Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

Навчально-науковий інститут комп'ютерних наук та інформаційних технологій
(повне найменування інституту, назва факультету (відділення))

Кафедра інформаційних систем та комп'ютерного моделювання
(повна назва кафедри (предметної, циклової комісії))

Пояснювальна записка

до дипломної роботи

перший (бакалаврський)
(рівень вищої освіти)

на тему: *"Розроблення вебзастосунку афілейт-маркетингу на основі Vue.js та Laravel"*

Виконав: студент 2 курсу групи ІСТС-21
спеціальності
126 " Інформаційні системи та технології "
(шифр і назва напряму підготовки, спеціальності)

Дзіндзюра О.В.
(прізвище та ініціали)

Керівник Сторожук О.Л.
(прізвище та ініціали)

Рецензент Крошний І.М.
(прізвище та ініціали)

Львів – 2024

Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

ННІ комп'ютерних наук та інформаційних технологій
Кафедра інформаційних систем та комп'ютерного моделювання
Рівень вищої освіти перший (бакалаврський)
Спеціальність 126 " Інформаційні системи та технології"
(шифр і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Сторожук О.Л.
"07" 02 2024 року

З А В Д А Н Н Я
НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Дзіндзюри Олександра Володимировича
(прізвище, ім'я, по батькові)

1. Тема роботи *«Розроблення вебзастосунку афілейт-маркетингу на основі Vue.js та Laravel»*
керівник роботи *Сторожук Олександр Леонідович, к.т.н., доцент*,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від *"06"02 2024 року № С-87*
2. Термін подання студентом роботи *10.06.2024р.*
3. Вихідні дані до роботи: Постановка задачі та її формалізація. Основні параметри та вимоги до проектування вебзастосунку афілейт-маркетингу.

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити):
1) Стан проблемної області;
2) Інформаційне та математичне забезпечення;
3) Програмне та технічне забезпечення.

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

6. Дата видачі завдання *7 лютого 2024 року*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1.	*Огляд літературних даних.*	*07.02.2024р. – 21.02.2024р.*	Виконано
2.	*Розділ 1. Стан проблемної області.*	*22.02.2024р. – 03.03.2024р.*	Виконано
3.	*Розділ 2. Інформаційне та математичне забезпечення.*	*04.03.2024р. – 17.03.2024р.*	Виконано
4.	*Розділ 3. Програмне та технічне забезпечення.*	*18.03.2024р. – 19.05.2024р.*	Виконано
5.	*Аналіз отриманих результатів та написання висновків. Оформлення дипломної роботи.*	*20.05.2024р. – 09.06.2024р.*	Виконано
6.	*Здача пояснювальної записки на перевірку керівнику, виправлення помилок та здача роботи рецензенту.*	*10.06.2024 р.*	Виконано

Студент _____ *Дзіндзюра О.В.*
(підпис) (прізвище та ініціали)

Керівник роботи _____ *Сторожук О.Л.*
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Дана дипломна робота містить 52 сторінки поянювальної записки, 14 рисунків, 1 додаток і 15 джерел використаної літератури.

Ця робота зосереджується на розробці та оптимізації пошуково-рекламних веб-додатків. Вона аналізує сучасні методи пошуку, фільтрації та рекомендації товарів, а також інтеграцію аналітичних інструментів. Автор використовує передові веб-технології для швидкої обробки запитів і збереження даних з високою цілісністю.

Робота також розглядає проблеми статистики сайтів з AJAX та виклики перерозподілу обробки даних. Висновки підкреслюють значення розробленого веб-ресурсу для покращення користувацького досвіду онлайн-шопінгу та підвищення ефективності рекламних стратегій.

***Ключові слова**: PHP, JavaScript, API, Мова, Інтерфейс, PostgreSQL*

ABSTRACT

This thesis contains pages of 52 explanatory notes, 14 illustrations, 1 appendices, and 15 sources of used literature.

This paper focuses on the development and optimization of search and advertising web applications. It analyzes modern methods of searching, filtering, and recommending products, as well as the integration of analytical tools. The author uses advanced web technologies to process requests quickly and store data with high integrity.

The paper also discusses the problems of statistics for AJAX sites and the challenges of redistributing data processing. The conclusions emphasize the importance of the developed web resource for improving the user experience of online shopping and increasing the effectiveness of advertising strategies.

Keywords: *PHP, JavaScript, API, Language, Interface, PostgreSQL*

ТЕХНІЧНЕ ЗАВДАННЯ

Необхідно розробити сервіс порівняння покупок (CSS - Comparison Shopping Service) для вебсайту, який буде включати наступні функції:

- пошукова стрічка для виконання пошукових запитів на отримання вибірки продуктів за запитом користувача;
- фільтрація результатів за конкретним магазином;
- основні фільтри по діапазону цін, наявності знижки, безкоштовної доставки тощо;
- можливість порівняння однакових продуктів з різних магазинів;
- простий та зрозумілий інтерфейс для користувачів з інтуїтивною навігацією;
- адміністративний інтерфейс з детальною інформацією про стан сервісів, звітами про кліки, доходи, витрати на рекламу тощо;
- система автоматизованого доопрацювання текстових файлів різного формату для покращення категоризації продуктів;
- система аналізу звітів по трафіку та якості для автоматичної або напівавтоматичної зміни рекламних налаштувань.

Для досягнення поставленої мети необхідно виконати наступні завдання:

- дослідити предметну область;
- зробити огляд літературних джерел для кращого теоретичного розуміння;
- виконати проектування та розробку архітектурних рішень системи;
- побудувати діаграми, що описують функції системи та її функціональних блоків;
- розробити програмне рішення для сервера та клієнта відповідно до діаграм та функцій системи.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

Фреймворк - заготовки, шаблони для програмної платформи, що визначають архітектуру програмної системи; програмне забезпечення, що полегшує розробку і об'єднання різних модулів програмного проекту.

Клік - кириличне написання англійського слова «click» – клацання. У контексті пошукової оптимізації клік означає перехід користувача на сайт за посиланням, розміщеної на сторінці пошуковика або сторонньому сайті.

Куки - (англ. Cookie, множина англ. Cookies – тістечка, печиво) – у комп'ютерній термінології поняття, яке використовується для опису інформації у вигляді текстових або бінарних даних, отриманих від вебзастосунку на веб-сервері, яка зберігається у клієнта, тобто браузера, а потім відправлена на той самий сайт, якщо його буде повторно відвідано.

Синтаксичний цукор - (англ. syntactic sugar) – узагальнена назва, яка позначає доповнення синтаксису мови програмування, які не додають нових можливостей, а роблять використання мови програмування зручнішим для людини. Зазвичай це додаткові легкозасвоювані синтаксичні конструкції, часто, схожі на інші мови програмування до яких могла звикнути людина.

JSON - (англ. JavaScript Object Notation, укр. запис об'єктів JavaScript, вимовляється джейсóн) – це текстовий формат обміну даними між комп'ютерами. JSON базується на тексті, може бути прочитаним людиною. Формат дає змогу описувати об'єкти та інші структури даних. Цей формат використовується переважно для передачі структурованої інформації через мережу (завдяки процесу, що називають серіалізацією).

ВСТУП

Останні роки можливості Інтернету значно розширилися, пропонуючи нові можливості для взаємодії. Тепер ми все частіше використовуємо веб-додатки для пошуку та замовлення одягу, продуктів, побутової техніки та інших речей.

Створення пошуково-рекламних веб-додатків стає дедалі популярнішим, оскільки вони пропонують простий спосіб взаємодії для кінцевих користувачів та значно спрощують пошук товарів за найвигіднішими цінами. Основою таких додатків є висока швидкодія та релевантність результатів, що потребує використання ефективної архітектури пошукової системи.

Також важливо забезпечити можливість фільтрації результатів за ключовими характеристиками, оскільки користувачі часто шукають конкретний товар. Інтеграція з рекламними сервісами дозволить:

- охопити ширше коло потенційних користувачів, які можуть скористатися веб-додатком;
- залучити цільовий трафік до веб-додатку, збільшуючи кількість переходів на продукти.

Беручи до уваги все вищезазначене, можна стверджувати, що правильно спроектований веб-додаток дозволить швидко знайти та порівняти пропозиції на товари, які користувач хоче придбати. Це дозволяє економити час і задовольняти більшість потреб користувачів, адже будь-хто може знайти потрібний товар за допомогою веб-браузера на своєму пристрої.

Об'єкт дослідження: сучасні інтернет-додатки для пошуку та купівлі товарів, які вирішують потреби онлайн-шопінгу.

Предмет дослідження: методи пошуку та рекомендації в пошуково-рекламних веб-додатках, що підвищують ефективність вибору та купівлі товарів.

Мета: розробка та оптимізація веб-додатків для поліпшення досвіду користувачів при онлайн шопінгу, забезпечення швидкої та ефективної взаємодії між покупцями та продавцями, та підвищення рівня конверсії веб-трафіку в продажі.

Практичне значення: забезпечення більшої зручності та швидкості для користувачів під час вибору та покупки товарів, підвищення ефективності рекламних кампаній для бізнесів, та стимулювання економічного зрос тання через збільшення онлайн торгівлі.

РОЗДІЛ 1. СТАН ПРОБЛЕМНОЇ ОБЛАСТІ

1.1. Характеристика об’єкту проектування та постановка задачі

Об'єктом проектування є вебзастосунок афілейт-маркетингу на основі Vue.js та Laravel, що спрямований на покращення ефективності пошуку та реклами товарів в інтернет-магазинах. Враховуючи стрімкий розвиток технологій та зростаючу популярність онлайн-шопінгу, актуальність створення високоефективних пошуково-рекламних вебзастосунків є надзвичайно високою.

Основні завдання включають:

- вивчення існуючих методів і технологій для пошуку, фільтрації та рекомендації товарів;
- інтеграція аналітичних інструментів для оптимізації веб-ресурсів;
- розробка програмного комплексу з використанням сучасних технологій веб-розробки, забезпечення швидкої обробки запитів та високої цілісності даних;
- висвітлення проблематики актуальності статистики сайтів, що використовують AJAX, та вирішення ускладнень, які виникають через перерозподіл обробки даних.

1.2. Побудова дерева проблеми та дерева цілей

Після ідентифікації характеристик об'єкта проектування та формулювання завдань продукту, наступним кроком у розробці системи є вибір методів і стратегій для досягнення цілей. Для оптимального вибору методів розробки необхідно здійснити системний аналіз проектованої системи. Системний аналіз включає в себе застосування комплексу методологічних інструментів, спрямованих на раціональне і обґрунтоване прийняття рішень в умовах складних та часто неочевидних завдань.

Цей аналітичний підхід дозволяє розглядати об'єкти як системи, а зокрема як цілеспрямовані системи. Він базується на принципах системного підходу, який

допомагає аналізувати зв'язки та взаємодії між елементами системи для підвищення їх загальної ефективності та стійкості. Основна мета такого аналізу – виявлення і структурування проблем, які можуть виникнути під час реалізації проекту, та визначення шляхів їх рішення.

Для систематизації аналізу використовуються такі інструменти як дерево проблем та дерево цілей:

Дерево проблем дозволяє визначити основну проблему та розкласти її на більш дрібні підпроблеми. Це допомагає зрозуміти корінь проблеми та залежності між її компонентами.

Дерево цілей використовується для визначення головної мети проекту та розбивки її на конкретніші підцілі. Цей інструмент сприяє формулюванню чітких, вимірюваних та досяжних завдань, необхідних для досягнення кінцевої мети проекту.

Перший етап системного аналізу з декомпозиції проблеми на складові допомагає визначити основну структуру та напрямки роботи, необхідні для ефективної розробки та реалізації системи. Цей процес є критично важливим для розуміння всіх аспектів проекту та ефективного управління ресурсами.

Опис структури діаграми:

- корінь дерева (верхній вузол) представляє загальну проблему низької ефективності;
- гілки, що виходять з кореня, представляють основні категорії проблем;
- листя (кінцеві вузли) представляють конкретні причини в рамках кожної категорії проблем.

Ціль діаграми.

Діаграма допомагає візуалізувати всі можливі причини проблем, з якими стикається вебзастосунок, та систематизувати їх для подальшого аналізу і пошуку рішень.

Призначення діаграми:

- аналіз проблем - допомагає ідентифікувати та структурувати проблеми;

- планування рішень - сприяє розробці стратегій для вирішення кожної проблеми;
- комунікація - служить інструментом для пояснення проблем іншим членам команди або зацікавленим сторонам.

Ця діаграма є важливою частиною системного аналізу, що дозволяє розробникам та аналітикам краще розуміти існуючі проблеми та працювати над їх ефективним вирішенням.

В контексті нашої системи, дерево проблем матиме вигляд наступним чином.

Дерево проблем.

Діаграма "Дерево проблем" представлена у вигляді графічного дерева, яке демонструє різні проблеми, пов'язані з низькою ефективністю пошуково-рекламних вебзастосунків. Кожен вузол (кружечок) у дереві представляє окрему проблему, а стрілки між вузлами вказують на причинно-наслідкові зв'язки між цими проблемами.

Корінь дерева.

Низька ефективність пошуково-рекламних вебзастосунків.

Це головна проблема, яка розбивається на кілька основних категорій:

Основні категорії проблем.

1. Низька швидкість обробки запитів:
    1.1. Неефективна архітектура вебзастосунків;
    1.2. Відсутність оптимізації коду;
    1.3. Невикористання сучасних технологій.
2. Відсутність релевантності результатів пошуку:
    2.1. Неправильне налаштування алгоритмів пошуку;
    2.2. Недостатня кількість даних для навчання алгоритмів;
    2.3. Відсутність персоналізації пошуку.
3. Проблеми з інтеграцією аналітичних інструментів:
    3.1. Несумісність аналітичних інструментів із системою;
    3.2. Відсутність підтримки сучасних форматів даних;
    3.3. Високі витрати на інтеграцію.

4. Труднощі в обробці даних через використання AJAX та розподіленої обробки даних:

4.1. Проблеми з синхронізацією даних;

4.2. Високий рівень складності розподіленої обробки;

4.3. Нестабільність роботи через високе навантаження.

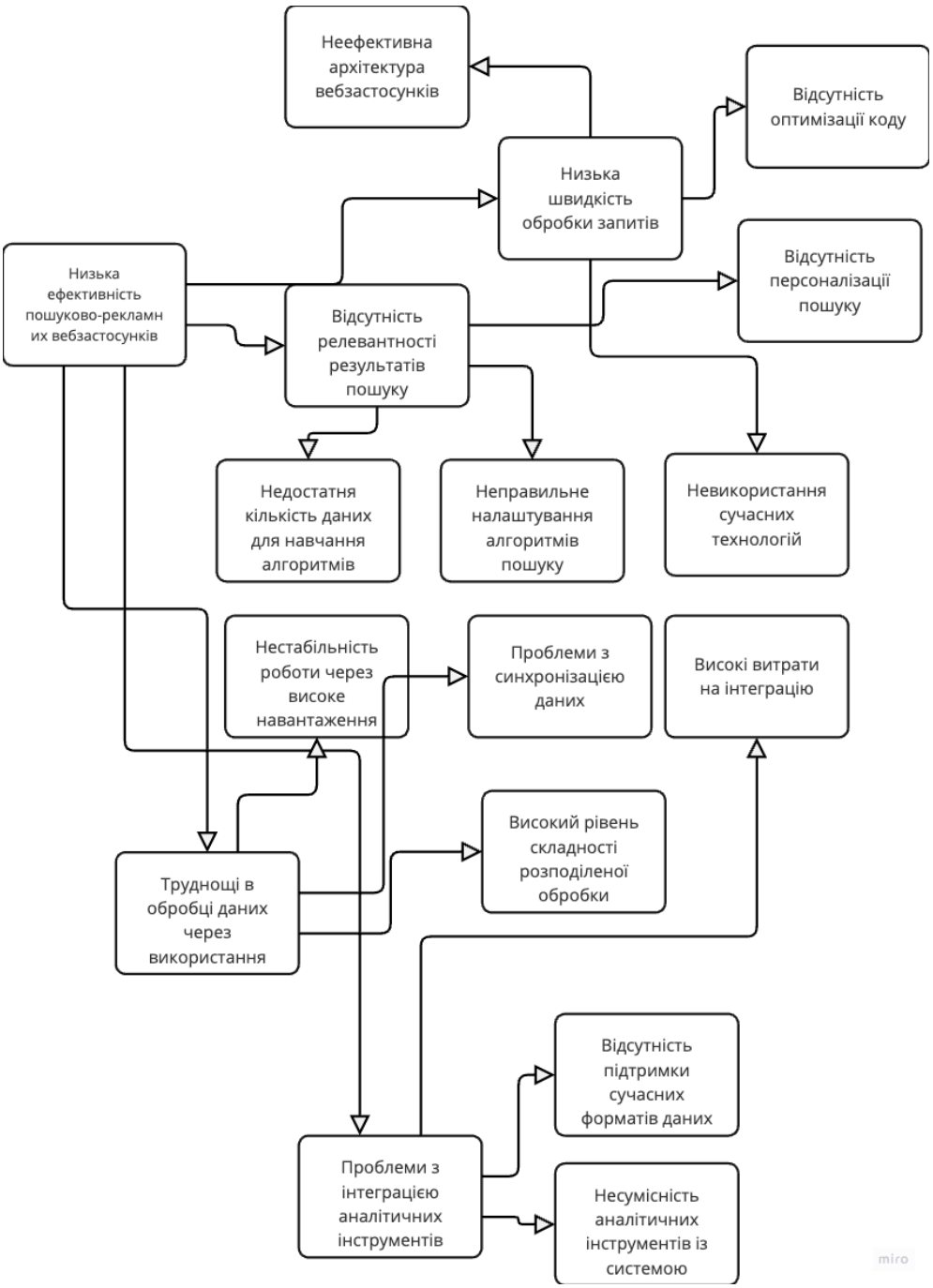

Рисунок 1.1 - Дерево проблем

Після створення та аналізу дерева проблем, наступним кроком є побудова і вивчення дерева цілей. Дерево цілей визначає основні завдання та цілі системи, які спрямовані на вирішення виявлених проблем. Цей етап є критично важливим для формування стратегії розробки та вдосконалення інформаційної системи.

Основна задача дерева цілей полягає в ідентифікації конкретних завдань, що дозволять вирішити проблеми, визначені на попередньому етапі. Кожна ціль представляє собою важливий аспект функціональності системи та сприяє поліпшенню користувацького досвіду і ефективності вирішення завдань. Створення дерева цілей стане фундаментом для подальших кроків у розробці та вдосконаленні пошуково-рекламного вебзастосунку.

Корінь дерева цілей, як і в дереві проблем, відповідає головній меті системи. Під час декомпозиції цієї основної мети на менші цілі відбувається процес деталізації. Вершини нижчих рівнів дерева представляють деталізовані завдання та цілі, що спрямовані на досягнення основної мети. Декомпозиція продовжується до тих пір, поки не стане можливим пов'язати нижчі цілі з конкретними засобами або функціоналом системи. Оскільки глобальну ціль не завжди можна прямо пов'язати з ресурсами для її досягнення, декомпозиція є необхідним кроком для забезпечення конкретності та можливості використання засобів для виконання завдань на нижчих рівнях ієрархії дерева цілей.Таким чином, на початковому етапі слід визначити основну мету системи – підвищення ефективності пошуково-рекламного вебзастосунку. Ця мета визначає загальний напрямок та характер функціонування системи, враховуючи сучасні вимоги до віддалених послуг і необхідність зручного користування.

Система пропонуватиме ряд наступних функцій:

- швидкі операції текстового пошуку, а також сортування й фільтрування результатів;
- інтеграція аналітичних інструментів для оптимізації веб-ресурсів;
- забезпечення високої релевантності результатів пошуку за допомогою налаштування та оптимізації алгоритмів;

- збір та аналіз даних для покращення роботи алгоритмів та персоналізації пошуку;
- впровадження механізмів синхронізації даних для забезпечення їх цілісності та актуальності;
- оптимізація розподіленої обробки даних для стабільної роботи системи під високим навантаженням.

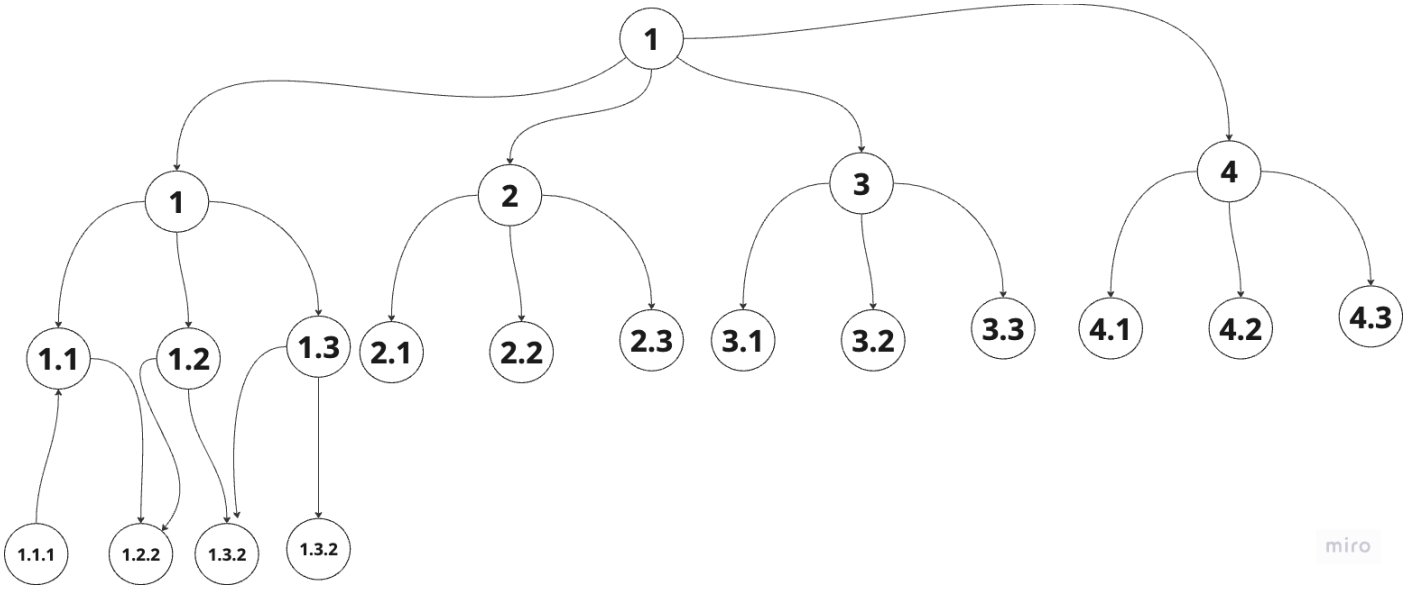

Рисунок 1.2 - Дерево цілей

0. Генеральна ціль. Створення високоефективного пошуково-рекламного вебзастосунку.

1. Розробка високоефективної архітектури вебзастосунку:

1.1 Використання сучасних фреймворків та бібліотек:

1.1.1 Реалізація ендпоінту для фільтрації;

1.1.2 Розробка компоненту фільтрації на рівні користувача.

1.2 Оптимізація коду для підвищення швидкості роботи:

1.2.1 Реалізація ендпоінту для пошуку;

1.2.2 Розробка компоненту пошуку на рівні користувача.

1.3 Впровадження методів кешування даних:

1.3.1 Реалізація ендпоінту для сортування;

1.3.2 Розробка компоненту сортування на рівні користувача.

2. Забезпечення високої релевантності результатів пошуку:

2.1 Налаштування та оптимізація алгоритмів пошуку;

2.2 Збір та аналіз даних для покращення алгоритмів;

2.3 Впровадження персоналізації пошуку на основі попередніх запитів користувачів.

3. Інтеграція сучасних аналітичних інструментів для оптимізації веб-ресурсів:

3.1 Вибір та впровадження сумісних аналітичних інструментів;

3.2 Забезпечення підтримки сучасних форматів даних;

3.3 Оптимізація процесів збору та аналізу даних для підвищення ефективності.

4. Розробка методів обробки даних для забезпечення їх цілісності та актуальності:

4.1 Впровадження механізмів синхронізації даних;

4.2 Оптимізація розподіленої обробки даних;

4.3 Забезпечення стабільної роботи системи під високим навантаженням.

Проведений аналіз стану проблемної області дозволив виявити основні проблеми, пов'язані з розробкою та оптимізацією пошуково-рекламних вебзастосунків. Було визначено, що головними проблемами є низька швидкість обробки запитів, відсутність релевантності результатів пошуку, складнощі з інтеграцією аналітичних інструментів та труднощі в обробці даних через використання AJAX.

Для вирішення цих проблем було запропоновано розробити вебзастосунок на основі сучасних технологій веб-розробки, зокрема Vue.js та Laravel, що дозволить забезпечити високу ефективність та релевантність пошукових запитів, а також інтеграцію аналітичних інструментів для оптимізації веб-ресурсів.

Завдяки запропонованому підходу користувачі зможуть швидко знаходити та порівнювати пропозиції на товари, що дозволить економити час та підвищити ефективність рекламних стратегій, забезпечуючи кращий користувацький досвід

РОЗДІЛ 2. ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

2.1. Організація та принципи роботи вебсервісу

Сучасні веб-додатки розробляються з використанням різних програмних платформ. Деякі додатки написані на PHP, в той час як інші використовують фреймворки, такі як Vue.js, Laravel та інші. Таке розмаїття породжує питання про призначення і використання веб-сервісів, які часто залишаються дещо незрозумілими для пересічного користувача.

Веб-сервіси - це самодостатні, незалежні модулі, які дозволяють взаємодіяти додаткам, написаним різними мовами програмування. По суті, веб-сервіс - це стандартизоване середовище, яке полегшує комунікацію між клієнтськими та серверними додатками через Інтернет. Це середовище можна змінювати та оновлювати за потреби.

Веб-сервіс можна описати як «програмний модуль, призначений для виконання певного набору функцій». Ці структурні елементи доступні через мережу і можуть бути викликані за необхідності. Після виклику веб-сервіс майже миттєво надає клієнту, який його викликав, функціональні можливості.

Щоб зрозуміти, як працюють веб-сервіси, необхідно розглянути їх функціонування в системі. Клієнти роблять послідовні запити до хосту веб-сервісу за допомогою віддаленого виклику процедур (RPC). Це означає, що у відповідь на конкретний запит виконується метод.

Наприклад, інфраструктура Amazon - одна з багатьох систем онлайн-магазинів і доставки, які використовують веб-сервіси. Amazon пропонує веб-сервіс для запиту цін на товари на сайті amazon.com. Зовнішній інтерфейс, який може бути реалізований на .Net, Java або будь-якій іншій мові програмування, взаємодіє з цим веб-сервісом.

Дані, якими обмінюються клієнт і сервер, мають вирішальне значення для роботи веб-сервісів в Інтернеті. XML і JSON є основними форматами, що

використовуються для цього обміну даними, і більшість мов програмування можуть ефективно аналізувати ці формати. Це дозволяє додаткам, розробленим на різних мовах програмування, безперешкодно обмінюватися даними незалежно від платформи.

Реалізація веб-сервісу вимагає певних компонентів, які не залежать від конкретної мови програмування. Для кожного типу запитів веб-сервіс можна розробити самостійно, створивши портал для його розміщення, розробивши необхідне середовище або інші залежні сервіси. Дві основні класифікації веб-сервісів - SOAP та RESTful. Проаналізувавши поставлене завдання та шляхи його вирішення, було прийнято рішення використовувати найсучасніші та найефективніші технології у веб-розробці. Такий підхід забезпечує розробку сервісу, здатного ефективно обробляти великі обсяги запитів користувачів.

2.2. Протокол HTTP

HTTP (HyperText Transfer Protocol) – це прикладний протокол, створений для обміну даними між додатками в розподілених і гетерогенних інформаційних системах. Протокол HTTP дозволяє додаткам передавати дані в людинозрозумілому форматі [3].

Спочатку HTTP був створений для передачі гіпертексту – типу даних, сформованого за допомогою HTML (HyperText Markup Language). Гіпертекстові документи містять текст, зображення, гіперпосилання та інші елементи представлення інформації, позначені тегами HTML. Гіперпосилання є інтерактивними елементами, взаємодія з якими приводить до отримання пов'язаних даних, що дозволяє користувачам навігацію через інформаційний контент.

HTTP функціонує поверх протоколу TCP (Transmission Control Protocol) і забезпечує контроль над передачею даних. На відміну від TCP, який не враховує структуру переданих пакетів, HTTP додає метаінформацію до корисних даних,

що дозволяє одержувачеві правильно інтерпретувати отриману інформацію. Глобальна мережа Інтернет базується на протоколі HTTP [3].

HTTP передбачає взаємодію двох сторін: клієнта і сервера. Клієнт, зазвичай веббраузер, ініціює з'єднання і надсилає запити на сервер для отримання даних або послуг. Сервер відповідає, надсилаючи необхідні дані та метаінформацію.

Основні характеристики HTTP:

- безстанність - кожен запит і відповідь є незалежними;
- гнучкість - підтримка різних форматів даних, таких як HTML, JSON, XML;
- розширюваність - можливість передачі додаткової інформації через заголовки.

Хост, який передає дані через HTTP, називається вебсервером. Ця програма обробляє запити від клієнтів і надсилає відповіді, які, окрім корисних даних, також містять метаінформацію, що їх описує.

Запит, який надсилається клієнтом до сервера, призначений для точної ідентифікації запитуваного ресурсу та містить дані, необхідні для коректної обробки цього запиту [3].

Зі структурної точки зору, запит складається з трьох частин:

- рядок запиту;
- блок заголовків;
- об’єкт.

Існує кілька методів HTTP-запитів, кожен з яких має своє призначення:

- GET: отримання даних з сервера;
- POST: надсилання даних на сервер для створення або оновлення ресурсу;
- PUT: повне оновлення ресурсу;
- DELETE: видалення ресурсу;
- PATCH: часткове оновлення ресурсу;
- HEAD: отримання заголовків відповіді без тіла;

- OPTIONS: отримання підтримуваних методів запитів для ресурсу.

Статусний рядок відповіді містить три поля, розділені пробілами і завершується символами CR, LF. Елементи статусного рядка:

1. Версія HTTP: наприклад, "HTTP/1.1".

2. Код статусу: трицифровий числовий код, який ідентифікує результат виконання запиту. Коди статусів бувають наступними:

- 1xx: інформаційні: інформація про процес передачі.

- 2xx: успішні:

  - 200 OK: запит виконано успішно;

  - 201 Created: ресурс створено успішно;

  - 202 Accepted (прийнято): запит прийнято для обробки;

  - 204 No Content (немає вмісту): тіло повідомлення не передається.

- 3xx: переадресація:

  - 301 Moved Permanently: ресурс переміщено постійно;

  - 303 See Other: ресурс тимчасово переміщено;

  - 304 Not Modified: ресурс не змінено.

- 4xx: помилки клієнта:

  - 400 Bad Request: помилковий запит;

  - 401 Unauthorized: потрібна автентифікація;

  - 403 Forbidden (заборонено): доступ заборонено;

  - 409 Conflict (конфлікт): конфлікт при спробі змінити ресурс.

- 5xx: помилки сервера: ці коди використовуються для повідомлення про невдале виконання операції через помилку з боку сервера:

  - 501 Not Implemented: сервер не підтримує можливостей для обробки запиту;

  - 503 Service Unavailable: сервер перевантажений або недоступний.

2.3. Переваги використання REST над SOAP

Сучасний Інтернет являє собою глобальну мережу, яка об'єднує комп'ютери для обміну ресурсами та інформацією. Модель "клієнт-сервер" є основою цього зв'язку.

REST (Representational State Transfer) – це архітектурний стиль для створення розподілених та масштабованих вебдодатків, який використовує специфікацію HTTP для маніпулювання ресурсами. REST регламентує взаємодію між серверами, мережевими шлюзами, проксі і клієнтами без накладення обмежень на окремих учасників.

SOAP (Simple Object Access Protocol) – це протокол для обміну структурованими інформаційними повідомленнями в розподілених обчислювальних середовищах, заснований на XML і використовуваний для інтеграції різних вебсервісів. SOAP регламентує взаємодію між клієнтом і сервером через XML-повідомлення, що передаються за допомогою різних транспортних протоколів [1].

Порівнюючи REST і SOAP, можна виділити кілька причин, чому REST частіше обирають для сучасних вебсервісів:

1. Простота: REST використовує стандартні методи HTTP, що спрощує його реалізацію та інтеграцію. SOAP вимагає додаткових протоколів і специфікацій;

2. Легкість у навчанні та використанніREST базується на добре відомих принципах HTTP і використовує легкі формати даних, такі як JSON;

3. Масштабованість: REST підходить для масштабованих систем завдяки легким HTTP-запитам і мінімальним службовим даним;

4. Продуктивність: REST є більш продуктивним завдяки меншому об'єму даних і меншій кількості службових даних;

5. Кросплатформність: REST підтримується будь-якою мовою програмування, що підтримує HTTP;

6. Кешування: REST запити можуть бути кешовані, що знижує навантаження на сервери;

7. Гнучкість: REST дозволяє адаптувати і оптимізувати його під конкретні потреби проекту.

Таким чином, REST пропонує більш простий, легкий та гнучкий підхід до розробки вебсервісів, що робить його кращим вибором для багатьох сучасних додатків у порівнянні з SOAP.

Були детально проаналізовані організація та принципи роботи вебсервісу, який є основною складовою веборієнтованої інформаційно-пошукової системи бібліотеки. Описані архітектурні рішення, що забезпечують ефективну роботу сервісу.

Розглянуто основні принципи функціонування протоколу HTTP, який є ключовим засобом комунікації між клієнтом та сервером в Інтернеті, та його роль у взаємодії елементів системи. Проведено порівняння двох популярних технологій для реалізації вебсервісів – REST та SOAP. Оцінено переваги та недоліки кожної технології з урахуванням вимог та специфіки бібліотечного вебсервісу.

2.4. Мови програмування JavaScript, TypeScript та фреймворки Vue.js, Element Plus

JavaScript – це динамічна, об'єктно-орієнтована мова програмування з прототипним наслідуванням, яка широко використовується для створення сценаріїв веб-сторінок. Ця мова дозволяє здійснювати інтеракцію з користувачем, керувати браузером, здійснювати асинхронний обмін даними з сервером, а також модифікувати структуру та зовнішній вигляд веб-сторінок [6].

JavaScript класифікують як прототипну скриптову мову з динамічною типізацією, яка також підтримує імперативні та функціональні парадигми програмування. Серед архітектурних особливостей мови – динамічна і слабка

типізація, автоматичне керування пам'яттю, наслідування за допомогою прототипів та розглядання функцій як об'єктів першого класу.

Основні застосування JavaScript:

- розробка інтерактивних веб-сторінок;
- створення односторінкових веб-додатків за допомогою таких фреймворків як React, AngularJS, і Vue.js;
- програмування на серверній стороні за допомогою Node.js;
- створення мобільних застосунків з використанням React Native та Cordova.

Хоча JavaScript має схожу назву з Java, ці мови мають різну семантику і використовуються для різних цілей. Обидві мови синтаксично походять від мови C, але дизайн та філософія JavaScript були засновані на мовах Self і Scheme.
Під час розробки великих веб-додатків, інструменти для налагодження стають критично важливими, особливо з огляду на відмінності в реалізації JavaScript і об'єктних моделей документів різними браузерами. Через динамічну природу та слабку типізацію, програмістам важливо забезпечити, що їх код коректно працює у всіх браузерах і середовищах. Розділення JavaScript коду на кілька файлів може допомогти уникнути "падіння" всієї сторінки через синтаксичні помилки і сприяє більш чистому та управліному коду.

Мова програмування TypeScript є надбудовою над мовою JavaScript і пропонує вирішення основних архітектурних проблем, які має в собі JavaScript. TypeScript додає можливість строгої типізації змінних в JavaScript і цим допомагає уникнути багатьох проблем при розробці.

Модуль перевірки типізації перевіряє типи кожного елементу та змінної у програмних файлах та шукає помилки невідповідності типів, якщо такі існують. Так можна дуже швидко виявити проблеми несумісних типів, які будуть видавати помилки на початку запуску у веб-браузері.

Для JavaScript існують кілька фреймворків, серед яких є Vue.js та Vuetify.

Vue.js – JavaScript-фреймворк, який використовує архітектуру MVVM для створення користувацьких інтерфейсів на основі моделей даних через реактивне зв'язування даних. Vue використовує HTML-базований шаблонний синтаксис, який дозволяє декларативно прив'язувати рендеринг DOM до моделей даних. Одна з ключових особливостей Vue – це її ненав'язлива реактивна система. Моделі представлені простими JavaScript-об'єктами, що спрощує управління станами.

Element Plus – відкритий набір інструментів для розробки веб-сайтів та додатків. Він включає CSS та HTML шаблони для типографіки, форм, кнопок, навігації та інших елементів інтерфейсу, а також доповнення на JavaScript, що полегшує створення динамічних веб-сайтів та додатків.

PHP – скриптова мова програмування, створена для генерації HTML-сторінок на стороні сервера. Вона є однією з найпопулярніших мов у сфері веб-розробки та підтримується більшістю хостинг-провайдерів. PHP – це проект з відкритим вихідним кодом, який інтерпретується сервером у HTML-код, який потім відправляється клієнту. Відмінно від JavaScript, код PHP залишається на сервері, забезпечуючи більшу безпеку, але меншу інтерактивність сторінок порівняно з клієнтськими скриптами [1].

PHP має вбудовані бібліотеки для роботи з багатьма базами даних, включно з MySQL, PostgreSQL та Oracle, та підтримує зв'язок з іншими базами через відповідні драйвери. Популярність PHP зумовлена її гнучкістю та широким набором функцій для роботи з веб-додатками, включно з автоматичним вилученням POST і GET-параметрів, взаємодією з різними СУБД, автоматизованою відправкою HTTP-заголовків, керуванням сесіями, cookies, файлами, які завантажуються, та багатьма іншими можливостями

PHP-код починає виконуватися після першої екрануючої послідовності (<?) і продовжується до зустрічі парної екрануючої послідовності (?>).

PHP знайомий багатьом програмістам завдяки своїм конструкціям, які запозичені з мов C та Perl. Код PHP схожий на типовий код мов C або Pascal, що полегшує його вивчення.

PHP є універсальною мовою з зрозумілим синтаксисом, спеціально призначеною для роботи в Інтернеті. Попри свою відносну молодість, PHP здобув велику популярність серед веб-розробників і зараз є однією з найпопулярніших мов для створення веб-додатків.

PHP має велику різноманітність функцій, що дозволяє уникнути написання багаторядкових функцій користувача на C або Pascal. Серед важливих переваг PHP варто зазначити його інтерпретованість, що забезпечує високу швидкість обробки сценаріїв. Хоча компільовані файли завжди працюють швидше, продуктивність PHP достатня для створення серйозних веб-додатків.

Одним із застосувань PHP є розробка парсеру – програми, що виконує синтаксичний аналіз. Парсер переходить за заданими URL-посиланнями, завантажує HTML-сторінку, аналізує її та обробляє дані відповідно до поставленої задачі.

Ефективність є важливим чинником у програмуванні для середовищ, розрахованих на багато користувачів, до яких належить і Web. PHP забезпечує достатню продуктивність для створення різноманітних веб-додатків, перевершуючи за швидкістю обробки сценаріїв багато аналогічних програм на Perl.

Laravel – відкритий та безкоштовний PHP фреймворк, створений Тейлором Отвелом, який використовується для розробки веб-додатків на основі архітектурного шаблону Model-View-Controller (MVC) [1].

Особливості Laravel включають:

- модульну систему пакетування з інтегрованим менеджером залежностей Composer;
- різноманітні методи доступу до реляційних баз даних;
- інструменти для розгортання та технічного обслуговування додатків;

- фокус на зручний та "солодкий" синтаксис.

Цей фреймворк пропонує високоефективну архітектуру для розробки веб-додатків будь-якого рівня складності. Він побудований за модульним принципом, що дає можливість з легкістю підключати нові модулі, які розробляються дуже широким суспільством розробників цього проекту. Також він пропонує зручні інтерфейси взаємодії з HTTP запитами, вбудованими сервісами та функціями.

Одною із ключових особливостей фреймворку є підтримка асинхронних задач і черг таких задач. Такі черги дозволяють виконувати складну логіку програми, або досить довгі задачі в фоновому режимі. Таким чином ми посилаємо задачі на асинхронне виконання і не блокуємо основний потік вводу/виводу програми. Це в свою чергу забезпечує високу швидкість виконання кожного HTTP запиту, який залучає виконання складних, або довгих операцій.

Вибір Nginx як основного веб-сервера для проекту є стратегічним рішенням, враховуючи його високу продуктивність та ефективність. Nginx – це безкоштовний веб-сервер і проксі-сервер, доступний для багатьох UNIX-подібних систем, а також для Windows. Він відомий своєю потужністю та широким набором функціональних можливостей, що робить його ідеальним вибором для різноманітних веб-застосунків.

Ось декілька ключових можливостей Nginx, які сприяють його популярності та широкому застосуванню:

- обробка статичних файлів: Nginx ефективно обробляє статичні файли, індексні файли, автоматично створює списки файлів і використовує кешування для покращення швидкості відгуку;
- продвинуте проксіювання: підтримка кешування при проксіюванні забезпечує покращену продуктивність та ефективність в обробці запитів;
- підтримка серверів FastCGI та memcached: Nginx оптимізує взаємодію з FastCGI і memcached серверами, що забезпечує більшу відмовостійкість і простоту розподілу навантаження;

- модульність та гнучкість: включення модулів для gzip-стиснення, обробки byte-ranges (докачування), chunked відповідей, HTTP-аутентифікації та SSI-фільтрів збільшує можливості налаштування сервера;
- паралельна обробка запитів: можливість виконання вкладених запитів на одній сторінці паралельно значно збільшує продуктивність обслуговування веб-запитів;
- підтримка SSL та HTTP/2: забезпечення безпеки та сучасних протоколів передачі даних для підвищення швидкості і захищеності веб-комунікацій;
- використання Nginx для виконання PHP скриптів, особливо з його здатністю обслуговувати велику кількість паралельних процесів, робить його ідеальним рішенням для високонавантажених веб-додатків і веб-сайтів з великим потоком одночасних запитів. Ці особливості роблять Nginx одним з найбільш вибраних веб-серверів у сучасних веб-технологіях.

Також можна виділити те, що цей веб-сервер підтримує конфігурації будь-якого рівня складності, які можна самому написати та помістити в окремий файл конфігурації. Це дозволяє підлаштувати веб-сервер під будь-які потреби та навантаження.

PostgreSQL – вільна система управління реляційними базами даних та ефективний багатопотоковий сервер, відомий своєю швидкістю, надійністю та простотою використання. У реляційній базі даних інформація розподіляється по окремих таблицях, що дозволяє збільшити швидкість обробки даних та гнучкість системи. Таблиці зв'язані відносинами, що дозволяє комбінувати дані з різних таблиць під час виконання запитів. SQL, мова структурованих запитів, використовується в PostgreSQL для доступу до даних.

PostgreSQL є ідеальним вибором для середніх та великих застосувань. Його серверні коди можна компілювати на різних платформах, але найкраще він

працює на UNIX-системах з підтримкою багатопотоковості, що підвищує загальну продуктивність системи.

Особливості сервера PostgreSQL включають:

- легкість встановлення та використання;
- підтримка необмеженої кількості користувачів, які одночасно працюють з базою даних;
- можливість зберігати до 50 мільйонів рядків у таблицях;
- висока швидкість виконання команд;
- проста та ефективна система безпеки.

PostgreSQL – програмне забезпечення з відкритим кодом, яке може використовувати та модифікувати будь-хто. Користувачі мають можливість вивчити та адаптувати вихідний код до своїх потреб.

Система складається з серверної та клієнтської частин. Сервер PostgreSQL неперервно працює на комп'ютері, обробляючи SQL-запити від клієнтських програм, які надсилають запити через сокети. Клієнтські програми, такі як PHP скрипти, інструктують сервер щодо типу інформації, яку вони хочуть отримати, а сервер відповідає на ці запити, повертаючи результати.

Також варто зазначити, що PostgreSQL підтримує повнотекстовий пошук по полям з типом tsvector. Головною задачею в такій ситуації є вибір правильного індексу для такої колонки. Правильний індекс дозволить уникнути проблем з довгим виконанням запитів на повнотекстовий пошук і зменшить використання ресурсів на запит.

Для цього для PostgreSQL потрібно було встановити окреме розширення RUM, яке додає однойменний індекс в СКБД. Цей індекс дає дуже великий приріст швидкості та ефективності на запитах, що вміщають в собі операцію повнотекстового пошуку по окремій таблиці.

Технологія AJAX – підхід до створення інтерактивних веб-інтерфейсів, що дозволяє проводити обмін даними між браузером та веб-сервером у фоновому режимі. Це означає, що під час оновлення інформації на сторінці не потрібно

перезавантажувати її цілком, завдяки чому веб-додатки працюють швидше і є більш зручними для користувача. AJAX не є самостійною технологією, а використовує комбінацію декількох технологій. Основні принципи включають:

1. Динамічний доступ до сервера «на льоту»:
- за допомогою XMLHttpRequest (основний об'єкт);через динамічне ;
- через динамічне створіння тега <script>;
- через динамічне створення фреймів.
2. Використання DHTML для динамічної зміни змісту сторінки. Зазвичай дані передаються у форматах JSON або XML.

Переваги:

- економія трафіку завдяки завантаженню лише змінених частин сторінки;
- зниження навантаження на сервер;
- швидка відповідь інтерфейсу, оскільки завантажується лише необхідна частина даних.

Недоліки:

- створені динамічно сторінки не реєструються в історії браузера, що може ускладнювати навігацію;
- динамічно завантажуваний вміст може бути недоступний для пошукових машин;
- ускладнення проекту через перерозподіл логіки даних;
- необхідність увімкненого JavaScript у браузері.

XMLHttpRequest – API, який дозволяє веб-клієнтам (браузерам) здійснювати HTTP-запити до веб-сервера без перезавантаження сторінки, використовуючи JavaScript, JScript, VBScript та інші мови програмування. Він підтримує як синхронний, так і асинхронний обмін інформацією в текстових форматах, зокрема XML і JSON.

Google Ads API – головна та єдина система управління системою Google Ads, що надає централізований API інтерфейс. Вона була запроваджена

компанією Google для спрощеного та автоматизованого керування ресурсами акаунтів Google Ads на програмному рівні.

Ця технологія є невід'ємною частиною будь-якої системи, що вимагає тісної інтеграції з цілою системою Google Ads.

Серед основних особливостей, які пропонує технологія можна виділити такі:

- висока швидкість виконання операцій;
- безпечне передавання даних про ресурси акаунтів;
- уніфікований інтерфейс взаємодії з системою;
- зручне налаштування доступу з будь-якими видами прав;
- стабільність виконання операцій.

РОЗДІЛ 3. ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1. Вибір архітектури програмного комплексу

Для виконання поставленої задачі було обрано триланкову архітектуру, яка включає такі компоненти: клієнт, сервер і база даних. Схема цієї архітектури представлена на рисунку 3.1.

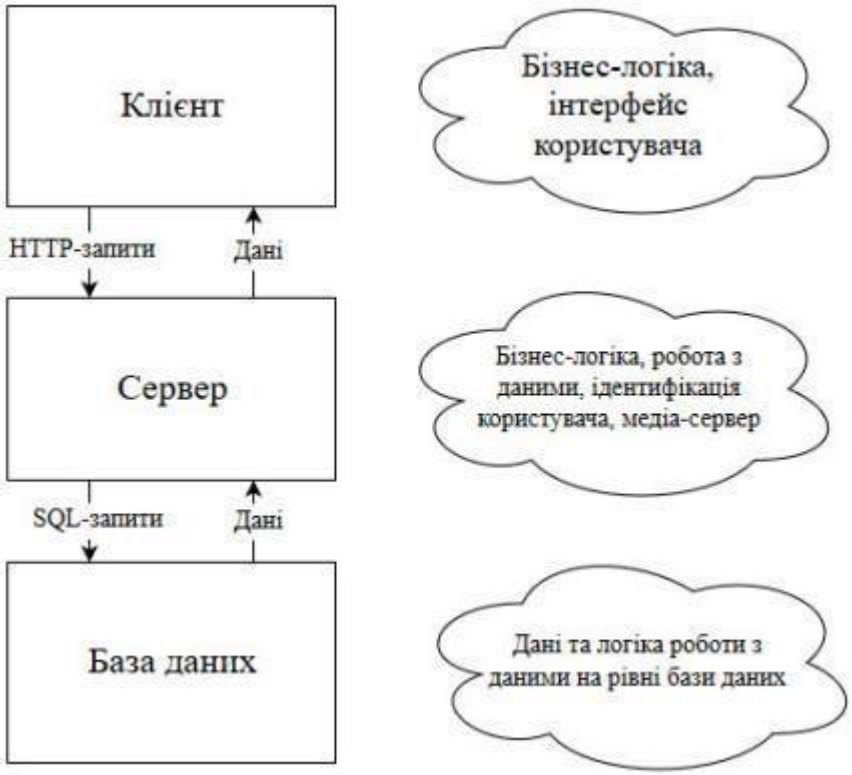

Рисунок 3.1 – Триланкова архітектура програмного комплексу

Головною складовою програмного комплексу є сервер. Він зосереджує основну бізнес-логіку та логіку доступу до бази даних. Сервер здійснює ідентифікацію користувача для надання індивідуального доступу до програмного застосунку. Він є єдиним посередником між користувачем та базою даних, що запобігає пошкодженню даних та їх нецільовому використанню. Під час користування програмою, користувач взаємодіє з клієнтським додатком, який у цьому випадку є вебзастосунком.

На користувацькому рівні реалізовано інтерфейс, через який відправляються пошукові запити на сервер. Крім того, на цьому рівні відбувається попередня обробка даних перед їх відправленням на сервер та обробка отриманих результатів. Тут також здійснюється первинна аутентифікація користувача для обмеження неконтрольованого доступу до програми.

Рівень бази даних відповідає за зберігання інформації, яку сервер використовує для подальшої роботи, та гарантує цілісність даних через використання зовнішніх зв'язків і ключів. На цьому етапі також можливе впровадження деяких аспектів бізнес-логіки, якщо для їх реалізації достатньо даних, що містяться в базі даних і її таблицях.

3.2. Опис архітектури серверу

Для створення сервера використано API, яке є інтерфейсом прикладного програмування. API визначає набір підпрограм, протоколів та інструментів для розробки програмного забезпечення, і є засобом для взаємодії між різними компонентами програми. У простіших термінах, це набір методів, які дозволяють різним компонентам спілкуватися між собою. API дозволяє розробникам швидко розробляти нове програмне забезпечення, використовуючи уже існуючі блоки або функції, що забезпечується бібліотеками або системами. API застосовується в різноманітних областях, таких як веб-сервіси, операційні системи, бази даних, апаратне забезпечення та програмні бібліотеки.

Часто API використовується для доступу до вже існуючих функцій, наприклад, для рендерингу вікон чи іконок на екрані, що дозволяє розробникам економити час та ресурси, не винаходячи ці функції заново. Таке використання API спрощує процес розробки і покращує сумісність між різними програмними рішеннями. Хоча API часто є частиною комплекту розробки програмного забезпечення (SDK), вони не є взаємозамінними, оскільки SDK може включати

додаткові інструменти, документацію або апаратне забезпечення, що розширюють можливості розробника.

У контексті веб-розробки API зазвичай визначається набором запитів HTTP та структурою відповідей, які зазвичай використовують XML або JSON. Історично API в веб-контексті було майже синонімом веб-служби, але з появою Web 2.0 спостерігається тенденція переходу від веб-служб на основі SOAP до більш прямої передачі репрезентативного стану (REST) і ресурсно-орієнтованої архітектури (ROA). Ця тенденція пов'язана з розвитком Семантичного вебу, інженерії онтологій та опису платформ (RDF).

Прикладні програмні інтерфейси у веб-середовищі, що дозволяють комбінувати декілька API в нові додатки, називають гібридними.

3.3. Опис архітектури клієнтського застосунку

Для створення клієнтського застосунку було застосовано фреймворк, що базується на шаблоні проектування MVVM (Model-View-ViewModel). Цей шаблон дозволяє ефективно розділити розробку графічного інтерфейсу користувача та бізнес-логіки, яка відома як модель. Основна ідея MVVM полягає у відокремленні логіки представлення від моделі даних, що сприяє зменшенню залежностей між компонентами системи.

Модель представлення (ViewModel) відіграє ключову роль у шаблоні MVVM, забезпечуючи перетворення даних для їх подальшої обробки та відображення. Вона функціонує як міст між представленням та моделлю, обробляючи більшість логіки, пов'язаної з даними. ViewModel може також використовувати шаблон медіатора для організації доступу до бекенд-логіки, забезпечуючи взаємодію між графічним інтерфейсом та бізнес-логікою через добре визначені правила.

Однією з ключових переваг шаблону MVVM є його здатність використовувати "зв'язування даних" (data binding), яке допомагає

синхронізувати модель і представлення, тим самим забезпечуючи легкість внесення змін у дані без необхідності вручну оновлювати елементи інтерфейсу. Це забезпечує більш структуровану та легку у підтримці архітектуру клієнтського застосунку, зробивши MVVM вельми популярним вибором для розробників сучасних застосунків.

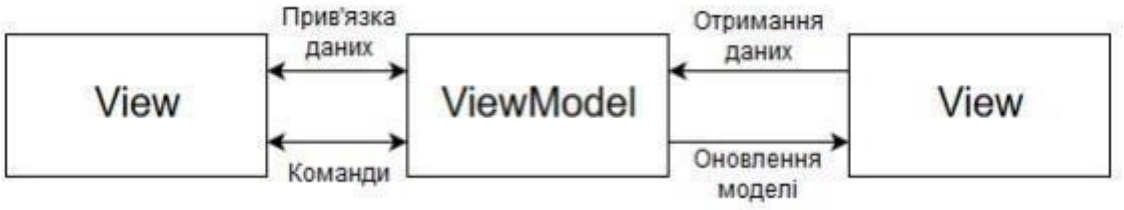

Рисунок 3.2 - Схема взаємодії View і ViewModel у патерні MVVM

Шаблон MVVM складається з таких частин:

- модель (Model): як і в класичному шаблоні MVC, модель представляє фундаментальні дані, необхідні для роботи застосунку;
- представлення (View): так само, як і в класичному шаблоні MVC, представлення є графічним інтерфейсом користувача, тобто включає вікна, кнопки та інші елементи;
- модель вигляду (ViewModel, що означає “Model of View”): з одного боку, це абстракція представлення, а з іншого – обгортка даних з моделі, які повинні зв'язуватись з представленням.

3.4. Інструменти розробки

Програмний комплекс розроблено згідно з принципами тришарової архітектури, де кожен шар реалізовано за допомогою різних технологій з метою створення мультиплатформного рішення на базі відкритих технологій.

На клієнтському рівні у роботі застосовано JavaScript та TypeScript для розробки динамічних та інтерактивних інтерфейсів, що дозволяють користувачам

ефективно взаємодіяти з веб-застосунком, використовуючи такі функціональні можливості, як асинхронні запити та реактівність компонентів.

На клієнтському рівні застосовано такі технології:

- JavaScript та TypeScript – мови програмування, які забезпечують динамічну інтерактивність інтерфейсів;
- Vue.js – прогресивний JavaScript-фреймворк для побудови веб-застосунків;
- Vuetify – фреймворк для створення графічних інтерфейсів, який використовує матеріальний дизайн.

На серверному рівні використано PHP для реалізації серверної логіки, а також використав фреймворк Laravel, що забезпечив швидкий та зручний процес розробки веб-додатків, включаючи управління базами даних, маршрутизацію запитів і обробку аутентифікації користувачів.

Отже, на серверному рівні використано:

- PHP – мова програмування, яка часто використовується для розробки серверної логіки;
- Laravel – фреймворк PHP для більш зручної і швидкої розробки веб-додатків;
- Nginx – потужний і гнучкий веб-сервер, що відомий своєю стабільністю та високою продуктивністю.

На рівні бази даних:

- PostgreSQL – об'єктно-реляційну систему управління базами даних, що підтримує складні запити та транзакції.

Технологія AJAX була використана для забезпечення асинхронної взаємодії між клієнтом і сервером, що дозволяє оновлювати частини веб-сторінок без необхідності перезавантаження усієї сторінки.

Цей багаторівневий підхід дозволяє забезпечити гнучкість, масштабованість та легкість управління комплексом, роблячи його ефективним у використанні і підтримці.

Нижче приставлений комопнент login.blable.php

```
<x-guest-layout>
  <x-auth-card>
   <x-slot name="logo">
     <a href="{{ route('membersSite.home') }}">
      <x-application-logo class="w-20 h-20 fill-current text-gray-
500" />
       </a>
        </x-slot>

  <!-- Session Status -->
  <x-auth-session-status class="mb-4" :status="session('status')" />

  <!-- Validation Errors -->
  <x-auth-validation-errors class="mb-4" :errors="$errors" />

  <form method="POST" action="{{ route('login') }}">
   @csrf

  <!-- Email Address -->
  <div>
  <x-label for="email" :value="__('Email')" />

  <x-input  id="email"  class="block  mt-1  w-full"  type="email"
name="email" required autofocus />
  </div>

  <!-- Password -->
  <div class="mt-4">
  <x-label for="password" :value="__('Password')" />

  <x-input id="password" class="block mt-1 w-full"
```

```
  type="password"
  name="password"
  required autocomplete="current-password" />
  </div>

  <!-- Remember Me -->
  <div class="block mt-4">
  <label for="remember_me" class="flex items-center">
  <input id="remember_me" type="checkbox" class="rounded border-
gray-300 text-indigo-600 shadow-sm focus:border-indigo-300
focus:ring focus:ring-indigo-200 focus:ring-opacity-50"
name="remember">
  <span class="ml-2 text-sm text-gray-600">{{ __('Remember me')
}}</span>
  </label>
  </div>

  <div class="flex items-center justify-end mt-4">
  @if (Route::has('password.request'))
  <a class="underline text-sm text-gray-600 hover:text-gray-900"
href="{{ route('password.request') }}">
  {{ __('Forgot your password?') }}
  </a>
  @endif

  <x-button class="ml-3">
  {{ __('Login') }}
  </x-button>
  </div>
  </form>
  <div class="mt-4 pt-2 border-t-2 text-sm">
  Don't have an account? <a class="underline" href="{{
route('register') }}">Sign up</a>
  </div>
```

```
  </x-auth-card>
</x-guest-layout>
```

Сторінка з продуктами:

```
<template>
  <div>
  <section class="body-font hero">
  <div class="container mx-auto flex px-5 py-24 md:flex-row flex-
col items-center">
  <div class="lg:flex-grow flex flex-col md:items-start md:text-
left mb-16 md:mb-0 items-center text-center">
  <h1 class="text-2xl mb-4 p-1 max-w-3xl font-normal bg-gray-100
rounded bg-opacity-50" v-html="$t('mainPage.heroTitle')"></h1>
  <search-bar class="relative text-gray-600 w-11/12 md:w-3/5 lg:w-
2/5"></search-bar>
  </div>
  </div>
  </section>
  <section>
  <div class="container px-3 py-12 mx-auto">
  <div class="grid grid-cols-12">
  <product-card
  v-for="product in products"
  :key="product.id"
  :data-object="product"
  class="col-span-6 md:col-span-4 lg:col-span-2 p-2 m-2 shadow-sm
hover:shadow-xl transition ease-out duration-500 rounded"
  ></product-card>
  </div>
  </div>
  </section>
  </div>
</template>
```

```
<script lang="ts">
import Vue from "vue";
import Component from "vue-class-component";
import ProductCard from "./utility/ProductCard.vue";
import SearchBar from "./utility/SearchBar.vue";
import {Product} from "./models/Product";
import {ProductContract} from "./contracts/ProductContract";

@Component({
  components: {ProductCard, SearchBar}
})
export default class MainPageContent extends Vue {
  private products: Array<Product> = [];

  public created() {
  this.$fsApi.requestProducts(60, 1).then((response:
ProductContract[]) => {
  this.products = response.map((value: ProductContract) => {
  return Product.createFromPlain(value)
  });
  });
  }
}
</script>
<style scoped>
</style>
```

3.5. Загальна структура програмного проекту

Дипломна робота виконувалася із використанням мов програмування JavaScript, TypeScript, Vue.js та PHP. Розроблено серверну та клієнтську частини.

Нижче буде представлена структура програмного рішення як для сервера, так і для клієнта.

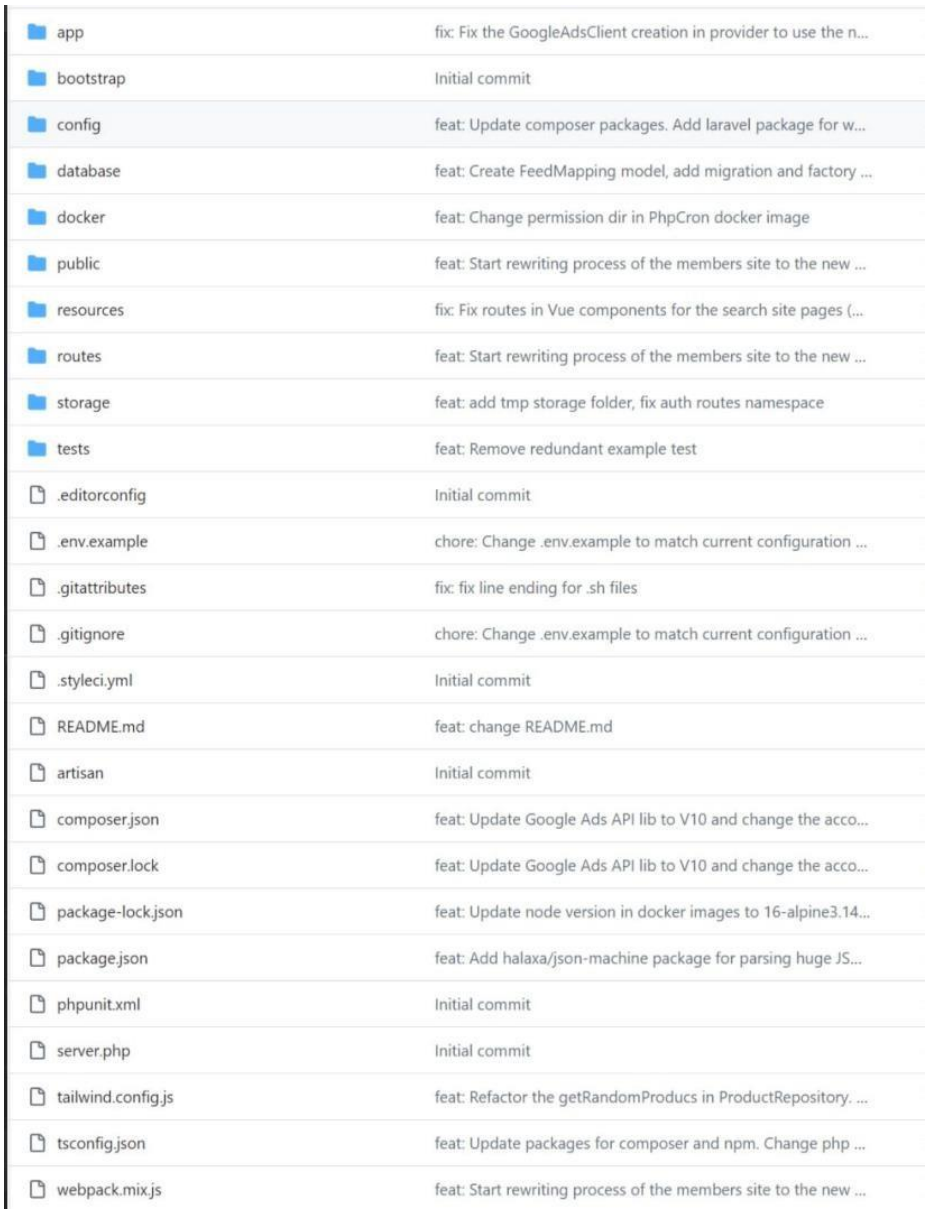

Рисунок 3.3 - Файлова структура серверної та клієнтської частин

Визначившись зі списком технологій та компонентів можна приступити до безпосередньої реалізації нашого програмного продукту.

3.6. Налаштування веб-серверу Nginx

Спочатку було встановлено веб-сервер Nginx на цільовий сервер.

Було проаналізовано та розроблено конфігурацію для веб-серверу відповідно до усіх вимог, що були описані:

- виставлено домен вебзастосунку та IP-адресу серверу;
- налаштовано SSL сертифікат для підтримки безпечних підключень до нашого вебзастосунку;
- підключено кешування статичних файлів для зменшення потоку повторних запитів цих файлів;
- налаштовано конфігурацію виконання PHP скриптів на сервері;
- додана перевірка та перенаправлення на захищений протокол передачі HTTPS з протоколу HTTP .

Конфіг Nginx.

Перший блок server (HTTPS налаштування)

```
server {
  listen 443 ssl http2;                    # Слухати порт 443 з підтримкою SSL і HTTP/2
  listen [::]:443 ssl http2 ipv6only=on;   # Слухати порт 443 для IPv6 з підтримкою SSL і HTTP/2, тільки для IPv6
  server_name ~^(members|www)\.squicket\.(.*)$; # Задає імена серверів за шаблоном для доменів типу members.squicket.* або www.squicket.*
  root /var/www/public;                    # Кореневий каталог файлів для сервера
  index index.php index.html index.htm;    # Файли, які будуть обслуговуватися як індексні
  location / {
  try_files $uri $uri/ /index.php$is_args$args; # Спробувати знайти файл, директорію або перенаправити запит на index.php
```

```
    }
    location ~ \.php$ {
     try_files $uri /index.php =404;       # Спробувати знайти PHP
файл або перенаправити на index.php, інакше 404 помилка
     fastcgi_pass php-upstream;           # Перенаправлення обробки
PHP-скриптів на FastCGI процес
     fastcgi_index index.php;              # Визначення index.php
як основного скрипта для FastCGI
     fastcgi_buffers 16 16k;                # Кількість і розмір
буферів для FastCGI
     fastcgi_buffer_size 32k;                # Розмір буфера для
читання заголовків FastCGI
     fastcgi_param                                SCRIPT_FILENAME
$document_root$fastcgi_script_name; # Параметр для визначення імені
скрипта
     fastcgi_read_timeout 600;             # Тайм-аут читання для
FastCGI у секундах
     include fastcgi_params;               # Включення додаткових
параметрів FastCGI
     }
     location ~ /\.ht {
     deny all;                              # Заборона доступу до
файлів, що починаються на .ht
      }
     location /.well-known/ {
     root /var/lib/certbot;                # Кореневий каталог для
файлів .well-known, використовуваних Let's Encrypt
      }
     location                                                    ~*
.(ico|css|js|gif|jpeg|jpg|png|woff|ttf|otf|svg|woff2|eot|webp)$ {
     expires 365d;                          # Час збереження кешу
статичних файлів (365 днів)
     add_header  Cache-Control  "public,  max-age=31536000";  #
Налаштування кешування для браузерів
```

```
    }
  }
```

Другий блок server (HTTP та HTTPS редирект)

```
  server {
    listen 80;                                  # Слухати HTTP запити
на порт 80
    listen 443 ssl http2;                      # Слухати HTTPS запити
на порт 443 з підтримкою SSL і HTTP/2
    server_name ~^squicket\.(.*)$;             # Задає імена серверів
за шаблоном для доменів типу squicket.*
    return     301     $scheme://www.$host$request_uri     ;     #
Перенаправлення всіх запитів на www версію з використанням поточного
протоколу (HTTP або HTTPS)
  }
```

Третій блок server (HTTP редирект для members і www)

```
  server {
    listen 80;                                  # Слухати HTTP запити
на порт 80
    server_name ~^(members|www)\.squicket\.(.*)$; # Задає імена
серверів за шаблоном для доменів типу members.squicket.* або
www.squicket.*
    return 301 https://$host$request_uri;  # Перенаправлення всіх
HTTP запитів на HTTPS версію сайту
  }
```

Цей конфігураційний файл сервера містить три секції сервера (блоки `server`), які використовуються для налаштування веб-сервера NGINX для обробки HTTP та HTTPS запитів для доменів, що відносяться до `squicket`. Кожен блок має свої специфічні директиви та параметри, які дозволяють забезпечити коректну роботу вебзастосунків.

Перший блок `server` (HTTPS налаштування):

1. Директиви `listen`: сервер слухає порт 443, що є стандартним для захищених SSL/TLS з'єднань. Використовується також `http2` для підтримки

HTTP/2 протоколу, який забезпечує ефективніше та швидше завантаження веб-сторінок.

2. `server_name`: задається регулярний вираз, що визначає імена серверів `members.squicket.*` та `www.squicket.*`, де `*` вказує на будь-який домен верхнього рівня.

3. `root` і `index`: визначають кореневий каталог файлів вебзастосунку і файли індексів, що будуть використовуватись за замовчуванням при запитах до каталогів.

4. `location /`: визначає обробку запитів до кореня сайту, використовуючи директиву `try_files` для спроби знайти файл або директорію, що відповідає запиту, або перенаправлення на `index.php`.

5. `location ~ \.php$`: конфігурація для обробки PHP файлів, з підтримкою FastCGI для зв'язку між NGINX та PHP-FPM.

6. `location ~ /\.ht`: забороняє доступ до файлів `.htaccess`, що є стандартною мірою безпеки.

7. `location /.well-known/`: спеціальний розділ для Let's Encrypt та інших сервісів автоматизації SSL/TLS сертифікації.

8. `location ~* \.(ico|css|js|gif|jpeg|jpg|png|woff|ttf|otf|svg|woff2|eot|webp)$`**: налаштування кешування для статичних файлів.

Другий блок `server` (HTTP та HTTPS редирект):

1. `listen`: слухає як порт 80 (стандартний HTTP), так і 443 (HTTPS).

2. `server_name`: задає сервер для домену `squicket.*`.

3. Редирект: всі запити перенаправляються на `www` версію сайту з використанням HTTPS.

Третій блок `server` (HTTP редирект для `members` і `www`):

1. `listen`: Слухає тільки порт 80.

2. `server_name`: Задає сервери для `members.squicket.*` і `www.squicket.*`.

3. Редирект: всі HTTP запити перенаправляються на HTTPS версії відповідних доменів.

Цей файл NGINX конфігурації забезпечує ефективну роботу вебзастосунку з використанням сучасних технологій та стандартів безпеки.

3.7. Реалізація основної логіки веб-додатку на Laravel

Спочатку налаштовано конфігурацію фреймворку та веб-додатку для сумісності з ресурсами сервера та налаштуваннями веб-сервера Nginx. Відбулась першопочаткова конфігурація модулів та пакетів, які будуть згодом використані.

Запрограмовано та реалізовану логіку віддачі веб-сторінок кінцевому користувачу з контроллерів використовуючи шаблони та двигун шаблонізації Blade. Також реалізовано механізм аутентифікації. Використаний стандартний механізм аутентифікації базований на куки (англ. Cookie-based authentication). Такий механізм забезпечує відслідковування попередніх та теперішніх сесій користувача, для того, щоб зберігати інформацію про користувача, який пройшов аутентифікацію. Таким чином, після успішної аутентифікації користувацький веб-браузер отримує куки з унікальним ідентифікатором сесії користувача в системі. Після повторних запитів до нашого вебзастосунку модуль аутентифікації в Laravel використовує такий ідентифікатор, щоб зібрати інформацію про користувача з нашої бази даних.

В додаток до цього запрограмовані спеціальні модулі імпорту та обробки даних для кожного з партнерів, які дають доступ до магазинів та їх продуктів. Зберігаємо всю потрібну інформацію про магазин в БД і після цього запускаємо імпорт продуктів в БД використовуючи відповідний модуль.

Імпортування кожного магазину може займати досить великий час, тому використані черги задач та виконавці таких задач. Кожна задача імпорту відправляється в загальну чергу задач, з якої кожен вільний на даний момент виконавчий процес берез задачі та опрацьовує їх в фоновому режимі, використовуючи окремий процес.

Також запрограмовано та реалізовано API контроллери, до яких звертаємось з клієнтської частини сайту та робимо запити на ті чи інші дані. Отже, буде загальний інтерфейс доступу до даних, які потрібно показати користувачу.

3.8. Налаштування бази даних PostgreSQL

Для кінцевого налаштування БД були зроблені такі кроки:

- встановлення серверу БД;
- опис конфігураційного файлу для серверу;
- завантаження, компіляція та встановлення розширення RUM для підтримки однойменного індексу для виконання повнотекстового пошуку на цільових таблицях;
- створення бази даних для нашого додатку;
- запуск команди виконання файлів міграції для створення потрібних таблиць для додатку;
- виконання команди підготовки таблиць, для заповнення частини таблиць в БД даними, які є критично необхідними для запуску веб-додатку.

3.9. Реалізація клієнтської частини

Для реалізації клієнтської частини вебзастосунку було зроблено:

- простий та мінімалістичний дизайн інтерфейсів;
- компоненти, що вміщають в себе основні секції сайту;
- компоненти, що реалізовують пошукові запити та запити даних з сервера використовуючи API;
- автоматичну локалізацію компонентів сайту відповідно до цільової країни користувача;
- партнерські сторінки реєстрації та аутентифікації;

- кабінет адміністратора з компонентами, що відповідають за моніторинг стану веб-додатку та компонентами;
- компоненти звітів для адміністраторів з детальною інформацією.

3.10. Демонстрація проекту

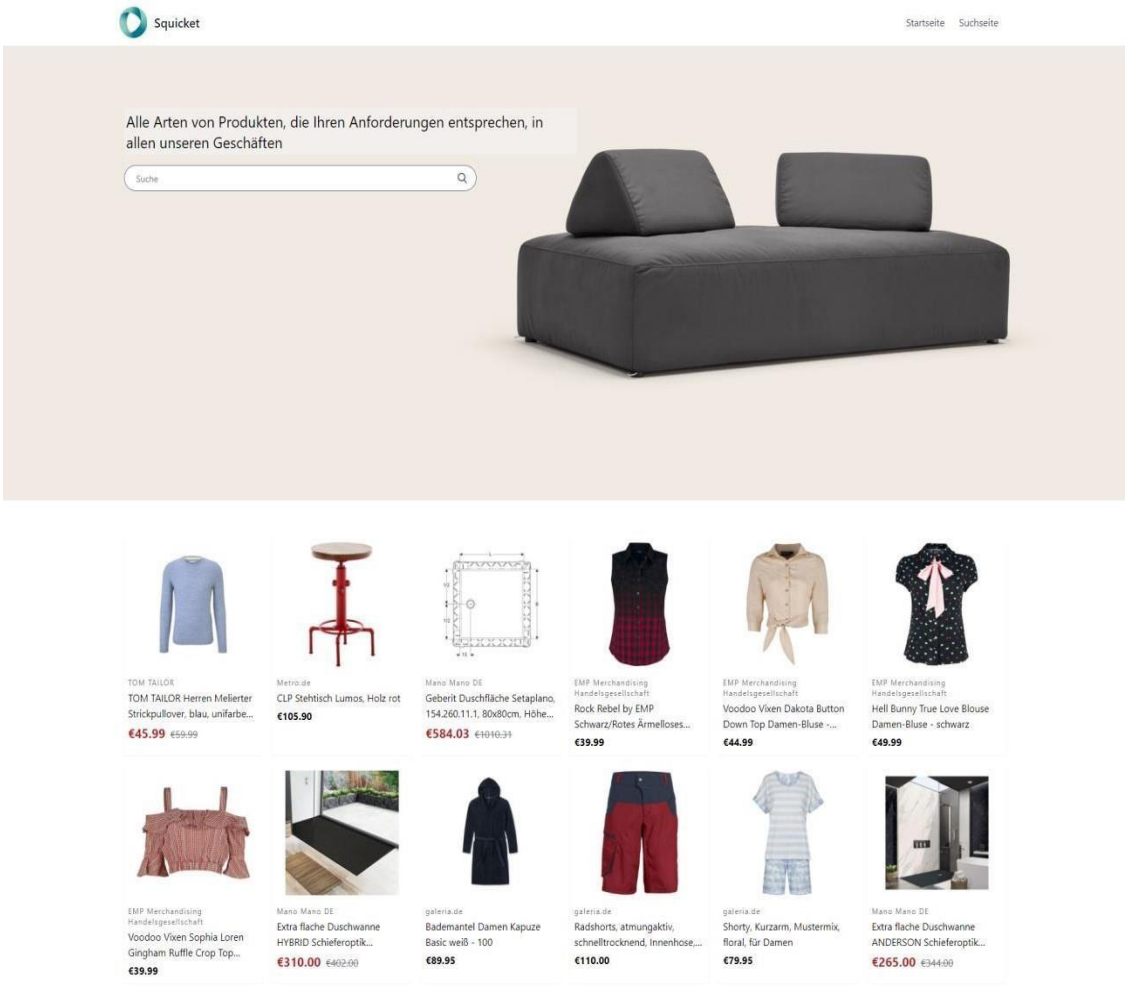

Рисунок 3.4 - Демонстрація проекту

Запускаючи сайт, відображається головна сторінка як на рисунку 3.4. У верхній частині виводиться пошукова стрічка, за допомогою якої можемо здійснювати пошукові запити. Нижче виводиться перелік доступних продуктів. Цей список завжди оновлюється з кожним перевантаженням сторінки і заповнюється продуктами, які були попередньо завантажені в нашу базу даних.

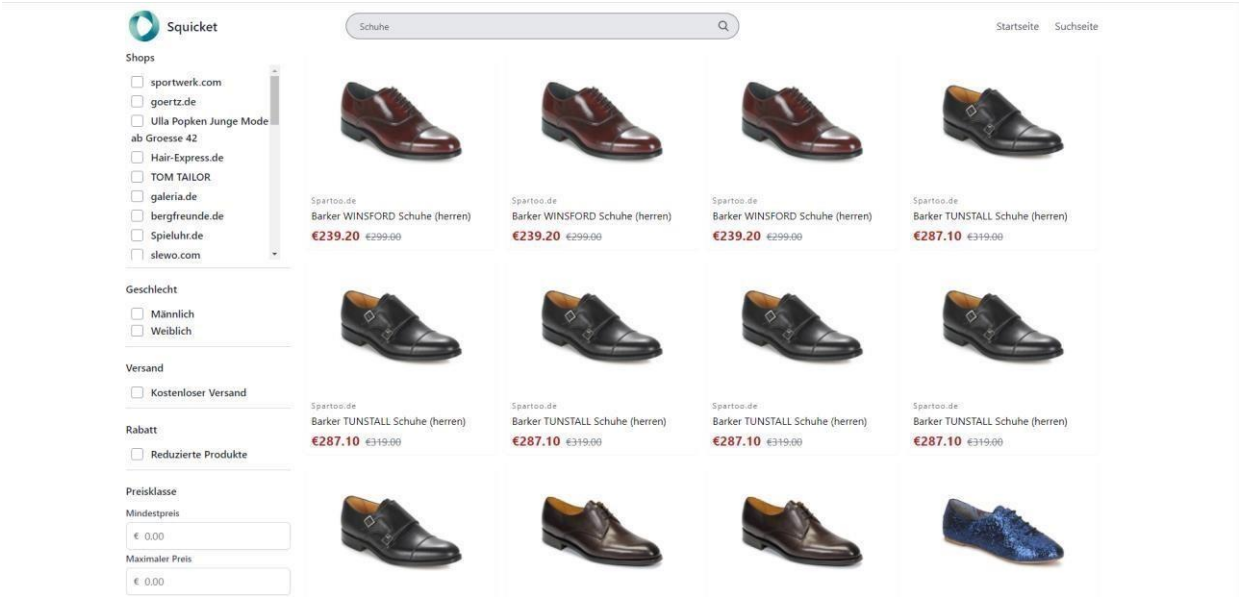

Рисунок 3.5 - Демонстрація сторінки пошуку

На рисунку 3.5 зображенно сторінку результатів пошуку. До прикладу, був заданий пошук по слову «взуття» (нім. Schuhe). Бачимо, що серед усіх можливих продуктів були вибрані та відфільтровані правильні результати. Також передбачена можливість фільтрів по конкретному магазину, ціновому діапазону, знижці, тощо.

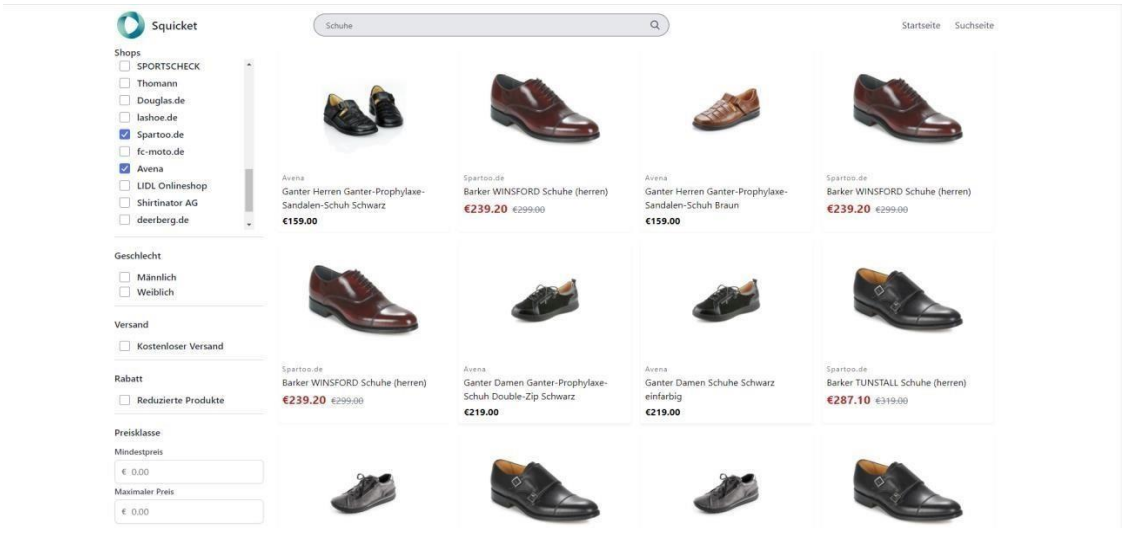

Рисунок 3.6 - Застосування філтрів

На рисунку 3.6 можна побачити, як застосований фільтр змінює кінцеву вибірку продуктів. Застосовано фільтр двох магазинів на пошуковий запит по слову «взуття». Також варто зазначити, що результати цих магазинів чергуються,

для того, щоб дати можливість користувачам саме порівнювати різні пропозиції та типи продуктів.

Рисунок 3.7 - Реалізація партнерської сторінки

На рисунку 3.7 зображена реалізація партнерської сторінки для користувачів, що хотіли б співпрацювати з системою. Ця сторінка має посилання на реєстрацію, аутентифікацію, а також інформаційні сторінки системи.

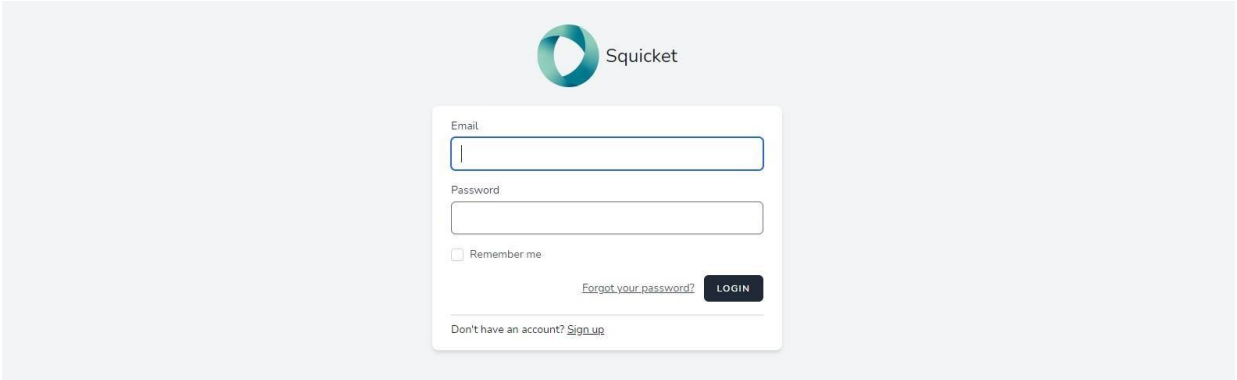

Рисунок 3.8 - Форма аутентифікації для клієнтів

На рисунку 3.8 зображено форма аутентифікації для клієнтів та сторінка реєстрації, де користувач може зареєструватись в систему.

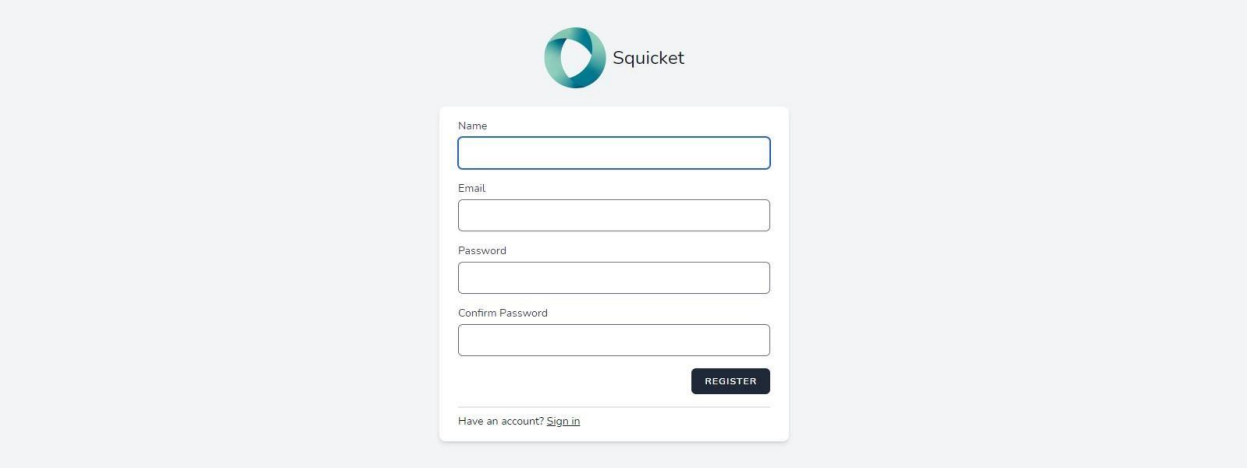

Рисунок 3.9 - Сторінка реєстрації

На рисунку 3.9 зображена сторінка реєстрації, на якій користувач може зареєструватись в систему. Вона приймає основні дані користувача, по яким він згодом зможе зайти в систему. Якщо дані мають неправильний формат, або користувач з такою поштою вже існує в системі, то він побачить помилки з повідомленнями, виведені у цю ж форму.

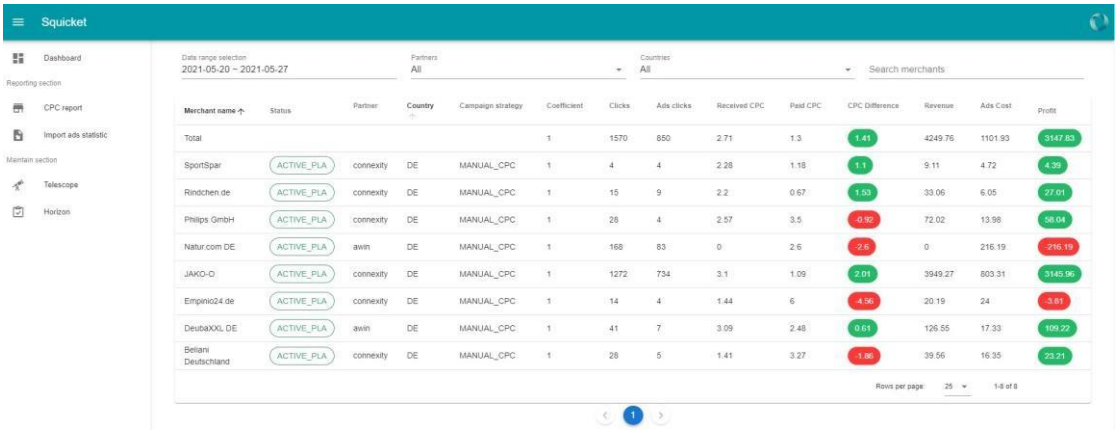

Рисунок 3.10 - Інтерфейс користувача (аутетифікований)

На рисунку 3.10 наведенно інтерфейс користувача, який виконав аутентифікацію у ролі адміністратора. Тут знаходиться таблиця з детальною статистикою про кожен з магазинів. Якщо користувач має на меті переглянути статистику за певний період, то йому варто скористатись передбаченим функціоналом та ввести дві дати. Також є можливість обрати партнера та країну, для яких користувач бажає переглянути статистику.

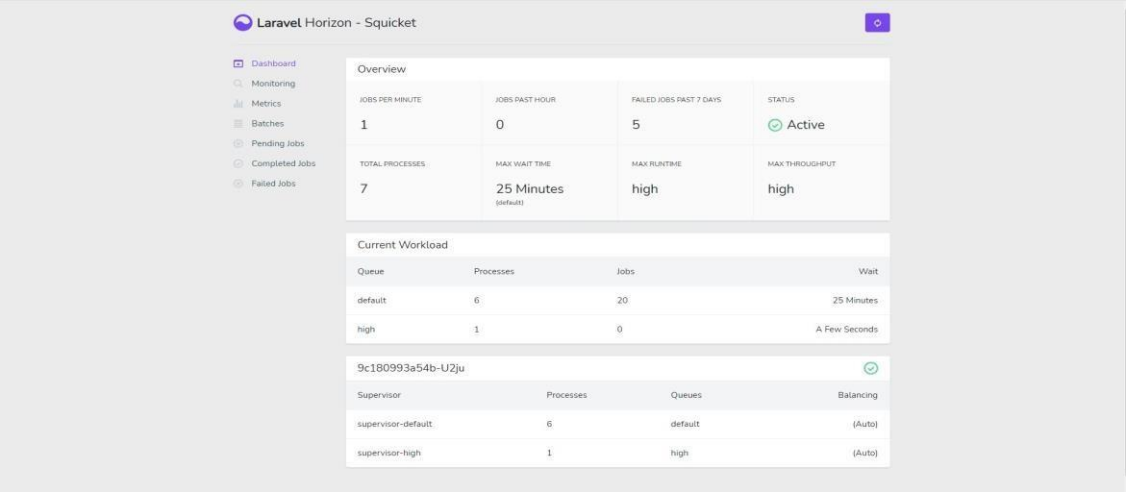

Рисунок 3.11 - Інтерфейс моніторингу черг та задач

На рисунку 3.11 зображено інтерфейс моніторингу черг та задач у нашій системі. Він в реальному часі показує інформацію про кількість виконаних задач, середню кількість часу, яка пішла на виконання, назви та кількість різних черг виконання.

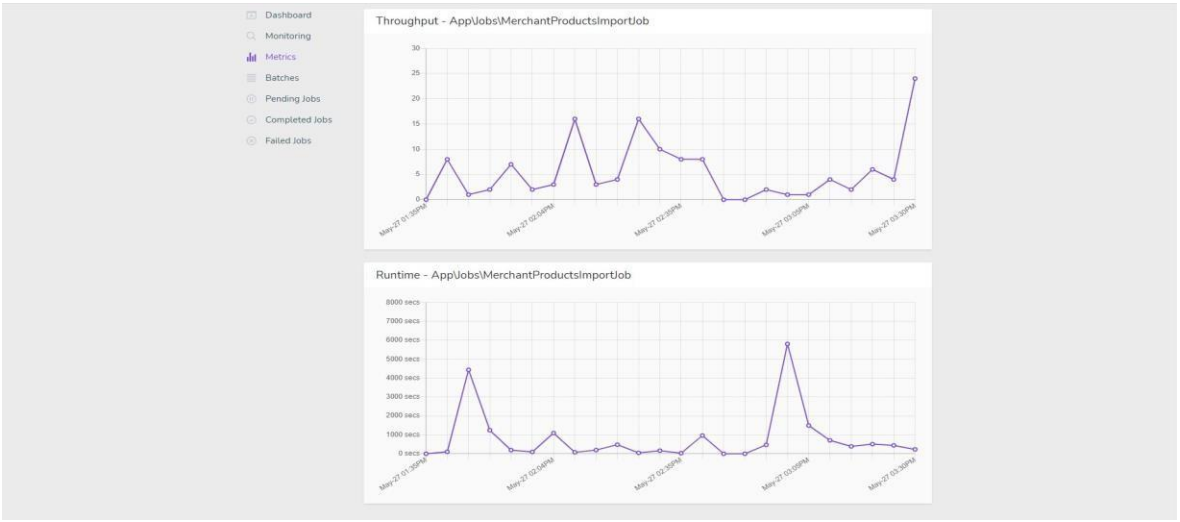

Рисунок 3.12 - Виконанні задачі

На рисунку 3.12 наведенно графіки з кількістю виконаних задач за останній час та кількість часу, яка пішла на ці задачі в кожен момент часу. Система автоматично оновлює інформацію кожні 5 хвилин і дописує у дані графікі нові значення.

ВИСНОВКИ

На основі виконаної роботи можна зробити наступні висновки:

1. *Ефективність технологічних рішень*: застосування мови програмування PHP та фреймворку Laravel дозволило створити ефективний та надійний фундамент для рекламно-пошукового вебзастосунку. Використання сучасних технологій сприяло реалізації повнотекстового пошуку, що є ключовим для забезпечення високої швидкості та точності обробки запитів користувачів.
2. *Дизайн і користувацький досвід*: мінімалістичний дизайн та простий інтерфейс вебзастосунку забезпечили високий рівень користувацької зручності. Це знижує поріг входження для нових користувачів та покращує загальне сприйняття вебзастосунку.
3. *Аналіз роботи та перспективи розвитку*: проведений аналіз функціоналу вебзастосунку підтвердив його ефективність та відкрив можливості для подальшого розвитку.
4. *Покращення системи управління*: ініціативи щодо автоматизації управління трафіком та якістю контенту значно покращують ефективність рекламних кампаній.

При розробці вебзастосунку були застосовані сучасні технології та методи, що забезпечують високу ефективність впроваджених рішень. Завдяки стратегічному підходу до планування та розробки проєкту досягнуто поставлених цілей у вирішенні технічних завдань. Реалізація повнотекстового пошуку та створення мінімалістичного дизайну забезпечили не тільки високу функціональність вебзастосунку, але й підвищили його привабливість для користувачів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Brent Roose. “LARAVEL BEYOND CRUD”. BeyondCRUD Publishing, 2020 - 350 с.
2. Vaughn Vernon. “Implementing Domain-driven Design”. Addison-Wesley Professional, 2013 - 560 с.
3. Robert C. Martin. “Clean Code: A Handbook of Agile Software Craftsmanship”. Prentice Hall, 2008 - 464 с.
4. Eric Evans. “Domain-Driven Design: Tackling Complexity in the Heart of Software”. Addison-Wesley Professional, 2003 - 560 с.
5. Martin Fowler. “Refactoring: Improving the Design of Existing Code”. Addison-Wesley Professional, 2018 - 448 с.
6. Kyle Simpson. “You Don't Know JS: Scope & Closures”. O'Reilly Media, 2014 - 98 с.
7. Nicholas C. Zakas. “JavaScript: The Good Parts”. O'Reilly Media, 2008 - 176 с.
8. Sandi Metz. “Practical Object-Oriented Design in Ruby”. Addison-Wesley Professional, 2012 - 272 с.
9. Michael Feathers. “Working Effectively with Legacy Code”. Prentice Hall, 2004 - 456 с.
10. Kent Beck. “Test-Driven Development: By Example”. Addison-Wesley Professional, 2003 - 240 с.
11. PHP: Hypertext Preprocessor [Електронний ресурс] Режим доступу: https://www.php.net (дата звернення 22.05.2024).
12. World Wide Web Consortium (W3C) [Електронний ресурс] Режим доступу: https://www.w3.org (дата звернення 23.05.2024).
13. Tailwind CSS [Електронний ресурс] Режим доступу: https://tailwindcss.com (дата звернення 24.05.2024).
14. Laravel - The PHP Framework For Web Artisans [Електронний ресурс] Режим доступу: https://laravel.com (дата звернення 12.05.2024).

15. Vue.js [Електронний ресурс] Режим доступу: https://vuejs.org (дата звернення 24.05.2024).

ДОДАТКИ

ДОДАТОК А

Представимо файл конфігурації сервера squicket.conf

```
server {
    listen 443 ssl http2;
    listen [::]:443 ssl http2 ipv6only=on;
    server_name ~^(members|www)\.squicket\.(.*)$;
    root /var/www/public;
    index index.php index.html index.htm;
    location / {
        try_files $uri $uri/ /index.php$is_args$args;
    }
    location ~ \.php$ {
        try_files $uri /index.php =404;
        fastcgi_pass php-upstream;
        fastcgi_index index.php;
        fastcgi_buffers 16 16k;
        fastcgi_buffer_size 32k;
        fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
        #fixes timeouts
        fastcgi_read_timeout 600;
        include fastcgi_params;
    }
    location ~ /\.ht {
        deny all;
    }
    location /.well-known/ {
        root /var/lib/certbot;
    }
    location ~* .(ico|css|js|gif|jpeg|jpg|png|woff|ttf|otf|svg|woff2|eot|webp)$ {
        expires 365d;
        add_header Cache-Control "public, max-age=31536000";
    }
```

```
}
server {
    listen 80;
    listen 443 ssl http2;
    server_name ~^squicket\.(.*)$;
    return 301 $scheme://www.$host$request_uri ;
}
server {
    listen 80;
    server_name ~^(members|www)\.squicket\.(.*)$;
    return 301 https://$host$request_uri;
}
```

Представимо файл конфігурації Nginx.Dockerfile

```
FROM nginx:alpine
RUN apk --update --no-cache add certbot tzdata openrc py3-pip
RUN pip3 install certbot-dns-cloudflare
WORKDIR /var/www
COPY nginx.conf /etc/nginx/nginx.conf
COPY renew /etc/periodic/weekly/renew
RUN chmod +x /etc/periodic/weekly/renew
RUN mkdir /var/lib/certbot
COPY startupConfiguration.sh /docker-entrypoint.d
RUN chmod +x /docker-entrypoint.d/startupConfiguration.sh
CMD ["nginx"]
EXPOSE 80 443
```

Представимо файл конфігурації бази даних Postgres

```
FROM postgres:14.4-alpine
RUN apk --update --no-cache add git make gcc libc-dev clang llvm-dev tzdata py-pip openssl-dev
libzip-dev lz4-dev gawk
RUN pip install pgxnclient
RUN pgxn install pg_repack
RUN mkdir -p /var/rum \
```

```
&& cd /var/rum \
&& git clone https://github.com/postgrespro/rum . \
&& make USE_PGXS=1 \
&& make USE_PGXS=1 install
RUN chown -hR postgres /var
COPY init_scripts/ /docker-entrypoint-initdb.d
RUN mkdir -p /var/pgconfigs
COPY pgconfigs/ /var/pgconfigs
EXPOSE 5432
```

Представимо файл основної сторінки користувача

```
<template>
  <div>
    <section class="body-font hero">
      <div class="container mx-auto flex px-5 py-24 md:flex-row flex-col items-center">
        <div class="lg:flex-grow flex flex-col md:items-start md:text-left mb-16 md:mb-0 items-center text-center">
          <h1 class="text-2xl mb-4 p-1 max-w-3xl font-normal bg-gray-100 rounded bg-opacity-50" v-html="$t('mainPage.heroTitle')"></h1>
          <search-bar class="relative text-gray-600 w-11/12 md:w-3/5 lg:w-2/5"></search-bar>
        </div>
      </div>
    </section>
    <section>
      <div class="container px-3 py-12 mx-auto">
        <div class="grid grid-cols-12">
          <product-card
            v-for="product in products"
            :key="product.id"
            :data-object="product"
            class="col-span-6 md:col-span-4 lg:col-span-2 p-2 m-2 shadow-sm hover:shadow-xl transition ease-out duration-500 rounded"
```

```
                ></product-card>
            </div>
        </div>
    </section>
  </div>
</template>

<script lang="ts">
import Vue from "vue";
import Component from "vue-class-component";
import ProductCard from "./utility/ProductCard.vue";
import SearchBar from "./utility/SearchBar.vue";
import {Product} from "./models/Product";
import {ProductContract} from "./contracts/ProductContract";

@Component({
  components: {ProductCard, SearchBar}
})
export default class MainPageContent extends Vue {
  private products: Array<Product> = [];

  public created() {
    this.$fsApi.requestProducts(60, 1).then((response: ProductContract[]) => {
      this.products = response.map((value: ProductContract) => {
        return Product.createFromPlain(value)
      });
    });
  }
}
</script>

<style scoped>

</style>
```

Представимо файл сторінки пошуку продуктів

```
<template>
    <div class="container m-auto px-2">
        <div class="sidebar">
            <div class="sidebar-backdrop" @click="toggleResponsiveSidebar()" v-
if="responsiveSidebarActive"></div>
            <transition name="slide">
                <filters-menu
                    :close-button="true"
                    class="sidebar-panel"
                    v-if="responsiveSidebarActive"
                ></filters-menu>
            </transition>
        </div>
        <div class="grid grid-cols-12">
            <filters-menu class="mb-2 col-span-2 hidden md:block"></filters-menu>
            <div class="px-4 col-span-12 md:col-span-10">
                <div class="grid grid-cols-12">
                    <transition name="fade">
                        <div v-show="isLoading && loadedProducts.length === 0" class="col-span-12 h-
screen flex items-center">
                            <loader class="mx-auto"></loader>
                        </div>
                    </transition>
                    <transition name="fade">
                        <div v-show="!isLoading && loadedProducts.length === 0" class="col-span-12 h-
screen flex">
                            <h4 class="mx-auto mt-12 text-xl font-medium" v-
html="$t('search.noProducts')"></h4>
                        </div>
                    </transition>
```

```
                <product-card
                    v-for="product in loadedProducts"
                    :key="product.id"
                    class="col-span-6 md:col-span-4 lg:col-span-3 p-2 m-2 shadow-sm hover:shadow-xl transition ease-out duration-500 rounded"
                    :data-object="product"
                ></product-card>
            </div>
            <div class="my-3">
                <button class="block bg-indigo-500 hover:bg-indigo-600 text-white font-semibold mx-auto py-2 px-3 transition ease-out duration-500 rounded-xl disabled:opacity-50"
                    v-html="$t('utility.loadMore')"
                    @click="loadProducts()"
                    :disabled="isLoading || (!isLoading && loadedProducts.length === 0)"
                ></button>
            </div>
        </div>
    </div>
</template>

<script lang="ts">
import Vue from "vue";
import Component from "vue-class-component";
import FiltersMenu from "./FiltersMenu.vue";
import ProductCard from "../utility/ProductCard.vue";
import {Root, RootMapper} from "../stores/modules/Root";
import {SidebarFilters, SidebarFiltersMapper} from "../stores/modules/SidebarFilters"
import {ProductsManager, ProductsManagerMapper} from "../stores/modules/ProductsManager"
import {store} from "../stores/main";
import Loader from "../utility/Loader.vue";

const rootContext = Root.context(store);
const sidebarFiltersContext = SidebarFilters.context(store);
```

```
const productsManagerContext = ProductsManager.context(store);

const SearchPageContentInit = Vue.extend({
    beforeRouteUpdate(to, from, next) {
        rootContext.mutations.setText(to.query.text as string);
        sidebarFiltersContext.mutations.loadFromQuery(to.query);
        productsManagerContext.actions.loadProducts(true);
        sidebarFiltersContext.actions.requestStores();
        next();
    },
    computed: {
        ...SidebarFiltersMapper.mapState([
            'responsiveSidebarActive'
        ]),
        ...ProductsManagerMapper.mapState([
            'loadedProducts',
            'isLoading'
        ])
    },
    methods: {
        ...RootMapper.mapMutations([
            'setText'
        ]),
        ...SidebarFiltersMapper.mapMutations([
            'toggleResponsiveSidebar',
            'loadFromQuery'
        ]),
        ...ProductsManagerMapper.mapActions([
            'loadProducts'
        ])
    }
})

@Component({
```

```
    components: {ProductCard, FiltersMenu, Loader}
})
export default class SearchPageContent extends SearchPageContentInit {
    public created() {
        this.setText(this.$route.query.text as string);
        this.loadFromQuery(this.$route.query);
        this.loadProducts(true);
    }
}
</script>

<style lang="scss">
.slide-enter-active,
.slide-leave-active {
    transition: transform 0.2s ease;
}

.slide-enter,
.slide-leave-to {
    transform: translateX(-100%);
    transition: all 150ms ease-in 0s
}

.sidebar-backdrop {
    background-color: rgba(0,0,0,.5);
    width: 100vw;
    height:  100vh;
    position: fixed;
    top: 0;
    left: 0;
    cursor: pointer;
    z-index: 15;
}
```

```
.sidebar-panel {
    overflow-y: auto;
    background-color: #ffffff;
    position: fixed;
    left: 0;
    top: 0;
    height: 101vh;
    z-index: 20;
    padding: 3rem 20px 2rem 20px;
    width: 300px;
}

.fade-enter-active, .fade-leave-active {
    transition: opacity .5s;
}
.fade-enter, .fade-leave-to {
    opacity: 0;
}
</style>

<style scoped>

</style>
```

Представимо файл для входу на кабінет

```
<x-guest-layout>
    <x-auth-card>
        <x-slot name="logo">
            <a href="{{ route('membersSite.home') }}">
                <x-application-logo class="w-20 h-20 fill-current text-gray-500" />
            </a>
        </x-slot>
```

```
        <!-- Session Status -->
        <x-auth-session-status class="mb-4" :status="session('status')" />

        <!-- Validation Errors -->
        <x-auth-validation-errors class="mb-4" :errors="$errors" />

        <form method="POST" action="{{ route('login') }}">
            @csrf

            <!-- Email Address -->
            <div>
                <x-label for="email" :value="__('Email')" />

                <x-input id="email" class="block mt-1 w-full" type="email" name="email" required
autofocus />
            </div>

            <!-- Password -->
            <div class="mt-4">
                <x-label for="password" :value="__('Password')" />

                <x-input id="password" class="block mt-1 w-full"
                                type="password"
                                name="password"
                                required autocomplete="current-password" />
            </div>

            <!-- Remember Me -->
            <div class="block mt-4">
                <label for="remember_me" class="flex items-center">
                    <input id="remember_me" type="checkbox" class="rounded border-gray-300 text-
indigo-600 shadow-sm focus:border-indigo-300 focus:ring focus:ring-indigo-200 focus:ring-opacity-
50" name="remember">
```

```
                <span class="ml-2 text-sm text-gray-600">{{ __('Remember me') }}</span>
            </label>
        </div>

        <div class="flex items-center justify-end mt-4">
            @if (Route::has('password.request'))
                <a class="underline text-sm text-gray-600 hover:text-gray-900" href="{{
route('password.request') }}">
                    {{ __('Forgot your password?') }}
                </a>
            @endif

            <x-button class="ml-3">
                {{ __('Login') }}
            </x-button>
        </div>
    </form>
    <div class="mt-4 pt-2 border-t-2 text-sm">
        Don't have an account? <a class="underline" href="{{ route('register') }}">Sign up</a>
    </div>
  </x-auth-card>
</x-guest-layout>
```

Представимо файл з кабінет користувача щоб витягнути звіт

```
<template>
  <v-container>
    <v-row>
      <v-col lg="3" sm="6" cols="12">
        <date-picker-field
          @update:date-range="selectDateRangeAction($event)"
```

```
    ></date-picker-field>
</v-col>
<v-col lg="3" sm="6" cols="12">
    <v-select
        :items="partners"
        :value="selectedPartner"
        @input="selectPartnerAction($event)"
        label="Partners"
    ></v-select>
</v-col>
<v-col lg="3" sm="6" cols="12">
    <v-select
        :items="countries"
        :value="selectedCountry"
        @input="selectCountryAction($event)"
        label="Countries"
    ></v-select>
</v-col>
<v-col lg="3" sm="6" cols="12">
    <v-text-field
        clearable
        :value="searchMerchantName"
        @input="debounceSearch($event)"
        label="Search merchants"
    ></v-text-field>
</v-col>
<v-data-table
    :headers="tableHeaders"
    :items="reports"
    :server-items-length="totalReportsNumber"
    :must-sort="true"
    :page="page"
    :items-per-page="perPage"
    :footer-props="{
```

```
    itemsPerPageOptions: [25, 50, 75],
    disablePagination: true,
    prevIcon: ",
    nextIcon: "
}"
    :sort-by="sortingOption.sort_field"
    :sort-desc="sortingOption.sort_field === 'DESC'"
    @update:page="selectPageAction($event)"
    @update:items-per-page="selectPerPageAction($event)"
    @update:sort-by="setSortByAction($event)"
    @update:sort-desc="setSortOrderAction($event === true ? 'DESC' : 'ASC')"
    class="elevation-1"
    style="width: 100%"
>
    <template
        v-slot:body.prepend="{ headers }"
    >
        <tr v-if="totalRow">
            <template v-for="header in headers">
                <td v-if="header.value === 'cpc_difference' || header.value === 'profit'">
                    <v-chip
                        :color="getColorForProfitLike(totalRow[header.value])"
                        v-html="totalRow[header.value]"
                        class="white--text"
                    >
                    </v-chip>
                </td>
                <td v-else v-html="totalRow[header.value]"></td>
            </template>
        </tr>
    </template>
    <template
        v-slot:item.merchant_status="{ item }"
    >
```

```
        <v-chip
          :color="getColorForStatus(item.merchant_status)"
          outlined
          v-html="item.merchant_status"
        >
        </v-chip>
      </template>
      <template
        v-slot:item.cpc_difference="{ item }"
      >
        <v-chip
          :color="getColorForProfitLike(item.cpc_difference)"
          v-html="item.cpc_difference"
          class="white--text"
        >
        </v-chip>
      </template>
      <template
        v-slot:item.profit="{ item }"
      >
        <v-chip
          :color="getColorForProfitLike(item.profit)"
          v-html="item.profit"
          class="white--text"
        >
        </v-chip>
      </template>
    </v-data-table>
    <v-pagination
      class="mx-auto"
      circle
      :length="totalPagesNumber"
      :value="page"
      @input="selectPageAction($event)"
```

```
            ></v-pagination>
        </v-row>
    </v-container>
</template>

<script lang="ts">
import Vue from "vue";
import Component from "vue-class-component";
import DatePickerField from "../../Components/DatePickerField.vue";
import {CpcReportManagerMapper} from "../../Stores/Modules/CpcReportManager";
import Timeout = NodeJS.Timeout;

const CpcReportInit = Vue.extend({
    computed: {
        ...CpcReportManagerMapper.mapState([
            'partners',
            'selectedPartner',
            'countries',
            'selectedCountry',
            'searchMerchantName',
            'sortingOption',
            'page',
            'perPage',
            'totalPagesNumber',
            'reports',
            'totalRow',
            'totalReportsNumber'
        ])
    },
    methods: {
        ...CpcReportManagerMapper.mapActions([
            'selectDateRangeAction',
            'selectPartnerAction',
            'selectCountryAction',
```

```
            'setSearchMerchantNameAction',
            'setSortingAction',
            'setSortByAction',
            'setSortOrderAction',
            'selectPageAction',
            'selectPerPageAction',
            'fetchReports',
            'fetchTotalRow',
            'fetchPartners',
            'fetchCountries'
        ])
    }
})

@Component({
    components: {DatePickerField}
})
export default class CpcReport extends CpcReportInit {
    created() {
        this.fetchPartners();
        this.fetchCountries();
        this.fetchReports();
        this.fetchTotalRow();
    }

    protected tableHeaders = [
        {
            text: 'Merchant name',
            value: 'merchant_name'
        },
        {
            text: 'Status',
            value: 'merchant_status'
        },
```

```
{
   text: 'Partner',
   value: 'partner'
},
{
   text: 'Country',
   value: 'country'
},
{
   text: 'Campaign strategy',
   value: 'campaign_strategy'
},
{
   text: 'Coefficient',
   value: 'revenue_coefficient'
},
{
   text: 'Clicks',
   value: 'clicks'
},
{
   text: 'Ads clicks',
   value: 'ads_clicks'
},
{
   text: 'Received CPC',
   value: 'received_cpc'
},
{
   text: 'Paid CPC',
   value: 'paid_cpc'
},
{
   text: 'CPC Difference',
```

```
        value: 'cpc_difference'
    },
    {
        text: 'Revenue',
        value: 'revenue'
    },
    {
        text: 'Ads Cost',
        value: 'ads_cost'
    },
    {
        text: 'Profit',
        value: 'profit'
    }
]

protected debounce: Timeout | null = null

protected debounceSearch(event: string) {
    if (this.debounce)
        clearTimeout(this.debounce);
    this.debounce = setTimeout(() => {
        this.setSearchMerchantNameAction(event);
    }, 600);
}

protected getColorForStatus(status: string) {
    switch (status) {
        case 'ACTIVE_PLA':
            return '#3D9970';
        case 'INACTIVE':
            return '#FF851B';
        case 'DEACTIVATED':
            return '#FF3838';
```

```
            case 'NOT_SET':
                return '#9EA7AD';
            case 'SUSPENDED':
                return '#FFB302';
            case 'OPTED_OUT':
                return '#FCE83A';
            case 'WAIT_FOR_SPACE':
                return '#FFB302';
            case 'ON_HOLD':
                return '#2DCCFF';
            case 'ACTIVE_SEARCH':
                return '#2DCCFF';
        }
    }

    protected getColorForProfitLike(profit: number) {
        if (profit >= 0)
            return '#23BA67';
        else
            return '#FF3838'
    }
}
</script>

<style scoped>

</style>
```

Представимо файл як виконується пошук продуктів на сайті

```
<?php
```

```
namespace App\Http\Controllers\FrontSiteApi;

use App\Http\Controllers\Controller;
use App\Http\Resources\ProductResource;
use App\Models\Product;
use App\Repositories\ProductRepository;
use App\Service\Support\BoundServiceNames;
use Illuminate\Http\Request;

class ProductsSearchController extends Controller
{
    private ProductRepository $productRepository;

    public function __construct(ProductRepository $productRepository)
    {
        $this->productRepository = $productRepository;
    }

    public function index(Request $request)
    {
        if (filter_var($request->input('random'), FILTER_VALIDATE_BOOLEAN)) {
            return ProductResource::collection($this->productRepository-
>getRandomProductsFast($request->input('perPage')));
        }
        /**
         * @var Product $products
         */
        $products = Product::with(['merchant:id,name'])->where('country_id',
\App::get(BoundServiceNames::REQUEST_COUNTRY_INFO)['id']);
        if ($request->input('text')) {
            $products = $products->fullTextSearch($request->input('text'))
                ->orderByTextSearch($request->input('text'))
                ->alternateBetweenMerchants();
        }
```

```
        if ($request->input('stores'))
            $products = $products->whereIn('merchant_id', $request->input('stores'));
        if ($request->input('gender'))
            $products = $products->where('gender', $request->input('gender'));
        if (filter_var($request->input('freeShipping'), FILTER_VALIDATE_BOOLEAN))
            $products = $products->where('shipping', 0);
        if (filter_var($request->input('discounted'), FILTER_VALIDATE_BOOLEAN))
            $products = $products->discountedProducts();
        if ($request->input('minPrice'))
            $products = $products->where('price', '>=', (float)$request->input('minPrice'));
        if ($request->input('maxPrice')) {
            if ((float)$request->input('maxPrice') > 0)
                $products = $products->where('price', '<=', (float)$request->input('maxPrice'));
        }
        return ProductResource::collection($products->simplePaginate(
            $request->input('perPage'),
            ['*'],
            'page',
            $request->input('page')
        ));
    }
}
```