

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Інститут інформатики та радіоелектроніки
Факультет комп'ютерних наук і технологій
(повне найменування інституту, факультету)

Кафедра системного аналізу та обчислювальної математики
(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

магістра

(ступінь вищої освіти)

на тему Оптимізація процесів CMS Magento 2 засобами топології черги
повідомлень

Виконав: студент(ка) 2 курсу, групи КНТ-810М

Спеціальності 124 «Системний аналіз»
(код і найменування спеціальності)

Освітня програма (спеціалізація)
«Інтелектуальні технології та прийняття рішень»

 Литвиненко А.В.

(прізвище та ініціали)

Керівник Терещенко Е.В.

(прізвище та ініціали)

Рецензент Шишканова А.А.

(прізвище та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
 (повне найменування вищого навчального закладу)

Інститут, факультет Інститут інформатики та радіоелектроніки,
Факультет комп'ютерних наук і технологій
 Кафедра Системного аналізу та обчислювальної математики
 Ступінь вищої освіти магістр
 Спеціальність 124 Системний аналіз
 (код і найменування)
 Освітня програма (спеціалізація) Інтелектуальні технології та прийняття рішень
 (назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри

проф. Г.В. Корніч

«20» грудня 2021 року

ЗАВДАННЯ

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

Литвиненка Артема Валерійовича

(прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) Оптимізація процесів CMS Magento 2 засобами топології черги повідомлень

керівник проєкту (роботи) Терещенко Еліна Валентинівна, к.ф.-м.н., доц.

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНТУ від «10» грудня 2021 року № 490-к

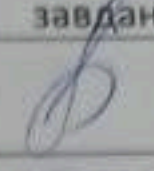
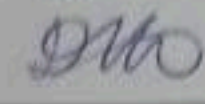
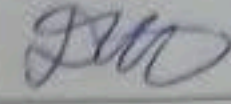
2. Строк подання студентом проєкту (роботи) 21 грудня 2021 року

3. Вихідні дані до проєкту (роботи) Теоретичні відомості черги повідомлень, система менеджменту контенту Magento 2

4. Зміст розрахунково-пояснювальної записки (перелік питань, які належить розробити) Проведено дослідження проблеми оптимізації завантаження веб-сторінок та причини, через яку користувачі залишають веб-ресурси. Наведено власну програмну реалізацію для оптимізації часу завантаження веб-ресурсу за допомогою черги повідомлень

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

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-3	Терещенко Еліна Валентинівна, к.ф.-м.н., доцент		
Нормоконтроль	Широкорад Дмитро Вікторович, к.ф.-м.н.		

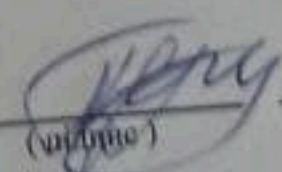
7. Дата видачі завдання « 2 » жовтня 20 21 року.

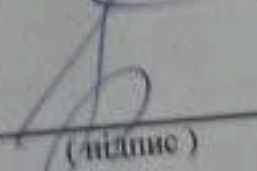
КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Сформулювати мету та основні завдання дипломної роботи	20.09.2021 – 30.09.2021	
2	Проаналізувати задачу та опрацювати літературу	01.10.2021 – 10.10.2021	
3	Підключення та налаштування брокеру черги повідомлень за допомогою технології контейнерів	11.10.2021 – 30.10.2021	
4	Розробка модуля для CMS Magento 2	01.11.2021 – 10.11.2021	
5	Демонстрація роботи модуля та порівняння результатів роботи веб-ресурсу до та після реалізації модуля	11.11.2021 – 15.11.2021	
6	Підсумки щодо дипломної роботи. Оформлення пояснювальної записки	16.11.2021 – 19.11.2021	
7	Попередній захист дипломної роботи та отримання рецензій	14.12.2021	
8	Захист дипломної роботи	21.12.2021	

Студент

Керівник проєкту (роботи)


(підпис)


(підпис)

Литвиненко А.В.
(прізвище та ініціали)

Терещенко Е.В.
(прізвище та ініціали)

РЕФЕРАТ

ПЗ: 54с., 27 рис., 17 джерел, 2 таблиці.

Об'єкт дослідження – процес обробки черги повідомлень для основних процесів інтернет-магазину.

Предмет дослідження – програмні засоби для оптимізації швидкості виконання основних процесів інтернет-магазину.

Мета роботи – створити модуль для CMS Magento 2, який буде керувати основними процесами інтернет-магазину за допомогою черги повідомлень та який оптимізує швидкість виконання основних процесів щонайменше, як у 1,5 рази. Реалізувати середовище для брокеру повідомлень за допомогою сучасного програмного забезпечення Docker та менеджера контейнерів Warden.

Метод дослідження – використання інструментів браузера для порівняння швидкості виконання процесів та швидкості завантаження сторінок.

CONTENT MANAGEMENT SYSTEM, MAGENTO 2, DOCKER,
CONTAINER, IMAGE, WARDEN, MESSAGE QUEUE, OPTIMIZATION, PAGE
SPEED, PHP, RABBIT MQ

ЗМІСТ

ЗАВДАННЯ

РЕФЕРАТ

ВСТУП.....	6
1. ТЕОРЕТИЧНА ЧАСТИНА.....	7
1.1 Content management system.....	7
1.2 CMS Magento 2	8
1.3 Черга повідомлень.....	6
1.4 Роз'єднання та масштабованість черги повідомлень	9
1.5 Програмне забезпечення Docker.....	15
1.6 Брокер повідомлень Rabbit MQ	17
2. РЕАЛІЗАЦІЯ СЕРЕДОВИЩА ТА МОДУЛЯ.....	25
2.1 Аналіз процесу реєстрації без використання черги повідомлень	25
2.2 Створення Docker контейнера для сервісу Rabbit MQ.....	27
2.3 Створення модулю для Magento 2.....	34
2.4 Створення нової черги повідомлень	37
3 ДОСЛІДЖЕННЯ РЕАЛІЗОВАНОГО МОДУЛЯ.....	40
3.1 Дослідження швидкості виконання процесів з використанням черги повідомлень.....	45
3.2 Дослідження швидкості виконання процесів на реальних проектах.....	44
ВИСНОВКИ.....	47
ПЕРЕЛІК ПОСИЛАНЬ.....	48

ВСТУП

В епоху розвитку електронних ресурсів досить швидко набрали популярності інтернет-магазини. Багато оффлайн сервісів та постачальників почали покращувати продажі за допомогою онлайн сервісів та веб-сайтів. Але, процес розвитку не стояв на одному місці, та за останні 20 років відбулося досить багато змін. Наприклад, було проведено конференції, які затверджували нові стандарти веб-розробки. А мови, за допомогою яких створювались веб-сайти, отримали досить багато оновлень та нових версій. З часом почали акцентувати увагу на оптимізацію роботи веб-ресурсів та на швидкість завантаження сторінок. В даний момент було розроблено достатню кількість інструментів для оптимізації завантаження візуальної частини веб-сайту. Але, варто пам'ятати про те, що будь-який веб-сайт не складається лише з візуальної частини, а також з частини, яка відповідає за обробку інформації, та не є видимою для користувача. До таких процесів можна віднести роботу з базою даних, ведення розрахунків, оформлення онлайн-замовлення, відправлення електронного листа тощо. Так, як ці процеси є ресурсномісткими, більшість власників інтернет-магазинів в останній час акцентують увагу на оптимізації цих процесів.

У дипломній роботі досліджується проблема завантаженості веб-сайту, реалізованому за допомогою CMS Magento 2, ресурсномісткими процесами, котрі потребують оптимізації та зменшення часу виконання. Для розв'язку цієї проблеми було розроблено модуль, який має можливість виконувати складні процеси за допомогою черги повідомлень.

1. ТЕОРЕТИЧНА ЧАСТИНА

1.1 Content management system

CMS (Content Management System) - це програмне забезпечення, яке дозволяє користувачам співпрацювати у створенні, редагуванні та виробництві цифрового вмісту: веб-сторінок, публікацій у блогах тощо. Цифровий вміст розміщується у більш надійну систему, яка є основою управління загальним цифровим досвідом у багатьох різних каналах, таких як електронна пошта, мобільні програми, соціальні медіа, веб-сайти тощо.

Будь-яке базове рішення для управління контентом допомагає завантажувати або писати контент власноруч, формувати його, додавати заголовки та зображення тощо. Але у світі з кількома пристроями, орієнтованим на користувача, цього недостатньо. Цифровий маркетинг розвивався, приділяючи більше уваги досвіду клієнтів, включаючи мобільні додатки. Все залежить від системи управління веб-контентом, яка відокремлює управління контентом від презентації вмісту.

Важливі можливості рішення для управління контентом включають:

- 1) Управління вмістом: можна легко створювати, редагувати, публікувати та керувати веб-вміст (включаючи текст, зображення, відео та аудіо);
- 2) Презентація: клієнти переходять від пристрою до пристрою, і жодна маркетингова організація не може докласти зусиль для оптимізації кожної публікації в блозі або сторінки каталогу для ноутбуків, планшетів та найпопулярніших форматів смартфонів. За допомогою CMS , потрібний контент буде створено лише один раз, а далі система сама оптимізує фінальний результат під потрібний пристрій;
- 3) Персоналізація: є можливість націлювати релевантний вміст на унікальних відвідувачів за персональністю, місцем розташування чи навіть окремо, на основі минулої взаємодії з брендом;

4) Аналітика: відстежується активність користувачів в будь-якому місці веб-ресурсу та підраховується статистика відвідувань, яку сторінку чи продукт найчастіше відвідували. Допомогає аналізувати, чи є попит на обслугований вміст та що користувачі роблять далі з цим вмістом;

5) Управління: будь-який опублікований матеріал досить легко знайти. Також є можливість гнучких налаштувань відносно матеріалу. Наприклад: кому дозволено схвалювати його, змінювати, публікувати, знімати тощо;

6) Багатомовна підтримка: всі CMS підтримують можливість перекладу контенту на будь-яку мову сучасного світу;

7) Масштабованість/продуктивність/стабільність: більшість CMS є досить надійними з фундаментальної технологічної точки зору.

З самого початку йде розрахунок на високу навантаженість веб-ресурсу та заздалегідь закладається можливість розширення, спираючись на потреби користувача.

1.2 CMS Magento 2

Magento 2 - це платформа електронної комерції з відкритим вихідним кодом, яка була створена Роєм Рубіном та Йоавом Кутнером. CMS Magento 2 було розроблено основі мови програмування PHP. Ця CMS є досить простою, гнучкою, універсальною та швидкою у використанні [1].

В даний момент є популярною, актуальною та використовуваною системою управління контентом для створення інтернет-магазинів по всьому світу завдяки своїй потужній, масштабованій архітектурі. Ця система сумісна з найновішими технологіями, такими як – мова PHP версії 7, що означає більш швидкий час завантаження в порівнянні з попередньою версією Magento. Зручна і зрозуміла структура Magento дозволяє швидко вносити зміни та

карувати торговельною мережею з декількома сайтами на різних мовах. Вона забезпечує повне управління зовнішнім виглядом, змістом і функціональністю інтернет-магазину.

Також варто звернути увагу на слоган компанії "Designed to grow, built for flexibility" – "Створена для зростання, розроблена для гнучкості". Але що це означає? Ця платформа дає користувачам чудову можливість розвивати бізнес відповідно до потреб. Власники онлайн магазинів можуть почати свій бізнес з базових функцій, але, з розвитком бізнесу, вони можуть переключитися на більш просунуту версію платформи.

А як на рахунок гнучкості платформи? Тут теж все вірно, оскільки Magento має відкритим вихідним кодом, що дозволяє інтегрувати будь-яке розширення або модуль. Існує безліч розширень для цієї платформи. Якщо бізнесу буде потрібно щось особливе, то платформа надає можливість розробити власний модуль, який буде задовольняти потребам.

Щодо долі всього ринку, то на зображенні (Рис.1) ми можемо спостерігати, що на цій платформі створені близько 4.55% всіх онлайн магазинів, що робить її четвертою за популярністю онлайн платформою. На даний момент, близько 141.800 онлайн магазинів створені саме на Magento 2.

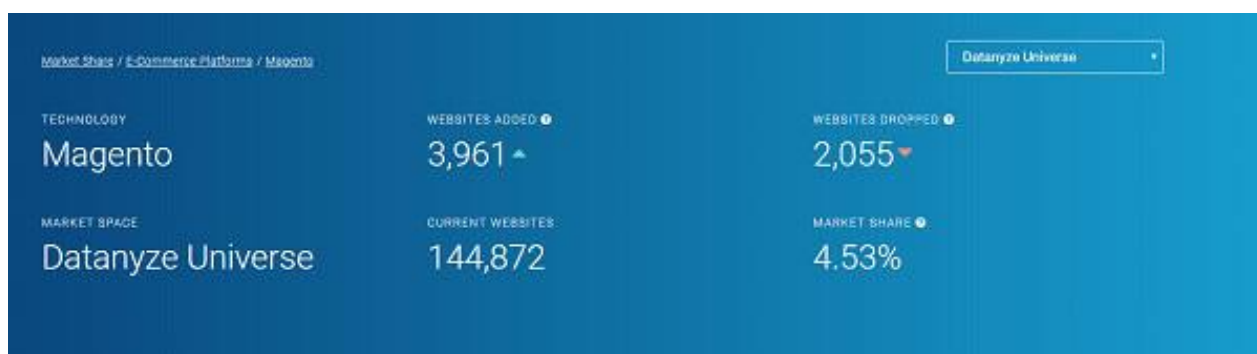


Рисунок 1.1 – Актуальна статистика платформи Magento 2

Серед самих відомих брендів, платформою Magento користуються:

1. **Coca Cola** – відомий бренд прохолодних напоїв. Компанія створила онлайн магазин на платформі Magento для продажу подарункових наборів, персоналізованих пляшок і одягу.

2. **Ford** – одна з найвідоміших компаній в США. Ford вирішили створити онлайн магазин на цій платформі для продажу запчастин.

3. **Warby Parker** – компанія відома завдяки продажу сонячних окулярів та окулярів з діоптріями. Володіючи 80% часткою ринку в США, вони вибрали саме Magento для розробки онлайн магазину.

4. **Olympus** – компанія, яка відома завдяки виробництву оптики, фототехніки і високоякісного оптичного устаткування, також обрала Magento для свого онлайн магазину.

Загалом, система пропонує багато функцій та розширювану кодову базу, яка дійсно розрахована на високу навантаженість. Завдяки цим особливостям, організації з малими та великими операціями, і навіть світові магнати використовують дану платформу для різноманітних проєктів.

Але, щоб зрозуміти, що таке Magento, нам потрібно розібратись, що таке інтернет комерція (або електронна комерція) [2].

Електронна комерція – це процес купівлі та продажу товарів та послуг через Інтернет та передає гроші та дані для здійснення цих транзакцій. Електронну комерцію також називають електронною комерцією або Інтернет-комерцією.

Є такі види електронної комерції:

1. **Бізнес до бізнесу (Business To Business):** Ця транзакція здійснюється між підприємствами. Це відбувається між оптовим та роздрібним торговцем або між виробником та оптовим продавцем.

2. **Бізнес до споживача (Business To Customer):** Ця операція здійснюється між підприємствами та споживачами. У B2C продавці продають продукцію безпосередньо споживачеві за допомогою програмного забезпечення кошика для покупок.

3. Споживач до споживача (Customer To Customer): Ця угода здійснюється між споживачами, де споживачі торгують продуктами один з одним в Інтернеті. Транзакції відбуваються у вигляді одного споживача або клієнта, котрий взаємодіє з іншими споживачами через Інтернет.

4. Від споживача до бізнесу (Customer To Business): Ця угода здійснюється між споживачами та підприємствами чи організаціями, де споживачі роблять продукти та послуги доступними в Інтернеті для придбання організаціями.

1.3 Черга повідомлень

Черга повідомлень - це форма одночасної комунікації між сервісами, що застосовується в багатьох архітектурах [3]. Повідомлення зберігаються в черзі, але не будуть оброблятися або видалятися. Кожне повідомлення звертається тільки до одного і тільки до одного користувача. Зареєстровані повідомлення можуть використовуватися для роз'єднань обробки процесів або для згладжування пікових навантажень.

У сучасній архітектурі додатки розділяють на невеликі незалежні елементи, які ще більш простіше розраховувати та обслуговувати. Черги повідомлень забезпечують для розповсюджених додатків можливість взаємодії та координації. Також черги повідомлень можуть значно спростити написання коду, що додається з роз'єднаними компонентами, а також підвищить їхню продуктивність, надійність та масштабіність.

Черга може використовувати численні джерела та одержувачі, але кожне повідомлення обробляється одним одержувачем тільки один раз. З цієї причини такий спосіб обміну повідомленнями часто називають взаємно-однозначним або прямою передачею повідомлень [4].

Отже, можна ствердити, що черга повідомлень дозволяє програмам спілкуватися, надсилаючи повідомлення один одному. Черга повідомлень забезпечує тимчасове зберігання повідомлень, коли цільова програма зайнята або не підключена. Далі сконцентруємо увагу, як саме використовувати чергу повідомлень та переваги використання черги повідомлень в архітектурі.

Черга - це ряд речей, які очікують на обробку, починаючи з початку рядка та обробляючи його в послідовному порядку.

Черга повідомлень - це набір повідомлень, надісланих між програмами, який включає послідовність робочих об'єктів або даних, які очікують на обробку.

Повідомлення - це дані, що передаються між відправником та одержувачем. Насправді, це байтовий масив з будь-якими даними всередині. Прикладом повідомлення може бути те, що говорить одній системі розпочати обробку завдання, воно може містити інформацію про готове завдання або бути простим повідомленням. Наприклад, електронна адреса користувача після процесу реєстрації для подальшої обробки даних.

Розберемо, як працює черга повідомлень за зразком зображення (Рис.2).



Рисунок 1.2 – Робота черги повідомлень

Уявимо, що в логічній частині веб-сайту виконується код, який обробляє електронну пошту користувача та додає будь-які додаткові дані. Наприклад, розраховує, які є поштові сервіси поруч з адресою, де проживає користувач. Отже, представимо, що ми зареєструвались в інтернет-магазині,

використовуючи поштову адресу *'RegisteredCustomer@mail.shop'*. Одразу ж формується повідомлення (*Message* на *Рис.2*), яке містить електронну пошту користувача *'RegisteredCustomer@mail.shop'*. Так, як черга повідомлень (*Message Queue* на *Рис.2*) не мала повідомлень, тому щойно надіслане повідомлення отримає першу позицію в черзі (*Message#1* на *Рис.2*). Через секунду, після реєстрації першого користувача, процес реєстрації закінчує другий користувач з поштовою адресою *'RegisteredCustomer2@mail.shop'*. Для нього так само формується повідомлення, яке містить всередині дані про поштову адресу користувача. Так, як черга повідомлень вже мала одне повідомлення, то наступне повідомлення з користувачем, який має поштову адресу *'RegisteredCustomer2@mail.shop'*, отримає другу позицію в черзі (*Message#2* на *Рис.2*). Наступний крок – відправлення повідомлень на процес обробки (*Enqueue* на *Рис.2*). Так, як повідомлення#1 потрапило в чергу першим, то на обробку воно також буде відправлено першим. Наступним буде відправлено повідомлення #2.

Черга повідомлень забезпечує одночасний зв'язок, який є системою, що розміщує повідомлення в черзі повідомлень і не вимагає негайної відповіді на продовження обробки. Більш яскравий приклад, коли треба надіслати електронного листа з підтвердженням профілю після реєстрації. Коли електронний лист надсилається, відправник продовжує обробляти інші речі, не потребуючи негайної відповіді одержувача. Такий спосіб обробки повідомлень відокремлює виробника від споживача, тому їм не потрібно одночасно взаємодіяти з чергою повідомлень.

1.4 Роз'єднання та масштабованість черги повідомлень

Роз'єднання - це процес відокремлення функцій, щоб вони були більш автономними [5]. Роз'єднана система досягається, коли дві або більше систем здатні спілкуватися без з'єднання. Системи можуть залишатися повністю автономними і не знати про інші функції. Роз'єднання часто є ознакою добре структурованої комп'ютерної системи, оскільки її легше обслуговувати.

Якщо одному процесу в роз'єднаній системі не вдається обробити повідомлення з черги, інші повідомлення все ще можна додавати до черги та обробляти, коли система відновиться. Також можна використовувати чергу повідомлень для затримки обробки - наприклад, виробник публікує повідомлення в черзі. В призначений час споживачі запускають і обробляють повідомлення в черзі. Повідомлення в черзі можна зберігати та пересилати, а повідомлення можна повторно доставити, поки воно не буде оброблено (Рис.3).



Рисунок 1.3 – Зберігання повідомлення в черзі

Замість створення однієї великої програми, корисно роз'єднати різні частини програми та асинхронно спілкуватися між ними за допомогою черг повідомлень. Це дозволяє різним частинам програми розвиватися незалежно, писатися різними мовами та/або обслуговуватися окремими командами розробників.

Черга повідомлень зберігатиме процеси у програмі окремими та незалежними один від одного. Першому процесу ніколи не потрібно буде

викликати інший процес, надсилати сповіщення в інший процес або стежити за перебігом інших процесів. Він може просто помістити повідомлення в чергу, а потім продовжити обробку. Інші процеси також можуть самостійно обробляти свою роботу, беручи повідомлення з черги, коли вони можуть їх обробляти. Такий спосіб обробки повідомлень створює систему, яку легко обслуговувати та масштабувати.

1.5 Програмне забезпечення Docker

У сучасному процесі розробки часто зустрічається проблема, пов'язана з підключенням додаткових сервісів або програм. Зазвичай це процес, який потребує багато часу на вивчення документації сервісу, правильного встановлення та індивідуального налаштування під власні потреби. Так, як під час підключення додаткового сервісу виконувались майже одні й ті самі дії, команда розробників вирішила автоматизувати даний процес та розробити програмне забезпечення, яке матиме містку бібліотеку з додатковими сервісами. Тому, у 2013 році було запущено проєкт з відкритим кодом під назвою **Docker**.

Docker - це програмне забезпечення для створення програм на основі контейнерів [6] - невеликих і легких середовищ виконання, які спільно використовують ядро операційної системи, але працюють ізольовано один від одного. Незважаючи на те, що контейнери використовувались у таких операційних системах, як Linux і Unix, саме проєкт Docker допоміг популяризувати технологію з контейнерами, надаючи можливість розробникам підготувати власне програмне середовище, яке можна збудувати один раз і запускати де завгодно та скільки завгодно.

Контейнери — це спосіб упакувати програму та всі її залежності в єдиний образ [7]. Даний образ запускається в ізольованому середовищі, що не

впливає на основну операційну систему. Контейнери дозволяють відокремити додаток від інфраструктури: розробникам не потрібно замислюватися, в якому оточенні працюватиме програма, чи будуть там потрібні налаштування та залежності. Вони просто створюють програму, упаковують всі залежності та налаштування в єдиний образ. Потім даний образ можна запускати на інших системах, не турбуючись, що програма не запускатиметься.

Контейнеризація схожа на віртуалізацію, але це не одне й те саме. Віртуалізація працює як окремий комп'ютер з віртуальним обладнанням та власною операційною системою [8]. При цьому всередині однієї ОС можна запустити іншу ОС. У разі контейнеризації віртуальне середовище запускається прямо з ядра основної операційної системи та не віртуалізує обладнання. Тобто, використовується фізичний процесор, оперативна пам'ять та місце на місці на жорсткому диску. Це означає, що контейнер може працювати тільки в тій самій операційній системі, що й основна система. Наприклад, ми маємо основну операційну систему Linux, тоді контейнер працюватиме у випадку, якщо він також матиме всередині операційну систему Linux. Приємною особливістю контейнерів є те, що вони споживають набагато менше ресурсів, так як контейнери не віртуалізують обладнання.

Які саме переваги надає використання **Docker** контейнерів [9]?

1. Docker вирішує проблеми залежностей та робочого оточення.

Контейнери дозволяють упаковати в єдиний образ програму та всі її залежності: бібліотеки, системні програми та файли налаштування. Це полегшує перенесення програми на іншу інфраструктуру.

Наприклад, розробники створюють програму в системі розробки, в якій все налаштовано та програма стабільно працює. Коли програма готова, її потрібно перенести в систему тестування, а наступним кроком - у продуктивне середовище. І якщо в системі не вистачатиме будь-якої залежності, то додаток не буде працювати. У цьому випадку програмістам доведеться відволіктися від

розробки та спільно з командою підтримки витратити час, щоб розібратися в ситуації.

Контейнери дозволяють уникнути даної проблеми, тому що вони містять все необхідне для запуску програми. Програмісти зможуть зосередитись на розробці, а не вирішенні інфраструктурних проблем.

2. Ізоляція та безпека

Контейнер – це набір процесів ізольованих від основної операційної системи. Програми працюють тільки всередині контейнерів і не мають доступу до основної операційної системи. Це підвищує безпеку програм, тому що вони не зможуть випадково чи навмисне нашкодити основній системі. Якщо програма в контейнері завершиться з помилкою або зависне, це не торкнеться основної операційної системи.

3. Прискореність, автоматизація розвертання додатків та масштабованість

Контейнери спрощують розгортання програм. У класичному підході для встановлення програми може знадобитися виконати кілька дій: виконати скрипт, змінити файли налаштувань тощо. У даному процесі не виключена можливість людської помилки: користувач запустить скрипт два рази, переплутає послідовність або щось не зрозуміє. Контейнери дозволяють повністю автоматизувати цей процес, тому що включають всі необхідні залежності і порядок виконання дій. Також контейнери спрощують розгортання на кількох серверах. У класичному підході для того, щоб розгорнути один і той же додаток на декількох машинах, потрібно буде повторювати одні й ті самі дії. Контейнери позбавляють від рутинної роботи і дозволяють автоматизувати розгортання.

4. Можливість розгорнути декілька контейнерів одночасно

Docker дозволяє розгортати та налаштовувати кілька контейнерів одночасно. Наприклад, для веб-програми потрібно розгорнути такий набір сервісів: Linux, база даних MySQL та PHP. Кожна з програм — це окремий контейнер. Але в даній ситуації нам потрібні саме всі контейнери разом, а не окремий додаток. Docker дозволяє розгорнути та налаштувати всі програми однією командою, як комплексний додаток та дозволяє користуватися будь-яким запущеним додатком в будь-який час.

Ознайомившись з перевагами Docker, не менш важливо розуміти, з яких компонентів складається додаток та як саме він працює. Тому, наступним кроком розглянемо компоненти, з яких складається Docker [10].

Docker daemon - це сервіс, через який здійснюється вся взаємодія з контейнерами: створення та видалення, запуск та зупинка. Цей сервіс працює у фоновому режимі та отримує команди від інтерфейсу командного рядка.

Docker client (клієнт) - це інтерфейс командного рядка для керування *docker daemon*. Використовується клієнтом, коли створюються та розгортаються контейнери, а клієнт надсилає ці запити в *docker daemon*.

Docker image (образ) - це незмінний файл (образ), з якого розгортаються контейнери. Програми упаковуються саме в образи, з яких потім створюються контейнери.

Наведемо аналогію з прикладу установки операційної системи. У дистрибутиві (образі) операційної системи є все, що потрібне для її установки. Але даний образ не можна запустити, спочатку його потрібно «розгорнути» в готову операційну систему. Саме дистрибутив для встановлення операційної системи і є **Docker image**, а встановлена і працююча операційна система це

Docker container. Але контейнери зазвичай розгортаються однією командою - це набагато простіше і швидше, ніж встановлення операційної системи.

Volume (виділений об'єм або місце) - це емуляція файлової системи для здійснення операцій читання та запису. Вона створюється автоматично з контейнером, оскільки деякі програми здійснюють збереження даних.

Docker Registry (реєстр) - зарезервований сервер, який використовується для зберігання образів docker.

Docker Hub або сховище даних – репозиторій, призначений для зберігання образів із різним програмним забезпеченням.

Docker Host - машинне середовище для запуску контейнерів із програмним забезпеченням.

Docker Networks - використовуються для організації мережевого інтерфесу між програмами, розгорнутими в контейнерах.

Дізнавшись, з яких саме компонентів складається Docker, наступним кроком розберемо, як саме працює даний додаток [11]. Робота Docker базується на принципах клієнт-серверної архітектури, яка базується на взаємодії клієнта з веб-сервером (хостом). Клієнт-сервер надсилає запити отримання даних, а веб-сервер їх надає. Детальніше зі схемою можна ознайомитись на *Рис.4*.

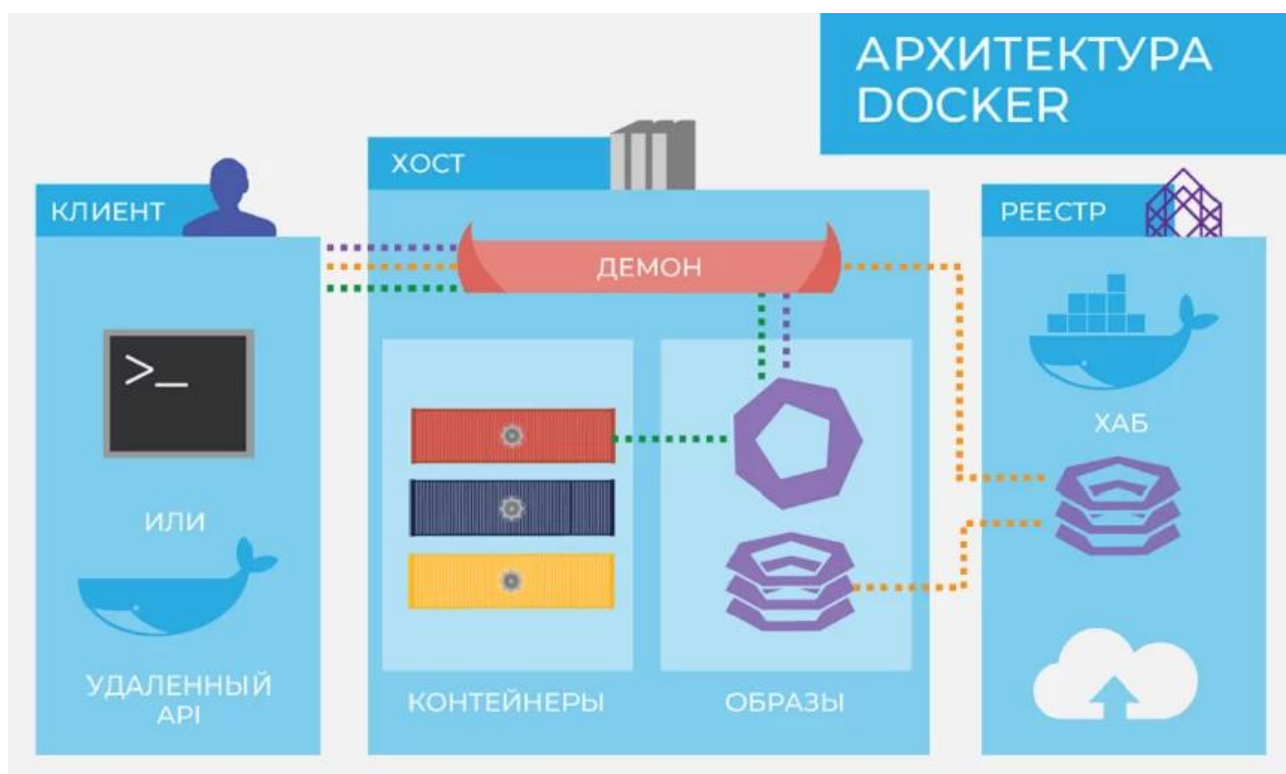


Рисунок 1.4 – Схема роботи додатку Docker

Розберемо більш детально, як працюють контейнери. Кожен контейнер будується на основі образів (*image*) Docker. Контейнери запускаються безпосередньо з ядра Linux. Завдяки цьому вони споживають набагато менше ресурсів, ніж при віртуалізації.

Ізоляція робочого середовища здійснюється за допомогою технології *namespace*. Для кожного ізольованого простору (контейнера) створюється унікальний простір імен, який і забезпечує доступ до нього. Будь-який процес, що виконується всередині контейнера, обмежується *namespace*. В операційній системі Linux за допомогою Docker Engine використовується трохи інша технологія – *контрольні групи (cgroups)*. При цьому програма обмежується деяким набором ресурсів. Контрольні групи здійснюють обмін доступних апаратних ресурсів із контейнерами, куди додатково встановлюються необхідні обмеження (використання пам'яті, прав доступу до іншого ресурсу тощо).

Двигун Docker об'єднує простір імен (*namespace*), контрольні групи (*cgroups*) та файлову систему у формат контейнера.

Наступним кроком розберемо, що відбувається під час запуску контейнеру.

- 1) Відбувається запуск зображення (Docker-image). Docker перевіряє існування образу. Якщо образ існує локально, Docker використовує його для нового контейнера. За його відсутності виконується завантаження з Docker Hub.
- 2) Створення контейнера із зображення.
- 3) Розмітка файлової системи та додавання шару для запису.
- 4) Створення інтерфейсу.
- 5) Пошук та присвоєння IP-адреси.
- 6) Запуск процесів та виконання інструкцій всередині контейнера.
- 7) Захоплення введення/виведення програми.

Після використання Docker на практиці, описані процеси та кроки стають більш зрозумілими. Тому, процес запуску контейнеру буде детально розглянуто у практичній частині.

Docker є важливим інструментом для кожного сучасного розробника як основа апаратної віртуалізації додатків [12]. Дана технологія володіє широким функціоналом та можливостями для контролю процесів. Докер дозволяє не тільки розгортати контейнери, а й оперативно масштабувати їх екземпляри, працювати з багатоконтейнерними програмами (Docker Compose), а також об'єднувати кілька Docker-хостів у єдиний кластер. Docker характеризується досить простим синтаксисом. Тому він досить простий у освоєнні як досвідчених для IT-фахівців, так і для новачків. Програмне забезпечення сумісне з усіма версіями операційних систем Linux і Windows, тому сфера застосування Docker практично не обмежена.

Так, як для подальшої розробки було обрано використання контейнерів Docker, для спрощення процесу можливо скористатися менеджером контейнерів, який автоматично налаштовує всі необхідні процеси під час розгортання локального середовища. Існує багато різних варіантів для налаштування локального середовища розробки. Звичайно ж, можна

скористатися власним рішенням на основі Docker. Але, спираючись на досвід, даний процес завжди займає деякий час, щоб налаштуватися правильно. А після налаштування завжди є щось, що можна покращити, доробити або виконати додаткове налаштування. Також досить часто виникає ситуація, коли потрібно запустити декілька версій Magento паралельно. Даний критерій виключив досить багато підходів, які включають синхронізацію, оскільки початковий час синхронізації займає деякий час, і параметри, які краще працюють для окремих екземплярів. Тому, потрібна була додаткова програма, яка налаштовує комунікацію між контейнерами та коректно налаштовує дані між ними. Найпопулярніший додаток, який спрощує сучасний процес розробки, називається Warden. Це зручний менеджер контейнерів, який має прозору офіційну документацію та детальні інструкції, як налаштовувати контейнери для кожного середовища. Хоч у нашому випадку не знадобиться організовувати складний веб-додаток, перевагу буде віддано саме даному додатку через можливість одразу налаштувати веб-додаток відповідно до власних потреб.

1.6 Брокер повідомлень Rabbit MQ

Rabbit MQ - це програмне забезпечення для черги повідомлень, також відоме як брокер повідомлень або менеджер черг. Простіше кажучи - це програмне забезпечення, де визначаються черги, до якого підключаються програми для передачі повідомлення або повідомлень.

Брокер повідомлень діє як посередник для різних послуг (наприклад, веб-додатки) [13]. Вони можуть бути використані для скорочення навантажень та термінів доставки серверів веб-додатків шляхом делегування завдань, які зазвичай забирають багато часу чи ресурсів третій стороні, яка не має іншої роботи.

Розглянемо сценарій, коли веб-додаток дозволяє користувачам завантажувати інформацію на веб-сайт. Сайт оброблятиме цю інформацію, генеруватиме PDF файл та надішле згенерований файл користувачеві по електронній пошті. Обробка інформації, створення PDF файлу та надсилання електронного листа у цьому прикладі займе кілька секунд. Це одна з причин, чому черга повідомлень буде використана для виконання завдання.

Коли користувач вводить інформацію у веб-інтерфейс, веб-додаток створює повідомлення "Обробка PDF", яке включає всю важливу інформацію, необхідну користувачеві, у повідомлення, і розміщує його у черзі, визначеній у RabbitMQ (Рис.5).



Рисунок 1.5 – Процес зберігання повідомлення в черзі за допомогою брокера Rabbit MQ

Чому та коли потрібно використовувати брокер повідомлень Rabbit MQ? Черга повідомлень дозволяє веб-серверам швидко реагувати на запити замість того, щоб змушувати сервер виконувати важкі ресурсні процедури на місці, що може затримати час відповіді. Черга повідомлень також корисна у випадку, коли потрібно розповсюдити повідомлення кільком споживачам або збалансувати навантаження між працівниками.

Споживач знімає повідомлення з черги і починає обробку PDF. Водночас джерело повідомлень постачає у чергу нові повідомлення. Споживач може перебувати на зовсім іншому сервері, ніж виробник, або вони можуть знаходитися на одному сервері. Запит може бути створений однією мовою

програмування та оброблений іншою мовою програмування. Справа в тому, що дві програми будуть спілкуватися лише через повідомлення, які вони надсилають один одному, а це означає, що відправник і одержувач мають зв'язок між собою лише за допомогою повідомлення.

Розглянемо процес відправлення та генерації PDF файлу, використовуючи Rabbit MQ (Рис.1.6).

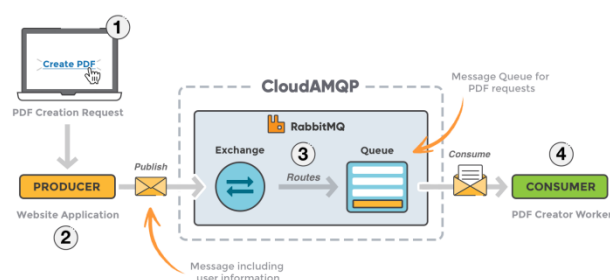


Рисунок 1.6 – Процес обробки запиту для створення PDF файлу через повідомлення

1. Користувач надсилає веб-програмі запит на створення PDF;
2. Веб-додаток (Producer) надсилає повідомлення до RabbitMQ, яке містить дані із запиту, такі як ім'я та електронна адреса користувача;
3. Менеджер повідомлень (Exchange) приймає повідомлення від виробника (Producer) і направляє їх для виправлення черг повідомлень для створення PDF;
4. Працівник з обробки PDF (Consumer) отримує повідомлення про завдання та починає обробку PDF.

2. РЕАЛІЗАЦІЯ СЕРЕДОВИЩА ТА МОДУЛЯ

2.1 Аналіз процесу реєстрації без використання черги повідомлень

Перед тим, як створювати підключення зі сторони Magento 2 до брокеру Rabbit MQ та змінювати логіку основних процесів, використовуючи чергу повідомлень, спочатку проаналізуємо, наскільки швидко виконуються процеси та за який проміжок часу можна отримати результат процесу. За основу було взято процес реєстрації нового користувача. Розберемо покроково, як саме відбувається процес реєстрації. В першу чергу, розглянемо, як саме виглядає стандартна форма реєстрації (Рис.2.1). Під час процесу розробки вона не буде змінюватись.

Аналізувати час завантаження та проміжок часу, за який було отримано відповідь, будемо за допомогою інструментів розробника в браузері Google Chrome (Рис.2.1).

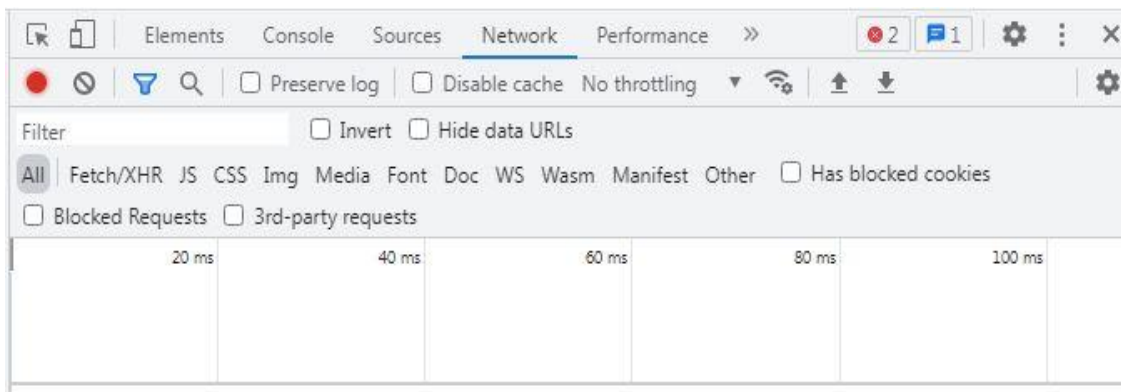


Рисунок 2.1 – Інструменти розробника в браузері Google Chrome

Create New Customer Account

Personal Information

First Name *

Last Name *

Email *

☐ Sign Up for Newsletter

Date of Birth

Sign-in Information

Password *

Confirm Password *

☒ Remember Me [What's this?](#)

Create an Account

Рисунок 2.2 – Стандартний вигляд форми для реєстрації нового користувача в Magento 2

Розпочнемо тестовий процес та заповнимо форму для реєстрації даними. Після того, як було натиснуто кнопку завершення реєстрації (*Create an Account*), відкриємо інструменти розробника та дізнаємось, скільки часу знадобилось на виконання запиту щодо реєстрації нового користувача та як швидко було отримано відповідь щодо закінчення процесу реєстрації нового користувача (Рис. 2.3).

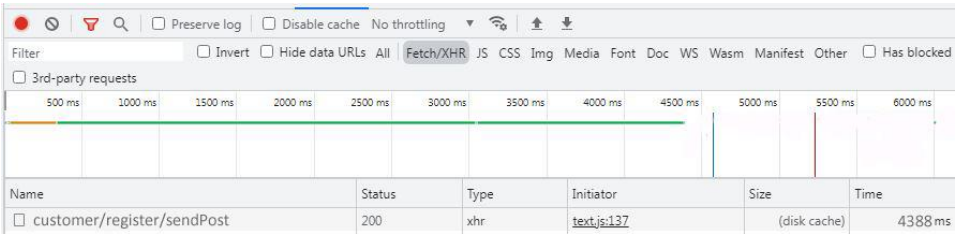


Рисунок 2.3 – Швидкість виконання процесу реєстрації без використання черги повідомлень

Можемо спостерігати, що процес обробки даних для реєстрації та час повернення результату загалом дорівнює **4,3** секунди. Цей результат демонструє нам, що процес реєстрації є ресурсномістким та витрачає досить багато часу. Тож, основне завдання відносно даного процесу – оптимізувати логічні процеси, які відповідають за процес реєстрації та зменшити час обробки даних до **2** секунд.

Загалом, процес реєстрації нового користувача має досить складну логіку та виконує багато операцій. Наприклад, спочатку треба перевірити, чи правильні значення були внесені в поля форми реєстрації та чи є вже існуючий користувач з такою електронною поштою. Наступним кроком потрібно внести новий запис з даними користувача до бази даних. Далі, слідкуючи специфікації Magento 2, треба внести додаткові дані та атрибути щодо користувача до додаткових таблиць в базі даних. А фінальним кроком є відправка електронного листа з підтвердженням нового зареєстрованого акаунту.

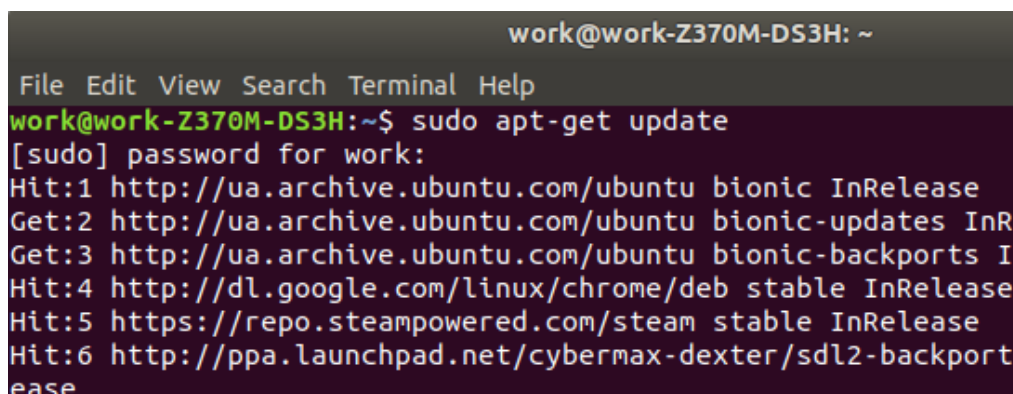
Також варто пам'ятати, що існує ще багато додаткових процесів, які також потребують часу та ресурсів на завантаження, а це може суттєво вплинути на сумарний час завантаження веб-сторінки. Тож, спробуємо розробити модуль для підключення черги повідомлень та перенести основні процеси Magento 2 на обробку з використанням черги повідомлень.

2.2 Створення Docker контейнера для сервісу Rabbit MQ

Перед тим, як розпочати процес розробки модуля, необхідно підготувати та налаштувати середовище з менеджером черги повідомлень. Так, як у нашому випадку було обрано такі технології, як Docker для контейнерів та менеджер контейнерів Warden, то спочатку обрані технології треба встановити. В першу чергу потрібно встановити додаток Docker. Також варто врахувати особливість,

що практична робота виконується на операційній системі Linux. Тому, весь процес встановлення та налаштування буде виконано через консоль.

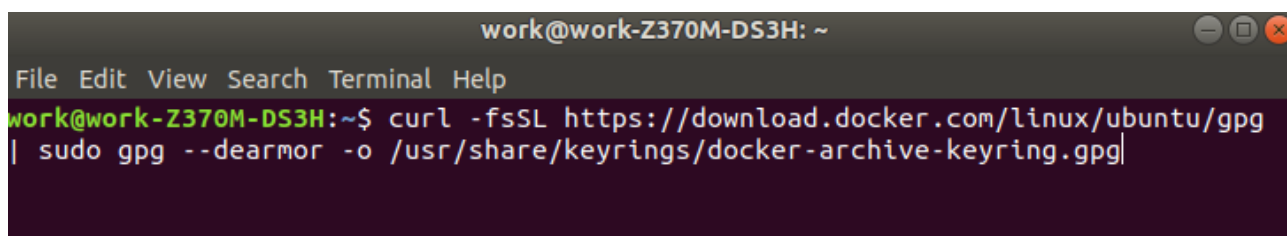
Перед тим, як встановлювати будь-яку програму, спочатку треба виконати порграму, яка перевіряє та оновлює важливий функціонал та основні додатки операційної системи (Рис. 2.4)



```
work@work-Z370M-DS3H: ~  
File Edit View Search Terminal Help  
work@work-Z370M-DS3H:~$ sudo apt-get update  
[sudo] password for work:  
Hit:1 http://ua.archive.ubuntu.com/ubuntu bionic InRelease  
Get:2 http://ua.archive.ubuntu.com/ubuntu bionic-updates InRelease  
Get:3 http://ua.archive.ubuntu.com/ubuntu bionic-backports InRelease  
Hit:4 http://dl.google.com/linux/chrome/deb stable InRelease  
Hit:5 https://repo.steampowered.com/steam stable InRelease  
Hit:6 http://ppa.launchpad.net/cybermax-dexter/sdl2-backport/ubuntu bionic InRelease  
Fetched 10.8 kB in 1s (10.8 kB/s)  
Reading package lists... Done
```

Рисунок 2.4 – Перевірка оновлень операційної системи Linux

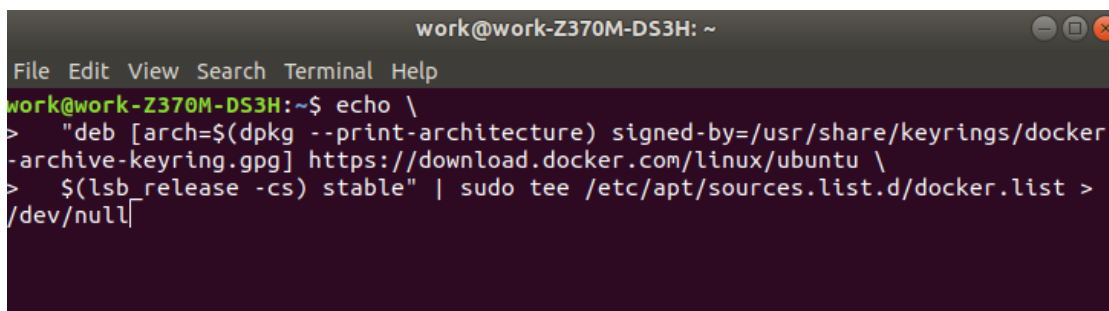
Після перевірки та встановлення важливих оновлень треба сповістити Linux про те, що нам потрібно додати сторонній репозиторій. Для цього додамо власний ключ для комунікації з веб-сервісами та джерелами Docker (Рис. 1.11).



```
work@work-Z370M-DS3H: ~  
File Edit View Search Terminal Help  
work@work-Z370M-DS3H:~$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg
```

Рисунок 2.5 – Встановлення ключа для комунікації з джерелами Docker

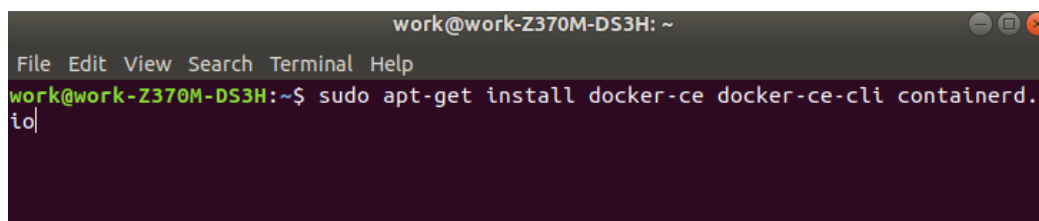
Наступний крок — виконати консольну команду, яка вказує, що нам потрібно завантажити необхідні додатки з джерел Docker (Рис.2.6).



```
work@work-Z370M-DS3H: ~  
File Edit View Search Terminal Help  
work@work-Z370M-DS3H:~$ echo \  
> "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker  
-archive-keyring.gpg] https://download.docker.com/linux/ubuntu \  
> $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list >  
/dev/null
```

Рисунок 2.6 – Отримання необхідних додатків з джерел Docker

Після того, як необхідні додатки було завантажено, їх необхідно встановити в операційній системі. Для цього виконаємо спеціальну команду для встановлення додатків, вказавши, які саме додатки встановити (Рис. 1.13).

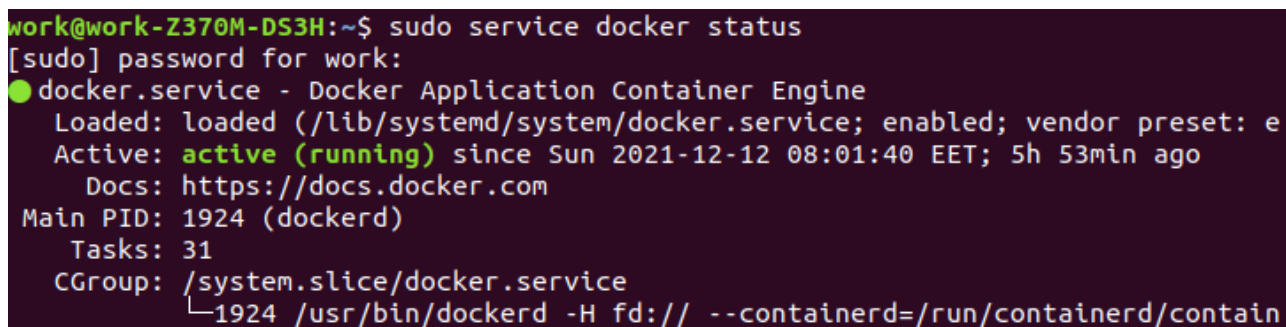


```
work@work-Z370M-DS3H: ~  
File Edit View Search Terminal Help  
work@work-Z370M-DS3H:~$ sudo apt-get install docker-ce docker-ce-cli containerd.  
io
```

Рисунок 2.7 – Встановлення додатків Docker в операційну систему Linux

Щоб пересвідчитись у тому, що додатки Docker було встановлено вірно, виконаємо команду, яка переглядає статус сервісів. Так, як нам потрібно переглянути статус конкретного сервісу, то вкажемо його, як додатковий параметр. Результат роботи команди можна переглянути на Рис.2.8. Статус відображається у рядку *Active*. В нашому випадку можемо спостерігати, що

статус зафарбовано у зелений колір, є активним та коректно виконується на даним момент. Якщо б була помилка, то статус мав би червоний колір.



```
work@work-Z370M-DS3H:~$ sudo service docker status
[sudo] password for work:
● docker.service - Docker Application Container Engine
   Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: e
   Active: active (running) since Sun 2021-12-12 08:01:40 EET; 5h 53min ago
     Docs: https://docs.docker.com
    Main PID: 1924 (dockerd)
      Tasks: 31
     CGroup: /system.slice/docker.service
             └─1924 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/contain
```

Рисунок 2.8 – Статус сервісу Docker після встановлення

Так, як необхідні сервіси Docker були успішно встановлені, наступним кроком потрібно встановити менеджер контейнерів Warden. Для цього потрібно також виконати консольну команду. Але, спираючись на те, що у Warden чудова офіційна документація, тому консольна команда вже описана та заготовлена в самій інструкції для встановлення:

```
sudo mkdir /opt/warden
sudo chown $(whoami) /opt/warden
git clone -b master https://github.com/davidalger/warden.git /opt/warden
echo 'export PATH="/opt/warden/bin:$PATH"' >> ~/.bashrc
PATH="/opt/warden/bin:$PATH"
warden svc up
```

Описана вище команда створює папку в необхідному місці ядра Linux, завантажує необхідну програму та встановлює її одразу ж після завантаження.

Програма Warden має трохи іншу логіку роботи, тому, щоб перевірити, чи успішно було встановлено програму, замість перегляду статусу у консолі, спробуємо запустити її. У випадку будь-яких помилок, на екран буде виведено повідомлення про те, що якусь дію було виконано некоректно. На рис.2.9

можемо спостерігати, що програму було успішно запущено, а це означає, що програму було встановлено вірно.

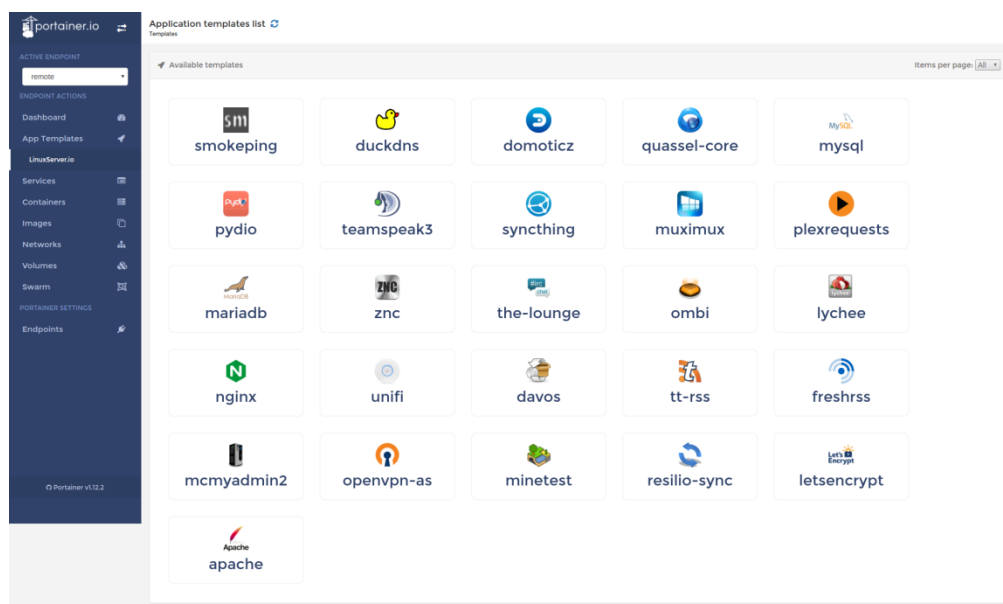


Рисунок 2.9 – Інтерфейс програми Warden

Маючи встановлені всі необхідні сервіси, можемо розпочати налаштування контейнеру з сервісом Rabbit MQ. Для цього потрібно завантажити образ (*Docker Image*) з інструкціями для сервісу Rabbit MQ. Для цього виконаємо у консолі команду для завантаження образу з джерел Docker (Рис.2.10).

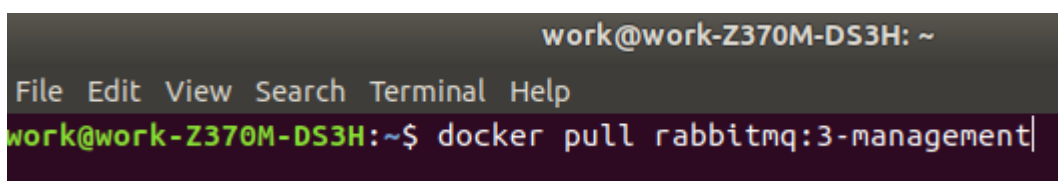


Рисунок 2.10 – Завантаження образу Rabbit MQ з джерел Docker

Щоб запустити контейнер з сервісом Rabbit MQ, відкриємо програму Warden та переглянемо наявність образу в списку всіх встановлених образів (рис. 2.11).

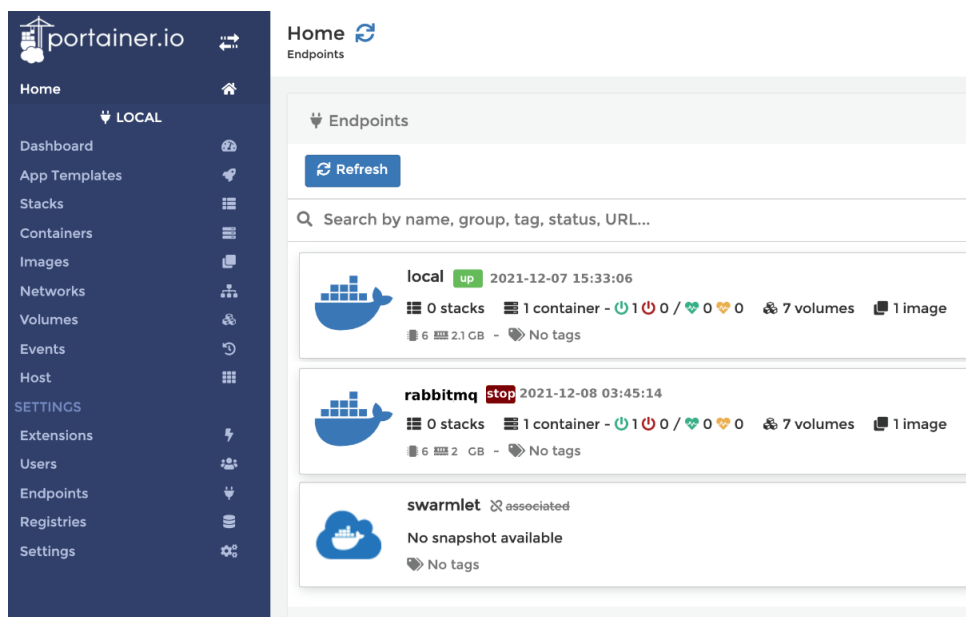


Рисунок 2.11 – Список образів в програмі Warden

Спостерігаємо, що з'явився контейнер під назвою **rabbitmq**, але на даний момент він виключений, про що свідчить напис **stop** на червоному фоні. Відповідно, для того, щоб перевірити, чи працює сервіс, спочатку треба запустити контейнер. Для цього достатньо натиснути 2 рази на потрібний елемент у списку та контейнер автоматично буде запущено. У цьому і є зручність менеджера контейнерів Warden. Тож, натиснемо 2 рази на елемент під назвою **rabbitmq** та перевіримо, чи було запущено потрібний сервіс (рис. 2.12).

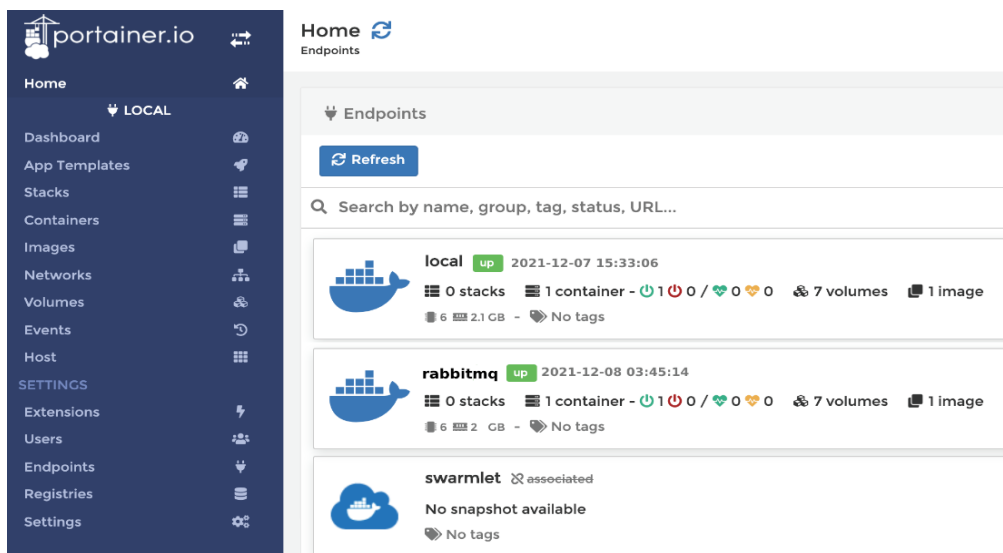


Рисунок 2.12 – Список сервісів після запуску контейнера

Спостерігаємо, що поруч з назвою **rabbitmq** з'явився напис **up** на зеленому фоні. Отже, контейнер було запущено без помилок. А це означає, що можна відкрити сервіс Rabbit MQ та перевірити, чи все функціонує вірно (рис. 2.13).

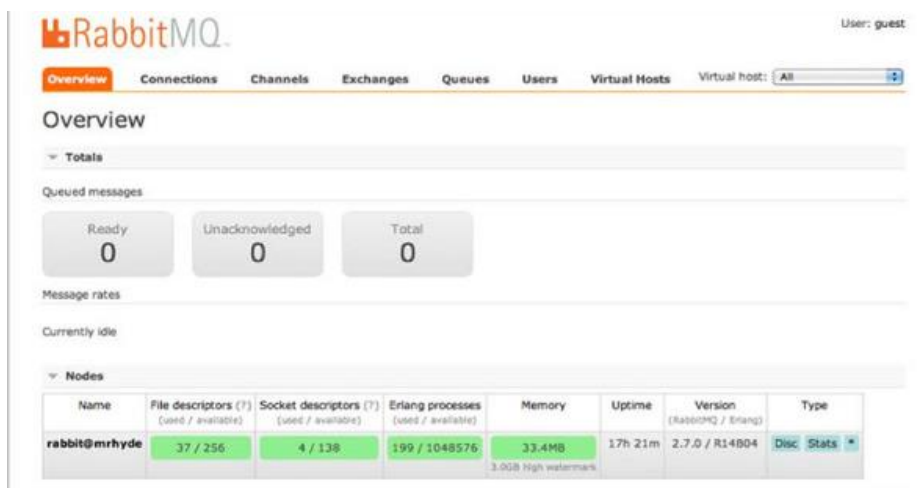


Рисунок 2.13 – Сервіс Rabbit MQ, запущений за допомогою контейнеру

Так, як середовище з брокером повідомлень було успішно підготовлено, можна приступати до наступного кроку — створення та розробка модулю для Magento 2.

2.3 Створення модулю для Magento 2

Для того, щоб створити підключення зі сторони Magento 2 до брокеру Rabbit MQ, спочатку треба створити новий модуль для Magento 2. Слідкуючи правилам розробки [14], треба створити два основні файли — *module.xml* (Рис.20) та *registration.php* (Рис.2.14). При створенні даних файлів, середовище Magento розуміє, що в директорії проєкту було створено новий модуль.

```

1  <?xml version="1.0"?>
2  <!--
3  /**
4   * @author Artem Lytvynenko
5   * @copyright 2021 Artem Lytvynenko. All Rights Reserved.
6   */
7  -->
8  <config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
9         xsi:noNamespaceSchemaLocation="urn:magento:framework:Module/etc/module.xsd">
10     <module name="Diplom_Alytvynenko">
11     </module>
12 </config>

```

Рисунок 2.14 — Контент файлу module.xml

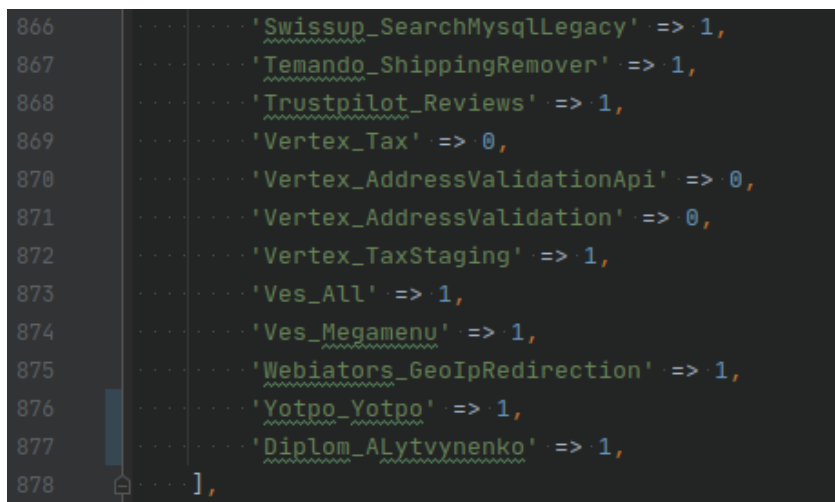
```

1  <?php
2  /**
3   * @author Artem Lytvynenko
4   * @copyright 2021 Artem Lytvynenko. All Rights Reserved.
5   */
6
7   use Magento\Framework\Component\ComponentRegistrar;
8
9   ComponentRegistrar::register(
10       ComponentRegistrar::MODULE,
11       'Diplom_Alytvynenko',
12       __DIR__
13   );

```

Рисунок 2.15 — Контент файлу registration.xml

Після створення даних файлів, середовище Magento 2 згенерує новий список модулів та ми зможемо побачити новий модуль у списку доступних модулів, який генерується в файлі *config.php* (Рис.2.16).



```

866     'Swissup_SearchMysqlLegacy' => 1,
867     'Temando_ShippingRemover' => 1,
868     'Trustpilot_Reviews' => 1,
869     'Vertex_Tax' => 0,
870     'Vertex_AddressValidationApi' => 0,
871     'Vertex_AddressValidation' => 0,
872     'Vertex_TaxStaging' => 1,
873     'Ves_All' => 1,
874     'Ves_Megamenu' => 1,
875     'Webiators_GeoIpRedirection' => 1,
876     'Yotpo_Yotpo' => 1,
877     'Diplom_ALytvynenko' => 1,
878 ]

```

Рисунок 2.16 – Новий модуль *Diplom_ALytvynenko* у згенерованому списку модулів

Наступним кроком потрібно створити та налаштувати файли, які будуть відповідати за підключення брокера повідомлень Rabbit MQ та за які процеси буде відповідати черга повідомлень.

Щоб створити нове підключення в Magento 2, треба створити файл *ModelConfig.php* та реалізувати функцію *getConnection()*. Реалізація даної функції виглядатиме так:

```

/**
 * Return Rabbit MQ client
 * @return RabbitMqContext
 * @throws FileSystemException
 * @throws RuntimeException
 */
public function getConnection()

```

```

{
    if (!isset($this->connection)) {
        $this->logger->debug(
            'Setting up connection',
            [
                'access_key' => $this->getValue(Config::ACCESS_KEY)
            ]
        );

        $config = (new RabbitMqClient([
            'region' => $this->getValue(Config::REGION),
            'version' => $this->getValue(Config::VERSION),
            'credentials' => [
                'key' => $this->getValue(Config::ACCESS_KEY),
                'secret' => $this->getValue(Config::SECRET_KEY),
            ],
            'endpoint' => $this->getValue(Config::ENDPOINT)
        ]));

        $this->connection = (new RabbitMqConnectionFactory($config))-
            >createContext();
    }

    return $this->connection;
}

```

Наступний крок — описати параметри для підключення до самого сервісу Rabbit MQ. Для цього потрібно створити новий масив даних у файлі *env.php*. У результаті отримаємо такий масив:

```

'queue' => [
    'amqp' => [
        'host' => 'localhost',

```

```

        'port' => '5672',
        'user' => 'guest',
        'password' => 'guest',
        'virtualhost' => '/',
    ],
]

```

За допомогою даних файлів, середовище Magento отримає інформацію щодо підключення нового сервісу. В нашому випадку, цим сервісом є брокер повідомлень Rabbit MQ. Щоб досвідчитись у тому, що підключення було виконано вірно, потрібно створити чергу для процесу реєстрації користувачів та перевірити наявність даної черги у веб-додатку Rabbit MQ.

2.4 Створення нової черги повідомлень

Щоб створити нову чергу повідомлень, спочатку треба реалізувати нові файли в модулі, які будуть відповідати за чергу та які інструкції повинні бути виконані під час обробки даної черги [15]. Тому, потрібно реалізувати такі файли:

communication.xml – створення загальної конфігурації для взаємодії модуля та черги повідомлень;

queue_topology.xml - опис характеристики та ідентифікація черги на стороні сервісу Rabbit MQ;

queue_publisher.xml – в даному файлі описується, до якого ідентифікатора потрібно звертатись зі сторони Magento 2 до сервісу Rabbit MQ;

queue_consumer.xml – вказується, який файл з реалізованою логікою буде відповідати за обробку даних, що були отримані з черги.

Маючи список необхідних файлів та опис кожного з них, розпочнемо реалізацію даних файлів. Почнемо з самого головного файлу

communication.xml. Опишемо нову чергу, яка буде відповідати за роботу з даними користувача:

```
<?xml version="1.0"?>
<config      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:Communication/etc/co
mmunication.xsd">
    <topic name="customer.after.register.handler" request="int"/>
</config>
```

Сама черга описується у вузлі *<topic>* та має параметри *name* – назва черги та *request* – який тип даних буде передано за допомогою черги. В нашому випадку буде передано ідентифікатор запису користувача з бази даних, який має числовий тип. Наступний файл, який буде реалізовано – *queue_topology.xml* та матиме такий контент:

```
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework-message-
queue/etc/topology.xsd">
    <exchange name="diplom.customer.register" type="topic"
connection="rabbitmq">
        <binding id="DiplomCustomerRegister"
            topic="customer.after.register.handler"
            destinationType="queue"
            destination="customer_after_register_handler"/>
    </exchange>
</config>
```

В даному файлі спостерігаємо два вузли – *exchange* та *binding* [16]. У вузлі *exchange* описується ім'я вхідної точки на стороні сервісу Rabbit MQ, яка буде приймати повідомлення, та буде розміщувати щойно отримані

повідомлення у черзі. Атрибут *name* відповідає за назву вхідної точки, *type* – вказує на тип вхідної точки, а *connection* зберігає в собі назву підключення до сервісу. Так, як ми створили підключення з назвою *rabbitmq*, то саме цю назву було вказано в атрибуті *connection* і саме даний атрибут буде вказувати, в якому сервісі потрібно створити вхідну точку.

У наступному вузлі, який має назву *binding*, розміщується інформація щодо зв'язку між чергою повідомлень та її вхідною точкою. Атрибут *id* містить в собі лише ідентифікатор зв'язку та не має надзвичайної важливості. Атрибут *topic* вказує на маршрут, за яким буде відправлено повідомлення до черги, а атрибути *destination* та *destinationType* вказують на те, що повідомлення буде розміщено саме у черзі, яка має певне ім'я.

Наступним файлом для реалізації буде *queue_consumer.xml* і в ньому будуть розміщено такі налаштування:

```
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework-message-
queue:etc/consumer.xsd">
    <consumer name="CustomerRegisterConsumer"
queue="customer_after_register_handler"
connection="rabbitmq"
handler="Diplom\ALytvynenko\Model\Customer\RegisterCustomerConsumer
::processMessage"/>
</config>
```

Налаштування були реалізовані у вузлі *consumer*, який має такі атрибути:

queue – назва черги, яку буде «прослуховувати» споживач та з якої будуть отримувані повідомлення;

name – назва самого споживача;

connection – назва підключення до сервісу

handler – назва класу з логікою, який буде обробляти дані з повідомлення;

Завершуючий крок – реалізація файлу queue_publisher.xml, в якому описується видавець, прив'язка до маршруту та обмінник (exchange), який буде користуватись видавцем. Даний файл має такий контент:

```
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework-message-
queue:etc/publisher.xsd">
    <publisher topic="customer.after.register.handler">
        <connection name="rabbitmq" exchange="diplom.customer.register" />
    </publisher>
</config>
```

Опис інструкцій відбувається у вузлі publisher. Атрибут topic відповідає за назву маршруту, який було описано у файлі queue_topology.xml. Всередині реалізовано інструкцію щодо підключення за допомогою вузла connection, в якому атрибут name відповідає на назву підключення до сервісу, а exchange відповідає на заву вхідної точки на стороні сервісу Rabbit MQ.

Після реалізації потрібних файлів, ми можемо перевірити наявність черги у веб-додатку Rabbit MQ з ім'ям черги, яке було описано у файлах (рис.2.17).

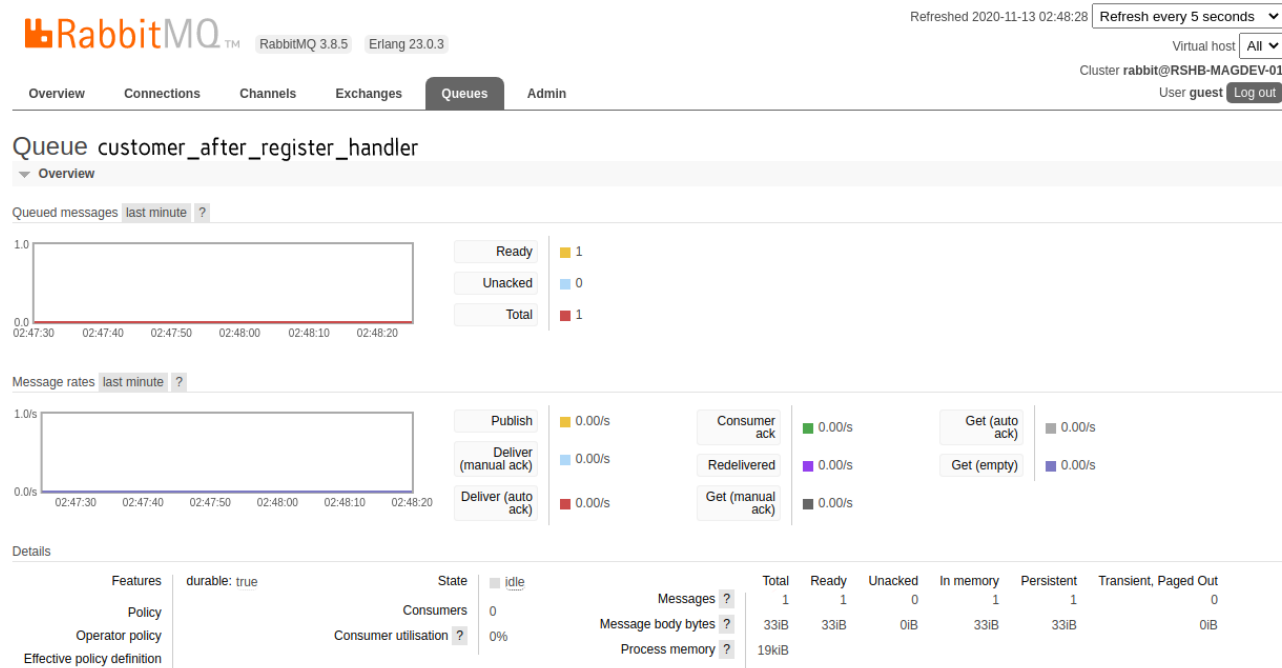


Рисунок 2.17 – Наявність створеної черги у веб-додатку Rabbit MQ

Тепер потрібно акцентувати увагу на створенні логіки, яка буде відправляти ідентифікатор користувача до черги повідомлень а також реалізувати логіку, яка буде прослуховувати чергу повідомлень та обробляти дані. Для цього потрібно реалізувати файли, які були описані в атрибуті handler файлу `queue_consumer` та окремий файл, який буде відправляти дані лише у чергу з потрібною назвою. Перший файл може бути розміщено у будь якому місці та може мати будь яку назву. Тому, файл для відправки даних до черги буде мати назву `RegisterCustomerPublisher.php` та в ньому буде реалізована така логіка:

```
<?php
```

```
namespace Diplom\ALytvynenko\Model\Customer;
```

```
use Magento\Framework\MessageQueue\PublisherInterface;
```

```
/**
```

```
* Class RegisterCustomerPublisher
```

```

*/
class RegisterCustomerPublisher
{
    /**
     * RabbitMQ Topic name
     */
    const TOPIC_NAME = 'customer.after.register.handler';

    /**
     * @var PublisherInterface
     */
    private $publisher;

    /**
     * RegisterCustomerPublisher constructor
     *
     * @param PublisherInterface $publisher
     */
    public function __construct(
        PublisherInterface $publisher
    ){
        $this->publisher = $publisher;
    }

    /**
     * Add message to queue
     *
     * @param int $customerId
     */
    public function execute(int $customerId)

```

```

    {
        $this->publisher->publish(self::TOPIC_NAME, $customerId);
    }
}

```

За відправку даних до черги в даному файлі відповідає функція *execute*, в яку передається ідентифікатор користувача, як параметр. В тілі самої функції відбувається виклик функції *publish*, яка відправляє до черги повідомлень потрібні дані, використовуючи описане підключення у файлах *xml*, які були реалізовані вище.

Наступний крок – реалізація файлу для прослуховування та обробки повідомлень з черги. На відміну від файлів, які відправляють дані до черги, файл-обробник повинен бути єдиним та мати таку саму назву, яку було описано в атрибуті *handler* файлу *queue_consumer.xml*. В нашому випадку файл повинен мати повну назву, відповідно до значення, яке було розміщено в атрибуті - *Diplom\ALytvynenko\Model\Customer\ RegisterCustomerConsumer.php*. В даному файлі повинна бути реалізована функція *processMessage*, яка приймає ідентифікатор користувача, як параметр функції. Тому, контент файлу виглядатиме таким чином:

```

<?php

namespace Diplom\ALytvynenko\Model\Customer;

/**
 * Class RegisterCustomerConsumer
 */
class RegisterCustomerConsumer
{
    /**
     * Process message from queue
     *

```

```

* @param int $customerId */
public function processMessage(int $customerId)
{
    // Handle with default magento 2 logic
}
}

```

Дані, котрі були відправлені до черги повідомлень, в кінці шляху, будуть відправлені на обробку до щойно створеного файлу *RegisterCustomerConsumer.php*. За логічну частину відповідає функція *processMessage*. В тілі функції описуються стандартні процедури середовища Magento, код яких має декілька тисяч строк. Тому, в даному випадку, заради зразка, в тілі функції було залишено коментар.

Маючи всі необхідні файли, які відповідають за відправлення та обробку повідомлення до черги, можемо акцентувати увагу на наступному кроці – тестуванні процесу реєстрації та дізнаємось, чи дійсно підключення черги повідомлень оптимізувала ресурсомісткий процес реєстрації нового користувача та чи змінився час виконання процесу.

3. ДОСЛІДЖЕННЯ РЕАЛІЗОВАНОГО МОДУЛЯ

3.1 Дослідження швидкості виконання процесів з використанням черги повідомлень у тестовому середовищі

Пересвідчившись у тому, що нову чергу було успішно створено, та маючи всі необхідні файли-обробники, ми маємо можливість дослідити процес реєстрації нового користувача, використавши чергу повідомлень. Для цього повернемось до форми реєстрації (Рис.2.2) та створимо користувача з тестовою електронною адресою *'testCustomerQueue@mail.com'*. Так, як більшість важких процесів було перенесено на обробку за допомогою черги повідомлень, то відтепер лише буде внесено новий запис щодо користувача до бази даних (Рис. 3.1), а далі до черги повідомлень буде передано ідентифікатор запису з бази даних на подальшу обробку даних користувача, що ми можемо перевірити у веб-інтерфейсі Rabbit MQ.

	entity_id Entity ID	website_id Website ID	email Email	group_id Group ID
лить	1759	1	testCustomerQueue@mail.com	1

Рисунок 3.1 – Новий зареєстрований користувач у базі даних

Спостерігаємо, що користувача з потрібною електронною адресою було успішно внесено до бази даних. Так, як логіка обробки повідомлень буде працювати з ідентифікатором користувача, то до черги повідомлень буде надіслано значення з поля *entity_id*. В нашому випадку дане значення дорівнює 1759. Тому, перевіримо, чи було відправлено саме це значення до черги повідомлень (Рис. 3.2).



Рисунок 3.2 – Ідентифікатор користувача в черзі повідомлень

Спостерігаємо, що в черзі повідомлень було розміщено вірний ідентифікатор користувача (поле Payload), а отже, всі налаштування зі сторони модуля були виконані вірно.

Після перевірки черги повідомлень та її функціоналу, ми можемо дослідити, чи дійсно черга повідомлень оптимізувала час виконання процесу реєстрації. Для цього знову відкриємо інструменти розробника браузера Google Chrome та порівняємо, наскільки швидше або повільніше став виконуватися процес (рис.3.3).

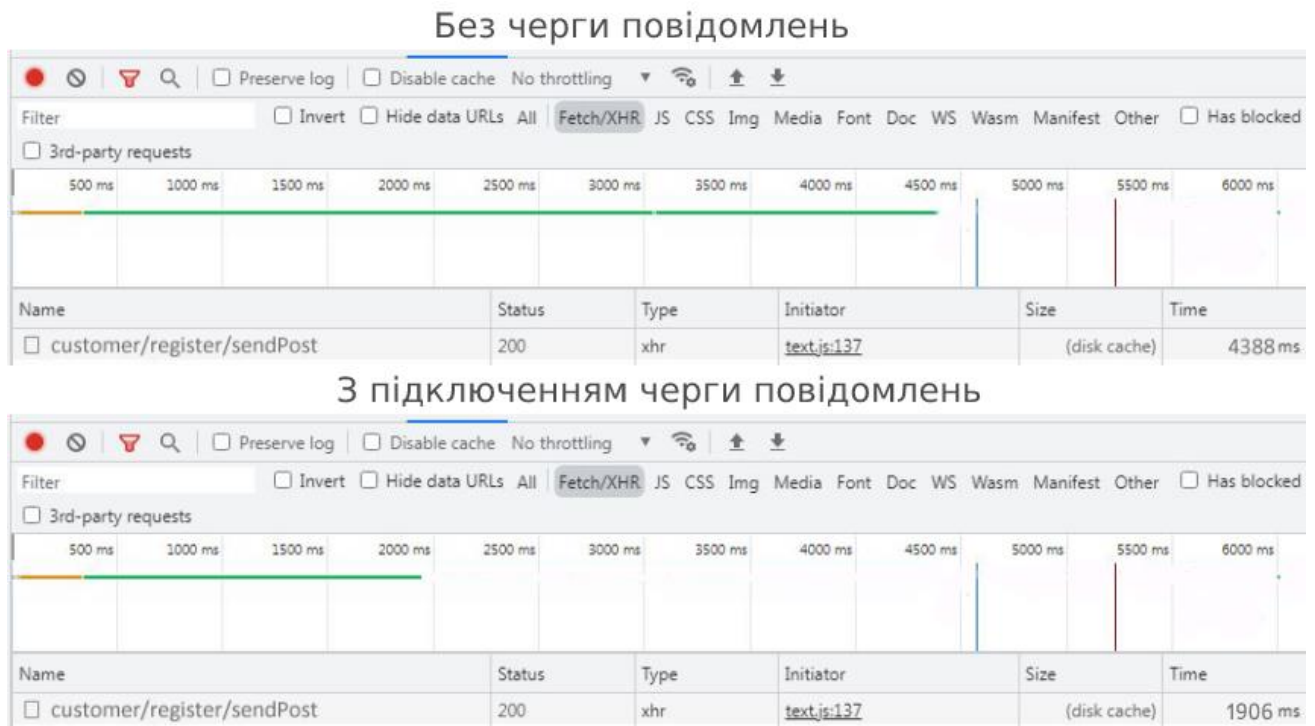


Рисунок 3.3 – Порівняння швидкості виконання процесу реєстрації

Можемо спостерігати досить велику різницю у часі. Процес виконується за 1,95 секунду, що є швидше в 2 рази, ніж було спочатку, та відповідає критеріям оптимізації поставленої задачі.

Тепер проведемо додаткове дослідження з процесом створення замовлення. Створимо потрібні файли за аналогією файлів для користувача та порівняємо швидкість виконання процесу без черги повідомлень та з чергою повідомлень (рис.3.4).

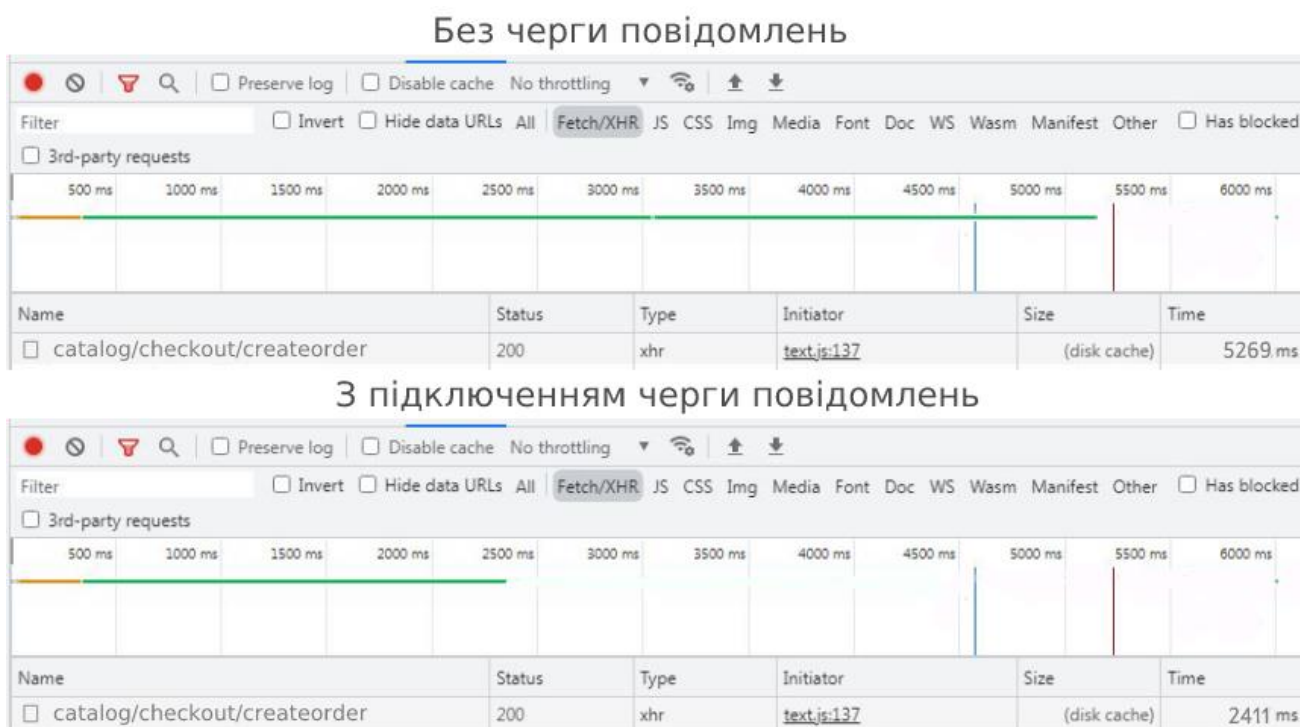


Рисунок 3.4 – Порівняння швидкості виконання процесу створення
замовлення

Спостерігаємо значне прискорення процесу створення замовлення — 2,4 секунди відносно початкових 5,2 секунд, що також є показником успішно виконаної задачі. Тому, можна стверджувати, що модуль було реалізовано вірно, а задачу оптимізації часу було успішно розв’язано. Тому, черга повідомлень є дійсно важливим та потужним інструментом, який розвантажує ресурсомісткі процеси та прискорює їх виконання, що і було досліджено у практичній частині.

Щоб досвідчитись у тому, що модуль є робочим та виконує свої функції, наступним кроком встановимо реалізований модуль на різні проекти та проаналізуємо отриманий результат.

3.2 Дослідження швидкості виконання процесів на реальних проектах

Після проведення аналізу в тестовому середовищі Magento 2, також важливо досвідчитись, що модуль прискорює процеси й на реальних робочих проектах. Тому, для проведення тестів маємо такі вхідні дані:

1. Маємо 5 проектів, встановлених у хмарному середовищі. Відповідно, технічні характеристики серверів, де розміщено проекти, абсолютно ідентичні. Інформація щодо характеристик не надається офіційними джерелами. Список проектів виглядає таким чином:

- Проект з продажу м'яса. Не має сторонніх додаткових сервісів та має середнє навантаження;
- Проект з продажу дорогоцінних металів. Має додаткові сервіси Amazon, де додатково зберігаються дані щодо нового користувача. Тому, саме на даному проекті робота модуля буде показниковою. Має високе навантаження;
- Проект з продажу автомобілів. Не має сторонніх сервісів та має середнє навантаження;
- Проект з продажу напоїв. Має додатковий сервіс, в якому виконуються API запити. Має високе навантаження;
- Проект з продажу сантехніки. Має додатковий сервіс з менеджментом акаунтів у сторонній системі. Комунікація відбувається через API запити. Має високу навантаженість.

2. Робочий комп'ютер для доступу до веб-ресурсів з такими характеристиками:

- Операційна система: Ubuntu 18.04.5 LTS;
- Материнська плата: Asus Prime H310M-R R2.0;
- Процесор: Intel Core i5-9400 2.90GHz;
- Оперативна пам'ять: Kingston HyperX 16GB;
- Жорсткий диск: SSD Samsung 970 Evo Plus 500GB.

3. Об'єкт дослідження — процес реєстрації.

4. Кількість запусків процесу для кожного проекту — 10, 20 та 50 разів.

5. Кожен проект має свій набір додаткових сервісів та технологій.

Наприклад, сервіси Amazon, які значно можуть впливати на швидкість виконання.

Перед тим, як розпочати тестування, перевіримо швидкість виконання процесів на кожному проекті без встановленого модуля, виконавши 10, 20 та 50 запусків. Результати роботи процесу на кожному проекті можна розглянути у таблиці 3.1.

Таблиця 3.1 – швидкість виконання процесу реєстрації без використання модуля

Проект	Продаж м'яса	Продаж дорогоцінних металів	Продаж авто	Продаж напоїв	Продаж сантехніки
К-ть запусків					
10	4634	7305	4812	5537	4956
20	4722	7240	4900	5541	5034
50	4650	7395	4826	5495	4997
Середній час виконання, мс					

Встановимо створений модуль на кожен з проектів та розпочнемо тестування. Для кожного проекту послідовно буде виконано 10, 20 та 50 запусків. Результати тестування буде занесено до таблиці 3.2.

Таблиця 3.2 – Швидкість виконання процесу реєстрації з використанням модуля

Проект	Продаж м'яса	Продаж дорогоцінних металів	Продаж авто	Продаж напоїв	Продаж сантехніки
К-ть запусків					
10	1945	4043	1857	2305	1964
20	2176	3956	1980	2308	2001
50	2053	4099	1933	2267	1943
Середній час виконання, мс					

Спостерігаємо значне прискорення процесу реєстрації. Для кожного проекту швидкість виконання було значно збільшено. У результаті, маємо такі коефіцієнти прискорення швидкості:

- Продаж м'яса — прискорення приблизно у 2,27 разів;
- Продаж дорогоцінних металів – прискорення у 1,81 разів;
- Продаж авто - прискорення у 2,51 разів;
- Продаж напоїв - прискорення у 2,4 рази;
- Продаж сантехніки - прискорення у 2,53 рази;

Отримавши такі результати, спостерігаємо, що реалізований модуль значно прискорює виконання процесів на всіх проектах, де було встановлено даний модуль. Кожен показник співвідношення значно більший, ніж 1,5 разів, які було поставлено у меті. Найменшим результатом зі співвідношенням прискорення продемонстрував проект з продажу дорогоцінних металів – 1,81 рази. Але, навіть такий результат задовольняє критеріям задачі.

ВИСНОВКИ

В результаті виконаної роботи було розроблено модуль для CMS Magento 2, який оптимізує час виконання важких процесів. Модуль виконує такі завдання:

- 1) створює нову чергу повідомлень;
- 2) має всі необхідні файли для налаштування черги;
- 3) реалізовано всі важливі файли для обробки даних з черги;
- 4) модуль значно прискорює виконання процесу, а саме:
 - розвантажує сервер, виконуючи складну логіку за допомогою черги повідомлень;
 - модуль прискорив процес щонайменше, як у 1,5 разів, на кожному досліджуваному проекті, що задовольняє критеріям задачі;
- 5) було використано сучасне програмне забезпечення Docker та сучасну технологію контейнерів для спрощення роботи з додатковими сервісами;
- 6) також було використано зручний менеджер контейнерів Warden для керування процесами контейнерів.

Розроблений модуль може бути модифікований та розширений для використання на інших проектах або для обробки будь-яких процесів у межах Magento 2. В ньому було передбачено можливість перенести будь-які важкі процеси до обробника черги повідомлень а також можливість створювати будь-яку кількість нових черг, залежно від потреб розробника.

ПЕРЕЛІК ПОСИЛАНЬ

1. **Рожков, І. В.** Інформаційні системи і технології в маркетингу : монографія [Текст] / І. В. Рожков – М. : Русайнс, 2014. – 199 с.
2. **Kopp, V.** Grokking Magento Book 1: Basics & Request Flow [Текст] / V. Kopp. – 1st Edition. – CreateSpace Independent Publishing Platform, 2014. – 170 p.
3. **Dickman, A.** Designing Applications with Msmq: Message Queuing for Developers [Текст] / A. Dickman. – Addison-Wesley Professional, 2016. – 400 p.
4. **Johansson L.** RabbitMQ for beginners — What is RabbitMQ. [Електронний ресурс] / L. Johansson. – Режим доступу: <https://www.cloudamqp.com/blog/part1-rabbitmq-for-beginners-what-is-rabbitmq.html>
5. **Johansson, L.** RabbitMQ Essentials: Build distributed and scalable applications with message queuing using RabbitMQ [Текст] / L. Johansson, D. Dossot. – 2nd Edition. – Packt Publishing, 2020. – 154 p.
6. Docker у розробці. [Електронний ресурс]. – Режим доступу: <https://eternalhost.net/blog/razrabotka/chto-takoe-docker>.
7. **Kocher, P.** Microservices and containers [Текст] / P. Kocher. – Addison-Wesley Professional, 2018. – 304 p.
8. **Іванов Д.** Що таке Docker та для чого він використовується [Електронний ресурс] / Д. Іванов. – Режим доступу: <https://selectel.ru/blog/what-is-docker/>
9. **Nickoloff, J.** Docker in Action, Second edition [Текст] / J. Nickoloff, S. Kuenzli. – Manning, 2019. – 350 p.
10. **Fielding, R.** Architectural Styles and the Design of Network-based Software Architectures. [Електронний ресурс] / R. Fielding. – Режим доступу: <https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>

11. Як працює Docker. [Електронний ресурс]. – Режим доступу: <https://habr.com/ru/post/309556/>.
12. **Моует, Е.** Використання Docker. Розробка програмного забезпечення за допомогою технології контейнерів [Текст] / Е. Моует. – М. : ДМК Москва, 2017. – 354 с.
13. **Dossot, D.** RabbitMQ Essentials [Текст] / D. Dossot. – Packt Publishing, 2014. – 182 p.
14. **Miguel, F. G.** Magento 2 Development Essentials [Текст] / F. J.Miguel. – Packt Publishing, 2016. – 178 p.
15. **Ширяєв, П.** Інтеграція RabbitMQ в Magento 2 [Електронний ресурс] / П. Ширяєв. – Режим доступу: <https://habr.com/ru/company/rshb/blog/527834/>
16. **Ajzele, B.** Magento 2 Developers Guide [Текст] / B. Ajzele. – Packt Publishing, 2015. – 412 p.
17. ДСТУ 3008-95. Документація. Звіти у сфері науки і техніки. Структура і правила оформлення [Текст]. – К, 1995 – 38 с.