.

Слід визначити функціональні та нефункціональні вимоги до системи, що дозволить розробити ефективний та функціональний парсер, який зможе працювати з різними типами даних та забезпечувати швидке та безперебійне виконання парсингу.

Функціональні вимоги:

- створювати задачі для парсингу відповідно до вказаних даних з визначеною кількістю ресурсів;
- зупиняти та відновлювати задачі;
- отримувати інформацію про існуючі задачі;
- отримувати дані, отримані в результаті виконання задачі.

Нефункціональні вимоги:

- продуктивність;
- універсальність;

– відмовостійкість.

Слід зазначити, що створити універсальний парсер майже неможливо через різну структуру вебсайтів. Таким чином, було вирішено створювати парсер для сторінок певного типу – сторінки зі списком необхідних товарів, на яких є посилання на сторінки з деталями та на наступну сторінку.

До того ж, парсинг є складною задачею, яка вимагає багато ресурсів, але не потребує втручання в процес його роботи. Тому, краще виділити парсер у окремий процес – Worker, щоб забезпечити його інкапсуляцію та зменшити навантаження на систему в цілому. З таким підходом, Master може контролювати Worker та відповідати на запити користувачів. Така архітектура дозволить підтримувати баланс між навантаженням та продуктивністю системи.

Таким чином, завданням є створити Master, що виконує роль користувацького інтерфейсу та API серверу, і Worker, чиїм завданням є безпосередньо парсинг.

Висновки до розділу 1

Таким чином, в першому розділі було розглянуто поняття парсингу та скрапінгу, встановлено різницю та спільне між ними. Також, було проаналізовано існуючі продукти для парсингу вебсторінок, розглянуто три найпопулярніших додатки для парсингу вебсторінок: ParseHub, Octoparse та Import.io та встановлено, що незважаючи на великий вибір програм та сервісів, не всі з них можуть ефективно працювати з великими обсягами даних. Було проаналізовано сучасні технології парсингу вебсторінок на прикладі Scrapy, BeautifulSoup та Cheerio, встановлено що Cheerio є найшвидшим інструментом з трьох, а отже найбільш доречним. Але, так як ця бібліотека не може взаємодіяти зі сторінками, вміст яких генерується JavaScript, то було вирішено використовувати її разом з Puppeteer – бібліотекою для взаємодії з браузером.

Також, було поставлено завдання, встановлено функціональні та нефункціональні вимоги.

2 ПРОЄКТУВАННЯ

Для правильного проєктування системи необхідно пройти такі етапи:

- визначити вимоги до системи;
- визначити прецеденти;
- визначити схему даних;
- побудувати діаграму компонентів;
- побудувати діаграму класів;
- побудувати функціональну модель.

Так як, згідно з технічним завданням, система поділяється на Master та Worker, то деякі етапи необхідно буде провести окремо для кожної з частини системи.

2.1 Визначення прецедентів

Перед початком проєктування необхідно визначити основні функції системи. Для цього необхідно створити Use Case Diagram – діаграму прецедентів, що використовується для опису функціональної поведінки системи та для ідентифікації користувачів, які будуть взаємодіяти з системою та її функціоналом. Опис сценаріїв взаємодії допоможе краще зрозуміти функціональні вимоги системи та сприятиме більш точному проєктуванню та реалізації функцій системи. На рис. 2.1 наведено діаграму для парсеру вебсторінок.

Діаграма прецедентів відображає взаємодію акторів “Користувач” та “Worker” з системою для виконання прецедентів.

Користувач – це актор, що представляє цільову аудиторію додатку, що буде мати можливості, що вказано нижче.

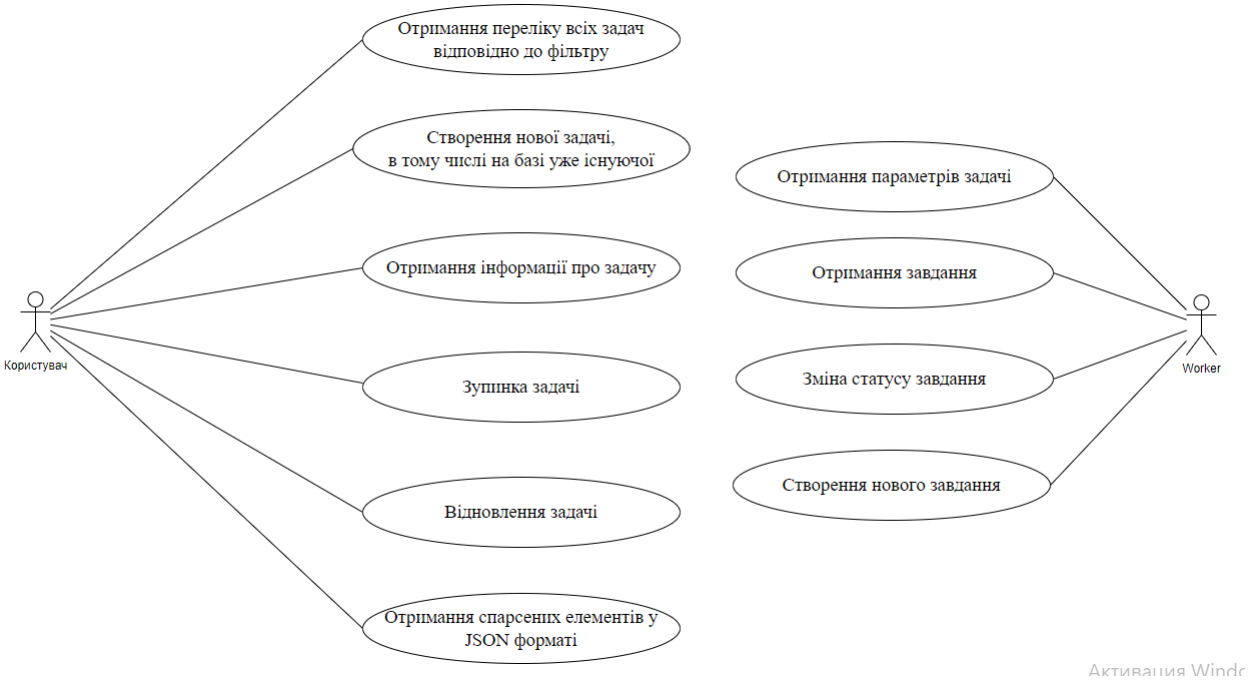

Рисунок 2.1 – Діаграма прецедентів

Отримання переліку всіх задач відповідно до фільтру. Цей прецедент передбачає можливість користувача отримати перелік задач, які відповідають заданим критеріям фільтрації за початковим посиланням та статусом.

Створення нової задачі, в тому числі на базі уже існуючої. Цей прецедент передбачає можливість користувача створити нову задачу в системі, вказуючи відповідні селектори та посилання, або створити її на базі вже існуючої задачі, скопіювавши необхідні дані та змінивши їх у разі необхідності.

Отримання інформації про задачу. Цей прецедент передбачає можливість користувача отримати детальну інформацію про задачу, включаючи її стан, кількість завдань та їх статус тощо.

Зупинка задачі. Цей прецедент передбачає можливість користувача зупинити виконання задачі в системі.

Відновлення задачі. Цей прецедент передбачає можливість користувача відновити виконання раніше зупиненої задачі в системі.

Отримання спарсених елементів у JSON форматі. Цей прецедент передбачає можливість користувача отримати результат парсингу вебсторінки у вигляді JSON об’єкту.

Таким чином, система надає користувачу можливість керувати задачами парсингу вебсторінок та отримувати необхідну інформацію про них, а також отримувати результати парсингу у зручному форматі.

Worker – актор, що представляє окремий компонент системи, що буде виконувати парсинг. Для цього йому необхідно мати можливості, що вказано нижче.

Отримання завдання. Цей прецедент передбачає можливість отримання безпосередньо задачі, що необхідно виконати.

Отримання параметрів задачі. Цей прецедент передбачає можливість отримати необхідні дані для виконання задачі парсингу, такі як селектори чи кількість сторінок, що необхідно спарсити.

Зміна статусу завдання. Цей прецедент передбачає можливість змінювати статус завдання, наприклад коли воно закінчилося чи отримало помилку.

Створення нового завдання. Цей прецедент передбачає можливість створювати завдання, коли це є необхідним.

2.2 Проєктування бази даних

При розробці кожного проєкту однією з важливих складових є правильно підібрана база даних.

Реляційні (РСКБД) та нереляційні (NoSQL) бази даних є двома основними типами баз даних, які використовуються в розробці програмного забезпечення. Однією з основних відмінностей між РСКБД та NoSQL СКБД є підхід до зберігання даних. У РСКБД дані зберігаються в таблицях, а реляційні зв’язки між даними визначаються за допомогою ключів. На відміну від цього,

NoSQL СКБД зберігають дані у вигляді документів, де структура даних не обмежена таблицями та полями, що дозволяє зберігати дані різної структури в одному документі. Якщо схема даних є статичною та відомо, які дані будуть зберігатися в базі даних, то РСБД можуть бути ефективнішим вибором, оскільки вони гарантують цілісність даних та дозволяють встановлювати зв’язки між ними. Однак, якщо схема даних не є статичною і не відомо, які дані будуть зберігатися, то NoSQL СКБД можуть бути кращим вибором, оскільки вони дозволяють додавати нові поля до документів за необхідності.

Для ефективної роботи парсера вебсторінок необхідна швидка та гнучка база даних, і оскільки структура даних, які будуть зберігатися, не відома заздалегідь, то в такому випадку найкраще буде використовувати NoSQL СКБД, одною з яких і є MongoDB.

MongoDB – це документ-орієнтована база даних, розроблена для спрощення розробки додатків та масштабування. Однією з головних переваг MongoDB є гнучкість схеми даних. Вона дозволяє зберігати дані у вигляді документів, які можуть містити будь-яку кількість полів та даних різних типів. Це дозволяє розробникам легко змінювати схему даних під час розвитку додатку, не перероблюючи всю базу даних.

Отже, з базою даних визначилися, тепер необхідно розробити схему даних, а саме ER-діаграму (Entity-Relationship diagram) – графічний інструмент для моделювання відносин між сутностями в базі даних. Вона складається з сутностей (entities), атрибутів (attributes) та зв’язків (relationships). На рис. 2.2 наведено цю діаграму.

Як можно побачити з рис. 2.2, присутні 3 сутності – Job, Task та Item.

Job – основна сутність проєкту, вона містить всі відомості про задачу парсингу: статус, початкове посилання, статус тощо. Також слід звернути увагу на вкладені документи – Params та ItemParam. Params містить селектори для переходу на наступну сторінку та на сторінку товару, а також масив ItemParam, що містить селектори для збирання інформації про товар.

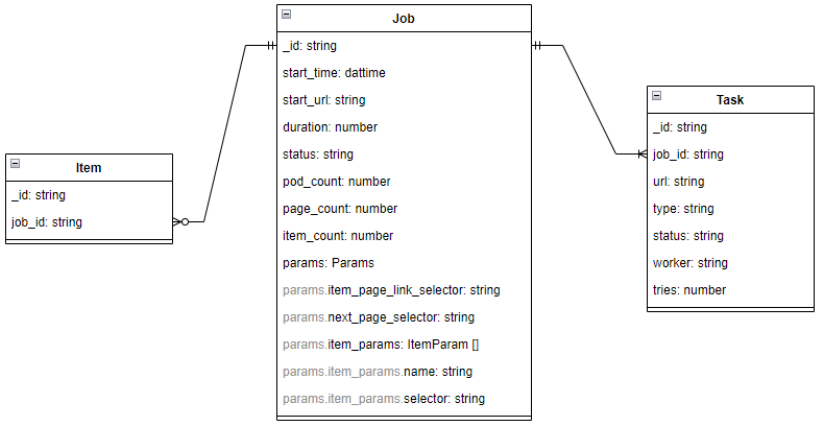

Рисунок 2.2 – Діаграма сутність-зв’язок

Item – це сутність товару, що в цій схемі містить лише ідентифікатор та id задачі, в якій його було спарсено, але він буде містити ті поля, що було зазначено в ItemParam.

Task – це сутність окремого завдання, що містить конкретне посилання, тип задачі тощо. Також слід звернути увагу на поле tries – кількість спроб виконати завдання, щоб, з одного боку, не було завдань, які намагаються виконатися вічно, а з іншого – що не виконуються через раптову затримку на вебсервісі.

Щодо зв’язків на цій діаграмі – то між Job та Task є зв’язок «один-до-багатьох», причому Job буде завжди містити принаймні один Task, бо він буде створюватися одразу ж після Job.

А між Job та Item зв’язок теж «один-до-багатьох», але потенційно, під час виконання задачі може бути не знайдено жодного Item (напевне, через помилку в селекторах), тому і зв’язок не строгий.

2.3 Проєктування Master

Master – це головний вузол системи, що буде на запит користувачів здійснювати відповідні дії щодо керування задачами та отримання інформації.

Для проєктування було вирішено використовувати патерн MVC.

MVC (Model-View-Controller) є архітектурним шаблоном, що широко використовується в розробці програмного забезпечення для розділення логіки, представлення та контролю над даними. Основні переваги MVC полягають у збільшенні масштабованості програми, поліпшенні повторного використання коду та зменшенні взаємодії між різними компонентами системи. Крім того, MVC дозволяє легше змінювати логіку програми та її представлення окремо одне від одного. Однак, є деякі недоліки у використанні шаблону MVC. Один з найбільших недоліків полягає в складності проєктування та розумінні архітектури. Крім того, розробка вебсайту з використанням MVC може вимагати більше зусиль та більшої кількості коду порівняно з іншими архітектурними шаблонами.

У загальному, MVC є потужним інструментом для розробки програмного забезпечення, який дозволяє легко розділяти логіку, представлення та контроль над даними. Однак, при розробці програмного забезпечення з використанням цього шаблону необхідно уважно планувати та проєктувати архітектуру системи, щоб уникнути можливих проблем.

Також було вирішено використовувати паттерн Репозиторій.

Паттерн «Репозиторій» – це один з популярних паттернів проєктування програмного забезпечення, який забезпечує абстракцію бази даних від решти програми та забезпечує уніфікований інтерфейс для роботи з даними. Основна ідея паттерну полягає в тому, що всі операції з базою даних (створення, оновлення, видалення, читання тощо) повинні бути виконуватись через спеціальний об’єкт – репозиторій. Репозиторій має відповідати за доступ до даних, збереження та відновлення. Цей підхід забезпечує ізольованість роботи з базою даних від решти програми, що сприяє зменшенню залежностей та спрощенню тестування коду.

В данному випадку, використання цього паттерну пояснюється тим, що навіть за наявності моделей для роботи з базою використовується досить велика кількість коду, який буде краще винести в окремий модуль.

Таким чином, з урахуванням наведеного вище, було розроблено декілька діаграм.

Як можна побачити з рис. 2.3, Master буде мати два контролери – один для запитів користувачів з браузерів, інший – для API. API контролер використовує сервіси для роботи з чергою та Kubernetes, а також репозиторії для взаємодії з базою даних. Також слід зазначити, що web контролер використовує API контролер для отримання даних для сторінок. Це призводить до того, щоб користувачі отримували інформацію однакового характеру як за допомогою вебінтерфейсу, так і за допомогою API запитів, і при цьому не було необхідності дублювати код.

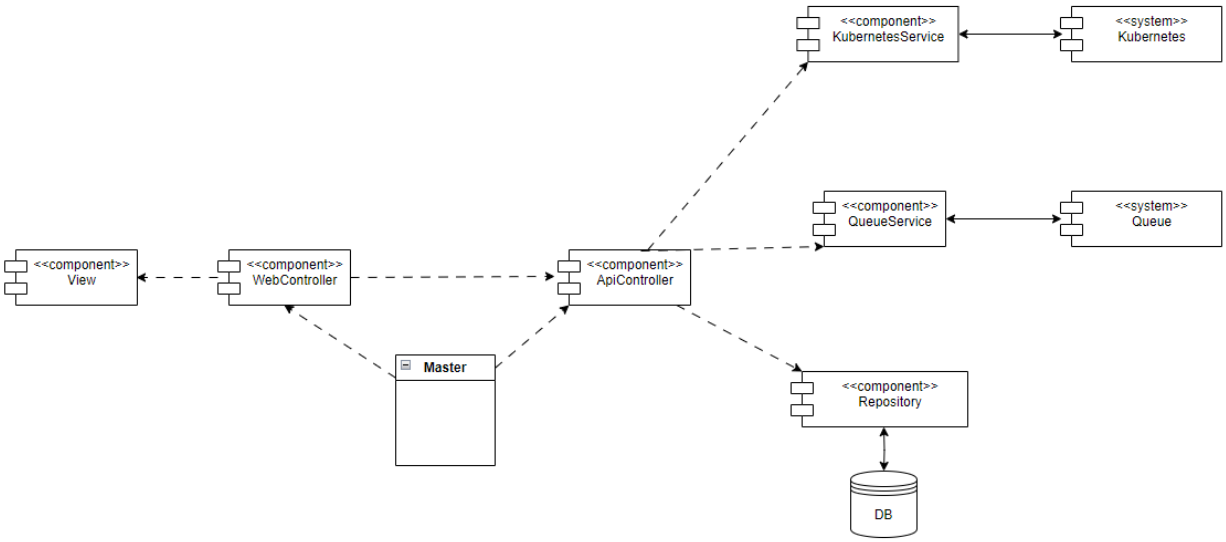

Рисунок 2.3 – Діаграма компонентів Master

На рис. 2.4 зображено діаграму класів, де можна детальніше розглянути архітектуру Master. Слід зазначити, що для біндингу сервісів використовується інтерфейси, що дозволить за необхідності замінити конкретну реалізацію сервісів без змін у залежних класах.

Також, було створено функціональну модель API Master, але через її розмір її винесено у додаток А.

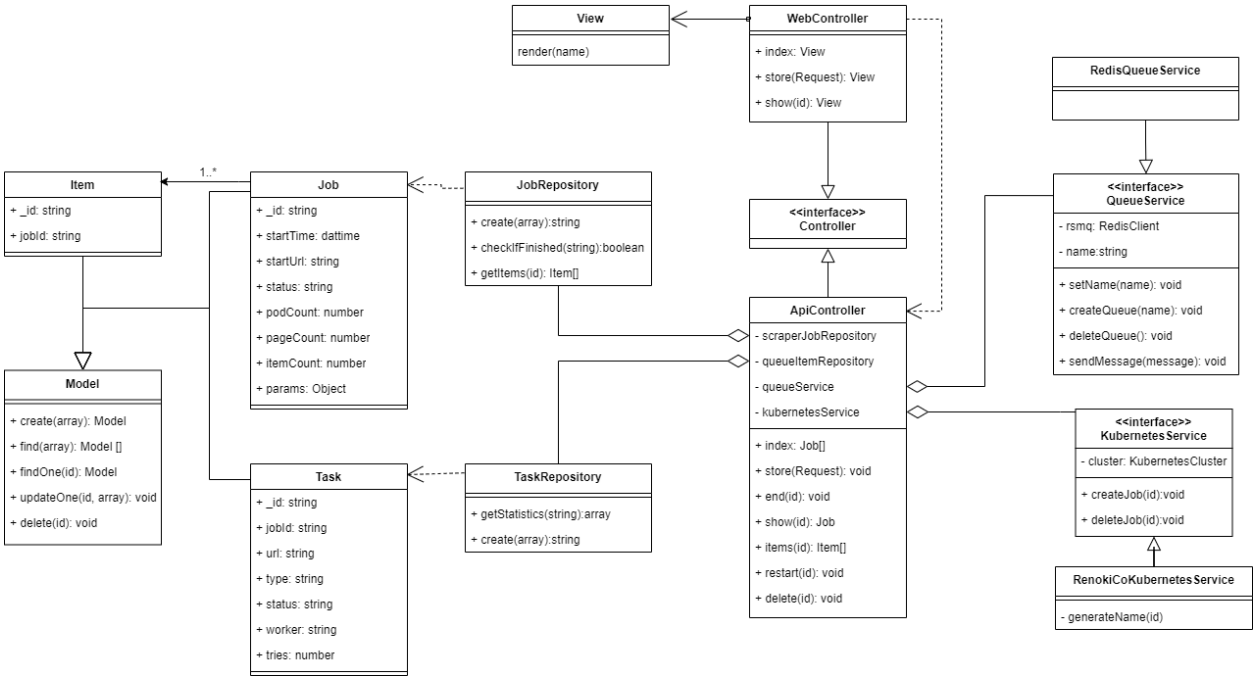

Рисунок 2.4 – Діаграма класів Master

2.4 Проєктування Worker

Worker – це вузол, що відповідає безпосередньо за парсинг. Він отримує з черги завдання та виконує його відповідно до вказаних даних.

Так як парсер буде працювати для збирання даних зі сторінок певного формату, то його поведінка повинна дещо відрізнятися в залежності від типу сторінки, але більша частина функціоналу буде спільною. В такому випадку буде доречним використання паттерну «Інвертований шаблонний метод»

Інвертований шаблонний метод – це підхід до розробки програмного забезпечення, що полягає у визначенні базового алгоритму, який може бути змінений дочірніми класами за допомогою використання віртуальних функцій. Шаблонний метод вимагає, щоб базовий клас визначав алгоритм та послідовність дій, які будуть виконуватися при виклику методу. Проте в інвертованому шаблонному методі послідовність дій визначається дочірніми класами, а базовий клас лише надає фреймворк для цього.

Такий підхід дозволяє розширювати функціональність програмного забезпечення без необхідності модифікації базового класу. Дочірні класи можуть визначати свої власні реалізації віртуальних функцій, що дає можливість змінювати поведінку програми в залежності від потреб користувача. Інвертований шаблонний метод є досить потужним інструментом для розробки програмного забезпечення, оскільки він забезпечує гнучкість та розширюваність системи.

Також, для забезпечення гнучкості та розширюваності парсера, було використано паттерн «Фабричний метод». Цей шаблон дозволяє створювати об’єкти без прив’язки до конкретних класів, що дозволяє легко розширювати функціонал програми. У випадку з парсером, фабричний метод може бути використаний для створення конкретних парсерів в залежності від типу сторінки, яку необхідно розібрати. Таким чином, застосування паттерна «Фабричний метод» дозволить нам створити гнучкий та розширюваний парсер, який зможе працювати з різноманітними типами сторінок, а також легко розширювати свій функціонал в майбутньому.

І так само, як і в Master, у Worker використовується паттерн «Репозиторій», який забезпечує ізоляцію бази даних від решти програми та забезпечує уніфікований інтерфейс для роботи з даними.

Таким чином, з урахуванням наведеного вище, було розроблено декілька діаграм.

На рис. 2.5 зображено діаграму компонентів Worker, де так само як в Master можна побачити використання сервісу для роботи з чергою та репозиторії, а також сервіс для парсингу.

На рис. 2.6 (див. с. 30) можна побачити діаграму класів Worker, що виглядає дещо складною для розуміння через використання інтерфейсів для кожної з підсистем додатку. Це дозволить без змін у залежних класах змінювати реалізації сервісів. Наприклад, якщо з’явиться необхідність змінити чергу з Redis на будь-яку іншу, то необхідно буде додати новий клас та у

service provider зазначити, що на запит про QueueService тепер необхідно повертати новий клас.

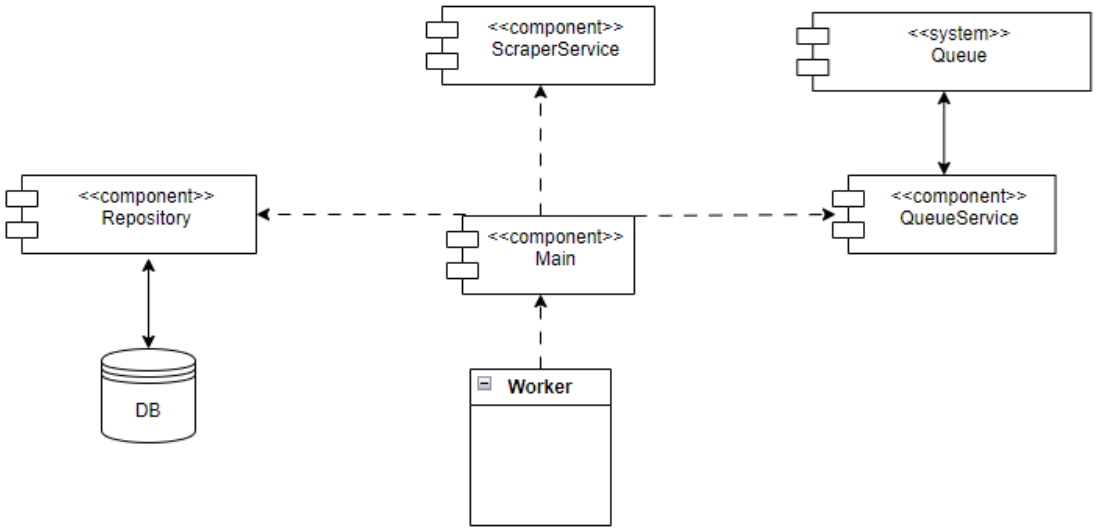

Рисунок 2.5 – Діаграма компонентів Worker

Також можна побачити зазначене вище використання інвертованого шаблонного метода – класи-нащадки ScrapeTask реалізують тільки два методи з шести – головний та метод запиту з бази необхідних селекторів, а інші, загальні для всіх завдання методи, реалізуються лише один раз у батьківському класі та далі використовуються у нащадках.

Також, було створено функціональну модель Worker, але через її розмір її винесено у додаток Б.

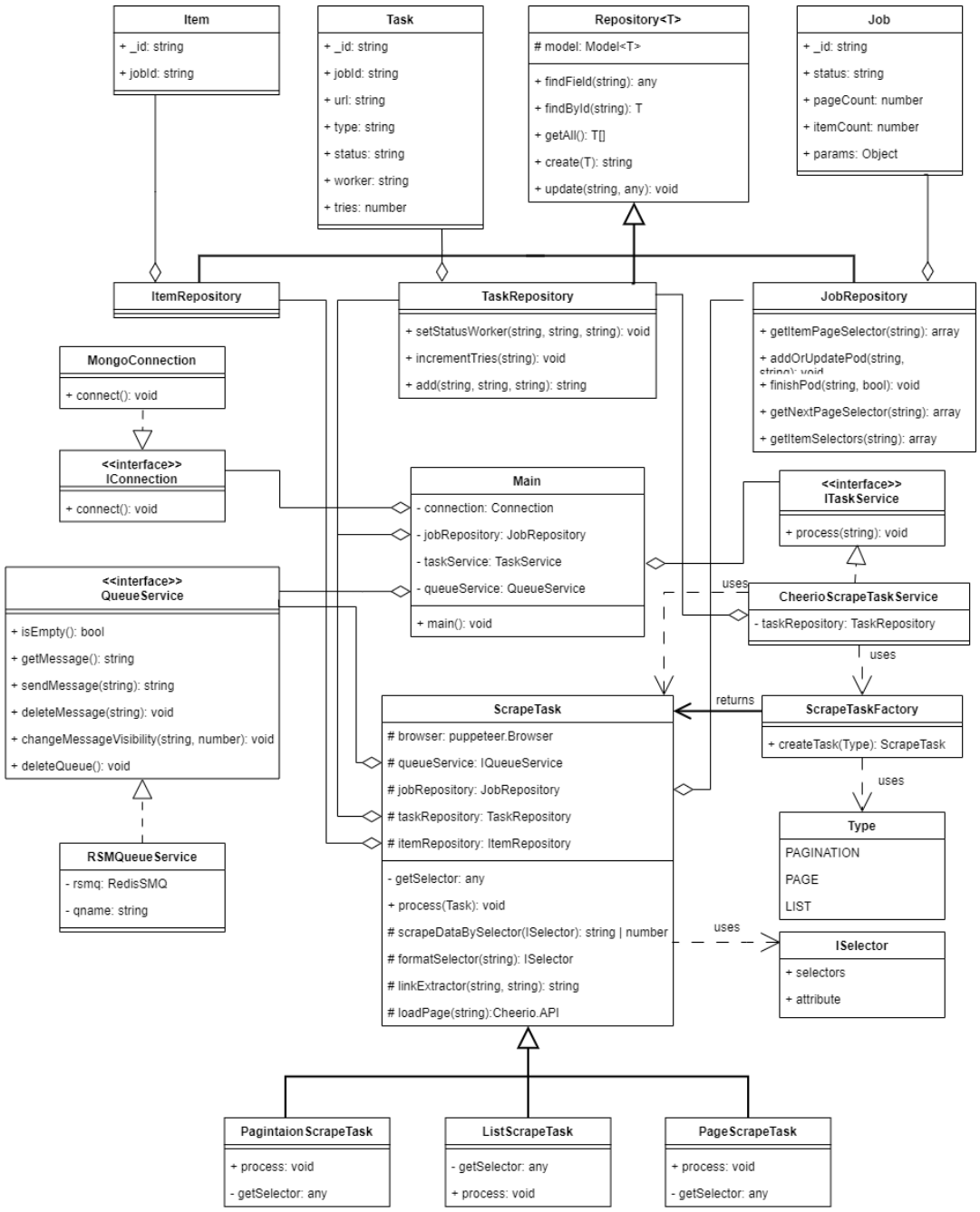

Рисунок 2.6 – Діаграма класів Worker

Висновки до розділу 2

Таким чином, у другому розділі було встановлено функціонал системи та на базі цього розроблено діаграму прецедентів і сплановано загальну архітектуру системи, в результаті було вирішено розділити систему на Master та Worker. Після цього було спроєктовано схему даних і побудовано ER-

діаграму для NoSQL СКБД MongoDB. Також було спроєктовано Master та Worker, і для кожного побудовано функціональну модель та діаграми компонентів та класів.

3 РЕАЛІЗАЦІЯ

3.1 Реалізація Master

Для реалізації Master було вирішено використовувати PHP Laravel.

Laravel – це фреймворк вебдодатків з елегантним синтаксисом. Фреймворк надає структуру та точку виходу для створення вашого додатку, що дозволяє сконцентруватися на створенні чогось неймовірного, поки ми працюємо над деталями [6].

Laravel за своєю суттю ставить перед собою завдання забезпечити розробників необхідними інструментами та можливостями. Його мета – надати зрозумілий, простий та красивий код, а також функціонал, що допоможе розробникам швидко навчатися, розпочати та розробляти проекти, писати код, який буде простим, зрозумілим і довговічним [7].

Головною функцією Master є створення нової задачі. Нижче наведено код методу store, в якому відбувається взаємодія всіх сервісів додатку. Завдяки вдалому проєктуванню, код виглядає лаконічно, незважаючи на велику кількість дій.

```
public function store(StartParserRequest $request)
    {
        $validated = $request->validated();
        $ jobId = $this->jobRepository->create(new
StoreJobRequest($validated));
        $queue_item_id = $this->taskRepository->create(new
StoreTaskRequest(['url' => $validated['url'], 'job_id'=>$jobId]));
        $this->queueService->createQueue($jobId);
        $this->queueService->sendMessage($queue_item_id);
        $this->kubernetesService->createJob($jobId, $request->pod_count);
        return response()->json(['jobId'=>$jobId]);
    }
```

Рисунок 3.1 – Метод створення нової задачі

Для реалізації користувацького інтерфейсу було вирішено використати Vue.js – легкий у використанні, продуктивний та універсальний фреймворк для побудови інтерфейсів користувача для вебсайтів [8]. Завдяки своїй простоті та гнучкості, Vue.js є одним з найбільш широко використовуваних фреймворків у сфері веброзробки.

Основною перевагою Vue.js є його компонентна структура. Кожен елемент інтерфейсу може бути розглянутий як окремий компонент, який можна використовувати в будь-якому місці проєкту. Це дає можливість розробникам швидко та легко змінювати, оновлювати та модифікувати вебінтерфейс [9].

Vue.js також відомий своєю високою продуктивністю, яку забезпечує використання віртуального DOM. Це дозволяє фреймворку швидко оновлювати та переробляти сторінки без перезавантаження сторінки.

Таким чином, Vue.js є потужним та зручним фреймворком для розробки вебінтерфейсів, який забезпечує швидку та ефективну роботу з вебдодатками будь-якої складності.

Таким чином, так як для фронтенду було обрано Vue.js, то тоді необхідно було у файлах Vue реалізовувати механізм запиту даних у сервера. Тоді було б доречно використовувати Vuex – бібліотеку керування станом для Vue.js. Вона дозволяє легко керувати станом додатка, зберігаючи дані в централізованому місці та надаючи доступ до них з будь-якої точки додатку. Але це призвело б до додавання ще досить великого масиву коду. А завдяки використанню Inertia, необхідно лише передати дані до компонента у відповідному роуті.

Inertia – це новий підхід до створення класичних вебдодатків, заснованих на сервері. Inertia дозволяє створювати повністю клієнтські, односторінкові додатки, без складності, яка супроводжує сучасні SPA [10].

Цей фреймворк дозволяє використовувати всю потужність фронтенд-бібліотек, таких як Vue.js, React або Svelte, без необхідності писати API-ендпоінти на бекенді. Замість цього, Inertia передає дані на фронтенд

безпосередньо з контролерів Laravel чи інших фреймворків на стороні сервера, що дозволяє вам швидко створювати масштабовані вебдодатки з мінімальною кількістю зусиль.

На коді нижче наведено метод контролера, який передає до компонента Vue масив задач та параметри для фільтрації.

```
public function index(Request $request): Response
    {
        $urlParams = $request->query();
        $jobs = ApiController::index($request);
        return Inertia::render('Index', [
            'jobs' => $jobs,
            'params'=>$urlParams
        ]);
    }
```

Рисунок 3.2 – Метод для повертання головної сторінки

А для того, щоб отримати дані, в Vue компоненті необхідно лише вказати їх як props, як зображено у коді нижче.

```
export default defineComponent({
  name: 'Index',
  components: {
    JobList,
    MainLayout,
  },
  props: {
    jobs: {
      type: Array,
      required: true,
    },
    params:{
      type: Object,
      required: false,
    }
  }});
```

Рисунок 3.3 – Отримання даних через props у компоненті Vue

3.2 Реалізація Worker

Для реалізації було вирішено обрати NodeJS та TypeScript.

Node.js – це відкрите програмне забезпечення для створення JavaScript-серверів [11]. Воно забезпечує движок JavaScript і десятки API, багато з яких дозволяють програмному коду взаємодіяти з операційною системою та зовнішнім світом [12]. Воно працює на різних операційних системах і дозволяє розробникам виконувати JavaScript-код за межами веббраузера. Node.js забезпечує високу продуктивність та ефективне використання ресурсів комп'ютера за рахунок використання неблокуючого вводу/виводу та однопоточної, подійної моделі програмування.

За допомогою Node.js можна створювати вебсервери, чат-боти, інструменти командного рядка, додатки для обробки даних та інші програми з високою швидкістю роботи та ефективним використанням ресурсів. Node.js також має величезну спільноту розробників, яка забезпечує безліч корисної документації, модулів та інструментів для розробки на цій платформі.

TypeScript – це мова програмування зі сильним типізацією, яка базується на JavaScript і надає кращі інструменти на будь-якому рівні масштабування [13]. TypeScript дозволяє програмістам створювати більш надійний та чистий код, зменшуючи можливість помилок на етапі написання коду та реалізації. Вона підтримує сучасні функції, які доступні у JavaScript, такі як лямбда-функції та асинхронний код, але додає також більше контролю над типами даних та визначенням класів.

Основні особливості TypeScript спрямовані на підвищення продуктивності розробника, особливо за допомогою використання статичних типів, які полегшують роботу з системою типів JavaScript. Інші функції, спрямовані на підвищення продуктивності, такі як ключові слова контролю доступу та лаконічний синтаксис конструктора класу, допомагають запобігати поширеним помилкам в кодуванні [14].

Нижче приведено код основної функції програми – парсингу задачі. У ній по отриманому з черги id задача знаходиться у базі даних та змінюється її статус. За її типом створюється відповідний ScrapeTask та передається йому на обробку. Якщо ж під час виконання задачі виникає помилка, то ставиться відповідний статус та збільшується кількість спроб, після чого задача повертається до черги. Якщо ж кількість спроб досягає п’яти, то задача завершується і видаляється з черги.

```
public async process(taskId:string):Promise<void> {
      const task:ITask | null = await this.taskRepository.findById(taskId);
      if(task == null || task._id == undefined){
            throw new Error('Task not found');
      }else{
            try{
            await this.taskRepository.setStatusWorker(task._id.toString(),
      'processing');
            const scrapeTask = await this.taskFactory.createTask
      (task.type);
            await scrapeTask.process(task);
            await this.taskRepository.setStatusWorker(task._id.toString(),
      'finished');
            }
            catch(error:any){
                  if(task.tries == undefined || task.tries<5){
                        await this.taskRepository.setStatusWorker(task._id.
                  toString(), 'finished_with_error');
                        await this.taskRepository.incrementTries(task._id.
                  toString());
                        throw error;
                  } else {
                        await
                  this.taskRepository.setStatusWorker(task._id.to String(),
                  'error');
                        await this.taskRepository.incrementTries(task._id.to
                  String());
                        return}}}};
```

Рисунок 3.4 – Метод для обробки задачі парсингу

Висновки до розділу 3

Таким чином, у третьому розділі було оглянуто технології, що використовувалися для реалізації, а саме Laravel, Vue.js та Inertia для Master та NodeJS Typescript для Worker. Також було наведено приклади коду, а саме створенння та відновлення задачі, особливості отримання даних на фронтенді з використанням Inertia у Master та обробки задачі у Worker.

4 РОЗГОРТАННЯ

4.1 Налаштування Kubernetes

В даному випадку проєкт буде запускатися на Windows, використовуючи Docker Desktop.

Необхідно встановити чекбокс Enable Kubernetes в налаштуваннях (рис. 4.1).

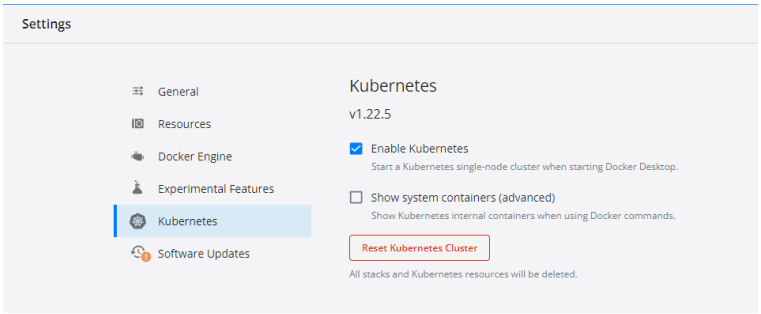

Рисунок 4.1 – Ввімкнення Kubernetes у Docker Desktop

Так як Master буде використовувати Kubernetes API, то необхідно надати йому права керувати Job. Для цього необхідно створити файл з наступним кодом та запустити його:

```
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  namespace: default
  name: service-reader
rules:
- apiGroups: ["*"]
  resources: ["jobs "]
  verbs: ["*"]

kubectl apply -f role.yaml
```

Рисунок 4.2 – Надання прав для керування job

4.2 Створення yaml

Для створення ресурсів в Kubernetes використовується мова YAML. Файли конфігурації Kubernetes у форматі YAML складаються з двох основних частин: метаданих та специфікації. Метадані включають ім’я ресурсу, його унікальний ідентифікатор та іншу інформацію про ресурс. Специфікація містить детальну інформацію про ресурс, таку як тип, параметри запуску та конфігурація.

За допомогою файлів конфігурації Kubernetes YAML можна визначати всі необхідні налаштування для різноманітних ресурсів, таких як деплойменти, служби, репліки та інші. Це дає змогу зберігати всю конфігурацію в одному місці та забезпечити консистентність налаштувань на різних середовищах. Крім того, YAML є розширюваною мовою конфігурації, яку можна використовувати для визначення власних ресурсів та додаткових параметрів для існуючих ресурсів. Це дозволяє налаштовувати кластер Kubernetes під потреби вашого додатку та дозволяє забезпечити його високу продуктивність та масштабованість.`

Для розгортання Master необхідно створити Deployment та Service.

Deployment (рис. 4.3) вказує необхідний Docker-образ, кількість реплік, в даному випадку 1, та порт, на якому додаток буде доступний в кластері.

```
apiVersion: apps/v1                    template:
kind: Deployment                         metadata:
metadata:                                  labels:
  name: parser-master                        app: parser-master
  labels:                                spec:
    app: parser-master                     containers:
spec:                                      - name: parser-master
  replicas: 1                                image: xiori007/parser-
  selector:                                  master
    matchLabels:                             ports:
      app: parser-master                 - containerPort: 80
```

Рисунок 4.3 – Створення Deployment для Master

Також необхідно створити Service. Перший Service надає доступ до Deployment в межах кластера Kubernetes, тоді як другий Service надає зовнішній доступ до Deployment за допомогою LoadBalancer. Код YAML файлів для сервісів зображено на рис. 4.4.

```
apiVersion: v1
kind: Service
metadata:
  name: parser-master
  labels:
    app: parser-master
spec:
  ports:
  - port: 80
    targetPort: 80
  selector:
    app: parser-master
---
apiVersion: v1
kind: Service
metadata:
  name: parser-master-svc
spec:
  type: LoadBalancer
  selector:
     app: parser-master
  ports:
  - name: parser-master
    protocol: TCP
    port: 80
    targetPort: 80
```

Рисунок 4.4 – Створення Service для Master

Worker буде запускатися за допомогою Job, для цього буде використовуватися метод KubernetesService в Master, що зображено на рис. 4.5.

```
    public function createJob($jobId, $podCount)
        {
            $jobName = $this->generateName($jobId);
            $container = K8s::container()
                ->setName('parser-job')
                ->setImage('xiori007/parser-worker')
                ->setEnv([
                    'JOB_ID' => $jobId,
                ]);
            $pod = K8s::pod()
                ->setName('parser-job')
                ->setLabels(['job-name' => $jobName])
                ->setContainers([$container])
                ->restartOnFailure();
            $job = $this->cluster
                ->job()
                ->setName($jobName)
                ->setSpec("parallelism", $podCount)
                ->setSpec("ttlSecondsAfterFinished: ", 20)
                ->setSpec("backoffLimit: ", 3)
                ->setSpec("activeDeadlineSeconds: ", 20)
                ->setTemplate($pod);
            $job->create();
        }
```

Рисунок 4.5 – Метод для створення нового Job

4.3 Розгортання

Для запуску Master необхідно виконати декілька команд.

Так як в роботі використовується черга на Redis, то необхідно для початку запустити його за допомогою офіційних прикладів Kubernetes.

```
kubectl apply -f https://k8s.io/examples/application/guestbook/redis-leader-deployment.yaml
kubectl apply -f https://k8s.io/examples/application/guestbook/redis-leader-service.yaml
```

Рисунок 4.6 – Створення Redis Deployment та Service

Після цього Redis буде доступний в кластері на порту 6379.

Для запуску Master необхідно виконати команду, що наведена на рис. 4.7.

```
kubectl apply -f parser-master.yaml
```

Рисунок 4.7 – Запуск Master

Після цього Master буде доступний на 80 порту і можна буде побачити інтерфейс.

Таким чином, парсер вдало запущено і приклади роботи програми можна переглянути у Додатку В.

Висновки до розділу 4

Таким чином, у цьому розділі було розглянуто таку систему як Kubernetes, встановлено що необхідно для розгортання розробленого парсера. Також Kubernetes було налаштовано та написано конфігураційні файли для Master та Worker. Після цього парсер було запущено.

ВИСНОВКИ

Отже, через постійне збільшення вимог до кількості даних та актуальністю автоматизації їх отримання метою даної роботи було обрано розробку парсера вебсторінок.

Під час досягнення поставленої мети було проаналізувано існуючі продукти для парсингу вебсторінок, серед яких ParseHub, Octoparse та Import.io, також було досліджено сучасні інструменти парсингу, серед яких Scrapy, BeautifulSoup та Cheerio, під час чого Cheerio було обрано для подальшої розробки. Також було встановлено вимоги до розроблюваної системи, внаслідок чого було вирішено розділити додаток на Worker та Master.

Далі було визначено прецеденти та побудовано відповідну діаграму, а також була визначена схема даних та побудована ER-діаграма. Після цього були спроєктовані Master та Worker та було побудовано діаграми компонентів і класів, та функціональну модель.

Після цього було реалізовано кожен з компонентів системи. Master було реалізовано засобами Laravel, VueJS та Inertia, Worker – NodeJS та Typescript.

Після цього було налаштовано Kubernetes, розроблено конфігураційні файли та розгорнуто парсер, після чого його було протестовано.

Внаслідок виконання роботи, мною було засвоєно багато корисної інформації щодо актуальних засобів парсингу даних, а також сучасних засобів розробки застосунків.

ПЕРЕЛІК ПОСИЛАНЬ

1. Mitchell R. Web Scraping with Python. Sebastopol: O'Reilly Media, Inc., 2015. 239 p.

2. Docker Documentation. URL: https://docs.docker.com/get-started/overview/ (дата звернення: 10.05.2023).

3. Goasguen S. Docker Cookbook: Solutions and Examples for Building Distributed Applications. Sebastopol: O'Reilly Media, Inc., 2015. 366 p.

4. Kubernetes Documentation. URL: https://kubernetes.io/docs/home/ (дата звернення: 10.05.2023).

5. Hohn F. The Book of Kubernetes: A Complete Guide to Container Orchestration. San Francisco: No starch press, Inc., 2022. 384 p.

6. Laravel Documentation. URL: https://laravel.com/docs/10.x (дата звернення: 10.05.2023).

7. Stauffer M. Laravel: Up & Running: A Framework for Building Modern PHP Apps. Sebastopol: O'Reilly Media, Inc., 2019. 552 p.

8. VueJS Documentation. URL: https://vuejs.org/guide/introduction.html (дата звернення: 10.05.2023).

9. Heitor R. Vue.js 3 Cookbook: Discover actionable solutions for building modern web apps with the latest Vue features and TypeScript. Birmingham: Packt Publishing, 2020. 562 p.

10. Inertia Documentation. URL: https://inertiajs.com (дата звернення: 10.05.2023).

11. NodeJS Documentation. URL: https://nodejs.org/api/documentation.html (дата звернення: 10.05.2023).

12. Hunter Th. II Distributed Systems with Node.js: Building Enterprise-Ready Backend Services. Sebastopol: O'Reilly Media, Inc., 2020. 377 p.

13. Typescript Documentation. URL: https://www.typescriptlang.org/docs/ (дата звернення: 10.05.2023).

14. Freeman A. Essential TypeScript 4: From Beginner to Pro. New York City: Apress, Inc., 2021. 581 p.

ДОДАТОК А

Функціональна модель API Master

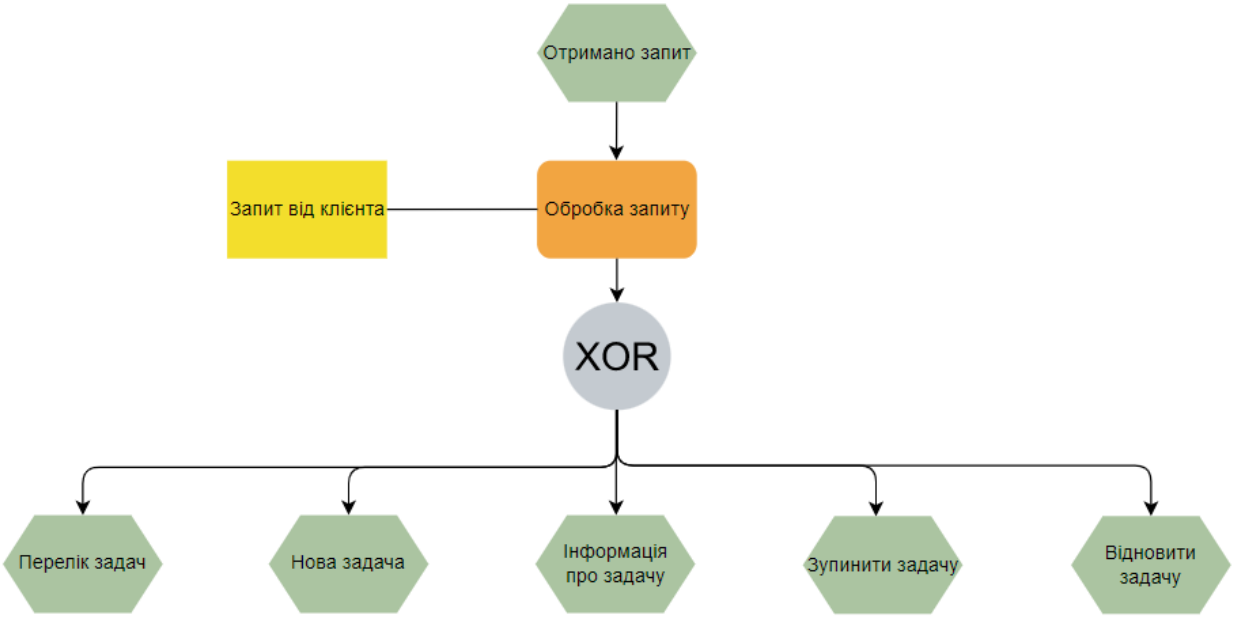

Рисунок А.1 – Модель основних процесів

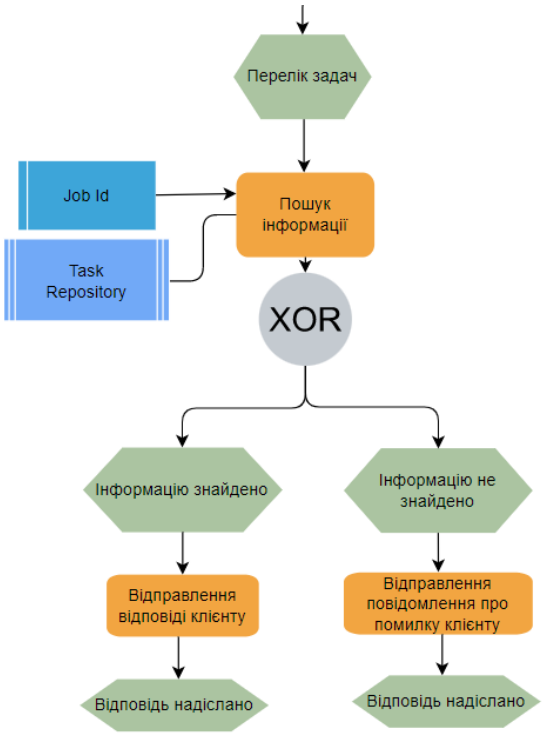

Рисунок А.2 – Модель процесу «Перелік задач»

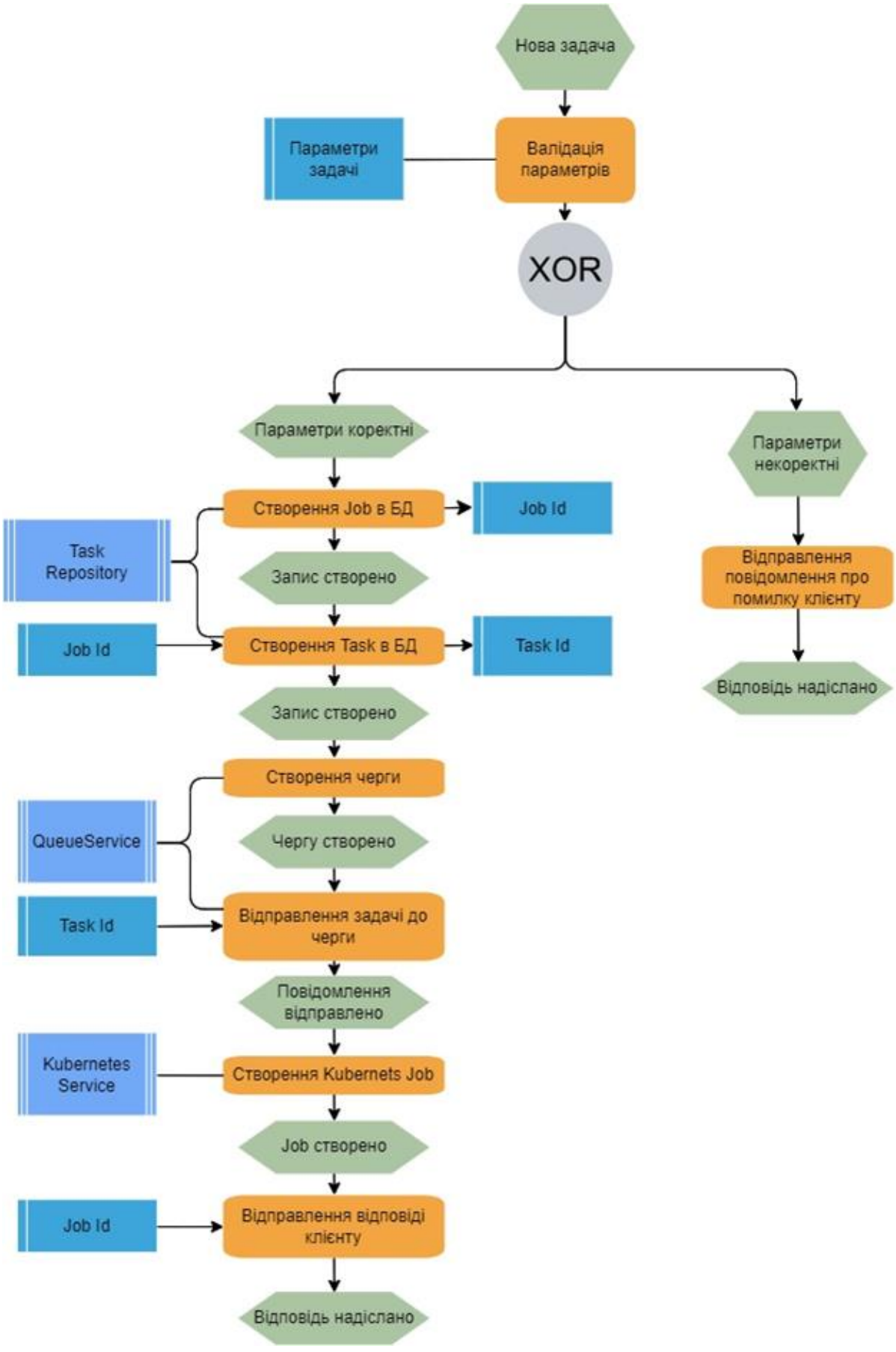

Рисунок А.3 – Модель процесу «Нова задача»

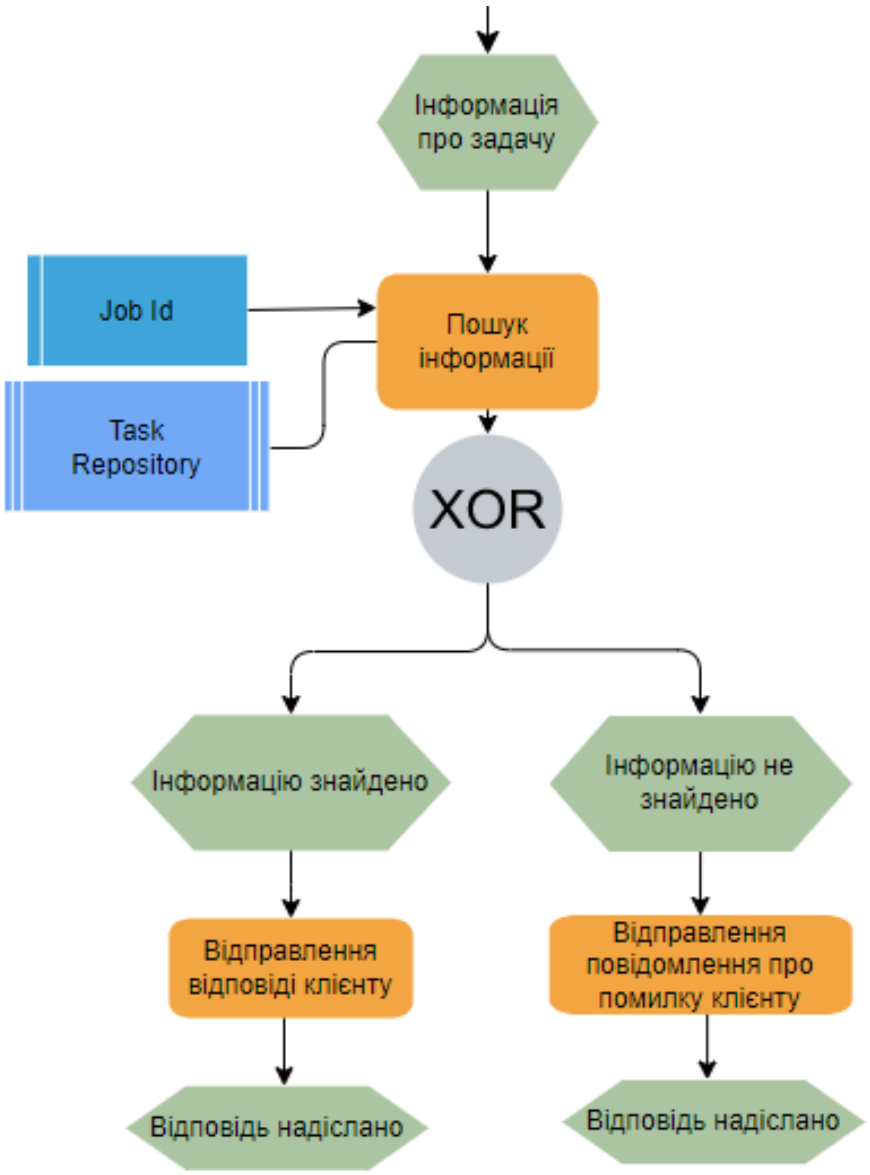

Рисунок А.4 – Модель процесу «Інформація про задачу»

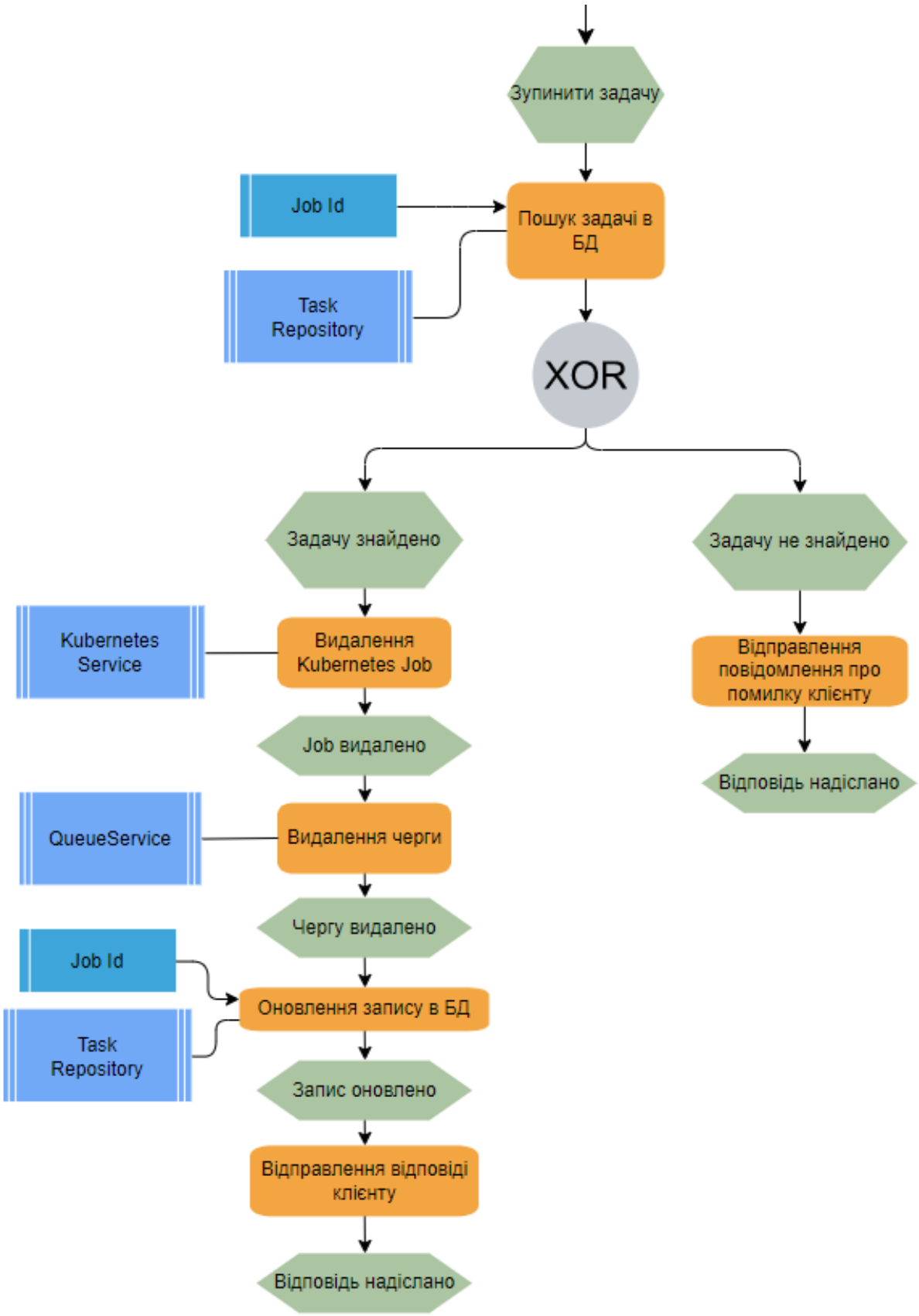

Рисунок А.5 – Модель процесу «Зупинити задачу»

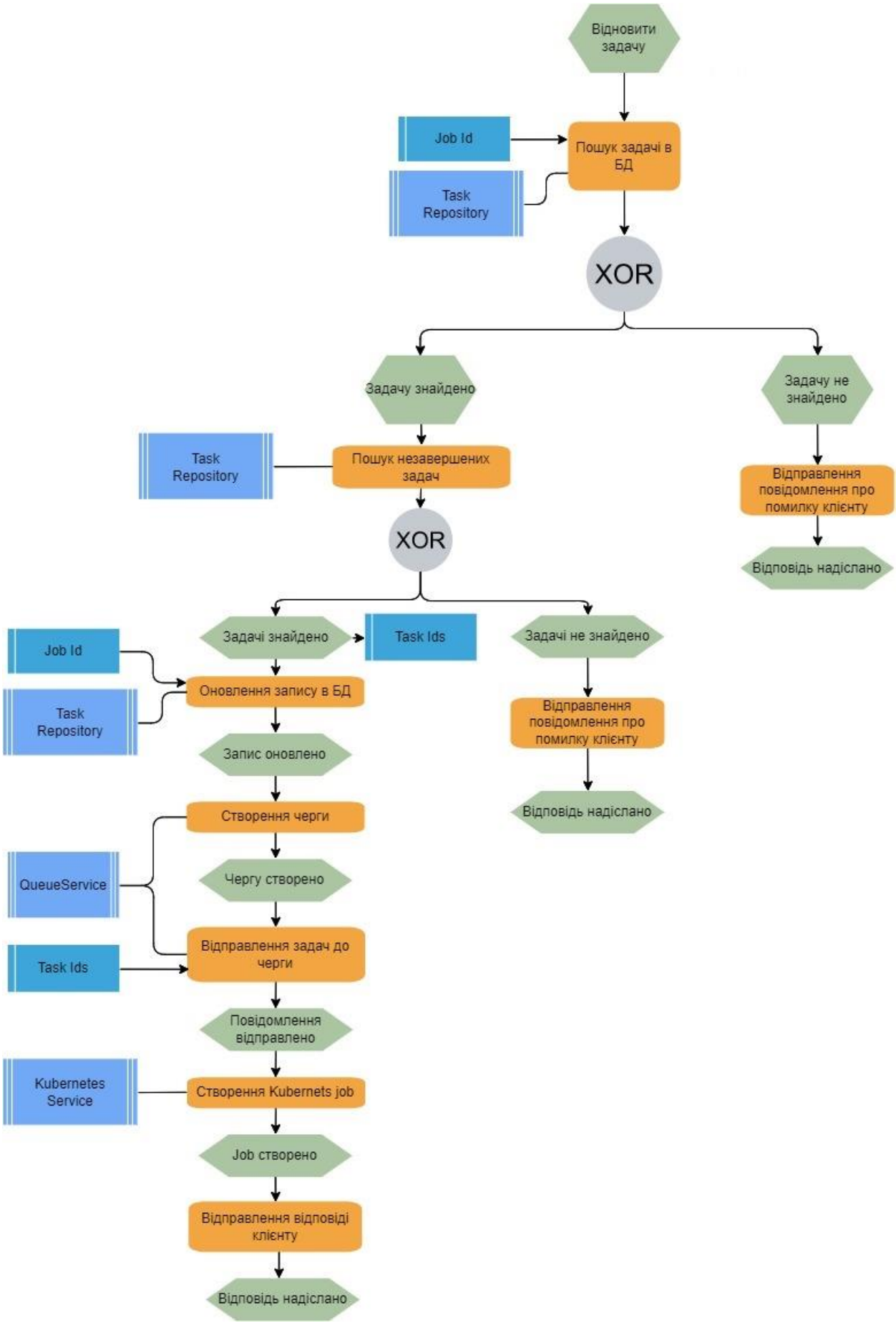

Рисунок А.6 – Модель процесу «Відновити задачу»

ДОДАТОК Б

Функціональна модель Worker

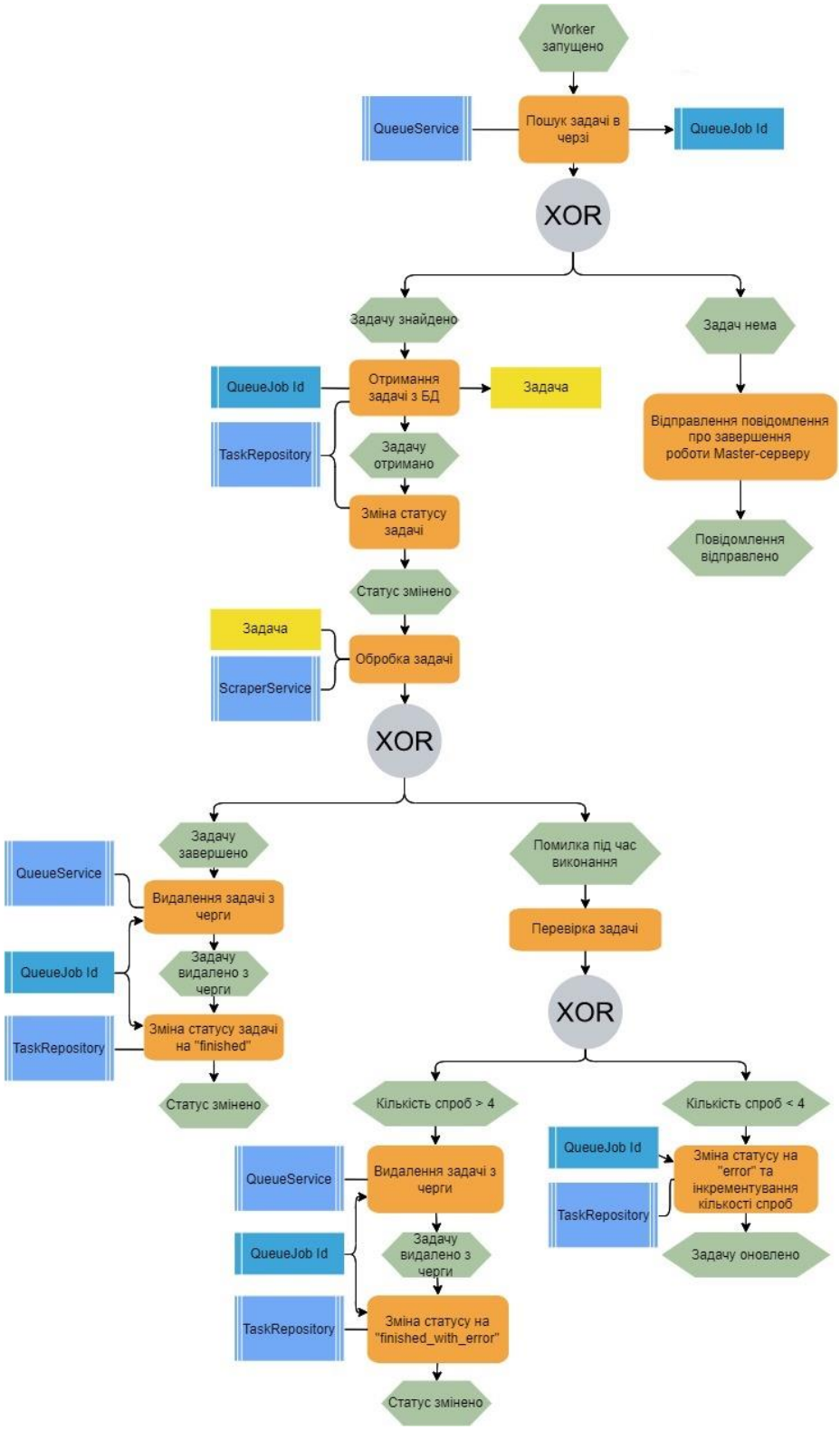

Рисунок Б.1 – Модель основного процесу

ДОДАТОК В

Приклади роботи програми

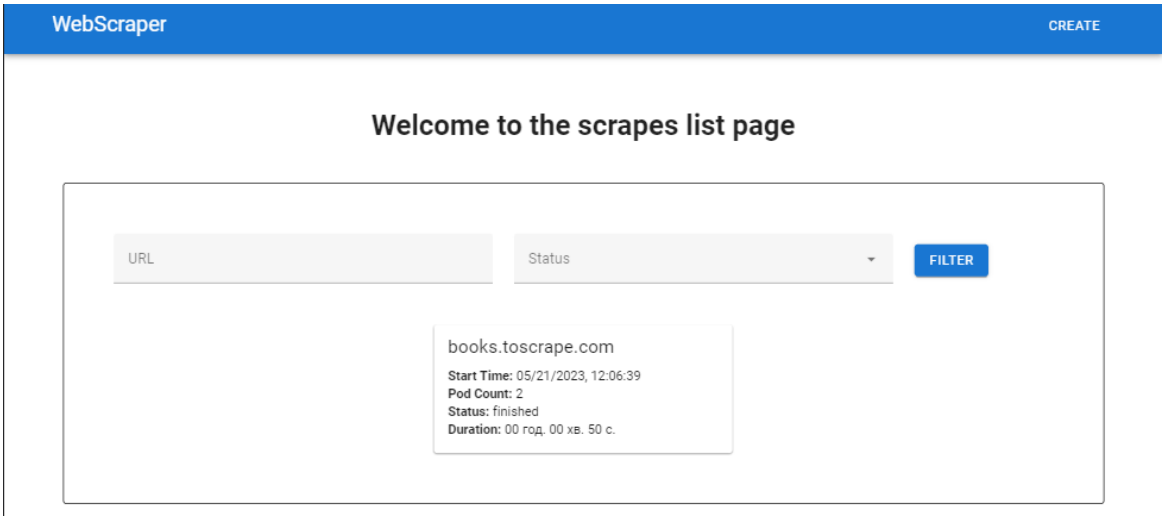

Рисунок В.1 – Головна сторінка

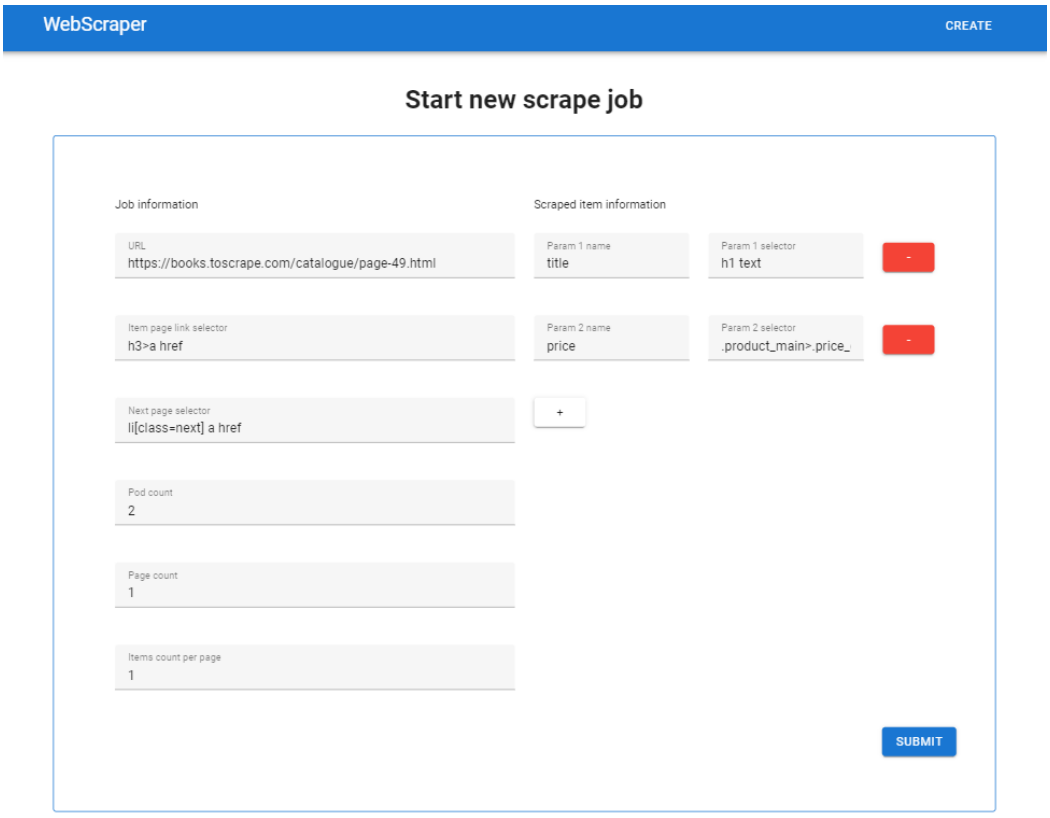

Рисунок В.2 – Сторінка створення завдання

Рисунок В.3 – Сторінка перегляду завдання