Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра комп'ютерних наук
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Створення backend частини мультисайтової мікросервісної архітектури засобами PHP Laravel, Docker, NGINX, Percona, Redis, Elasticsearch та Kong API Gateway

Виконав: студент IV курсу, групи СНс-42
спеціальності 122 Комп'ютерні науки
(шифр і назва спеціальності)

_____ Бойчук О. А.
(підпис) (прізвище та ініціали)

Керівник _____ Млинко Б.Б.
(підпис) (прізвище та ініціали)

Нормоконтроль _____ Шимчук Г.В.
(підпис) (прізвище та ініціали)

Завідувач кафедри _____ Боднарчук І.О.
(підпис) (прізвище та ініціали)

Рецензент _____ Осухівська Г.М.
(підпис) (прізвище та ініціали)

Тернопіль
2024

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра комп'ютерних наук
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Боднарчук І.О.
(підпис) (прізвище та ініціали)
«__» _червня_ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Бакалавр
(назва освітнього ступеня)

за спеціальністю 122 Комп'ютерні науки
(шифр і назва спеціальності)

Студенту Бойчук Олександр Антонович
(прізвище, ім'я, по батькові)

1. Тема роботи Створення backend частини мультисайтової мікросервісної архітектури засобами PHP Laravel, Docker, NGINX, Percona, Redis, Elasticsearch та Kong API Gateway

Керівник роботи Млинко Богдана Богданівна, к.т.н., доцент кафедри КН
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «*29*» *квітня* 2024 року № *4/7-47*

2. Термін подання студентом завершеної роботи 24 червня 2024р.

3. Вихідні дані до роботи Літературні та інтернет джерела щодо розробки backend частини мультисайтової мікросервісної архітектури засобами PHP Laravel, Docker, NGINX, Percona, Redis, Elasticsearch та Kong API Gateway

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ. 1. Аналіз предметної області та постановка завдання розробки. 1.1 Аналіз предметної області. 1.1.1 Особливості мультисайтової архітектури. 1.1.2 Переваги та недоліки мікросервісної архітектури. 1.2 Огляд існуючих рішень. 1.3 Планування розробки backend частини. 1.3.1 Вимоги до мультисайтової мікросервісної архітектури. 1.4 Формування структури мікросервісної архітектури. 1.4.1 Дизайн мікросервісів. 1.4.2 Взаємодія між мікросервісами. 1.5 Обґрунтування використовуваних технологій. 1.6 Висновок до першого розділу. 2. Проєктування backend частини мультисайтової мікросервісної архітектури. 2.1 Проєктування мікросервісної архітектури. 2.2 Проєктування баз даних мікросервісів. 2.3 Висновок до другого розділу. 3. Реалізація та тестування backend частини мультисайтової мікросервісної архітектури. 3.1 Розробка мікросервісів. 3.2 Інтеграція та налаштування мікросервісів. 3.3 Тестування мультисайтової мікросервісної архітектури. 3.3.1 Застосування розробленої системи. 3.4 Висновок до третього розділу. 4. Безпека життєдіяльності, основи охорони праці. Висновки. Перелік джерел. Додатки.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
1. Титульний сдайд. 2. Актуальність обраної теми. 3. Мета дослідження. 4. Огляд існуючих рішень. 5. Переваги та недоліки мікросервісної архітектури. 6. Розроблені мікросервіси. 7. Функціональні можливості розроблених мікросервісів. 8.Архітектура розробленої системи. 9. Використані технології та програмне забезпечення. 10. Практичне застосування розробленої мультисайтової мікросервісної архітектури. 11. Висновки.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці	Сенчишин В.С. доцент кафедри МТ		

7. Дата видачі завдання 29 січня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи		
2.	Підбір джерел про розробку мікросервісної архітектури засобами PHP Laravel, Docker, NGINX, Percona, Redis, Elasticsearch та Kong API Gateway		
3.	Опрацювання джерел по темі кваліфікаційної роботи		
4.	Виконання дослідження щодо розробки мультисайтової мікросервісної архітектури засобами PHP Laravel, Docker, NGINX, Percona, Redis, Elasticsearch та Kong API Gateway		
	Розроблення мультисайтової мікросервісної архітектури засобами PHP Laravel, Docker, NGINX, Percona, Redis, Elasticsearch та Kong API Gateway		
5.	Оформлення розділу «Аналіз предметної області та постановка завдання»		
6.	Оформлення розділу «Проєктування backend частини мультисайтової мікросервісної архітектури»		
6.	Оформлення розділу «Реалізація та тестування backend частини мультисайтової мікросервісної архітектури»		
7.	Виконання завдання до підрозділу «Безпека життєдіяльності»		
8.	Виконання завдання до підрозділу «Основи охорони праці»		
9.	Оформлення кваліфікаційної роботи		
10.	Нормоконтроль		
11.	Перевірка на плагіат		
12.	Попередній захист кваліфікаційної роботи		
13.	Захист кваліфікаційної роботи		

Студент _____ Бойчук О.А.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Млинко Б.Б.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Створення backend частини мультисайтової мікросервісної архітектури засобами PHP Laravel, Docker, NGINX, Percona, Redis, Elasticsearch та Kong API Gateway // Кваліфікаційна робота освітнього рівня «Бакалавр» // Бойчук Олександр Антонович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра комп'ютерних наук, група СНс-42 // Тернопіль, 2024 // С. 56, рис. – 16, табл. – 5, слайди. – 11, додат. – 4, бібліогр. – 35.

Ключові слова: мікросервісна архітектура, мультисайтовість, бази даних, веб-сайт, backend, API, PHP, Laravel.

Кваліфікаційна робота присвячена розробці backend частини мультисайтової мікросервісної архітектури з використанням PHP Laravel, Docker, NGINX, Percona, Redis, Elasticsearch та Kong API Gateway.

В першому розділі кваліфікаційної роботи детально проаналізовано предметну область, висвітлено переваги та недоліки мікросервісної архітектури, а також проаналізовано вимоги та сплановано розробку.

В другому розділі кваліфікаційної роботи досліджено процес проєктування мікросервісної архітектури, описано функціональність мікросервісів, сформовано базу даних для кожного мікросервісу.

В третьому розділі кваліфікаційної роботи описано розробку мікросервісів, їх інтеграцію та налаштування, проведено проєктування, тестування та застосування розробленої системи у реальному веб-сайті.

Об'єкт дослідження: процес розробки backend частина мультисайтової мікросервісної архітектури.

Предмет дослідження: засоби і методи розробки мікросервісів з підтримкою мультисайтовості, зокрема PHP Laravel, Docker, NGINX, Percona, Redis, Elasticsearch та Kong API Gateway.

ANNOTATION

Creating the backend part of a multisite microservices architecture using PHP Laravel, Docker, NGINX, Percona, Redis, Elasticsearch, and Kong API Gateway // Qualification work of the educational level "Bachelor" // Boichuk Oleksandr // Ternopil Ivan Pulyu National Technical University, Computer and Information Systems and Software Engineering Faculty, Computer Sciences Department, group SNs-42 // Ternopil, 2024 // P. 56, slides. - 11, tabl. - 5, chair. - 0, annexes. – 4, references - 35.

Keywords: microservices architecture, multisite capability, databases, website, backend, API, PHP, Laravel.

The qualification work is dedicated to developing the backend part of a multisite microservices architecture using PHP Laravel, Docker, NGINX, Percona, Redis, Elasticsearch, and Kong API Gateway.

In the first section of the qualification work, the subject area is thoroughly analyzed, highlighting the advantages and disadvantages of microservices architecture, as well as analyzing requirements and planning development.

The second section of the qualification work explores the process of designing a microservices architecture, describing the functionality of microservices, and forming a database for each microservice.

The third section of the qualification work describes the development, integration, and configuration of microservices, as well as the design, testing, and application of the developed system in a real website.

Research object: the process of developing the backend part of a multi-site microservices architecture.

Subject of the study: tools and methods for developing microservices with multisite support, including PHP Laravel, Docker, NGINX, Percona, Redis, Elasticsearch, and Kong API Gateway.

ПЕРЕЛІК СКОРОЧЕНЬ І ТЕРМІНІВ

API (англ. Application Programming Interface) – набір визначень і протоколів для створення і інтеграції прикладного програмного забезпечення.

API Gateway – сервер або сервіс, який виступає єдиною точкою входу для групи мікросервісів, забезпечуючи маршрутизацію запитів, агрегацію результатів і управління політиками безпеки.

Backend – серверна частина програмного забезпечення, що виконує основні обчислення, обробку даних і забезпечує зберігання інформації.

CRUD (англ. Create, Read, Update, Delete) – основні операції управління даними в базі даних.

Docker – платформа для автоматизації розгортання, масштабування та управління додатками в контейнерах.

Elasticsearch – розподілена пошукова та аналітична система з відкритим вихідним кодом для всіх типів даних.

JSON (англ. JavaScript Object Notation) – формат обміну даними, що використовується для передачі даних між сервером та клієнтом.

JWT (англ. JSON Web Token) – стандарт для створення токенів доступу, що використовуються для аутентифікації та передачі інформації між сторонами.

Kong API Gateway – платформа з відкритим вихідним кодом для керування API, яка забезпечує високу продуктивність та гнучкість у маршрутизації запитів.

Laravel – PHP фреймворк з відкритим вихідним кодом для розробки веб-додатків.

MySQL – система управління реляційними базами даних з відкритим вихідним кодом.

NGINX – веб-сервер з відкритим вихідним кодом, що також може виконувати роль зворотного проксі-сервера, балансувальника навантаження та HTTP кешу.

PHP – скриптова мова програмування, широко використовується для розробки веб-додатків.

Postman – інструмент для тестування API, що дозволяє розробникам створювати, тестувати та документувати свої запити.

Redis – база даних з відкритим вихідним кодом, яка використовує структури даних типу ключ-значення.

REST (англ. Representational State Transfer) – архітектурний стиль взаємодії між компонентами в мережі, який використовує стандартні методи HTTP.

Stripe – платформа для онлайн-платежів, яка забезпечує інтерфейс для прийому та управління онлайн-платежами.

Аутентифікація – процес перевірки автентичності користувача або системи.

БД – база даних.

Мікросервіс – невеликий, самостійний сервіс, що виконує одну функцію в межах мікросервісної архітектури.

Мікросервісна архітектура – підхід до проектування програмного забезпечення, в якому додаток складається з невеликих незалежних сервісів, що взаємодіють через API.

Мультисайтовість – можливість системи підтримувати та керувати декількома сайтами з однієї інсталяції.

ПЕОМ - узагальнена назва групи засобів обчислювальної техніки, які виконують задану програмою послідовність операцій (арифметичних і логічних) обробки даних на основі принципу програмного управління.

Фреймворк – програмне забезпечення, яке полегшує розробку шляхом поєднання в собі різних функціональних можливостей.

ЗМІСТ

ВСТУП

Актуальність теми. Внаслідок швидкого розвитку інтернету та інформаційних технологій, сучасні веб-сайти стають все більш складними і вимогливими до архітектури та інфраструктури. Мікросервісна архітектура з підтримкою мультисайтовості набуває популярності завдяки своїй здатності забезпечувати високу гнучкість, масштабованість і зручність в управлінні. Вибір і впровадження такої архітектури, дозволяє створювати надійні та ефективні веб-сайти, які можуть працювати на одній платформі. Тому розробка мікросервісної архітектури з підтримкою мультисайтовості є актуальним напрямком сучасних досліджень в галузі веб-розробки.

Мета і задачі дослідження. Метою даної кваліфікаційної роботи освітнього рівня «Бакалавр» є створення ефективної та масштабованої мікросервісної архітектури з підтримкою мультисайтовості для підвищення якості послуг. Для досягнення поставленої мети потрібно виконати ряд задач, зокрема:

- Проаналізувати обрану предметну область.
- Розглянути переваги та недоліки мікросервісної архітектури.
- Вибрати оптимальні технології та інструменти для розробки.
- Спроєктувати структуру мікросервісів та їх взаємодію.
- Розробити та протестувати окремі мікросервіси.
- Інтегрувати та налаштувати мікросервіси для забезпечення їх взаємодії.

Практичне значення одержаних результатів. Розроблена в ході виконання кваліфікаційної роботи backend частина мультисайтової мікросервісної архітектури дозволить створювати та ефективно управляти декількома веб-сайтами, кожен з яких може використовувати різні комбінації мікросервісів залежно від потреб. Це забезпечить гнучкість та масштабованість усієї мікросервісної архітектури, що є критичним для сучасних веб-сайтів.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1 Аналіз предметної області

Сучасні веб-сайти вимагають високої гнучкості, масштабованості та зручності в управлінні. Особливо це стосується компаній, які володіють кількома брендами, або потребують розміщення різних сайтів для різних географічних регіонів. Для таких задач підходять мультисайтові архітектури та мікросервісні підходи. Мультисайтові системи дозволяють ефективно керувати декількома сайтами з одного місця, забезпечуючи централізоване адміністрування та спільну кодову базу [1]. Мікросервісна архітектура, в свою чергу, забезпечує незалежність компонентів системи, що дозволяє досягти високої масштабованості та стійкості до збоїв .

1.1.1 Особливості мультисайтової архітектури

Мультисайтова архітектура – це підхід до розробки веб-застосунків, який дозволяє створювати декілька сайтів на базі єдиної платформи або інфраструктури. Кожен сайт може мати свою тему, функціональність та контент, але використовує загальну кодову базу та інфраструктуру [2]. Основні особливості мультисайтової архітектури включають:

1. Єдина платформа для управління декількома сайтами: адміністратори можуть управляти кількома сайтами з єдиної панелі управління, що спрощує керування та оновлення.

2. Поділ ресурсів: мультисайтова архітектура дозволяє спільно використовувати ресурси (бази даних, файлову систему тощо) між різними сайтами, що знижує витрати на інфраструктуру

3. Індивідуальні налаштування для кожного сайту: кожен сайт може мати свої унікальні налаштування, що дозволяє адаптувати функціональність під конкретні потреби.

4. Масштабованість: мультисайтова архітектура легко масштабується, що дозволяє швидко додавати нові сайти без значних змін в основній інфраструктурі [3].

1.1.2 Переваги та недоліки мікросервісної архітектури

Мікросервісна архітектура передбачає розбиття застосунку на невеликі, незалежні сервіси, кожен з яких виконує окрему функцію та може розроблятися, тестуватися, розгортатися та масштабуватися окремо [4]. Основні переваги та недоліки мікросервісної архітектури включають:

Переваги:

1. Масштабованість: кожен мікросервіс можна масштабувати окремо, що дозволяє ефективно розподіляти ресурси відповідно до навантаження.

2. Гнучкість у розробці: Різні команди можуть працювати над різними мікросервісами незалежно один від одного, що прискорює процес розробки та впровадження нових функцій [5].

3. Надійність: збої в одному мікросервісі не призводять до збоїв у всій системі, що підвищує загальну надійність застосунку.

4. Спрощене тестування та деплоймент: оскільки кожен мікросервіс є автономним, його легше тестувати та розгортати, що зменшує ризики при впровадженні змін.

Недоліки:

1. Складність управління: велика кількість мікросервісів ускладнює управління та моніторинг, вимагає ефективних інструментів оркестрації та координації [6].

2. Міжсервісна комунікація: забезпечення ефективної та надійної комунікації між мікросервісами може бути складним завданням, що вимагає додаткових рішень (наприклад, API Gateway).

3. Підвищені вимоги до інфраструктури: кожен мікросервіс потребує власних ресурсів (сервери, бази даних тощо), що збільшує загальні витрати на інфраструктуру.

4. Висока складність тестування інтеграції: незважаючи на спрощення тестування окремих мікросервісів, тестування інтеграції всієї системи стає більш складним та трудомістким [7].

Таким чином, мікросервісна архітектура надає значні переваги у гнучкості, масштабованості та надійності, проте вимагає ретельного планування та управління для подолання складнощів, пов'язаних з її реалізацією та підтримкою. [8]

1.2 Огляд існуючих рішень

На шляху розробки мультисайтової архітектури ключовим етапом є вибір відповідної платформи або інструменту, який найбільш ефективно відповідатиме потребам та вимогам проєкту. Важливо врахувати наявні рішення та платформи, що можуть задовольнити вимоги проєкту та оптимізувати робочий процес [9]. Існують такі платформи для створення мультисайтової архітектури:

WordPress Multisite. Це функція WordPress, яка дозволяє створювати та управляти декількома сайтами з єдиної встановленої копії WordPress. Ця платформа добре підходить для швидкого розгортання та керування багатьма сайтами [10].

Drupal Multisite. Аналогічно до WordPress Multisite, Drupal Multisite дозволяє керувати кількома сайтами з одного встановлення Drupal. Вона має більшу гнучкість у налаштуванні, але вимагає більше технічних знань для налаштування та підтримки [11].

Magento Commerce. Це платформа електронної комерції, яка надає можливості створення мультисайтових магазинів з одного адміністративного інтерфейсу. Вона добре підходить для інтернет-магазинів з різними темами та функціональністю [12].

Shopify Plus. Це розширена версія платформи електронної комерції Shopify, яка підтримує мультисайтовість. Ця платформа має високу

продуктивність та надійність, але обмежує можливості налаштування мікросервісів [13].

Custom Solution with Microservices Architecture: Написання власної системи з використанням мікросервісної архітектури є можливістю для максимальної гнучкості та контролю.

Кожна з цих платформ має свої переваги та обмеження. Під час вибору оптимального рішення необхідно врахувати потреби проєкту, рівень складності та гнучкість, якої потрібно досягти.

1.3 Планування розробки backend частини

Планування розробки backend частини мультисайтової мікросервісної архітектури є важливим етапом, який дозволяє визначити цілі проєкту, його функціональні можливості та те, як система буде відповідати очікуванням користувачів. Даний розділ описує процес планування розробки, включаючи визначення вимог до системи, формування функціональних та нефункціональних вимог, також опис методів, які використовуються для цього.

1.3.1 Вимоги до мультисайтової мікросервісної архітектури

Для розробки ефективної та гнучкої мікросервісної архітектури з підтримкою мультисайтовості, необхідно визначити вимоги до системи [14]. Ці вимоги можна розділити на дві категорії: функціональні та нефункціональні.

Функціональні вимоги описують те, що система повинна вміти робити. До них належать:

– Підтримка мультисайтовості: система повинна мати можливість підтримувати декілька сайтів з різною тематикою.

– Мікросервісна архітектура: система повинна складатися з незалежних мікросервісів, кожен з яких відповідає за певну функціональність.

– API: система повинна мати API, яке буде використовуватися для взаємодії з мікросервісами.

– Аутентифікація та авторизація: система повинна мати механізми аутентифікації та авторизації для захисту даних [15].

Нефункціональні вимоги описують те, як система повинна працювати. До них належать:

– Масштабованість: система повинна бути здатною обслуговувати велику кількість користувачів та запитів.

– Гнучкість: система повинна бути легко розширюваною та модифікованою для додавання нових функцій та сайтів.

– Продуктивність: система повинна мати високу продуктивність та мінімальний час відгуку.

– Безпека: система повинна бути стійкою до кібератак та забезпечувати захист даних [16].

Зазначені вимоги слід враховувати на кожному етапі розробки, починаючи від проєктування архітектури та закінчуючи впровадженням та підтримкою системи в експлуатаційному середовищі. Дотримання цих вимог дозволить забезпечити правильне функціонування всієї системи.

1.4 Формування структури мікросервісної архітектури

1.4.1 Дизайн мікросервісів

Для взаємодії між мікросервісами було використано REST API та Kong API Gateway. REST API надасть можливість мікросервісам обмінюватися даними за стандартними HTTP-запитами, Kong API Gateway забезпечить централізоване керування та маршрутизацію трафіку між мікросервісами.

Детальну структуру мультисайтової мікросервісної архітектури та основний принцип взаємодії веб-застосунків з нею представлено на рисунку 1.1

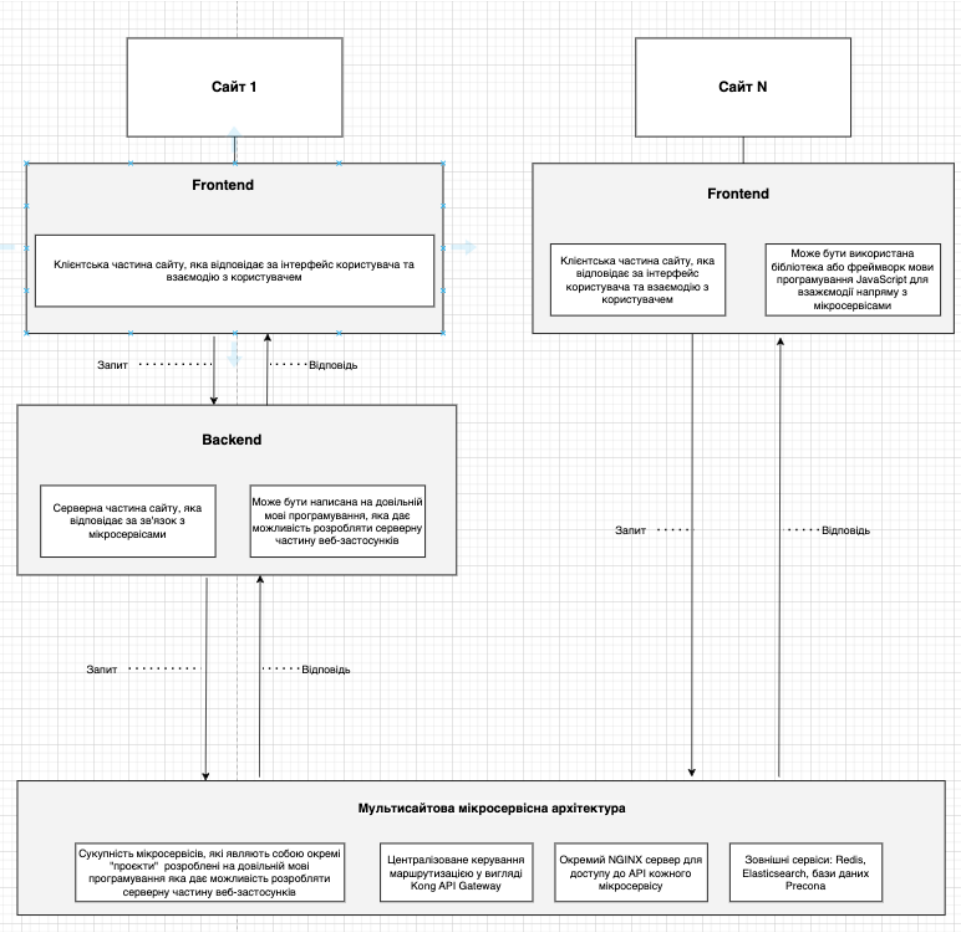


Рисунок 1.1 – Структура мікросервісної архітектури та основний принцип взаємодії веб-застосунків з нею

Кожен мікросервіс буде представлено у вигляді окремого Laravel застосунку, який буде взаємодіяти виключно по API. До стеку мікросервісу обовязково входить PHP (для інтерпретації коду), NGINX (для передачі запиту до застосунку Laravel для його обробки), також туди може входити БД, а саме Percona (альтернативний варіант MySQL), Redis (для збереження кешу) та Elasticsearch (для пошуку).

1.4.2 Взаємодія між мікросервісами

Взаємодія між мікросервісами в мультисайтовій мікросервісній архітектурі буде здійснюватися за допомогою REST API у форматі JSON. Для маршрутизації та агрегування запитів буде використовуватися API-шлюз Kong.

API-шлюз Kong буде відповідати за, тобто він буде перенаправляти запити від клієнтського додатку до відповідних мікросервісів.

Взаємодія відбуватиметься таким чином: користувач відкриває сторінку з списком товарів в категорії "Одяг" на сайті, після чого додаток надсилає GET-запит до API-шлюзу Kong, він в свою чергу перенаправляє запит до мікросервісу товарів, мікросервіс товарів отримує список товарів з категорії "Одяг" з бази даних, після надсилає відповідь API-шлюзу Kong, а той в свою чергу надсилає відповідь клієнтському додатку, після чого додаток відображає список товарів на екрані користувача.

В такому випадку використання REST API та API-шлюзу Kong дозволить створити гнучку та масштабовану систему, яка буде легко піддаватися змінам та доопрацюванням [17].

1.5 Обґрунтування використовуваних технологій

Використання конкретних технологій впливає на ефективність, масштабованість, безпеку та інші аспекти розробки і підтримки системи [18]. Нижче наведені технології, які ми вибрали для реалізації нашого проєкту та обґрунтування їх використання:

У якості основної технології розробки було обрано мову програмування PHP та її феймворк Laravel. Даний фреймворк є одним з найпопулярніших фреймворків для розробки веб-сайтів та сервісів на PHP. Він має широкий функціонал, простий синтаксис та потужну систему роутингу, яка ідеально підходить для побудови мікросервісної архітектури [19].

Задля забезпечення контейнеризації був використаний Docker. Він абезпечує контейнеризацію додатків, що дозволяє легко розгортати та масштабувати середовище розробки та виробництва. Також його використання дозволяє забезпечити однаковість середовищ усім членам команди розробки та спростити процес розгортання [20].

У якості веб-серверу було обрано NGINX, оскільки він є високопродуктивним веб-сервером, який використовується для обробки

великого обсягу запитів та забезпечення швидкості роботи веб-сайтів, а в даному випадку – мікросервісів [21].

База даних. За основу було взято MySQL, але при побудові архітектури використовувалась Percona. Це альтернативний варіант MySQL Server, який включає в себе деякі додаткові функції, які розширюють можливості стандартного MySQL. Ці функції можуть включати в себе покращений механізм оптимізації запитів, засоби моніторингу та управління, а також додаткові інструменти для підтримки високої доступності. Вона надає високу швидкодію та надійність, що робить її ідеальним вибором для забезпечення роботи бази даних в мікросервісах [22].

У якості сховища кешу було обрано Redis. Це програмне забезпечення представляє собою базу даних у форматі ключ-значення, яка використовується для кешування даних та підвищення продуктивності мікросервісів. Використання Redis дозволяє зберігати та отримувати дані з високою швидкістю, що важливо для швидкості відповіді мікросервісів, збереження ресурсів бази даних та швидкості завантаження сторінки на веб-сайті [23].

Для пошуку було використано Elasticsearch. Це потужний пошуковий рушій, який дозволяє здійснювати повнотекстовий пошук у великих обсягах даних. Використання Elasticsearch дозволяє ефективно забезпечити пошук серед товарів [24]. Також даний сервіс підходить для збору логу всіх мікросервісів та логу доступу NGINX.

У якості центральної точки управління API мікросервісної архітектури було обрано Kong API Gateway. Він забезпечує керування та маршрутизацію API запитів та перенаправлення запитів до мікросервісів, а також забезпечує захист API від стороннього доступу [25].

Кожна з цих технологій була обрана з урахуванням її відповідності вимогам проєкту, забезпечення ефективності та надійності системи, а також для спрощення процесу розробки та підтримки.

1.6 Висновок до першого розділу

У першому розділі даної кваліфікаційної роботи було детально проаналізовано обрану предметну область і аргументовано її актуальність у контексті розробки мультисайтової мікросервісної архітектури. Було розглянуто особливості мультисайтовості, а також переваги та недоліки мікросервісної архітектури.

Огляд існуючих рішень дозволив виявити та порівняти різні платформи, що використовуються для створення мультисайтових систем, таких як WordPress Multisite, Drupal Multisite, Magento Commerce, Shopify Plus та інші. Це дало змогу краще зрозуміти, які рішення найкраще відповідають поставленим завданням.

При плануванні розробки backend частини було визначено вимоги до системи, зокрема, як функціональні, так і нефункціональні. Функціональні вимоги включають підтримку мультисайтовості, мікросервісну архітектуру, API для взаємодії з мікросервісами, підтримку різних типів даних, а також механізми аутентифікації та авторизації. Нефункціональні вимоги зосереджені на масштабованості, гнучкості, надійності, продуктивності та безпеці системи.

Було сформовано структуру мікросервісної архітектури, включаючи дизайн мікросервісів та взаємодію між ними за допомогою REST JSON API та Kong API Gateway. Також було обґрунтовано використання певних технологій.

РОЗДІЛ 2. ПРОЄКТУВАННЯ BACKEND ЧАСТИНИ МУЛЬТИСАЙТОВОЇ МІКРОСЕРВІСНОЇ АРХІТЕКТУРИ

2.1 Проєктування мікросервісної архітектури

Проєктування архітектури мікросервісів являє собою ключовий етап у створенні складної і масштабованої системи, що забезбечує незалежність, гнучкість та ефективність як кожного окремого мікросервісу, так і сукупності мікросервісів [26].

Процес проєктування починається з визначення мікросервісів. Далі буде описано мікросервіси, які входитимуть до мультисайтової мікросервісної архітектури.

Мікросервіс категорій. Цей мікросервіс буде відповідати за роботу з категоріями товарів. Він буде забезпечувати CRUD-операції з категоріями, а також буде надавати API для отримання списку категорій, дерева категорій та інформації про конкретну категорію разом з її товарами.

Мікросервіс товарів. Даний мікросервіс буде відповідати за роботу з товарами. Він буде забезпечувати CRUD-операції з товарами, а також буде надавати API для отримання списку товарів, інформації про конкретний товар, товарів з певної категорії та товарів, що відповідають певним критеріям.

Мікросервіс оплат. Цей мікросервіс буде відповідати за роботу з оплатою товарів. Він буде забезпечувати API для створення, обробки та підтвердження платежів користувачів. Платежі будуть оброблятися платіжною системою Stripe. Одна з задач цього мікросервісу – організація взаємодії з API даної платіжної системи.

Мікросервіс блогу. Цей мікросервіс буде відповідати за роботу з блогами. Він буде забезпечувати CRUD-операції з публікаціями, групами категорій та категоріям блогу, а також буде надавати API для отримання списку публікацій, конкретну публікацію та категорії з відповідними до них публікаціями.

Мікросервіс авторизації буде відповідати за всі аспекти, пов'язані з аутентифікацією та авторизацією користувачів в системі. Він буде виконувати

реєстрацію нових користувачів, аутентифікацію користувачів за допомогою логіну та пароля або OAuth, видавати та оновлювати JSON Web Tokens (JWT) для авторизованих користувачів, перевіряти JWT токени при кожному запиті до відповідного API.

Детальну архітектуру мікросервісів з підтримкою мультисайтовості подано на рисунку 2.1.

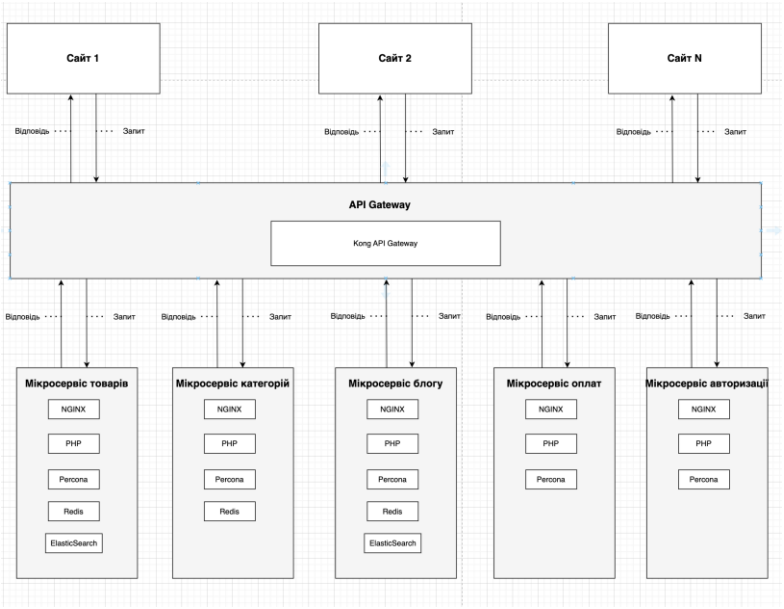


Рисунок 2.1 – Архітектура мікросервісів з підтримкою мультисайтовості

Ця архітектура дозволяє ефективно керувати різними аспектами системи, забезпечуючи високу продуктивність та надійність. Кожен мікросервіс має чітко визначені обов'язки та може масштабуватися незалежно, що сприяє гнучкості і простоті обслуговування всієї системи. Такий підхід значно полегшує впровадження нових функцій і адаптацію до змін у вимогах бізнесу.

2.2 Проєктування баз даних мікросервісів

Кожен мікросервіс повинен мати власну базу даних, яка забезпечує ізоляцію даних та підвищує незалежність мікросервісів. Такий підхід дозволяє уникнути проблем, пов'язаних з єдиною точкою відмови. Він також забезпечує

масштабованість та підвищує безпеку даних [27]. Тому кожен з вищеперерахованих мікросервісів повинен мати свою спроєктовану базу даних.

Мікросервіс товарів. БД товарів спроєктована та оптимізована для коректного та швидкого виконання запитів. В основній таблиці товарів присутня велика кількість індексів, які забезпечують шкидку вибірку з таблиці та мінімізацію часу, який витрачається на отримання даних про товар або товари з бази даних на запит користувача. ER-діаграма бази даних мікросервісу товарів представлена на рисунку 2.2.

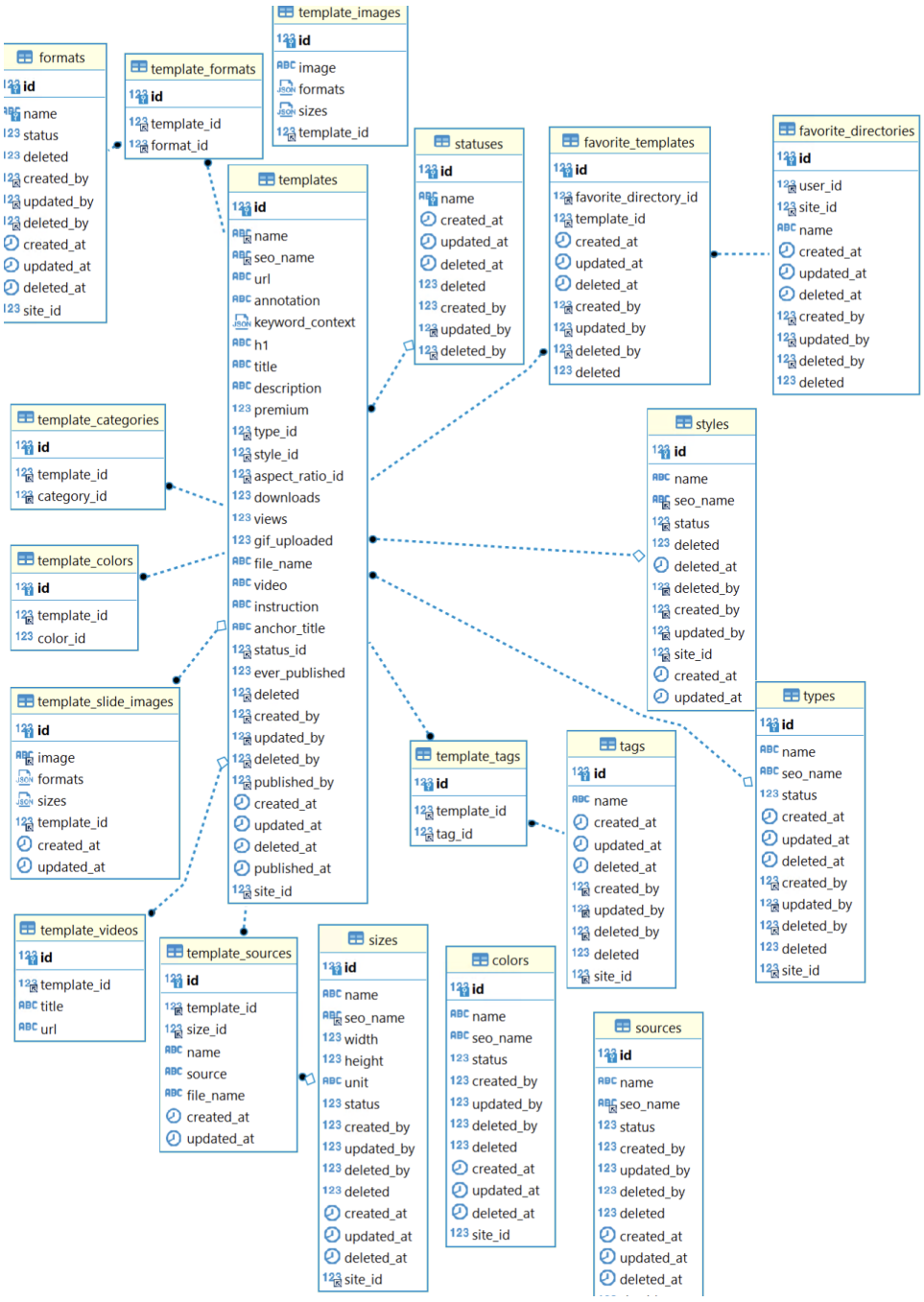


Рисунок 2.2 – ER-діаграма бази даних мікросервісу товарів

Відповідно до представленої ER-діаграми мікросервісу товарів на рисунку перелік та опис таблиць даної бази даних подано в таблиці 2.1. У даній БД представлено велику кількість таблиць, оскільки даний мікросервіс є основним для сайтів, в основі яких лежить продаж товарів, тому базі даних представлено багато таблиць з довідниками тих чи інших сутностей та властивостей товару.

Таблиця 2.1 – Опис таблиць бази даних мікросервісу товарів

Назва таблиці	**Опис**
1	**2**
products	Таблиця товарів, в якій зберігаються товари з усіх сайтів які взаємодіють з мікросервісом товарів
product_categories	Таблиця відповідає за зв'язок один-до-багатьох товарів з категоріями (в окремому мікросервісі)
product_images	Таблиця, в якій зберігаються дані про картинки товарів
product_videos	В даній таблиці представлені дані про відео на сторінці товару
product_slide_images	Таблиця, в якій зберігаються дані про картинки, представлені в слайдері на сторінці товару
product_styles	Таблиця, яка реалізовує зв’язок багато-до-багатьох товарів зі стилями
product_tags	Таблиця, яка реалізовує зв’язок багато-до-багатьох товарів з тегами
product_formats	Таблиця, яка реалізовує зв’язок багато-до-багатьох товарів з форматами товарів
product_sizes	Таблиця, яка реалізовує зв’язок багато-до-багатьох товарів з розмірами товарів
product_sources	Таблиця, яка реалізовує зв’язок один-до-багатьох товарів з посиланнями на товар електронної комерції (наприклад файл Google Docs)
favorite_directories	Таблиця, яка зберігає інформацію про колекції "обраних" товариі користувача
favorite_products	Таблиця, яка реалізовує зв'язок багато-до-багатьох товарів з колекціями "обраного" користувачів
statuses	Таблиця-довідник статусів публікації товару
formats	Таблиця-довідник форматів товару

Продовження таблиці 2.1

1	2
sizes	Таблиця-довідник розмірів товару
colors	Таблиця-довідник кольорів товару
types	Таблиця-довідник типів товару (товар електроної-комерції, чи інші товари)
sizes	Таблиця-довідник розмірів товару

Мікросервіс категорій. Даний мікросервіс матиме іншу окрему базу даних. ER-діаграму цієї бази даних подано на рисунку 2.3.

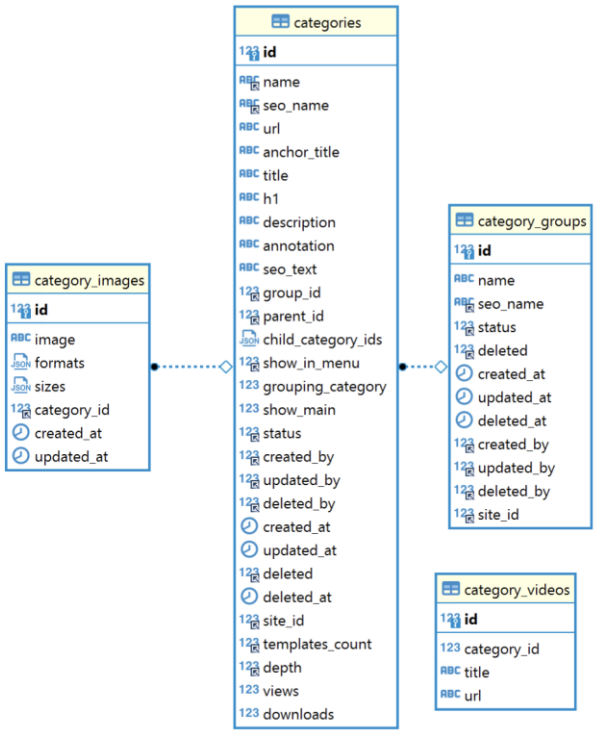


Рисунок 2.3 – ER-діаграма бази даних мікросервісу категорій

Відповідно до вищевказаної ER-діаграми можна сформувати опис кожної з таблиць бази даних. Таблиці бази даних мікросервісу категорій описано та подано у таблиці 2.2.

Таблиця 2.2 –Опис таблиць бази даних мікросервісу категорій

Назва таблиці	**Опис**
categories	Таблиця категорій товарів, в якій представлені категорії з усіх сайтів які взаємодіють з мікросервісом товарів
category_groups	В таблиці зберігаються групи категорій, які повязані з категоріями, які в свою чергу можуть бути узагальнені за допомогою групи категорій
category_images	В цій таблиці зберігаються дані про картинки категорій
category_videos	В даній таблиці зберігаються дані про відео на сторінці товару

Даний мікросервіс має меншу кількість таблиць, ніж мікросервіс товарів, але база даних мікросервісу категорій не менш оптимізована. В основну таблицю категорій додано необхідну кількість індексів для швидкої вибірки, а також спроєктовано та оптимізовано запити до всієї бази даних. Це дозволяє значно підвищити продуктивність при обробці великої кількості запитів, забезпечуючи швидку та ефективну роботу системи в цілому. Додатково, мікросервіс категорій включає механізми кешування для зменшення навантаження на базу даних і прискорення доступу до часто запитуваних даних. Це гарантує стабільну роботу навіть при значному збільшенні кількості користувачів.

Розглянемо мікросервіс блогу. Він також матиме окрему базу даних. ER-діаграму бази даних мікросервісу блогу подано на рисунку 2.4. Основна таблиця блогу містить поля для зберігання інформації про заголовки, тексти постів та дати публікацій. Крім цього, реалізовано таблиці для категорій, тгруп категорій та контенту посту блогу, а також типи контенту та статуси посту, що забезпечує гнучкість і зручність у використанні мікросервісу для організації повноцінного блогу на сайті.

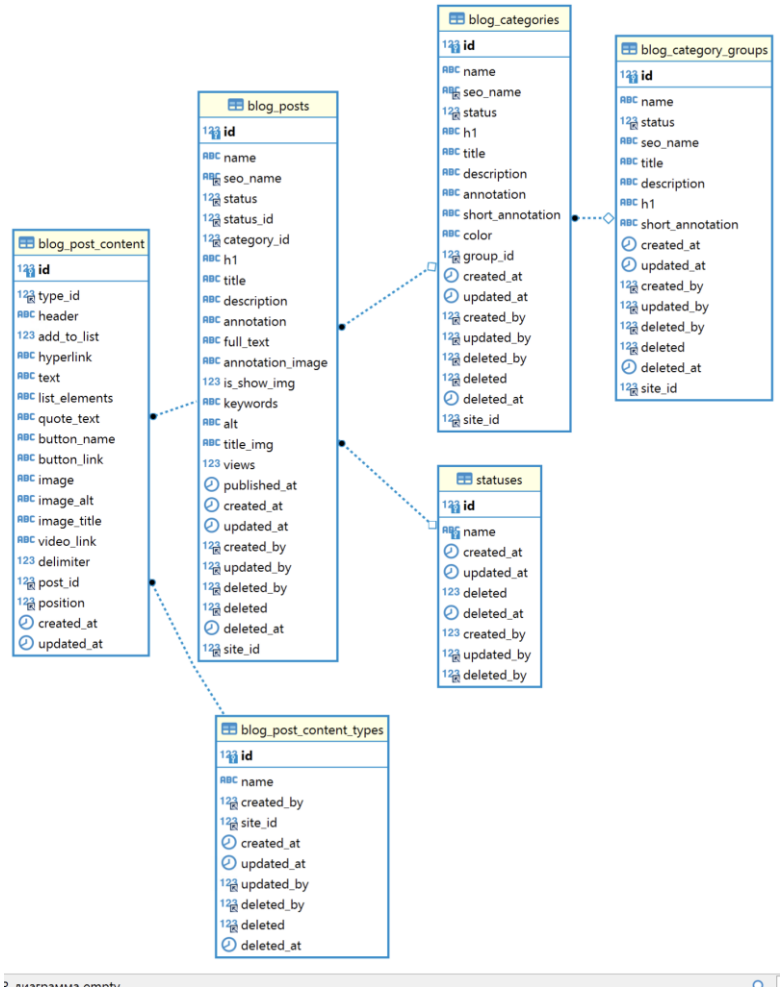


Рисунок 2.4 – ER-діаграма бази даних мікросервісу блогу

Таким чином відповідно до вищевказаної ER-діаграми можна сформувати опис кожної з таблиць бази даних. Таблиці бази даних мікросервісу блогу описано та подано у таблиці 2.3.

Таблиця 2.3 –Опис таблиць бази даних мікросервісу категорій

Назва таблиці	**Опис**
1	**2**
blog_posts	Таблиця постів блогу, в якій представлені пости з усіх сайтів, які використовують даний мікросервіс
blog_categories	В таблиці зберігаються дані про категорії, до яких належать пости блогу
blog_category_groups	Таблиця з групами категорій, які дозволяють згрупувати певні категорії постів блогу

Продовження таблиці 2.3

1	2
blog_post_content	Таблиця, в якій зберігаються дані про контент кожного з постів блогу: картинки, кнопки, текст та інше
statuses	Таблиця-довідник статусів публікації поста блогу

Мікросервіс оплат. ER-діаграму спроєктованої бази даних мікросервісу оплат подано на рисунку 2.5.

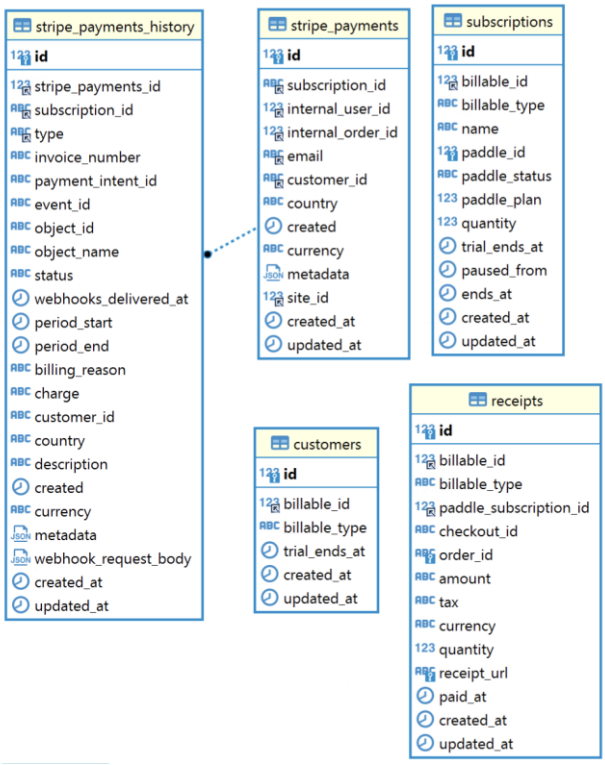


Рисунок 2.5 – ER-діаграма бази даних мікросервісу оплат

Відповідно до вищевказаної ER-діаграми можна сформувати опис кожної з таблиці бази даних мікросервісу. Опис таблиць бази даних мікросервісу оплат описано та подано у таблиці 2.4. Дана база даних має декілька таблиць, а саме таблицю інформації про оплати платіжною системою Stripe, таблицю інформації про підписки користувачів, таблицю користувачів та таблицю чеків оплат користувачів.

Таблиця 2.4 –Опис таблиць бази даних мікросервісу оплат

Назва таблиці	**Опис**
stripe_payments	Таблиця даних про оплати з використанням платіжної системи Stripe
stripe_payments_history	В даній таблиці представлена історія формування та оплати кожного з товарів з використанням платіжної системи Stripe
subscriptions	В цій таблиці зберігаються дані про підписки, які можуть бути оплачені в якості товару на сайтах
customers	В даній таблиці зберігаються дані про покупців, які хочаб один раз купували щось на сайті
receipts	Таблиця, в якій зберігаються дані про чеки оплати товарів користувачами

БД мікросервісу оплат спроєктована таким чином, щоб не привязуватись до конкретної платіжної системи, тобто, якщо якийсь сайт, який використовує даний мікросервіс буде підтримувати інакшу платіжну систему, то мікросервіс буде розширено, але не буде змінено існуючу таблицю з даними про оплати. Ця архітектурна особливість дозволяє зберігати дані про оплати уніфіковано, незалежно від конкретної платіжної системи, що використовується. Такий підхід забезпечує легкість внесення змін у функціональність платформи без потреби вносити складні модифікації до бази даних мікросервісу оплат. В разі підтримки нової платіжної системи лише розширюються функціональні можливості мікросервісу, зберігаючи при цьому стабільність і надійність збережених даних.

Мікросервіс авторизації. ER-діаграму спроєктованої бази даних мікросервісу авторизації подано на рисунку 2.6.

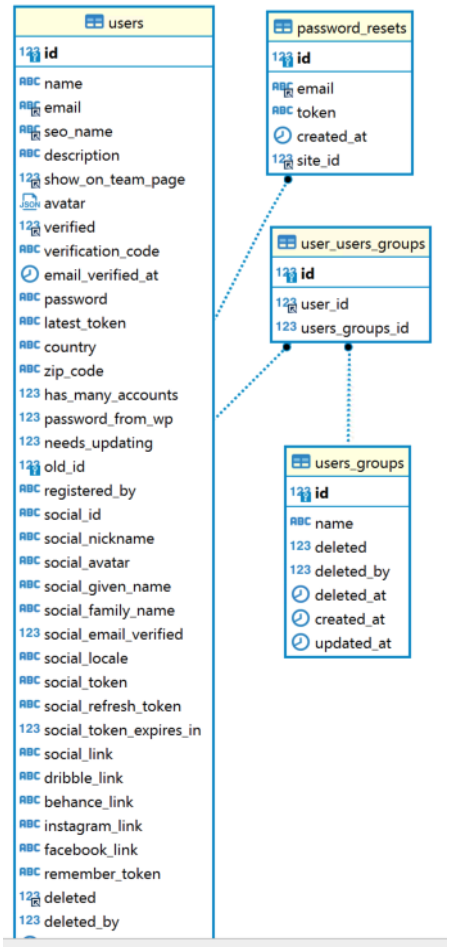


Рисунок 2.6 – ER-діаграма бази даних мікросервісу авторизації

Відповідно до вищевказаної ER-діаграми можна сформувати опис кожної з таблиці бази даних мікросервісу. Опис таблиць бази даних мікросервісу авторизації описано та подано у таблиці 2.5.

Таблиця 2.5 –Опис таблиць бази даних мікросервісу авторизації

Назва таблиці	**Опис**
users	Таблиця з даними про всіх користувачів
user_groups	Таблиця реалізує зв'язок один-до-багатьох, користувач до груп користувачів
groups	Таблиця-довідник груп користувачів
password_resets	Таблиця з даними про відновлення паролів користувачами

БД мікросервісу авторизації, а особливо таблиця користувачів спроєктована таким чином, щоб враховувати всю можливу інформацію про користувачів сайтів. Включаючи посилання на соціальні мережі, аватари, верифікацію, та інше. Також додано велику кількість індексів задля пришвидшення вибірки користувачів з таблиці та швидкого виконання входу до особистого кабінету користувача на сайті.

2.3 Висновок до другого розділу

В другому розділі кваліфікаційної роботи було визначено основні мікросервіси, що будуть входити до архітектури, включаючи мікросервіси для роботи з категоріями товарів, товарами, оплатами, блогом та авторизацією користувачів.

Також у даному розділі було спроєктовано базу даних для кожного окремого мікросервісу та детально розглянуто кожну з них. Особливу увагу було приділено базі даних мікросервісу товарів, яка спроєктована з урахуванням необхідності швидкої вибірки та коректного виконання запитів, що досягається завдяки великій кількості індексів в основній таблиці товарів.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ BACKEND ЧАСТИНИ МУЛЬТИСАЙТОВОЇ МІКРОСЕРВІСНОЇ АРХІТЕКТУРИ

3.1 Розробка мікросервісів

Для розробки кожного мікросервісу було використано фреймворк Laravel мови програмування PHP. Тобто кожен мікросервіс, як згадувалось у попередніх розділах являє собою окремий проєкт фреймворку з окремими налаштуваннями середовища. В основі фреймворку Laravel лежить архітектурний шаблон Модель-вигляд-контроллер (Model-View-Controller), який передбачає поділ системи на три взаємопов'язані частини: модель даних, вигляд (інтерфейс користувача) та модуль керування [28]. Діаграму взаємодії між компонентами шаблону MVC подано на рисунку 3.1 .

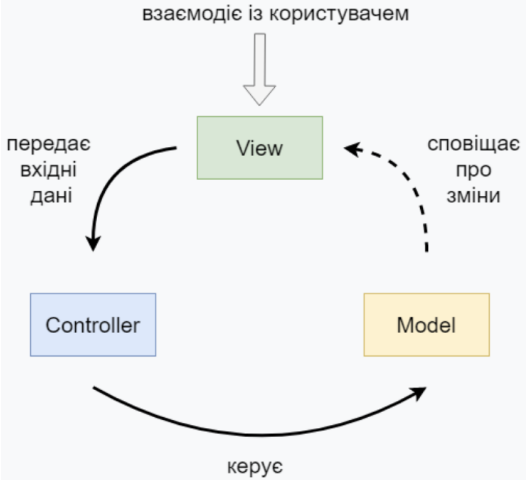


Рисунок 3.1 – Діаграма взаємодії між компонентами шаблону MVC

В основі побудови кожного мікросервісу знаходиться маршрутизація (роутинг), яка надає доступ до бізнес-логіки мікросервісу через API. За кожним маршрутом відповідно до вищеописаного архітектурного шаблону закріплений певний контролер, якому передається управління у разі надходження запиту за цим маршрутом. Після чого контролер передає вхідні дані для окремої функції, яка реалізовує конкретну бізнес логіку, отримує певний результат та надсилає

відповідь користувачеві у форматі JSON, який відповідає архітектурному стилю REST API [29].

3.1.1 Розробка мікросервісу товарів

Основним мікросервісом для деяких сайтів, які прагнуть продавати якісь товари є саме мікросервіс товарів. Як згадувалось у попередніх розділах, даний мікросервіс буде забезпечувати CRUD-операції з товарами, а також буде надавати API для отримання списку товарів, інформації про конкретний товар, товарів з певної категорії та товарів, що відповідають певним критеріям.

Маршрути API мікросервісу товарів наведено у лістингу 3.1.

Лістинг 3.1 – Маршрутизація мікросервісу товарів

```
Route::group(['prefix' => 'products'], function () {
     Route::resource('/admin', AdminController::class);
     Route::get('/all',
          [PublicController::class,'getAllProducts']
     );
     Route::get('/{category_slug}/{product_slug}',
          [PublicController::class, 'getItem']
     );
});
```

Після опису маршрутизації потрібно описати методи контролеру для передачі запиту та його вхідних даних. Для отримання всіх товарів сайту використовується функція контролеру, який наведено у лістингу 3.2.

Лістинг 3.2 – Функція контролеру для отримання усіх товарів на сайті

```
public function getAllProducts(Request $request): JsonResponse
{
     try{
          $items=Product::where('site_id',
               $request->header('x-app-site-id')
```

```
            )->get();
            return $this->successResponse(['items' => $items]);
        }catch (\Exception $exception){
            LogService::error($exception);
            return $this->failedResponse($exception->getMessage());
    }
}
```

Для виконання CRUD операцій було використано функції, наведені у лістингу 3.3

Лістинг 3.3 – Функції, які виконують CRUD операції з товарами

```
public function store(Request $request){
    return$this->successResponse(Product::createItem($request->
validated()));
}
public function show($id){
    return $this->successResponse(Product::findItem($id));
}
public function update(Request $request, $id){
    return$this->successResponse(Product::updateItem($id,
$request->validated());
}
public function destroy($id){
    return $this->successResponse(Product::deleteItem($id));
}
```

Також було реалізовано пошук за допомогою пошукового рушія ElasticSearch. Перед виконанням пошукових запитів було виконано індексацію товарів у пошуковий рушій для його коректних та швидких відповідей на пошукові запити.

Для запитів отримання контенту на сайті, які постійно використовуються, для яких також важлива швидкість відповіді та у невеликому проміжку часу (наприклад одна доба) не зазнають важливих змін, було впроваджено систему

кешування за допомогою Redis. Це дозволяє зменшити навантаження на базу даних та прискорити повернення відповіді на запит. Взаємодія з Redis відбувається за допомогою вбудованого фасаду Cache у фреймворк Laravel. Основну частину коду мікросервісу товарів наведено у додатку А.

3.1.2 Розробка мікросервісу категорій

Для категоризації товарів потрібно задіяти мікросервіс категорій, який буде відповідати за роботу з категоріями товарів. Також він буде забезпечувати CRUD операції з категоріями і надавати API для отримання списку категорій, дерева категорій та інформації про конкретну категорію разом з її товарами.

Реалізовані маршрути API мікросервісу категорій наведено у лістингу 3.4.

Лістинг 3.4 – Маршрутизація мікросервісу категорій

```
Route::group(['prefix' => categories], function () {
     Route::resource('/admin', AdminController::class);
     Route::get('/all',
          [PublicController::class,'getAllCategories']
     );
     Route::get('/{category_slug}',
          [PublicController::class, 'getCategoryItem']
     );
     Route::get('/menu',
          [PublicController::class, 'getMenuCategories']
     );
});
```

Для отримання всіх товарів сайту використовується функція контролеру, яку наведено у лістингу 3.5.

Лістинг 3.5 – Функція контролеру для отримання меню категорій на сайті

```
public function getMenuCategories(Request $request): JsonResponse{
```

```
    $items=Category::where('site_id',$request->header('x-app-site-
id'))->where('menu', 1)->get();
    return $this->successResponse(['items' => $items]);
}
```

Для виконання CRUD операцій було використано функції, наведені у лістингу 3.6.

Лістинг 3.6 – Функції, які виконують CRUD операції з категоріями

```
public function store(Request $request){
    return Category::createCategoryItem($request->validated());
}
public function show($id){
    return Category::findCategoryItem($id);
}
public function update(Request $request, $id){
    returnCategory::updateCategoryItem($id,$request->validated());
}
public function destroy($id){
    return Category::deleteCategoryItem($id);
}
```

Було реалізовано усі необхідні функції та API маршрути для коректного функціонування мікросервісу категорій. Основну частину коду мікросервісу категорій наведено у додатку Б.

3.1.3 Розробка мікросервісу блогу

Для розробки мікросервісу блогу було використано аналогічний підхід, як і у попередніх мікросервісах.

Реалізовані маршрути API мікросервісу наведено у лістингу 3.7.

Лістинг 3.7 – Маршрутизація мікросервісу блогу

```
Route::group(['prefix' => posts], function () {
     Route::resource('/admin', AdminController::class);
     Route::get('/all',
          [PublicController::class,'getAllPosts']
     );
     Route::get('/{post_slug}',
          [PublicController::class, 'getPostItem']
     );
     Route::get('/category/{category_slug}’,
          [PublicController::class, 'getBlogCategory']
     );
});
```

Після розробки маршрутизації було реалізовано функції контролерів, на які посилається кожен маршрут мікросервісу. Реалізовано функцію контролеру для отримання поста блогу наведено на лістингу 3.8.

Лістинг 3.8 – Функція контролеру для отримання поста блогу

```
public function getPostItem(Request $request,$postSlug){
     $items=Post::where(‘site_id’,$request->header('x-app-site-
id'))->where(‘slug, ,$postSlug)->first();
     return $this->successResponse(['items' => $items]);
}
```

Відповідно до вищевказаного лістингу можна побачити, як саме відбувається вибірка з бази даних, спершу додається умова на ідентифікатор сайту, після чого додається умова на slug самого посту блогу. Основну частину коду мікросервісу блогу наведено у додатку В.

3.1.4 Розробка мікросервісу оплат

Даний мікросервіс відповідає за роботу з оплатою товарів. Він забезпечує API для створення, обробки та підтвердження платежів користувачів. Платежі будуть обробляються за допомогою платіжної системои Stripe. Одна з задач

цього мікросервісу – організація взаємодії з API даної платіжної системи. Взаємодія з API даної платіжної системи відбувається за допомогою офіційної бібліотеки для мови програмування PHP. На лістингу 3.9 подано реалізацію взаємодії з API платіжної системи Stripe для створення сесії оплати замовлення.

Лістинг 3.9 – Реалізація створення сесії оплати замовлення

```
$stripe=new\Stripe\StripeClient('test_4eC39HqLyjWDarjtT1zdp7dc');
$stripe->checkout->sessions->create([
     'success_url' => 'https://example.com/success',
     'line_items' => [[
           'price' => 'price_1MotwRLkdIwHu7ixYcPLm5uZ',
           'quantity' => 2,
          ],
     ],
     'mode' => 'payment',
]);
```

Після створення сесії оплати замовлення формується посилання на сторінку оплати у платіжній системі. І одразу після того, як користувач оплачує замовлення – платіжна система надсилає Webhook запит до мікросервісу з даними про оплату. Код мікросервісу оплат наведено у додатку Г.

3.1.5 Розробка мікросервісу авторизації

Мікросервіс авторизації відповідає за всі аспекти, пов'язані з аутентифікацією та авторизацією користувачів в системі. Він виконує реєстрацію нових користувачів, аутентифікацію користувачів за допомогою логіну (ел. пошти) та пароля, генерує та оновлює JSON Web Tokens (JWT) для авторизованих користувачів, перевіряє JWT токени при кожному запиті до відповідного API, яке вимагає авторизації та, можливо, наявності прав доступу користувача.

Реалізовано функцію, яка реєструє користувача. Її представлено на лістингу 3.10. Код мікросервісу авторизації поданл у додатку Д.

Лістинг 3.10 – Функція реєстрації нового користувача

```
public function signUp(SignUpRequest $request){
     try{
          $user=User::create(array_merge(
               $request->validated(),
               ['site_id' => $request->header('x-app-site-id')]
          );
          return $this->successResponse(['user' => $user]);
        }catch (\Exception $exception){
          LogService::error($exception);
          return $this->failedResponse($exception->getMessage());
     }
}
```

Також було реалізовано функцію, яка здійснює аутентифікацію користувача. Дану функцію подано на лістингу 3.11.

Лістинг 3.11 – Функція реєстрації нового користувача

```
public function signIn(SignInRequest $request){
     try{
          $token = $user=auth()->attempt(
               $request->only('email', 'password')
          );
          if(!$token) return $this->unauthorizedResponse();
          return $this->successResponse(['token' => $token]);
        }catch (\Exception $exception){
          LogService::error($exception);
          return $this->failedResponse($exception->getMessage());
     }
}
```

Для роботи з JWT токенами було використано бібліотеку tymon/jwt-auth для PHP. Вона дозволяє генерувати JWT токени та перевіряти їх валідність. Також дозволяє додавати токени до чорного списку (blacklist), тим самим

забороняючи їм доступ навіть до терміну закінчення токену, який становить 24 години.

3.2 Інтеграція та налаштування мікросервісів

Інтеграція та налаштування мікросервісів є ключовим етапом у розробці системи, що базуються на мікросервісній архітектурі, оскільки мікросервіси повинні функціонувати незалежно один від одного та мати передбачувану поведінку.

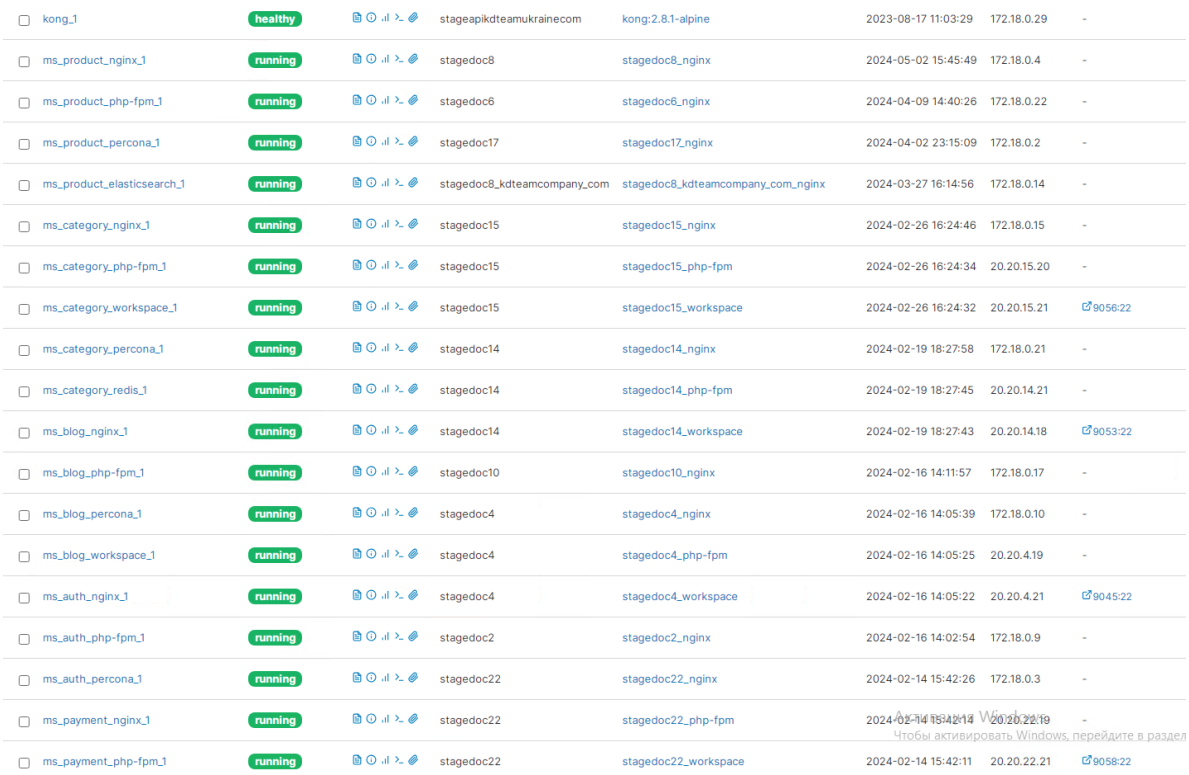
Одним із найважливіших аспектів інтеграції мікросервісів є використання технологій контейнеризації, таких як Docker. У даному випадку використовуватиметься саме Docker, так як це найпопулярніша та найзручніша технологія контейнеризації. Docker забезпечує ізоляцію мікросервісів у окремих контейнерах, що дозволяє знизити залежності між ними та забезпечити стабільність і передбачуваність їхньої роботи. Використання контейнеризації полегшує налаштування та розвиток системи, що важливо для того, щоб переконатися, що вона надійно та продуктивно працює [30].

Розгортання стеку мікросервісів передбачає налаштування контейнерів для кожного окремого мікросервісу, забезпечення їхньої взаємодії через мережеві інтерфейси та налаштування зворотного проксі-сервера для маршрутизації запитів.

Для кожного мікросервісу потрібно описати конфігурацію контейнерів для успішного їх "підняття" на сервері. Дані конфігурації та образи контейнерів вже описано та передбачено у проєкті Laradock. Він містить велику кількість готових налаштувань для великої кількості контейнерів, серед яких є всі, які використані в контексті розробки кожного мікросервісу [31].

Отже, спочатку було налаштувно файл середовища ".env" для кожного мікросервісу, в якому вказано інформацію про базу даних та зовнішні сервіси, також було окремо налаштовано конфігураційні файли для Docker Compose, після чого було виконано побудову (build) контейнерів окремо для кожного мікросервісу. Також, було виконано налаштовано контейнер Kong API Gateway.

Після успішної побудови контейнерів було виконано їх підняття (up) за допомогою Docker Compose. Список усіх контейнерів подано на рисунку 3.2.



kong_1	healthy	stageapikdteamukrainecom	kong:2.8.1-alpine	2023-08-17 11:03:29	172.18.0.29	-
ms_product_nginx_1	running	stagedoc8	stagedoc8_nginx	2024-05-02 15:45:49	172.18.0.4	-
ms_product_php-fpm_1	running	stagedoc6	stagedoc6_nginx	2024-04-09 14:40:26	172.18.0.22	-
ms_product_percona_1	running	stagedoc17	stagedoc17_nginx	2024-04-02 23:15:09	172.18.0.2	-
ms_product_elasticsearch_1	running	stagedoc8_kdteamcompany_com	stagedoc8_kdteamcompany_com_nginx	2024-03-27 16:14:56	172.18.0.14	-
ms_category_nginx_1	running	stagedoc15	stagedoc15_nginx	2024-02-26 16:24:46	172.18.0.15	-
ms_category_php-fpm_1	running	stagedoc15	stagedoc15_php-fpm	2024-02-26 16:24:34	20.20.15.20	-
ms_category_workspace_1	running	stagedoc15	stagedoc15_workspace	2024-02-26 16:24:32	20.20.15.21	9056:22
ms_category_percona_1	running	stagedoc14	stagedoc14_nginx	2024-02-19 18:27:58	172.18.0.21	-
ms_category_redis_1	running	stagedoc14	stagedoc14_php-fpm	2024-02-19 18:27:45	20.20.14.21	-
ms_blog_nginx_1	running	stagedoc14	stagedoc14_workspace	2024-02-19 18:27:43	20.20.14.18	9053:22
ms_blog_php-fpm_1	running	stagedoc10	stagedoc10_nginx	2024-02-16 14:11:57	172.18.0.17	-
ms_blog_percona_1	running	stagedoc4	stagedoc4_nginx	2024-02-16 14:05:39	172.18.0.10	-
ms_blog_workspace_1	running	stagedoc4	stagedoc4_php-fpm	2024-02-16 14:05:25	20.20.4.19	-
ms_auth_nginx_1	running	stagedoc4	stagedoc4_workspace	2024-02-16 14:05:22	20.20.4.21	9045:22
ms_auth_php-fpm_1	running	stagedoc2	stagedoc2_nginx	2024-02-16 14:02:54	172.18.0.9	-
ms_auth_percona_1	running	stagedoc22	stagedoc22_nginx	2024-02-14 15:42:26	172.18.0.3	-
ms_payment_nginx_1	running	stagedoc22	stagedoc22_php-fpm	2024-02-14 15:42:14	20.20.22.19	-
ms_payment_php-fpm_1	running	stagedoc22	stagedoc22_workspace	2024-02-14 15:42:11	20.20.22.21	9058:22

Рисунок 3.2 – Список функціонуючих контейнерів усіх мікросервісів

В результаті вищеописаних дій було розгорнуто усі мікросервіси, а також Kong API Gateway, як окремий контейнер.

3.3 Тестування мультисайтової мікросервісної архітектури

Тестування розробленої системи є критичним етапом у процесі створення надійного та ефективного програмного забезпечення.

Для тестування API використовуватиметься інструмент Postman, який надає можливість комплексної перевірки як окремих мікросервісів, так і всієї архітектури в цілому. Postman дозволяє створювати та автоматизувати запити до API, перевіряти правильність їх виконання та аналізувати результати [32].

Процес тестування було почато з перевірки кожного окремого мікросервісу на відповідність заданим функціональним вимогам. Це включає тестування CRUD операцій, аутентифікації та інших функцій, які забезпечують

основну бізнес-логіку кожного мікросервісу. Перевірено коректність виконання запитів на створення та отримання товару у мікросервісі товарів, результат відповіді API мікроервісу представлено на рисунку 3.3.

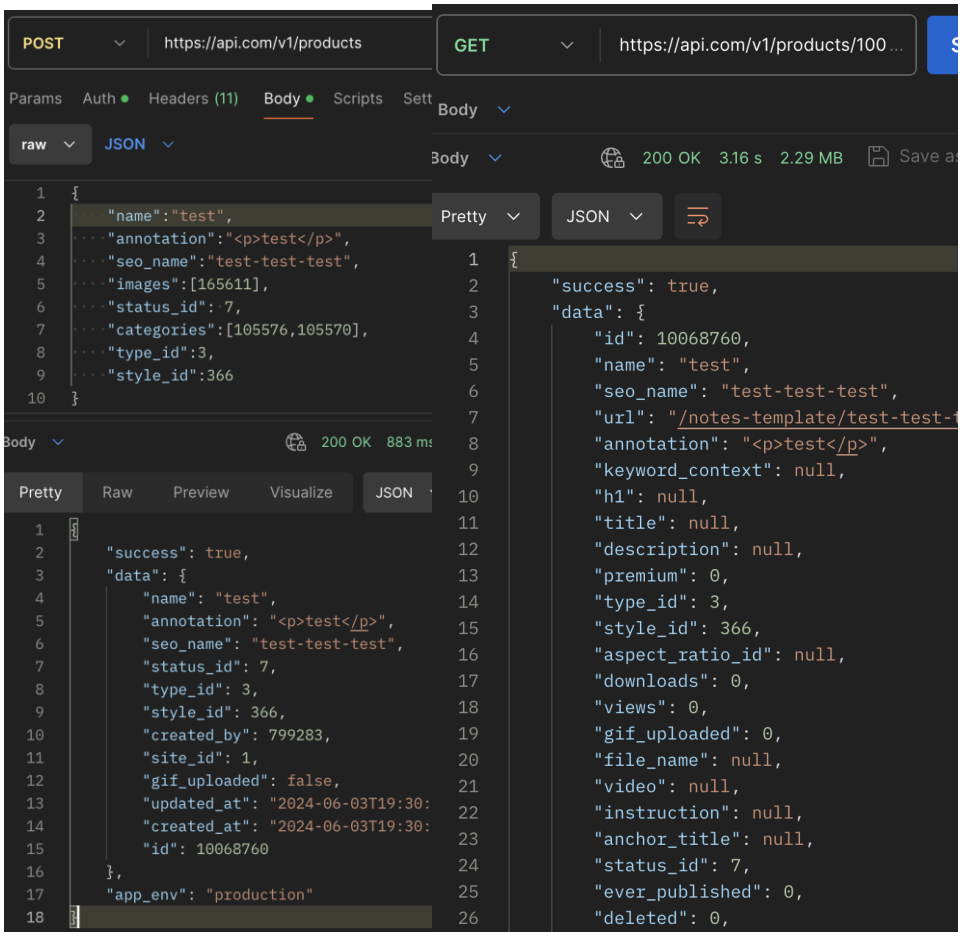


Рисунок 3.3 – Запити на створення та отримання товару

Після перевірки CRUD операцій було перевірено коректність роботи реєстрації та аутентифікації користувача. На рисунку 3.4 подано результати виконання цих запитів, які мають 200 код відповіді, тобто виконались успішно. Перший запит був надісланий з метою створення товару в мікросервісі товарів, у якості відповіді на який було відправлено дані про створений товар. Другий запит був надісланий для отримання товару по його ідентифікатору з мікросервісу товарів. Відповідь містить в собі повну інформацію про товар, яка зберігається в базі даних та у Elasticsearch.

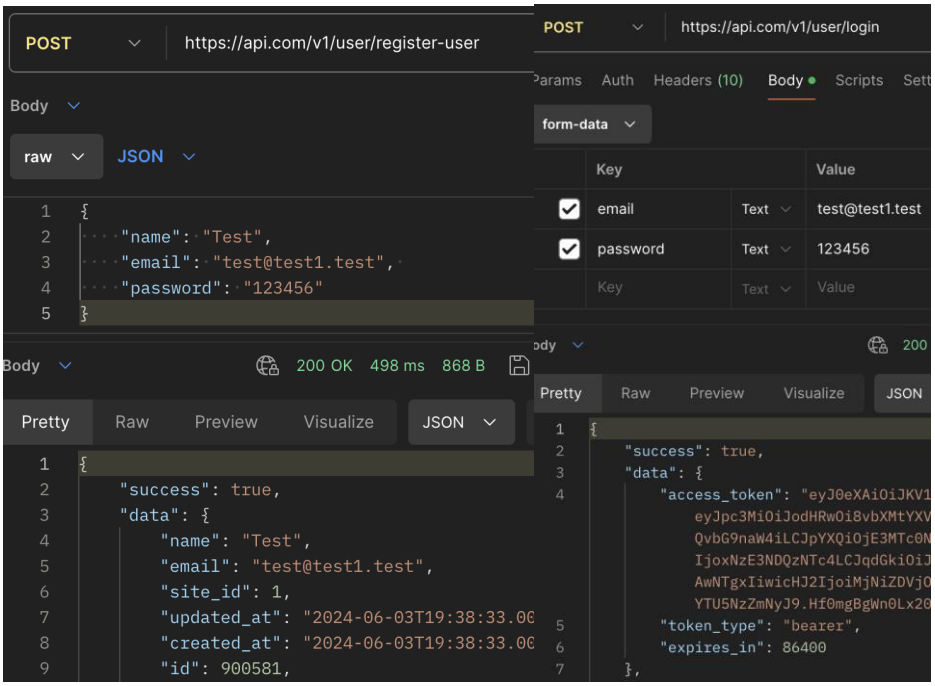


Рисунок 3.4 – Результат перевірки коректності роботи реєстрації та аутентифікації користувача

Далі було перевірено мікросервіс оплат, який відповідальний за обробку платежів. На рисунку 3.5 представлено результат виконання запиту на обробку платежу, дані до якого надає платіжна система одразу після оплати замовлення користувачем. Оскільки запит виконано успішно, можна дійти висновку, що тестування пройдено.

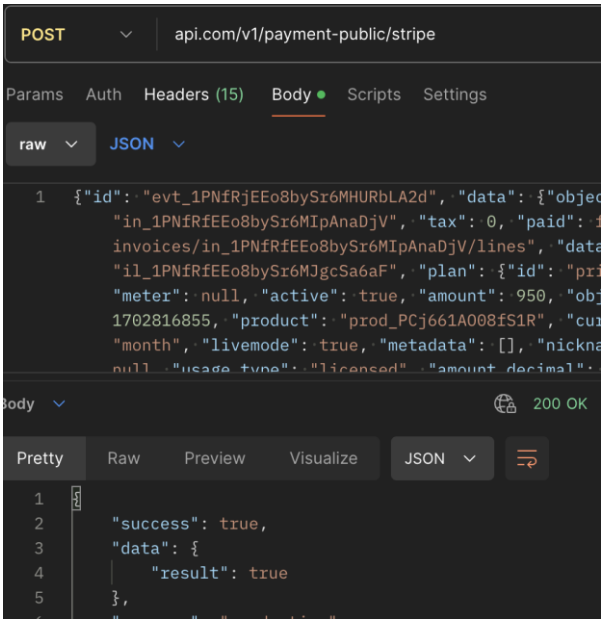


Рисунок 3.5 – Результат виконання запиту на обробку платежу

За допомогою програмного забезпечення Postman було перевірено роботу API усіх мікросервісів на відповідність вимогам мікросервісної архітектури та архітеткрному стилю REST.

3.3.1 Застосування розробленої мультисайтової мікросервісної архітектури

Для наглядної демонстрації розробленого програмного забезпечення, а саме мікросервісів з підтримкою мультисайтовості, було окремо розроблено публічний сайт, за зміст, наповнення та функціонал якого повністю відповідає впроваджена мікросервісна архітектура з підтримкою мультисайтовості.

На рисунку 3.6 представлено головну сторінку сайту Thegoodocs.

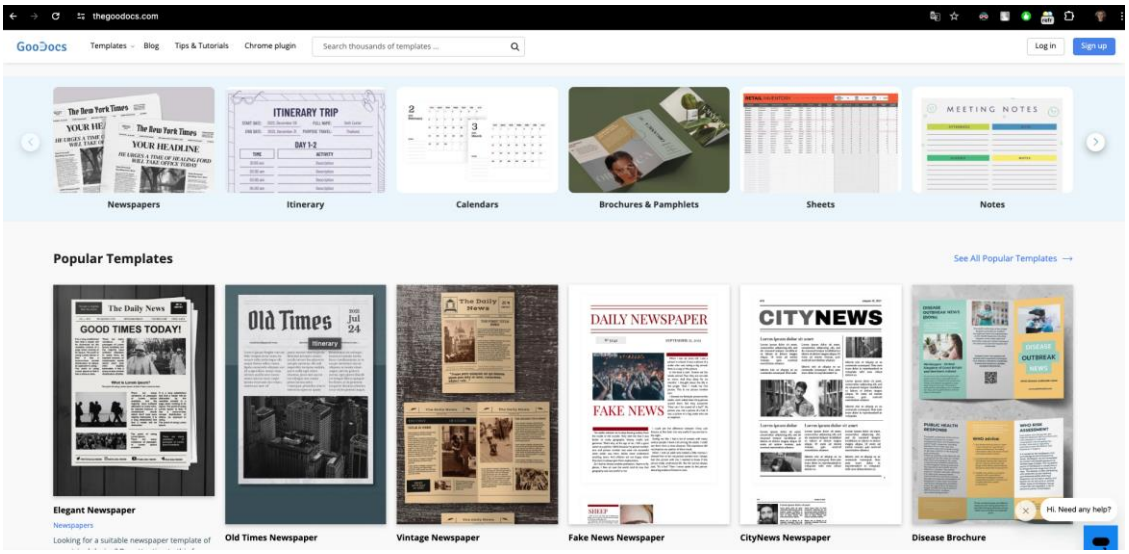


Рисунок 3.6 – Головна сторінка сайту Thegoodocs

Представлений контент на вищевказаному рисунку, такий як категорії, товари, категорії блогу, пошук, реєстрація та авторизація – повністю працює, як згадувалось раніше, використовуючи розроблені у даній кваліфікаційній роботі мікросервіси.

Товар на цьому сайті являє собою шаблон (template) Google Docs, Slides або Sheets для використання користувачами у своїх цілях. На рисунку 3.7 можна побачити як виглядає сторінка товару. На ній представлено меню

категорій зліва, також є можливість завантажити шаблон, або додати у колекцію. Також є можливість переглянути у сайдбарі перелік категорій інших товари, нижче представлена повна інформація про товар та товари від автора даного товару та схожі на нього. Оскільки на сайті є преміум контент, завантаження шаблонів доступно завдяки оплаті замовлення на купівлю підписки, яка дозволяє завантажувати преміум товари. Про покупку підписки буде описано далі.

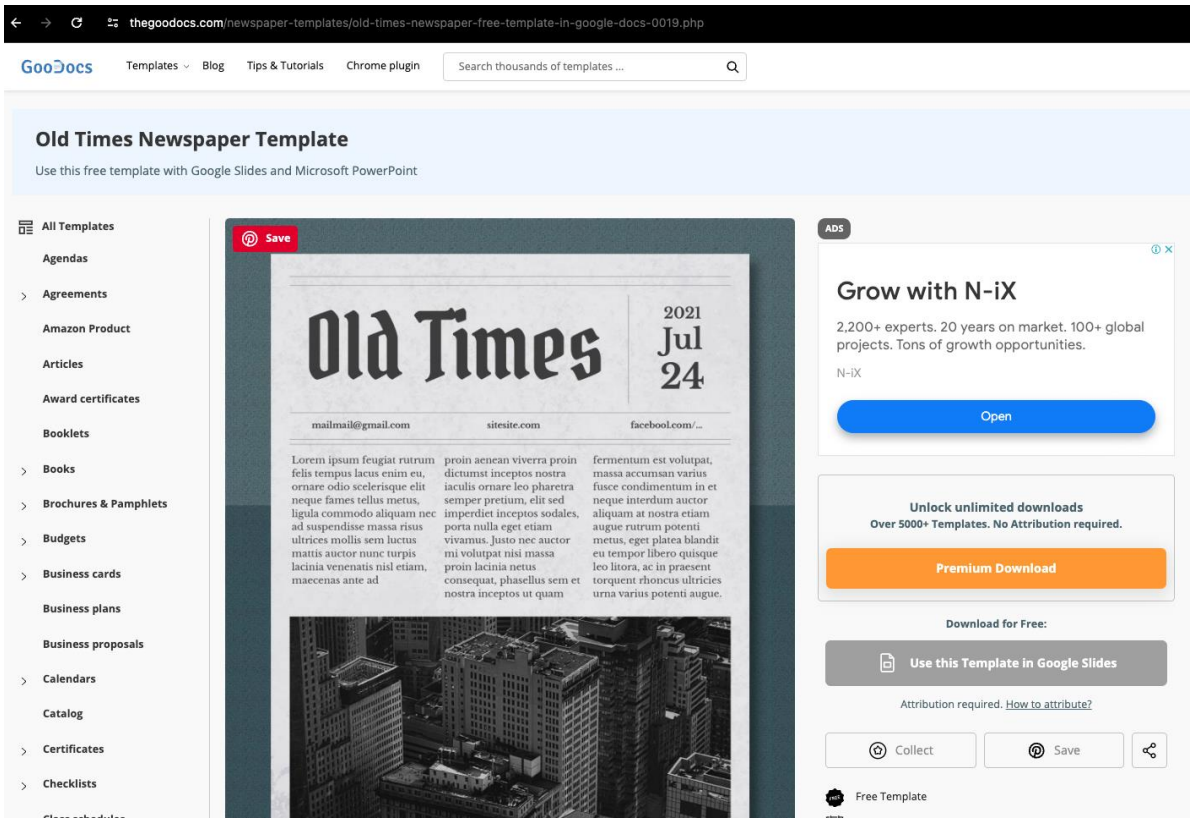


Рисунок 3.7 – Сторінка шаблону на сайті Thegoodocs

Також на сайті присутній розділ з блогом, мікросервіс для якого теж був окремо реалізований у даній кваліфікаційній роботі. На рисунку 3.8 показано як виглядає головна сторінка розділу блогу. На ній представлені категорії блогу та перелік самих постів. Кожен пост блогу має свою окрему категорію, до якої він належить та на сторінці якої він буде відображений. Блог на даному сайті описує основний принцип взаємодії з представленими товарами, пост може також містити в собі підбірку товарів, посилання на інші сайти, картинки, тощо.

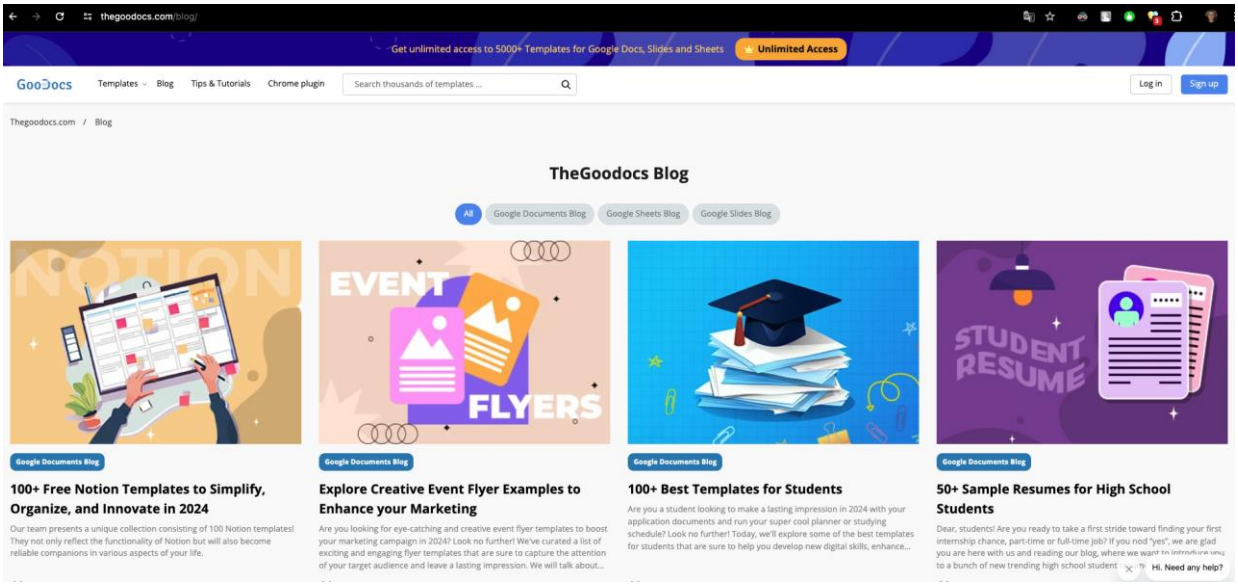


Рисунок 3.8 – Сторінка розділу блогу на сайті Thegoodocs

Також на вищевказаному сайті впроваджено функціонал покупки підписки за допомогою спроєктованого мікросервісу оплат. На рисунку 3.9 подано вигляд сторінки оплати замовлення за допомогою платіжної системи Stripe.

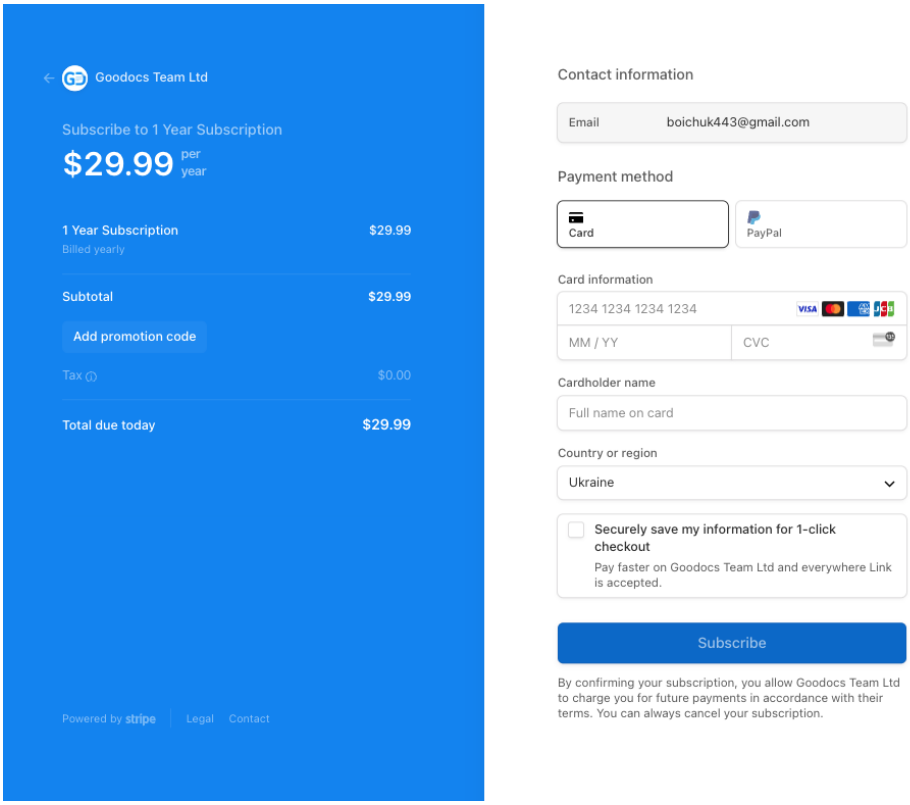


Рисунок 3.9 – Сторінка оплати замовлення на сайті Thegoodocs

Отже, весь функціонал, який був розроблений у мікросервісах, які в свою чергу були розроблені у даній кваліфікаційній роботі, було успішно застосовано у розробці реального сайту.

3.4 Висновок до третього розділу

В третьому розділі кваліфікаційної роботи було виконано кілька ключових етапів, які забезпечили успішну розробку, інтеграцію та тестування мікросервісної архітектури з підтримкою мультисайтовості.

Спочатку було детально описано розробку кожного окремого мікросервісу. Розробка включала реалізацію CRUD операцій на мікросервісах категорій, товарів та блогу, забезпечення необхідних механізмів аутентифікації та реєстрації на мікросервісі авторизації, а також обробки оплати замовлення на мікросервісі оплат. Кожен мікросервіс був спроєктований таким чином, щоб забезпечити максимальну незалежність та гнучкість, що дозволяє легко модифікувати або додавати нові функції без впливу на інші компоненти системи.

Далі було описано процес інтеграції та налаштування мікросервісів за допомогою Docker. Всі мікросервіси були контейнеризовані, що забезпечило ізоляцію середовищ виконання та спростило процес розгортання. Конфігурація Docker була розроблена таким чином, щоб забезпечити легке управління контейнерами та досягнути масштабованості. Крім того, було налаштовано Kong API Gateway, який забезпечує централізоване управління доступом до API мікросервісів та підвищує безпеку системи.

На завершення було проведено комплексне тестування розробленої системи. Тестування API виконувалося за допомогою Postman, що дозволило ретельно перевірити кожен мікросервіс окремо та у взаємодії з іншими сервісами. Крім того, розроблене програмне забезпечення було застосовано у розробці реального сайту (Thegoodocs), що підтвердило його працездатність та ефективність у умовах реального використання.

РОЗДІЛ 4. БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

4.1 Ергономічні проблеми безпеки життєдіяльності

Аналіз основоположних принципів ергономіки дозволяє зробити висновок про її тісний взаємозв'язок з охороною праці. Згідно з визначенням існуючого державного стандарту охорона праці — це система законодавчих, соціально-економічних, технічних, санітарно-гігієнічних, організаційних заходів і засобів, що забезпечують безпеку праці, зберігають здоров'я й працездатність людини в процесі праці. Вчені засвідчують, що зв'язки між безпекою праці та ергономікою настільки переплетені, що між ними важко провести чітку межу. Роботи, які пов'язані з обома галузями знань, взаємно доповнюють одна одну з метою пристосування обладнання, машин і механізмів до природних можливостей людини. Підтвердженням тісних зв'язків між охороною праці та ергономікою є взаємний вплив їх одна на одну в галузі нормування та стандартизації. Необхідно зауважити, що цілий ряд стандартів системи безпеки праці вміщує в себе загальні ергономічні вимоги. У той же час ергономічні рекомендації, особливо при висвітленні гігієнічних вимог до навколишнього середовища, практично базуються на нормативних вимогах охорони праці.

Завданням охорони праці є усунення небезпечних і шкідливих факторів виробничого середовища, зниження їх негативного впливу на людину у випадку неможливості їх повного усунення. Завданням ергономіки є забезпечення ефективної взаємодії людини й техніки. Розвиток ергономіки зумовлений об'єктивними процесами інтеграції й взаємного проникнення та збагачення соціально-економічних, природознавчих і технічних наук, які досліджують людину в процесі її діяльності.

Основними завданнями ергономіки є:

– вивчення умов праці;

– виявлення конструктивних недоліків виробничого обладнання;

– оцінка організації робочих місць з точки зору забезпечення нормальної робочої зони, допустимих швидкостей, траєкторій, кількості рухів і зусиль, необхідних для обслуговування виробничого обладнання;

– вивчення інформаційної взаємодії оператора й машини.

Для розв'язання різних ергономічних завдань застосовують методи дослідження характеру та організації праці, методи спостереження та опитування, операційно-структурного опису трудової діяльності, хронометражні, антропометричні, біомеханічні, фізичні, фізіологічні, психологічні, гігієнічні, економічні методи та ін. Залежно від особливостей досліджуваної системи "людина — машина — середовище" добирають комплекс методів, який може в одних випадках бути націлений на розкриття конструктивних недоліків виробничого обладнання, що призводять до погіршення умов праці, а в інших — на оцінку конструктивних особливостей органів керування, організації робочого місця і т. ін. Спеціальний комплекс методів повинен застосовуватись при вивченні інформаційної взаємодії оператора й машини. Важливе значення має застосування адекватних методів досліджень, за допомогою яких можна виявити працездатність найбільш завантажених систем організму людини [39].

Одержавши у своє розпорядження величезні запаси енергії, нову техніку й технології, вона змінила своє життя, але разом з тим постала перед складним завданням – забезпечити ефективне, стійке та безпечне керування цією технікою.

При вирішенні завдань пов’язаних з поліпшенням умов праці, необхідне детальне вивчення системи людина-машина (СЛМ). СЛМ – це складна багатофункціональна система, яка включає в себе людський і технічний фактори і має такі складові:

– машина – усе те, що штучно створено руками людини для задоволення своїх потреб (технічні пристрої, інформаційне забезпечення);

– людина – людина-оператор, при взаємодії з машиною виконує деякі функції для досягнення поставленого завдання;

– навколишнє середовище – визначається такими параметрами, як освітленість, шум, випромінювання, температура, вологість тощо;

– робоче місце – окреслюється положенням оператора при виконанні своїх обов’язків;

– органи керування (ОК) – за допомогою їх людина керує об'єктами;

– засоби відображення інформації (ЗВІ) – завдяки їм людина слідкує застаном машини (виробничого процесу).

Одним із важливих завдань СЛМ є розподіл функцій між людиною і машиною, який повинен ураховувати їх можливості. Однак загальне рішення складно отримати, оскільки кожна система характеризується своїми особливостями. На основі порівняння можливостей людини і машини в системах керування можна запропонувати наведений далі варіант розподілу функцій.

Людина виконує такі функції:

– індуктивно мислить, тобто приймає рішення на базі неповної інформації, узагальнення різних факторів, доповнюючи інформацію з власного досвіду;

– вирішує завдання, стосовно яких відсутні правила;

– вибирає шляхи вирішення завдань в умовах, що швидко змінюються.

Машині доцільно передати такі функції:

– виконання громіздких математичних розрахунків та вибір відомих варіантів розв’язання;

– здійснення одноманітних операцій за відомим алгоритмом;

– збереження великої кількості інформації;

– виконання швидких дій у відповідь на певну команду.

Процеси приймання, переробки інформації та прийняття рішень і виконання оператором керуючих дій поєднанні в цілісну діяльність, яка полягає у гарантуванні функціонування СЛМ. Для ефективного забезпечення роботи машини людина-оператор повинна зручно себе відчувати. Таке можливо при виконанні певних вимог, які ставляться до машини і навколишнього середовища, а саме: до розміщення засобів відображення інформації та засобів керування, робочого місця, а також до освітлення, клімату, шуму, вібрації, що

можуть впливати як на фізичний стан людини, так і на протікання технологічного процесу [40].

4.2 Загальні вимоги безпеки з охорони праці для користувачів ПК

Площа робочого місця для обслуговування, ремонту й налагодження ПЕОМ повинна бути не менш 10 м², причому робочі місця повинні бути відділені одне від іншого перегородками й розташовуватися не ближче одного метра від приладів опалення. Кришки робочих столів або стендів повинні бути покриті не струмопровідними гладкими матеріалами, які можна легко помити. На робочому місці для обслуговування, ремонту й налагодження ПЕОМ повинні бути електророзетки на напругу 12 і 36 В та підставки для паяльників. Для підключення ПЕОМ і установок для ремонту й налагодження на робочому місці повинен бути електрощит з ізоляційного матеріалу.

Щодня перед початком роботи необхідно проводити очищення екрана відеотермінала від пилу й інших забруднень.

Після закінчення роботи ПЕОМ і периферійні пристрої повинні бути відключені від електричної мережі, а при виникненні аварійної ситуації відключення необхідно виконати негайно. Монтаж, підключення й відключення кабелів, ремонт ПЕОМ варто виконувати тільки при повністю відключеному живленні.

При необхідності виконання робіт при включеному живленні роботи повинні виконуватися не менш чим двома працівниками, використовувати інструмент із ізоляційними ручками й стояти на діелектричному килимку.

При виконанні ремонтних робіт варто користуватися електроінструментом з номінальною напругою не більше 36 В.

Забороняється виконання ремонтних робіт з ручними годинниками з металевим браслетом.

Державними санітарними правилами встановлюється такий режим праці й відпочинку при роботі з ПЕОМ при 8-годинній робочій зміні залежно від характеру праці: для розроблювачів програм варто призначати регламентовані

перерви для відпочинку тривалістю 15 хвилин щогодини при роботі з ПЕОМ; для операторів ПЕОМ варто призначати регламентовані перерви для відпочинку тривалістю 15 хвилин через кожні дві години; для операторів комп'ютерного набору варто призначати регламентовані перерви для відпочинку тривалістю 10 хвилин щогодини роботи з ПЕОМ.

У всіх випадках, коли виробничі обставини не дозволяють використати регламентовані перерви, тривалість безперервної роботи з ПЕОМ не повинна перевищувати чотирьох годин.

При 12-годинній робочій зміні регламентовані перерви повинні бути аналогічними перервам при 8-годинній робочій зміні, а протягом інших чотирьох годин роботи, незалежно від характеру трудової діяльності, щогодини тривалістю 15 хвилин.

Для зниження нервово-емоційної напруги, стомлення очей, поліпшення мозкового кровообігу, подолання наслідків гіподинамії доцільно використати перерви для виконання вправ, наведених у додатку 7 «Державних санітарних правил...» ДсанПІН 3.32.007-98.

Активний відпочинок дозволяє зняти нервову напругу, оновити, відновити функції фізіологічних систем, які порушуються в процесі праці, зняти втому очей, поліпшити мозковий кровообіг. Крім того, рекомендоване психологічне розвантаження в спеціально відведених приміщеннях під час регламентованих перерв або наприкінці робочого дня (додаток 8 ДСанПІН 3.32.007-98).

Для збереження здоров'я користувачів ПЕОМ, виключення професійних захворювань і підтримки працездатності варто передбачати регламентовані перерви для відпочинку протягом зміни.

При виконанні протягом дня робіт з ПЕОМ, які займають не менш 50% тривалості робочої зміни, повинні передбачатися перерви: для відпочинку й прийому їжі (обідня перерва); для відпочинку й особливих потреб (відповідно до трудових норм); додаткові перерви, які вводяться для окремих професій з урахуванням особливості трудової діяльності.

В окремих випадках, при постійних скаргах на зорову втому тих, хто працює перед відеотерміналом, при дотриманні санітарно-гігієнічних вимог до режиму праці та відпочинку, а також вимог щодо застосування індивідуальних засобів локального захисту очей, допускається індивідуальний підхід до обмеження тривалості робіт перед відеотерміналом, зміни змісту роботи, чергування з іншими видами діяльності, не пов’язаними з відеотерміналом [40].

4.3 Висновок до четвертого розділу

В четвертому розділі кваліфікаційної роботи описано основні вимоги безпеки та охорони праці для користувачів персональних електронно-обчислювальних машин (ПЕОМ). Визначено, що площа робочого місця для обслуговування, ремонту й налагодження ПЕОМ повинна бути не меншою за 10 м², причому робочі місця повинні бути відділені перегородками та розташовуватися не ближче одного метра від приладів опалення.

Крім того, підкреслено необхідність регулярного очищення екрану ПЕОМ від пилу й інших забруднень перед початком роботи, а також відключення ПЕОМ і периферійних пристроїв від електромережі після завершення роботи. Особливу увагу приділено вимогам безпеки при проведенні ремонтних робіт: їх слід виконувати тільки при відключеному живленні або, якщо це неможливо, не менш ніж двома працівниками з використанням інструменту з ізоляційними ручками та діелектричного килимка. Забороняється виконання ремонтних робіт з ручними годинниками з металевим браслетом. Також визначено режим праці й відпочинку для користувачів ПЕОМ, включаючи регламентовані перерви для різних категорій працівників, з метою збереження їхнього здоров'я та запобігання професійним захворюванням.

ВИСНОВКИ

У даній кваліфікаційній роботі було досліджено та розроблено backend частину мультисайтової мікросервісної архітектури. Робота охоплює як теоретичні, так і практичні аспекти, спрямовані на створення ефективної, масштабованої та надійної системи.

В першому розділі кваліфікаційної роботи освітнього рівня «Бакалавр»:

– подано детальний аналіз предметної області, зокрема мультисайтової архітектури;

– розглянуто переваги та недоліки мікросервісної архітектури;

– висвітлено існуючі рішення забезпечення мультисайтовості;

– проаналізовано вимоги до backend частину мікросервісної архітектури та сплановано її розробку;

– обґрунтовано вибір використовуваних технологій.

В другому розділі кваліфікаційної роботи:

– досліджено процес проєктування мікросервісної архітектури;

– обґрунтовано вибір мікросервісів та їх функціональність;

– сформовано бази даних та стек (набір) контейнерів Docker для кожного мікросервісу.

В третьому розділі кваліфікаційної роботи:

– розроблено окремі мікросервіси для кожної окремої сутності, такі як товари, категорії, блог, оплата та авторизація.

– запропоновано інтеграцію та налаштування мікросервісів з використанням Docker, а також налаштовано Kong API Gateway;

– спроєктовано та протестовано backend частину мікросервісної архітектури, зокрема API за допомогою Postman, і застосовано її у розробці реального сайту.

У розділі «Безпека життєдіяльності, основи охорони праці» висвітлено ергономічні проблеми безпеки життєдіяльності, які виникають при веб-розробці. Також розглянуто загальні вимоги безпеки з охорони праці для користувачів ПК.

ПЕРЕЛІК ДЖЕРЕЛ

1. Multi-Site Solutions. *Integral Vision*. URL: https://integralvision.eu/en/expertise/multi-site-solutions (дата звернення: 04.02.2024).

2. Benefits of Using a Multisite for Your Multi-organizational websites. *Bring Results*. URL: https://bringresults.com/blog/2021/04/22/10-benefits-of-using-a-multisite-for-your-multi-organizational-websites/ (дата звернення: 04.02.2024).

3. Benefits of Using a Multisite. *Amplimark*. URL: https://www.amplimark.com/benefits-of-using-a-multisite/ (дата звернення: 04.02.2024).

4. Мікросервісна архітектура для початківців. *GlobalLogic*. URL: https://www.globallogic.com/ua/insights/blogs/microservices-architecture-for-beginners-part-one/ (дата звернення: 04.02.2024).

5. Мікросервісна архітектура: плюси та мінуси. *ITEDU Center*. URL: https://itedu.center/ua/blog/articles/microservices-architecture-advantages-and-disadventages/ (дата звернення: 04.02.2024).

6. Мікросервісна архітектура: переваги та недоліки. *Javarush*. URL: https://javarush.com/ua/groups/posts/uk.2015.mkroservsna-arkhtektura-pljusi-ta-mnusi (дата звернення: 04.02.2024).

7. Мікросервіси та мікросервісна архітектура: сучасний підхід до розробки програмного забезпечення. *Bizmag*. URL: https://bizmag.com.ua/arkhitektura-mikroservisiv-dlia-biznesu/ (дата звернення: 04.05.2024).

8. Мікросервіси. *Вікіпедія*. URL: https://uk.wikipedia.org/wiki/Мікросервіси (дата звернення: 04.02.2024).

9. Multi-Site: Key Features and How To Choose The Right One. *Sanity*. URL: https://www.sanity.io/multi-site-cms (дата звернення: 05.02.2024).

10. What is WordPress Multisite?. *HubSpot*. URL: https://blog.hubspot.com/website/wordpress-multisite (дата звернення: 05.02.2024).

11. Multisite Drupal. *Drupal*. URL: https://www.drupal.org/docs/getting-started/multisite-drupal (дата звернення: 05.05.2024).

12. Magento. *Вікіпедія*. URL: https://uk.wikipedia.org/wiki/Magento (дата звернення: 05.05.2024).

13. Shopify та Shopify Plus. *ADW Service*. URL: https://adwservice.com.ua/uk/porivnjannja-shopify-ta-shopify-plus (дата звернення: 06.02.2024).

14. Стадії циклу розробки ПЗ. *QALight*. URL: https://qalight.ua/baza-znaniy/stadiyi-tsiklu-rozrobki-pz/ (дата звернення: 05.02.2024).

15. Мікросервісна архітектура ПЗ. *QALight*. URL: https://qalight.ua/baza-znaniy/shho-take-mikroservisna-arhitektura-pz/ (дата звернення: 05.05.2024).

16. Вимоги до мікросервісної архітектури. *Medium*. URL: https://medium.com/@IvanZmerzlyi/microservices-architecture-461687045b3d (дата звернення: 07.05.2024).

17. Kong: an open-source API gateway. *Medium*. URL: https://medium.com/@aryak.deshpande0512/kong-an-open-source-api-gateway-ffc91f474216 (дата звернення: 08.05.2024).

18. Мови програмування для веброзробки та супутні технології. *Drukarnia*. URL: https://drukarnia.com.ua/articles/movi-programuvannya-dlya-vebrozrobki-ta-suputni-tekhnologiyi-wzeAr (дата звернення: 08.05.2024).

19. Огляд переваг Laravel для веб-розробки. *ASABIX*. URL: https://asabix.com.ua/what-is-laravel/ (дата звернення: 08.05.2024).

20. Що таке Docker і навіщо він?. *QA Group*. URL: https://qagroup.com.ua/publications/shcho-take-docker-i-navishcho-vin/ (дата звернення: 08.05.2024).

21. NGINX: продуктивний і популярний веб-сервер. *Brander*. URL: https://brander.ua/technologies/nginx (дата звернення: 08.05.2024).

22. Percona Server. *Вікіпедія*. URL: https://uk.wikipedia.org/wiki/Percona_Server (дата звернення: 08.05.2024).

23. Загальна інформація про Redis. *Ukraine hosting*. URL: https://www.ukraine.com.ua/wiki/redis/overview/ (дата звернення: 18.05.2024).

24. Elasticsearch розробка швидкого пошуку і фільтрації. *SEOTM Company*. URL: https://www.seotm.com/ua/technologies/elasticsearch-rozrobka.html (дата звернення: 18.05.2024).

25. Kong Gateway. *Kong Docs*. URL: https://docs.konghq.com/gateway/latest/ (дата звернення: 18.05.2024).

26. Мікросервісна архітектура: основні концепції. *Foxminded*. URL: https://foxminded.ua/mikroservisna-arkhitektura/ (дата звернення: 18.05.2024).

27. Архітектура мікросервісів: Особливості. *Hostzealot*. URL: https://www.hostzealot.com.ua/blog/about-solutions/arxitektura-mikroservisiv-osoblivosti-perevagi-realni-prikladi (дата звернення: 25.05.2024).

28. MVC: шаблон проектування архітектури додатку. *Brander*. URL: https://brander.ua/technologies/mvc (дата звернення: 25.05.2024).

29. Розробка веб-додатків з використанням Laravel. *Step2Dev*. URL: https://step2.dev/uk/blog/laravel (дата звернення: 25.05.2024).

30. Мікросервіси та контейнери Docker. *LiderBooks*. URL: https://liderbooks.com.ua/ua/p1852830587-mikroservisy-kontejnery-docker.html (дата звернення: 25.05.2024).

31. Laradock. *Laradock*. URL: https://laradock.io/ (дата звернення: 25.05.2024).

32. Використання Postman в тестуванні. *QATestLab*. URL: https://training.qatestlab.com/blog/technical-articles/use-postman-in-testing/ (дата звернення: 26.04.2024).

33. Fryz M., Mlynko B. Property Analysis of Conditional Linear Random Process as a Mathematical Model of Cyclostationary Signal // 2nd International Workshop on Information Technologies: Theoretical and Applied Problems (ITTAP 2022). Ternopil, Ukraine: CEUR Workshop Proceedings, 2022. Vol. 3309. P. 77–82.

34. Fryz M., Mlynko B. Determination of the characteristic function of discrete-time conditional linear random process and its application // Sci. J. TNTU. 2023. Vol. 109, № 1. P. 16–23.

35. Fryz M., Mlynko B. Property analysis of multivariate conditional linear random processes in the problems of mathematical modelling of signals // Technol. Audit Prod. Reserv. 2022. Vol. 3, № 2(65). P. 29–32.

36. Фриз М.Є., Млинко Б.Б. Умовні лінійні випадкові процеси з дискретним часом та їх властивості // Вісник Хмельницького національного університету. Серія: Технічні науки. 2022 (309), № 3. С. 7–12.

37. Fryz M., Mlynko B. Properties of Stationarity and Cyclostationarity of Conditional Linear Random Processes // 2020 IEEE 15th International Conference on Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering (TCSET). Lviv-Slavske, Ukraine: IEEE, 2020. P. 166–170.

38. Fryz M., Scherbak L., Mlynko B., Mykhailovych T. Linear Random Process Model-Based EEG Classification Using Machine Learning Techniques // Proceedings of the 1st International Workshop on Computer Information Technologies in Industry 4.0 (CITI 2023). Ternopil, Ukraine: CEUR Workshop Proceedings, 2023. Vol. 3468. P. 126–132.

39. ДСТУ 8604:2015 Дизайн і ергономіка. Робоче місце для виконання робіт у положенні сидячи. *Національний стандарт України.* URL: https://nau.edu.ua/site/variables/docs/science/ndi/dizain/0%A1%A2%D0%A3-D1%8F%D1%87%D0%B8.doc (дата звернення 26.05.2024).

40. Голінько В. І., Іконніков М. Ю., Лебедєв Я. Я. Охорона праці в галузі інформаційних технологій. Дніпропетровськ, Україна: Міністерство освіти і науки України, Національний гірничий університет. С. 246. ISBN 978-966-350-558-9.

ДОДАТКИ

Додаток А

Програмний код мікросервісу товарів

api.php

```
Route::get('healthcheck',
[\App\Http\Controllers\SystemController::class, 'healthcheck']);
Route::group(['middleware'=> 'check.auth'], function () {
Route::group(['middleware'=>
'check.permission:Admin|Designer|ContentManager'], function () {
Route::group(['prefix' => 'items'], function () {
     Route::get('admin/getStatistics',
[\App\Http\Controllers\Products\AdminController::class,
'getStatistics']);
       Route::put('admin/quickUpdate/{id}',
[\App\Http\Controllers\Products\AdminController::class,
'quickUpdate']);
       Route::get('admin/getListWithoutRelations',
[\App\Http\Controllers\Products\AdminController::class,
'getListWithoutRelations']);
       Route::resource('admin',
\App\Http\Controllers\Products\AdminController::class);
       Route::get('admin/checkBySeoName/{seoName}',
[\App\Http\Controllers\Products\AdminController::class,
'checkBySeoName']);
       Route::get('admin/getBySeoName/{seoName}',
[\App\Http\Controllers\Products\AdminController::class,
'getBySeoName']);
       Route::post('admin/filter',
[\App\Http\Controllers\Products\AdminController::class,
'filter']);
   Route::group(['prefix' => 'public'], function () {
});
Route::group(['prefix' => 'system'], function () {
Route::get('/getAllowableImageSizes',[\App\Http\Controllers\Produc
tAdditionalFiles\AdminController::class,
'getAllowableImageSizes']);
```

```
Route::get('/php-ini-all',
[\App\Http\Controllers\Products\SystemController::class,
'getPhpIniParameters']);
        Route::get('/php-ini-maxsize',
[\App\Http\Controllers\Products\SystemController::class,
'getMaxSizesPhpIniParameters']);
        Route::post('/clear-cache',
[\App\Http\Controllers\Products\SystemController::class,
'clearCache']);
    });
});
});
Route::group(['prefix' => 'products'], function () {
    Route::resource('/admin', AdminController::class);
    Route::get('/all',
    [PublicController::class,'getAllProducts']
    );
    Route::get('/{category_slug}/{product_slug}',
    [PublicController::class, 'getItem']
    );
});
```

ProductController.php

```
namespace App\Http\Controllers\Products;
use App\Jobs\UpdateProductstatistics;
use App\Models\TemplateModel;
use App\Models\ProductsourceModel;
use App\Services\CacheService;
use App\Services\LogService;
use App\Services\Translation;
use Illuminate\Contracts\Bus\Dispatcher;
use Illuminate\Http\JsonResponse;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Validator;
use Illuminate\Support\MessageBag;
use Psr\Container\ContainerExceptionInterface;
use Psr\Container\NotFoundExceptionInterface;
```

```
class PublicController extends \App\Http\Controllers\PublicController {
    private bool $cacheEnabled = false;
    public function __construct()
    {
        parent::__construct();
        $this->cacheEnabled = (bool)env('CACHE_ENABLED', false);
    }

    public function getAllProducts(Request $request): JsonResponse
    {
        try{
            $items=Product::where('site_id',
                $request->header('x-app-site-id')
            )->get();
            return $this->successResponse(['items' => $items]);
        }catch (\Exception $exception){
            LogService::error($exception);
            return $this->failedResponse($exception->getMessage());
        }
    }
    public function store(Request $request){
        return$this->successResponse(Product::createItem($request-> validated()));
    }
    public function show($id){
        return $this->successResponse(Product::findItem($id));
    }
    public function update(Request $request, $id){
        return$this->successResponse(Product::updateItem($id, $request->validated());
    }
    public function destroy($id){
        return $this->successResponse(Product::deleteItem($id));
    }
```

```
public function getPremiumField(Request $request): JsonResponse
    {
        try{
            $validator = Validator::make(['seo_name' => $request->get('seo_name')], ['seo_name' => 'required|string']);
            if($validator->fails()) {
                return   $this->validationErrorResponse($validator->errors());
            }
            $item      =      TemplateModel::getBySeoName($request->get('seo_name'), $request->header('x-app-site-id'), ['premium']);

            if(!$item){
                return $this->notFoundResponse();
            }
            return $this->successResponse([
                'premium' => (bool)$item->premium
            ]);
        }catch (\Exception $exception){

            LogService::error($exception);
            return                $this->failedResponse($exception->getMessage());
        }
    }
    public    function    getRelatedProducts(Request    $request): JsonResponse
    {
        try{
            $validator   =   Validator::make(['id'   =>   $request->get('id')], ['id' => 'required|integer']);

            if($validator->fails()) {
                return   $this->validationErrorResponse($validator->errors());
            }
            $items = $this->getTemplateAttributesByTemplateId(
                $request,
```

```
                'id',
                'retrieveRelatedProducts'
            );
            if(!$items){
                return $this->notFoundResponse();
            }
            return $this->successResponse([
                'items' => $items
            ]);
        }catch (\Exception $exception){
            LogService::error($exception);
            Return                $this->failedResponse($exception-
>getMessage());
        }
    }
```

Product.php

```
namespace App\Models;
use App\Concerns\HasRelationshipsRemotely;
use App\Events\UpdateTemplateUrlEvent;
use App\Jobs\AddTemplateToRatingsJob;
use App\Jobs\ImportSourceFilesJob;
use App\Jobs\SetTemplatePublishDateJob;
use App\Jobs\UpdateTemplateStatistics;
use App\Relations\BelongsToManyRemotely;
use App\Relations\HasRemotely;
use App\Services\CdnDoService;
use App\Events\CalculateTemplatesCount;
use App\Services\Client;
class Product extends BaseModel {
    use HasRelationshipsRemotely;
    protected $table = 'templates';
    protected $fillable = [
        'id',
        'name',
        'seo_name',
```

```
        'annotation',
        'keyword_context',
        'h1',
        'title',
        'description',
        'premium',
        'type_id',
        'style_id',
    ];
    protected function keywordContext(): Attribute
    {
        return Attribute::make(
            get: fn ($value) => $value ? json_decode($value) :
$value,
            set: fn ($value) => json_encode($value),
        );
    }
    protected $dates = ['created_at', 'updated_at', 'deleted_at'];
    public function categories(): HasRemotely
    {
        return $this->hasManyRemotely(
            TemplateCategoryModel::class,
            'template_id',
            'id',
            'category-system',
            '/getById',
            'category_id',
            ['id', 'name', 'seo_name', 'parent_id', 'depth',
'url',       'grouping_category',       'anchor_title',       'h1',
'templates_count']
        );
    }
    public function createdBy(): HasRemotely
    {
        return $this->hasManyRemotely(
            TemplateModel::class,
            'id',
```

```
            'id',
            'user-system',
            '/getById',
            'created_by',
            ['id',       'name',       'email',       'seo_name',
'show_on_team_page'],
            true
        );
    }
    public function updatedBy(): HasRemotely
    {
        return $this->hasManyRemotely(
            TemplateModel::class,
            'id',
            'id',
            'user-system',
            '/getById',
            'updated_by',
            ['id', 'email'],
            true
        );
    }
    public function rating(): HasRemotely
    {
        return $this->hasManyRemotely(
            TemplateModel::class,
            'id',
            'id',
            'rating-system',
            '/getByTemplateId',
            'id',
           ['grade', 'grades_count']
        );
    }
    public function translation(): BelongsToManyRemotely{
        return $this->belongsToManyRemotely(
            TemplateModel::class,
```

```
            'id',
            'id',
            'lang-system',
            '/templates/getById',
            'id',
            'template_id',
            ['id',    'name',    'annotation',    'title',    'h1',
'description', 'template_id', 'lang_id']
        );
    }
```

Додаток Б

Програмний код мікросервісу категорій

api.php

```
Route::group(['prefix' => 'items'], function () {
    Route::group(['prefix' => 'categories'], function () {
    Route::resource('/admin', AdminController::class);
    Route::get('/all',[PublicController::class,'getAllCategories']
    );
     Route::get('/{category_slug}',[PublicController::class,
    'getCategoryItem']);
     Route::get('/menu',[PublicController::class,
    'getMenuCategories']);
});
Route::group(['prefix' => 'system'], function () {
    Route::get('getById',[\App\Http\Controllers\Items\SystemContro
    ller::class,'getById']);
    Route::get('checkById',[\App\Http\Controllers\Items\SystemCont
    roller::class, 'checkById']);
    Route::get('/data-for-site-
    map',[\App\Http\Controllers\Items\SystemController::class,
    'getDataForSiteMap']);
    Route::post('/clear-
    cache',[\App\Http\Controllers\Items\SystemController::class,
    'clearCache']);
    Route::put('views/{id}',[\App\Http\Controllers\Items\SystemCon
    troller::class, 'updateViews']);
    Route::put('downloads/{id}',[\App\Http\Controllers\Items\Syste
    mController::class, 'updateDownloads']);
    Route::get('queue-status',
    [\App\Http\Controllers\SystemController::class,
    'queueStatus']);
       });
})
```

CategoryController.php

```
namespace App\Http\Controllers\Items;
use App\Exceptions\GetTemplatesException;
use App\Http\Controllers\Controller;
use App\Http\Middleware\FrontendSecurity;
use App\Models\CacheModel;
use App\Models\CategoryModel;

class                    PublicController                    extends
\App\Http\Controllers\PublicController {

    public function getPopularCategories(): JsonResponse
    {
        try {
            $cacheEnabled = (bool)env('CACHE_ENABLED', false);
            $cacheKey = env("APP_ENV", 'production') . '_popular-
categories.'.request()->header('x-app-site-id');
            if(Cache::has($cacheKey)                              &&
!empty(Cache::get($cacheKey)) && $cacheEnabled) {
                $items = Cache::get($cacheKey);
            } else {
       $items=CategoryModel::getPopularCategories($cacheKey);
            }
            return $this->successResponse([
                'items' => $items,
            ]);
        } catch (\Exception $exception) {
            LogService::error($exception);
            return                $this->failedResponse($exception-
>getMessage());
        }
    }
    public     function     getMenuCategories(Request     $request):
JsonResponse{
        $items=Category::where('site_id',$request->header('x-app-
site-id'))->where('menu', 1)->get();
        return    $this->successResponse(['items'    =>    $items]);
```

```
    }
    public function getTrendingCategories(Request $request): JsonResponse
    {
        try {
            $cacheEnabled = (bool)env('CACHE_ENABLED', false);
            $lang = $request->hasHeader('lang-id') ? (int)$request->header('lang-id') : null;
            $cacheKey = env("APP_ENV", 'production')
                . '_trending-categories.'
                . $request->header('x-app-site-id')
                . ($lang ? ".lang.$lang" : "");
            if(Cache::has($cacheKey) && !empty(Cache::get($cacheKey)) && $cacheEnabled) {
                $items = Cache::get($cacheKey);
            } else {
                $items = CategoryModel::getTrendingCategories($request);
                if($lang){
                    $items = (new Translation($items))->translateArrayItems();
                }
                CacheModel::createIfNotExists($cacheKey);

                Cache::put($cacheKey, $items, env('CACHE_DURATION_SEC', 3600));
            }
            return $this->successResponse(['items' => $items,]);
        } catch (\Exception $exception) {
            LogService::error($exception);
            return $this->failedResponse($exception->getMessage());
        }
    }
    public function store(Request $request){
        return Category::createCategoryItem($request->validated());
```

```
    }
    public function show($id){
        return Category::findCategoryItem($id);
    }
    public function update(Request $request, $id){
        returnCategory::updateCategoryItem($id,$request-
>validated());
    }
    public function destroy($id){
        return Category::deleteCategoryItem($id);
    }
```

Category.php

```
use namespace App\Models;
use App\Concerns\HasRelationshipsRemotely;
use App\Exceptions\GetTemplatesException;
use App\Jobs\UpdateChildCategoryIdsJob;
use App\Relations\BelongsToManyRemotely;
use App\Relations\HasRemotely;
use App\Services\CdnDoService;
use App\Services\Client;
class Category extends BaseModel
{
    use HasRelationshipsRemotely;
    protected $table = 'categories';
    protected $fillable = [
        'id',
        'name',
        'seo_name',
        'anchor_title',
        'title',
        'h1',
    ];
    public function videos(): HasMany
    {
```

```
        return            $this->hasMany(CategoryVideoModel::class,
'category_id', 'id')
            ->select('title','url', 'category_id');
    }
    public                    function                    group():
\Illuminate\Database\Eloquent\Relations\HasOne
    {
        return    $this->hasOne(CategoryGroupModel::class,    'id',
'group_id');
    }
    public                    function                    parent():
\Illuminate\Database\Eloquent\Relations\HasOne
    {
        $relations = ['parent'];
        return      $this->hasOne(CategoryModel::class,      'id',
'parent_id')->with($relations);
    }
    public function translation(): BelongsToManyRemotely
    {
        return $this->belongsToManyRemotely(
            CategoryModel::class, 'id', 'id',
            'lang-system',
            '/categories/getById',
            'id',
            'category_id'
        );
    }
```

Додаток В

Програмний код мікросервісу блогу

api.php

```
Route Route::group(['prefix' => posts], function () {
     Route::resource('/admin', AdminController::class);
     Route::get('/all',[PublicController::class,'getAllPosts']);
     Route::get('/{post_slug}',[PublicController::class,
     'getPostItem');
     Route::get('/category/{category_slug}',[PublicController::clas
     s, 'getBlogCategory']);
});
```

PostController.php

```
namespace App\Http\Controllers\Posts;
use App\Http\Controllers\Controller;
use App\Services\LogService;
class                  PublicController                  extends
\App\Http\Controllers\PublicController
{
    public function getItem(string $categoryGroupSeoName, string
$postSeoName): JsonResponse
    {
        try {
            $siteId = (int)\request()->header('x-app-site-id');
            $cacheKey                                          =
"front.posts.get_item.{$categoryGroupSeoName}.{$postSeoName}.{$sit
eId}";
            if($items = CacheService::get($cacheKey)) {
                $post = $items;
            } else {
                $post  =  BlogPostModel::getBySeoName($postSeoName,
$categoryGroupSeoName, $siteId);
                if(!$post) {
                    return              $this->notFoundResponse();
```

```
                }
                CacheService::put($cacheKey, $post);
            }
            return $this->successResponse([
                'items' => [
                    'item' => $post,
                ],
            ]);
        } catch (\Exception $exception) {
            LogService::error($exception);
            return                $this->failedResponse($exception-
>getMessage());
        }
    }
    public function getPostItem(Request $request,$postSlug){
        $items=Post::where('site_id',$request->header('x-app-site-
id'))->where('slug, ,$postSlug)->first();
        return $this->successResponse(['items' => $items]);
    }

    public function getLatestArticles(): JsonResponse
    {
        try {
            $cacheKey = "front.posts.latest_articles.".\request()-
>header('x-app-site-id');
            if($posts = CacheService::get($cacheKey)) {
                $items = $posts;
            } else {
                $items = BlogPostModel::getLatestArticles();
                CacheService::put($cacheKey, $items);
            }
            return $this->successResponse([
                'items' => $items,
            ]);

        } catch (\Exception $exception) {
            LogService::error($exception);
```

```
            return$this->failedResponse($exception->getMessage());
        }}
```

Blog.php

```
namespace App\Models;
use App\Jobs\AddPostToViewsIndex;
use App\Jobs\UpdatePostStatistics;
use App\Services\CdnDoService;
class Post extends BaseModel
{
    use HasFactory;

    protected $table = 'blog_posts';

    protected $fillable = [
        'id',
        'name',
        'seo_name',
        'title',
        'h1',
        'views',
        'created_at',
    ];
    public function category(): HasOne
    {
        return    $this->hasOne(BlogCategoryModel::class,    'id',
'category_id')
            ->select('id',    'name',    'seo_name',    'status',
'deleted', 'site_id', 'group_id', 'color', 'annotation');
    }
    public function content(){
        return          $this->hasMany(BlogPostContentModel::class,
'post_id', 'id');
    }
public function isActive($post = null): bool {
        if(!$post)                                                    {
```

```
            $post = $this;
        }

        if($post->deleted == 1 && !$post->status) {
            return false;
        }

        return true;
    }
```

Додаток Г

Програмний код мікросервісу оплат

api.php

```
Route::group(['prefix' => 'payment'], function() {
    Route::group(['prefix' => 'public'], function() {
        Route::post('/',
[\App\Http\Controllers\Payment\PublicController::class,
'payment']);
        Route::post('/stripe',
[\App\Http\Controllers\Payment\Stripe\PublicController::class,
'payment']);
        Route::get('/',
[\App\Http\Controllers\Payment\PublicController::class,
'getOrders']);
        Route::delete('/{id}',
[\App\Http\Controllers\Payment\PublicController::class,
'refundOrder']);
    });
});
```

StripePaymentController.php

```
namespace App\Http\Controllers\Payment\Stripe;
use App\Exceptions\OrderNotFoundException;
use App\Models\User;
use Illuminate\Support\Facades\Validator;
class                    PublicController                    extends
\App\Http\Controllers\PublicController {
    public function payment(Request $request): JsonResponse
    {
        try {
            $validator = Validator::make($request->all(), [
                'type' => 'required|string',
                'data' => 'required|array',
                'data.object' => 'required|array',
                'data.object.id' => 'required|string',
                'data.object.object'     =>     'required|string',
```

```
            ]);
            if($validator->fails()) {
                return   $this->validationErrorResponse($validator-
>errors());
            }
            sleep(5);
            $service  =  new  ManageSubscriptionsService($request-
>all());
            $service->processEvent();
            return $this->successResponse([
                'result' => true
            ]);
        }catch (OrderNotFoundException $exception){
            return       $this->notFoundResponseMessage($exception-
>getMessage());
        } catch (\Exception $exception) {
            LogService::error($exception);

            return                $this->failedResponse($exception-
>getMessage());
        }
    }
    public function paymentLink(Request $request) {
        try {
            $stripe=
new\Stripe\StripeClient('test_4eC39HqLyjWDarjtT1zdp7dc');
            $data = $stripe->checkout->sessions->create([
                    'success_url'                                    =>
'https://example.com/success',
                    'line_items' => [[
                'price' => 'price_1MotwRLkdIwHu7ixYcPLm5uZ',
                'quantity' => 2,
            ],
            ],
                    'mode' => 'payment',
                    ]);
                    return    successResponse(['data'    =>$data]);
```

```
        } catch (\Exception $exception) {
            return                $this->failedResponse($exception-
>getMessage());
        }
    }
```

StripePayment.php

```
namespace App\Models;
use App\Exceptions\OrderNotFoundException;
use Illuminate\Database\Eloquent\Builder;
use Illuminate\Database\Eloquent\Model;

class StripePaymentModel extends BaseModel {
    protected $table = 'stripe_payments';
    protected $fillable = [
        'subscription_id',
        'internal_user_id',
         'customer_id',
]
    public      function      getBySubscriptionId(string      $id):
Model|Builder|null
    {
        return $this->query()
            ->where('subscription_id', $id)
            ->where('site_id', request()->header('x-app-site-id'))
            ->first();
       }
}
```

Додаток Д

Програмний код мікросервісу авторизації

api.php

```
Route::group(['prefix' => 'auth'], function () {
  Route::post('/login', [AuthController::class, 'login']);
  Route::post('/register',[AuthController::class, 'register']);
  Route::post('/logout', [AuthController::class, 'logout']);
});
```

AuthController.php

```
namespace App\Http\Controllers\Auth;

use App\Http\Controllers\PublicController;
use App\Http\Requests\Auth\LoginRequest;
use App\Http\Requests\Auth\RegisterRequest;
use App\Http\Requests\Auth\SocialRegisterRequest;
use App\Interfaces\LogInterface;
use App\Interfaces\ResponseInterface;
use App\Services\Entities\UserService;
use Illuminate\Http\JsonResponse;
use Illuminate\Support\Arr;
use Illuminate\Support\Facades\Hash;

class AuthController extends PublicController
{
    public function __construct(
        ResponseInterface $response,
        LogInterface $log,
        protected UserService $userService
    )
    {
        parent::__construct($response, $log);
    }
```

```
    public function register(RegisterRequest $request): JsonResponse
    {
        try{
            $user = $this->userService->create($request->validated());
            return $this->response->success($user);
        } catch (\Exception $exception) {
            $this->log->error($exception);
            return $this->response->failed($exception->getMessage());
        }
    }
    public function login(LoginRequest $request): JsonResponse
    {
        try{
            $credentials = array_merge([
                'deleted' => 0,
            ], $request->validated());
            $user = $this->userService->retrieveByCredentials($credentials, [
                'id',
                'email',
                'password'
            ]);
            if (!$token = $this->attempt($user, $credentials)) {
                return $this->response->unauthorized('Email or password is incorrect');
            }
            return $this->response->withToken($token);
        } catch (\Exception $exception) {
            $this->log->error($exception);
            return $this->response->failed($exception->getMessage());
        }
    }
```

User.php

```
namespace App\Models;
use App\Concerns\HasRelationshipsRemotely;
use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Relations\BelongsToMany;
use Illuminate\Foundation\Auth\User as Authenticatable;
use Illuminate\Notifications\Notifiable;
use Tymon\JWTAuth\Contracts\JWTSubject;
class User extends Authenticatable implements JWTSubject
{
    use HasFactory, Notifiable, HasRelationshipsRemotely;
    protected $table = 'users';
    protected $fillable = [
        'id',
        'name',
        'email',
        'password',
        'developer_id',
        'created_by',
        'updated_by',
        'deleted',
        'deleted_by',
        'deleted_at',
    ];
    protected $hidden = [
        'password',
    ];
    public function getJWTIdentifier(): mixed
    {
        return $this->getKey();
    }
    public function getJWTCustomClaims(): array
    {
        return [];
    }
```