НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
"КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО"

Факультет електроніки
(повна назва інституту/факультету)

Кафедра акустичних та мультимедійних електронних систем
(повна назва кафедри)

«До захисту допущено»
Завідувач кафедри

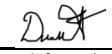
С.А. Найда
(ініціали , прізвище)
“ 14 01 202 р.

Магістерська дисертація

зі спеціальності 171 Електроніка
(код і назва)

на тему: "Розробка веб-сайту з функціональною адміністративною панеллю та інтеграцією платіжних систем"

Виконав: студент II курсу, групи ДВ-21мп
(шифр групи)

Димитров Артем 
(прізвище, ім’я, по батькові) (підпис)

Керівник к.т.н., доцент Філіпова Н.Ю. 
(посада, вчене звання, науковий ступінь, прізвище, ініціали) (підпис)

Рецензент доцент каф. ЕПС к.т.н., доц. Клен К.С.
(посада, науковий ступінь, вчене звання, прізвище, ініціали) (підпис)

Консультант
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище, ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент 
(підпис)

Київ – 2023 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Інститут (факультет) Факультет електроніки
(повна назва)

Кафедра Акустичних та мультимедійних електронних систем
(повна назва)

Рівень вищої освіти – другий (магістерський) за освітньо-професійною програмою;

Освітня програма Електронні системи мультимедіа та засоби Інтернету речей
(назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри

Сергій НАЙДА
(підпис) (ініціали, прізвище)
«01» 09 202 3р.

ЗАВДАННЯ

на магістерську дисертацію студенту

Димитрову Артему
(прізвище, ім'я, по батькові)

1. Тема роботи «Розробка веб-сайту з функціональною адміністративною панеллю та інтеграцією платіжних систем».

Науковий керівник дисертації к.т.н., доц. Філіпова Наталія Юріївна
(науковий ступінь, вчене звання, прізвище, ім'я, по батькові)
затверджені наказом по університету від « 09 » листопада 2023 р. №5218-с

2. Строк подання студентом дисертації 31.12.2023 р.

3. Об'єкт дослідження: Веб-сайт для інтернет магазину з адміністративною панеллю та системою оплати.

4. Предмет дослідження (Вхідні дані – для магістерської дисертації за освітньо-професійною програмою): використання сучасних веб-технологій для розробки веб-сайту з функціональною адміністративною панеллю та інтеграцією платіжних систем. Основні компоненти, які розглядаються, включають технології Next.js та Medusa для створення адміністративної панелі, бази даних для зберігання інформації про товари, інтеграцію платіжної системи для безпечних транзакцій, а також розробку користувацького інтерфейсу для оптимізації користувацького досвіду. Ці компоненти спрямовані на покращення управління контентом та ефективності електронної комерції.

5. Перелік завдань, які потрібно розробити: 1. Спроектувати структуру та архітектуру веб-сайту з використанням Next.js та Medusa Admin. 2. Реалізувати базу даних для зберігання інформації про товари. 3. Створити адміністративну панель для керування товарами та замовленнями. 4. Інтегрувати систему оплати та забезпечити безпеку транзакцій. 5. Протестувати та оптимізувати роботу веб-сайту.
6. Перелік графічного (ілюстративного) матеріалу: 37 рис., 1 презентація, основними назвами плакатів якої є сформульовані завдання, мета, постановка проблеми, особливості проведення процедур на основі інструментів та правил в спеціалізованій програмі.
7. Дата видачі завдання 09. 11. 2023 р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Строк виконання етапів магістерської дисертації	Примітка
1	Написання першого розділу	23.09.2023	Виконано
2	Написання другого розділу	01.10.2023	Виконано
3	Написання третього розділу	10.11.2023	Виконано
4	Написання четвертого розділу	05.12.2023	Виконано
5	Оформлення пояснювальної записки	20.12.2023	Виконано
6	Підготовка та оформлення презентації для доповіді	28.12.2023	Виконано

Студент _____ _____ Артем ДИМИТРОВ
(підпис) (ініціали, прізвище)

Науковий керівник _____ ___ Наталія ФІЛІПОВА
(підпис) (ініціали, прізвище)

УДК 004.738.5

РЕФЕРАТ

Димитров А. Розробка веб-сайту з функціональною адміністративною панеллю та інтеграцією платіжних систем: магістерська дис.: 171 Електроніка / Димитров Артем. – Київ, 2023. – 108 с.

Ключові слова: веб-розробка, електронна комерція, адміністративна панель, платіжні системи, Next.js, Medusa, безпека транзакцій, управління контентом, оптимізація інтерфейсу, користувацький досвід.

Актуальність дослідження. У сучасному світі величезну увагу приділяється цифровізації бізнес-процесів, зокрема в сфері електронної комерції. Ефективність, безпека та зручність веб-сайтів для електронної комерції мають вирішальне значення для успіху бізнесу в цифровому світі. Розвиток та інтеграція функціональних адміністративних панелей та платіжних систем є ключовими факторами, що впливають на якість обслуговування клієнтів та забезпечення безпеки фінансових операцій. Це дослідження актуальне, оскільки воно відповідає сучасним потребам ринку в створенні гнучких, безпечних та користувацьки-орієнтованих рішень для веб-сайтів, які сприяють ефективному управлінню електронною комерцією та покращенню загального досвіду користувача.

Метою дослідження є створення веб-сайту для електронної комерції з функціональною адміністративною панеллю та інтеграцією платіжних систем за допомогою технології Next.js та Medusa, а також надання ефективних, безпечних, та користувацьки-орієнтованих рішень для покупців

Предмет дослідження – процес розробки та імплементації функціональної адміністративної панелі для веб-сайту електронної комерції, а також інтеграція платіжних систем. Включає аналіз вимог до користувацького інтерфейсу, безпеки даних, та ефективності управління контентом і транзакціями.

Методи дослідження – розробка веб-сайту електронної комерції на основі готових технологій та аналіз компонентів для реалізації системи.

Наукова новизна одержаних результатів: створення інтегрованого рішення, що поєднує передові технології в області розробки веб-сайтів з інноваційними методами управління контентом та транзакціями. Особливу увагу приділено розробці

безпечних та ефективних механізмів інтеграції платіжних систем, які підвищують надійність та зручність використання.

Практичне значення одержаних результатів: В рамках цього дослідження було розроблено веб-сайт для електронної комерції з інтегрованою адміністративною панеллю та мануальною платіжною системою. Проведено детальний аналіз сучасних технологій, що використовувалися для реалізації проекту. Отримані результати можуть бути застосовані для подальшого розвитку та вдосконалення веб-сайту у галузі електронної комерції, забезпечуючи більш ефективне управління контентом, покращення безпеки транзакцій та поліпшення користувацького досвіду.

ABSTRACT

Dymytrov A. Development of a Website with a Functional Administrative Panel and Integration of Payment Systems: Master's Thesis: 171 Electronics / Dymytrov Artem. - Kyiv, 2023. - 108 p.

Keywords: web development, e-commerce, administrative panel, payment systems, Next.js, Medusa, transaction security, content management, user interface optimization, user experience.

This thesis presents the development of a comprehensive website for e-commerce, featuring a functional administrative panel and integrated payment systems. The initial part provides an analytical overview of the current trends and requirements in e-commerce website development. It includes a comparative study of various technologies and platforms like WordPress, Drupal, Shopify, with a specific focus on Next.js and Medusa for the project implementation.

The core of the thesis is dedicated to the design and architecture of the website, utilizing Next.js and Medusa Admin for efficient content management and transaction handling. The development process encompassed the creation of a user-friendly administrative panel, database design for product information storage, and seamless integration of payment systems ensuring transaction security.

The study also covers the aspects of user interface (UI) and user experience (UX) optimization, with the aim to enhance the effectiveness and accessibility of the website. The security analysis of the web application, including transaction safety and data protection, is elaborately discussed.

In the final part, the thesis showcases the testing and optimization strategies employed to ensure the website's functionality and performance. The practical application of the project is illustrated through its potential in improving e-commerce operations, contributing to a more secure and user-friendly online shopping environment. The research and development carried out in this thesis provide valuable insights and methodologies for future advancements in the field of e-commerce web development.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ І ТЕРМІНІВ

HTML – HyperText Markup Language. Мова розмітки гіпертексту, використовується для створення та структурування вмісту на веб-сторінках.

CSS – Cascading Style Sheets. Мова стилів, що використовується для оформлення вигляду веб-сторінок, наприклад, для визначення шрифтів, кольорів та розташування елементів.

JS – JavaScript. Мова програмування, що використовується для створення інтерактивних ефектів на веб-сторінках.

Next.js – JavaScript фреймворк для побудови серверно-рендерованих JavaScript-додатків, особливо веб-сайтів.

Medusa – Серверний фреймворк або бекенд-система, використовувана для розробки електронної комерції, забезпечує управління контентом та операціями з транзакціями.

UI – User Interface. Інтерфейс користувача, частина веб-сайту або додатку, з якою користувач взаємодіє.

UX – User Experience. Досвід користувача, загальний вплив, який отримує користувач від взаємодії з веб-сайтом або додатком.

API – Application Programming Interface. Набір процедур, протоколів та інструментів для створення програмного забезпечення і додатків.

SQL – Structured Query Language. Мова запитів для керування та маніпулювання базами даних.

HTTPS – HyperText Transfer Protocol Secure. Розширена версія HTTP з додаванням шару безпеки для шифрування даних.

ВСТУП

У сучасному світі електронна комерція зазнає стрімкого розвитку, завдяки широкому доступу до інтернету та збільшенню числа онлайн-платформ для торгівлі.

Веб-сайти є одними з основних засобів, які допомагають підприємствам взаємодіяти з клієнтами, презентувати та продавати свою продукцію. Однак, створення функціонального та ефективного веб-сайту є складним завданням, яке вимагає глибоких технічних знань та розуміння бізнес-процесів. Основна проблема, яка розглядається в цьому дослідженні, полягає у розробці веб-сайту з функціональною адміністративною панеллю та інтеграцією платіжних систем. Адміністративна панель є важливою частиною будь-якого веб-сайту електронної комерції, оскільки вона дозволяє управлінському персоналу легко керувати контентом сайту, замовленнями, клієнтами та іншими аспектами бізнесу. Інтеграція платіжних систем є критично важливою для обробки платежів та забезпечення безпечних фінансових транзакцій.

Актуальність дослідження визначається декількома факторами:

Розвиток технологій: Нові технології та фреймворки, такі як Next.js та Medusa, надають розробникам потужні інструменти для створення високопродуктивних веб-сайтів з відмінним користувацьким досвідом.

Зростання електронної комерції: Зі збільшенням обсягу онлайн-торгівлі зростає попит на якісні веб-сайти, які можуть задовольнити вимоги як підприємств, так і їхніх клієнтів.

Безпека платежів: Безпечні та надійні платіжні системи є ключовими для захисту фінансових даних клієнтів та забезпечення їхньої довіри.

Розробка веб-сайту з функціональною адміністративною панеллю та інтегрованими платіжними системами є важливим кроком на шляху до створення ефективних та безпечних платформ для електронної комерції, які можуть задовольнити потреби сучасного ринку.

Мета та завдання дослідження

Основна мета даного дослідження полягає у розробці веб-сайту для електронної комерції з функціональною адміністративною панеллю та інтеграцією платіжних систем за допомогою технології next-theme-medusa-admin. Сучасний ринок електронної комерції вимагає від підприємств надання ефективних, безпечних та користувацьки-орієнтованих рішень для покупців, і цей проект має на меті розробити веб-сайт, який відповідає цим вимогам.

Завдання дослідження:

1. Аналіз та вибір технологій:

- Проаналізувати існуючі технологічні рішення та фреймворки для розробки веб-сайтів, зокрема Next.js, Medusa та інші суміжні технології.
- Вибрати оптимальні технології для розробки веб-сайту, адміністративної панелі та інтеграції платіжних систем, звертаючи увагу на безпеку, продуктивність та швидкість розробки

2. Проектування архітектури веб-сайту:

- Розробити загальну архітектуру веб-сайту та адміністративної панелі, визначити основні модулі та їх взаємозв'язки.
- Скласти детальну схему інтеграції платіжних систем, звертаючи особливу увагу на безпеку транзакцій та конфіденційність даних користувачів.

3. Розробка веб-сайту та адміністративної панелі:

- Реалізувати фронтенд та бекенд веб-сайту з використанням вибраних технологій, забезпечити високу продуктивність та оптимізацію для пошукових систем.
- Розробити адміністративну панель, яка забезпечує легке управління продукцією, замовленнями, користувачами та іншими аспектами веб-сайту.

4. Інтеграція платіжних систем:

- Вибрати надійні та перевірені платіжні системи для інтеграції з веб-сайтом.
- Реалізувати безпечну та ефективну систему обробки платежів, забезпечити коректну обробку транзакцій та повернень грошей.

5. Тестування та оптимізація:

- Провести комплексне тестування веб-сайту, включаючи функціональне, безпекове та навантажувальне тестування.
- Оптимізувати продуктивність веб-сайту, швидкість завантаження сторінок та коректність роботи на різних пристроях та браузерах.

6. Розгортання та моніторинг:

- Розгорнути веб-сайт на сервері, налаш тувати резервне копіювання даних та систему моніторингу.
- Забезпечити стабільну роботу веб-сайту, відстежувати можливі проблеми та відгукуватися на відгуки користувачів для подальшого вдосконалення системи.

7. Документація та передача:

- Підготувати повну технічну документацію по проекту, включаючи опис архітектури, модулів, алгоритмів та інструкції з розгортання та обслуговування веб-сайту.
- Провести передачу проекту замовнику або команді підтримки, забезпечити необхідне навчання та консультування.

РОЗДІЛ 1. ТЕОРЕТИЧНИЙ АНАЛІЗ ТА ВИБІР ТЕХНОЛОГІЙ

1.1 Огляд існуючих рішень для розробки веб-сайтів з адміністративними панелями

Існує безліч технологій та платформ для розробки веб-сайтів з адміністративними панелями. Вони надають розробникам готові інструменти для створення, управління та моніторингу веб-сайтів. Ось декілька популярних рішень:

1. WordPress - є однією з найбільш популярних платформ для створення веб-сайтів та блогів. Вона має потужну адміністративну панель, яка дозволяє користувачам керувати контентом, користувачами, плагінами та темами без потреби в глибоких технічних знань.

2. Drupal - це ще одна популярна CMS, яка пропонує гнучкі можливості для розробки веб-сайтів з розширеними адміністративними функціями.

3. Joomla є відмінним вибором для створення веб-сайтів з адміністративними панелями, особливо для середніх та великих проектів.

4. Magento є спеціалізованою платформою для електронної комерції, яка має потужні адміністративні інструменти для управління магазинами.

5. Shopify - це одна з найлегших платформ для створення веб-сайтів для електронної комерції з адміністративними панелями.

6. Next.js + Medusa. Комбінація **Next.js** для фронтенду та Medusa для бекенду може надати потужні інструменти для створення високопродуктивних та гнучких веб-сайтів з адміністративними панелями.

7. Contentful + Gatsby. Contentful як headless CMS в комбінації з Gatsby може бути відмінним рішенням для створення модернізованих веб-сайтів з адміністративними панелями.

Ці платформи та технологічні стеки представляють широкий спектр можливостей для розробки веб-сайтів з адміністративними панелями. Вибір конкретного рішення залежить від багатьох факторів, включаючи технічні вимоги проекту, бюджет, терміни та ресурси, доступні для розробки та підтримки веб-сайту.

1.1.1 WordPress

WordPress — це популярна система керування вмістом, яка дозволяє користувачам легко створювати та керувати веб-сайтами[1]. Він надає широкий спектр функцій, включаючи настроювані теми, плагіни та віджети, що робить його потужним інструментом для веб-розробки. Крім того, WordPress зручний і не вимагає спеціальних навичок або знань для використання. За допомогою WordPress користувачі можуть створити гарний і функціональний веб-сайт без зайвих витрат. Однією з головних переваг використання WordPress для веб-розробки є його гнучкість, масштабованість і зручний індерфейс, який показано на рис. 1.1.

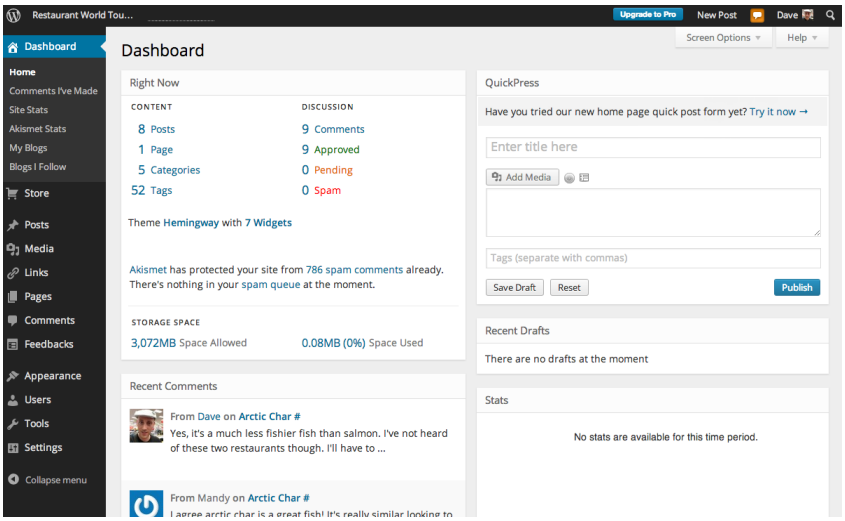


Рис. 1.1. Інтерфейс застосунку WordPress

WordPress можна використовувати для створення різноманітних веб-сайтів, від простих блогів до складних сайтів електронної комерції. Він також пропонує велику бібліотеку плагінів і тем, що дозволяє користувачам налаштовувати свій сайт відповідно до своїх конкретних потреб[1]. Крім того, WordPress є дружнім до SEO,

завдяки чому веб-сайтам легше займати вищі позиції в результатах пошуку. Незважаючи на деякі недоліки, такі як потенційна вразливість системи безпеки та необхідність регулярних оновлень, переваги використання WordPress для веб-розробки значно переважують недоліки[2].

Встановлення WordPress — це простий процес, який можна виконати за кілька простих кроків. Користувачі можуть встановити WordPress вручну або скористатися опцією встановлення одним клацанням миші, наданою їхнім постачальником веб-хостингу[2]. Після встановлення користувачі можуть почати налаштовувати свій сайт і створювати вміст. Також важливо переконатися, що сайт оптимізовано для пошукових систем, що можна зробити, створивши файл robots.txt і надіславши його в Google. Виконуючи ці дії, користувачі можуть створити функціональний та оптимізований веб-сайт за допомогою WordPress.

WordPress пропонує широкий вибір тем і плагінів, які можна налаштувати відповідно до потреб будь-якого веб-сайту. Ці теми та плагіни можна легко встановити та налаштувати для забезпечення бажаної функціональності веб-сайту. Команда підтримки WordPress.com готова надати допомогу в установці та усуненні несправностей цих тем і плагінів. За допомогою плагінів користувачі можуть додавати нові функції на свій веб-сайт, такі як контактні форми, інтеграція соціальних медіа та функції електронної комерції. Параметри налаштування, доступні з темами та плагінами WordPress, роблять його універсальною платформою для розробки веб-сайтів. Пошукова оптимізація (SEO) є важливою складовою успіху будь-якого веб-сайту, і WordPress пропонує кілька функцій, які можуть допомогти з SEO. WordPress розроблено для оптимізації пошукових систем із такими функціями, як чисті URL-адреси, настроювані мета-теги та автоматичні карти сайту. Крім того, для WordPress доступні кілька плагінів SEO, наприклад Yoast SEO та All in One SEO Pack. Ці плагіни можуть допомогти користувачам оптимізувати свій вміст для пошукових систем, проаналізувати продуктивність свого веб-сайту та покращити видимість свого сайту в результатах пошукової системи. За допомогою цих плагінів користувачі можуть розробити стратегію WordPress SEO для залучення трафіку на свій веб-сайт.

WordPress пропонує кілька варіантів розміщення, включаючи безкоштовний хостинг, спільний хостинг і керований хостинг WordPress. Коли справа доходить до розміщення веб-сайту WordPress, безпека є головною проблемою[2]. Послуги керованого хостингу WordPress забезпечують більш безпечну платформу для розміщення веб-сайтів WordPress із такими функціями, як автоматичне оновлення, резервне копіювання та сканування зловмисного програмного забезпечення. Крім того, користувачі можуть вжити додаткових заходів для захисту свого веб-сайту WordPress, наприклад вибрати надійного хостинг-провайдера та запровадити плагіни безпеки. Вибираючи надійного хостинг-провайдера та вживаючи належних заходів безпеки, користувачі можуть забезпечити безпеку та надійність свого веб-сайту WordPress[2].

Переваги та обмеження

Переваги:

1. Зручний інтерфейс, WordPress має простий та інтуїтивно зрозумілий інтерфейс, що дозволяє легко створювати та керувати веб-сайтами навіть для нетехнічних користувачів.

2. Параметри налаштування. Завдяки тисячам доступних тем і плагінів WordPress пропонує широкий спектр варіантів налаштування відповідно до потреб різних веб-сайтів.

3. Зручність для пошукових систем: WordPress розроблено з урахуванням оптимізації пошукових систем (SEO), що полегшує веб-сайтам отримувати вищі позиції в результатах пошуку.

4. Масштабованість: незалежно від того, чи є у вас невеликий блог чи великий веб-сайт електронної комерції, WordPress може працювати з веб-сайтами будь-якого розміру та масштабу.

5. Підтримка спільноти: WordPress має велику й активну спільноту користувачів і розробників, які надають підтримку, навчальні посібники та ресурси.

Недоліки:

1. Ризики в безпеці. Будучи платформою з відкритим кодом, WordPress вразливий до загроз безпеці. Користувачі повинні вживати належних заходів для забезпечення безпеки своїх веб-сайтів.

2. Крива навчання. Незважаючи на те, що WordPress є дружньою до користувача, все ще є крива навчання, особливо для новачків, які не знайомі з розробкою веб-сайтів.

3. Обмежений контроль. Оскільки WordPress є хостинговою платформою, користувачі мають обмежений контроль над серверним середовищем і функціональністю серверної частини. Це може бути обмеженням для досвідчених користувачів, яким потрібні додаткові параметри налаштування.

4. Сумісність плагінів: через величезну кількість доступних плагінів можуть виникнути проблеми сумісності. Іноді плагіни можуть погано працювати разом або спричиняти конфлікти з веб-сайтом.

5. Продуктивність. Залежно від теми та плагінів, які використовуються, веб-сайти WordPress іноді можуть мати повільніше завантаження або проблеми з продуктивністю.

Вирішення недоліків:

1. Плагіни безпеки: встановлення плагінів безпеки може допомогти підвищити безпеку веб-сайтів WordPress і захистити від потенційних уразливостей.

2. Навчальні посібники та документація: використовуйте величезну кількість навчальних посібників і документації, доступних в Інтернеті, щоб дізнатися та зрозуміти тонкощі WordPress.

3. Самостійний WordPress: подумайте про використання самостійного хостингу WordPress, щоб мати більше контролю над серверним середовищем і функціями серверної частини.

4. Регулярні оновлення та технічне обслуговування: оновлюйте WordPress, теми та плагіни, щоб мінімізувати проблеми сумісності та покращити продуктивність.

5. Кешування та оптимізація: запровадьте плагіни кешування або методи оптимізації, щоб покращити швидкість завантаження та продуктивність веб-сайтів WordPress.

Підсумовуючи WordPress пропонує численні переваги, такі як зручний інтерфейс, можливості налаштування, зручність для SEO, масштабованість і спільнота підтримки. Однак він також має обмеження, зокрема ризики безпеці, криву навчання, обмежений контроль, проблеми сумісності плагінів і потенційні проблеми з продуктивністю. Впроваджуючи заходи безпеки, використовуючи доступні ресурси, розглядаючи WordPress із власним розміщенням, оновлюючи програмне забезпечення та оптимізуючи веб-сайти, користувачі можуть пом’якшити ці обмеження та максимально використати переваги WordPress[3].

1.1.2 Drupal

Drupal — це популярна та потужна система керування контентом (CMS), яка дозволяє користувачам створювати широкий спектр цифрового контенту[3]. Він забезпечує інтуїтивно зрозумілий інтерфейс для керування сторінками, що дозволяє користувачам легко створювати, редагувати та впорядковувати вміст. Drupal пропонує різноманітні інструменти для керування вмістом, включаючи можливість створювати нові типи вмісту, додавати поля до існуючих типів вмісту та керувати відображенням вмісту. Завдяки цим функціям Drupal є гнучкою системою керування вмістом (CMS), яку можна налаштовувати та використовувати для різноманітних веб-сайтів, від невеликих блогів до великих складних корпоративних сайтів.

Однією з ключових переваг Drupal є його гнучкість і можливості налаштування. Завдяки понад 46 000 доступних модулів Drupal можна легко адаптувати до конкретних потреб будь-якого веб-сайту. Він також має високу масштабованість, що робить його ідеальним вибором для підприємств, яким потрібні розширені функції та налаштування[8]. Drupal Commerce, наприклад, є гнучкою та оптимізованою для SEO

платформою електронної комерції, яка підтримує обробку складних форм оплати. Ця гнучкість робить Drupal популярним вибором для підприємств будь-якого розміру, від стартапів до великих підприємств[3].

Ще однією важливою перевагою Drupal є його спільнота з відкритим кодом і підтримка. Ця спільнота розробників створює та підтримує широкий спектр модулів і тем, постійно вдосконалюючи та оновлюючи платформу. Крім того, природа Drupal з відкритим вихідним кодом означає, що він вільний у використанні та може бути легко інтегрований із програмами сторонніх розробників. Ця інтеграція в поєднанні з оптимізацією Drupal для SEO та підтримкою спільноти робить його привабливим варіантом для компаній, які прагнуть створити надійну та динамічну онлайн-присутність[3].

Переваги використання Drupal для розробки веб-сайтів

Drupal — це масштабована та продуктивна система керування вмістом, яка ідеально підходить для створення веб-сайтів із високим трафіком. Продуктивність і масштабованість Drupal є одними з його найважливіших переваг, що робить його чудовим вибором для веб-сайтів, які потребують швидкої та ефективної обробки даних[10]. Система кешування Drupal розроблена для скорочення часу завантаження сторінки та вимог до ресурсів, покращуючи продуктивність веб-сайту. Крім того, масштабованість Drupal гарантує, що веб-сайт може впоратися зі сплесками трафіку без будь-яких проблем з продуктивністю. Ці функції роблять Drupal популярним вибором для компаній і організацій, яким потрібен високомасштабований і ефективний веб-сайт. Drupal також відомий своєю винятковою безпекою та надійністю. Drupal розроблений для запобігання критичних вразливостей безпеки, включаючи 10 найпоширеніших ризиків безпеки, забезпечуючи безпеку та надійність веб-сайтів, створених на платформі. Drupal забезпечує надійне шифрування бази даних із багатьма опціями для захисту конкретної інформації, що додатково підвищує безпеку веб-сайту. Просте створення вмісту, надійна продуктивність і чудова безпека є стандартними функціями Drupal, що робить його ідеальним вибором для підприємств і організацій, яким потрібен безпечний і надійний веб-сайт[4].

Гнучкість Drupal і здатність інтегруватися зі сторонніми програмами та службами є додатковими перевагами. Гнучкість Drupal дозволяє розробникам створювати власні модулі та теми відповідно до конкретних вимог веб-сайту. Інтеграційні можливості Drupal дозволяють інтегрувати його зі сторонніми програмами, живим чатом, багатомовністю, платіжними системами, службами доставки та аналітичними службами[4]. Ці функції роблять Drupal універсальною та настроюваною платформою, яку можна адаптувати до конкретних потреб будь-якого бізнесу чи організації.

Переваги та обмеження

Переваги:

1. Гнучка та настроювана: Drupal надає дуже гнучку та настроювану платформу для розробки веб-сайтів. Він пропонує широкий спектр модулів і тем, які можна використовувати для створення унікальних і адаптованих веб-сайтів відповідно до конкретних вимог.

2. Масштабованість: Drupal здатний обробляти великі обсяги вмісту та трафіку, що робить його придатним для веб-сайтів із великим обсягом трафіку або складною структурою контенту. Його архітектура дозволяє легко масштабувати та рости відповідно до потреб веб-сайту.

3. Активна підтримка спільноти: Drupal має велику й активну спільноту розробників і користувачів, які роблять внесок у його розвиток і надають підтримку. Цей підхід, керований спільнотою, забезпечує регулярне оновлення, виправлення помилок і велику кількість ресурсів для користувачів.

4. Безпека: Drupal приділяє значну увагу безпеці та відомий своїми надійними функціями безпеки. Випускаються регулярні оновлення безпеки та виправлення, які гарантують, що веб-сайти, створені на Drupal, захищені від вразливостей і загроз.

Недоліки:

1. Крута крива навчання: Drupal має круту криву навчання, особливо для новачків, які мають обмежені технічні знання або досвід роботи з системами

керування контентом. Його складна архітектура та термінологія можуть ускладнити його розуміння та навігацію для нових користувачів.

2. Потрібна технічна експертиза: для створення та підтримки веб-сайту Drupal часто потрібні технічні знання, зокрема знання PHP, HTML і CSS. Це може бути обмеженням для користувачів, які не мають доступу до кваліфікованих розробників або не вміють працювати з кодом.

3. Обмежені готові теми: Хоча Drupal пропонує широкий спектр модулів, вибір готових тем відносно обмежений порівняно з іншими системами керування вмістом. Це може ускладнити пошук теми, яка відповідає конкретним уподобанням у дизайні чи вимогам до бренду[4].

Вирішення недоліків:

1. Навчання та документація: користувачі можуть подолати круту криву навчання, інвестуючи в навчальні ресурси та документацію, надані спільнотою Drupal. Онлайн-підручники, документація та форуми можуть допомогти користувачам краще зрозуміти платформу та орієнтуватися в ній.

2. Співпраця з розробниками: користувачі без технічних знань можуть співпрацювати з досвідченими розробниками Drupal або агентствами для створення та підтримки своїх веб-сайтів. Це може допомогти подолати технічні проблеми та забезпечити високоякісний веб-сайт.

3. Розробка спеціальної теми: якщо наявні попередньо створені теми не відповідають вимогам, користувачі можуть інвестувати в розробку спеціальної теми, щоб створити унікальний і візуально привабливий дизайн веб-сайту.

Підсумовуючи Drupal пропонує гнучкість, масштабованість і потужні функції безпеки, що робить його потужним вибором для створення веб-сайтів. Однак його швидка крива навчання та технічні вимоги можуть бути обмеженнями, особливо для початківців або користувачів без технічних знань. Інвестуючи в навчання, співпрацю з розробниками та розробку індивідуальних тем, користувачі можуть подолати ці обмеження та скористатися перевагами Drupal для потреб розробки веб-сайтів[4].

1.1.3 Shopify

Shopify — це популярна платформа електронної комерції, яка дозволяє окремим особам запускати свій онлайн-бізнес. Це універсальна комерційна платформа, яка пропонує різні інструменти для створення та ведення онлайн-бізнесу. Платформа надає користувачам онлайн-магазин, обробку платежів та інтеграцію доставки, серед інших функцій. Shopify широко відомий своєю простотою використання зі зручним інтерфейсом на рис. 1.2, що робить його чудовим варіантом для власників малого бізнесу, які хочуть створити свій онлайн-магазин[5].

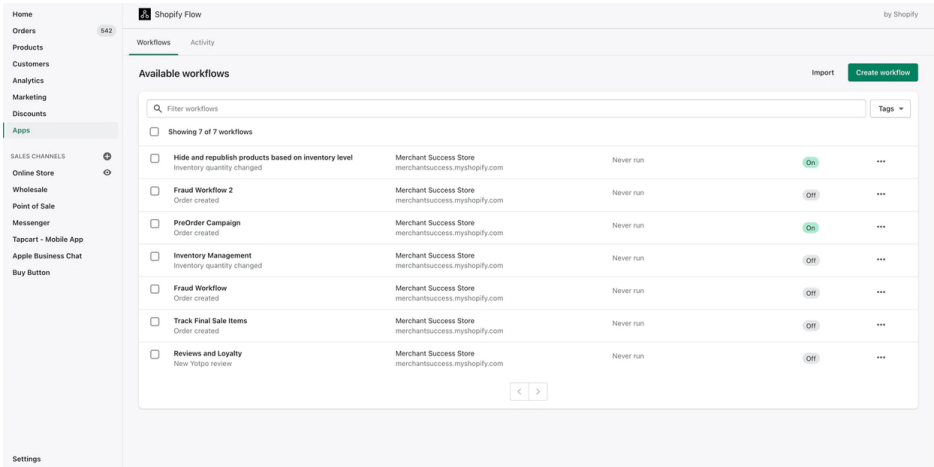


Рис. 1.2. Інтерфейс застосунку Shopify

Платформа також пропонує різноманітні функції та функції, які виділяють її серед конкурентів, що робить її привабливою для власників онлайн-бізнесу.

Однією з основних переваг використання Shopify є простота використання[5]. Платформа є зручною та інтуїтивно зрозумілою, що дозволяє людям із незначними технічними знаннями або без них легко створювати та керувати своїми онлайн-магазинами. Крім того, Shopify пропонує широкий вибір шаблонів і тем, які можна налаштувати відповідно до конкретних потреб бізнесу. Платформа також надає різні плагіни та інтеграції, які можна використовувати для покращення функціональності онлайн-магазину. Ще одна помітна перевага Shopify полягає в тому, що він пропонує

чудову підтримку клієнтів із спеціальною командою, яка працює цілодобово й без вихідних, щоб допомогти користувачам із будь-якими проблемами чи запитаннями. Використання Shopify має кілька переваг для власників онлайн-бізнесу. Платформа пропонує різні інструменти та функції, які допомагають бізнесу розвиватися та масштабуватися, включаючи інструменти маркетингу та пошукової оптимізації, аналітику та інтеграцію в соціальні мережі. Shopify також забезпечує безпечну обробку платежів і інтеграцію доставки, що полегшує для компаній керування своїми транзакціями та процесами доставки. Ще одна значна перевага використання Shopify полягає в тому, що він пропонує мобільний додаток, який дозволяє власникам бізнесу керувати своїми онлайн-магазинами на ходу[5]. Загалом Shopify є чудовим вибором для людей, які хочуть розпочати онлайн-бізнес або вивести свій існуючий бізнес на новий рівень.

Щоб почати використовувати Shopify для створення онлайн-магазину, першим кроком є створення облікового запису Shopify. Це передбачає створення облікового запису, вибір тарифного плану та введення основної інформації про магазин. Після налаштування облікового запису користувачі отримують доступ до інформаційної панелі Shopify, де вони можуть керувати всіма аспектами свого магазину. Це включає в себе налаштування дизайну магазину, налаштування варіантів оплати та керування продуктами та замовленнями. Важливо зазначити, що перед переходом на Shopify користувачам потрібно перенести такий вміст, як продукти, облікові записи клієнтів і дані, зі своєї попередньої платформи[5].

Розробка та налаштування інтернет-магазину є наступним кроком у використанні Shopify. Користувачі можуть вибирати з широкого спектру шаблонів веб-сайтів і налаштовувати їх відповідно до свого бренду та стилю. Shopify також пропонує конструктор блокових сторінок, що дозволяє користувачам створювати прості магазини без навичок програмування. Крім того, Shopify пропонує рішення для точки продажу (POS), яке дозволяє користувачам легко налаштувати свій магазин і синхронізувати офлайн-продажі з системою Shopify. Щоб оптимізувати службу підтримки, Shopify також пропонує вхід в облікові записи користувачів за допомогою програми одним клацанням миші[6].

Управління продуктами та платежами є важливим аспектом використання Shopify для створення інтернет-магазину. Користувачі можуть додавати продукти та керувати ними, зокрема налаштовувати варіанти продуктів, зображення й описи. Shopify також пропонує різні варіанти оплати, включаючи кредитні картки, PayPal та інші платіжні шлюзи. Щоб збільшити продажі та охопити ширшу аудиторію, користувачі також можуть налаштувати Google Shopping у своєму магазині Shopify. За допомогою Shopify користувачі можуть легко керувати своїм онлайн-магазином і розвивати свій бізнес на кількох платформах[6].

Переваги та обмеження

Переваги:

1. Простота використання: Shopify надає зручний інтерфейс, що дозволяє навіть тим, хто не має технічних знань, легко налаштувати онлайн-магазин і керувати ним.

2. Параметри налаштування: Shopify пропонує широкий спектр настроюваних тем і шаблонів, що дозволяє компаніям створювати унікальний і візуально привабливий онлайн-магазин.

3. Інтеграція додатків: Shopify має величезний магазин додатків із численними доступними інтеграціями, що дозволяє компаніям покращувати функціональність свого магазину та додавати такі функції, як маркетинг електронною поштою, підтримка клієнтів та інтеграція соціальних мереж.

4. Зручність для мобільних пристроїв: Shopify автоматично оптимізує онлайн-магазини для мобільних пристроїв, забезпечуючи безпроблемний досвід покупок для клієнтів на смартфонах і планшетах.

5. Безпечний і надійний: Shopify піклується про безпеку, хостинг і обслуговування, гарантуючи, що онлайн-магазини захищені та мають мінімальний час простою.

Недоліки:

1. Вартість. Незважаючи на те, що Shopify пропонує різні тарифні плани, деякі компанії можуть вважати щомісячні комісії та комісії за транзакції, пов’язані з використанням платформи, значними витратами.

2. Обмежений контроль: Shopify — це хостинг-платформа, що означає, що підприємства обмежені щодо доступу до сервера та можливостей налаштування порівняно з рішеннями, що розміщені самостійно.

3. Крива навчання. Незважаючи на те, що Shopify є зручною для користувачів, для тих, хто не знайомий із платформою чи електронною комерцією загалом, все ще може бути крива навчання.

4. Портативність даних: перенесення онлайн-магазину з Shopify на іншу платформу може бути складним через запатентований характер програмного забезпечення Shopify.

Вирішення недоліків:

1. Економічні варіанти. Підприємства можуть досліджувати різні тарифні плани та враховувати комісії за транзакції, пов’язані з кожним планом, щоб знайти найбільш прийнятний варіант для свого бюджету.

2. Налаштування за допомогою тем і додатків. Інтеграція тем і додатків Shopify надає широкий спектр варіантів налаштування, що дозволяє компаніям адаптувати свій онлайн-магазин до своїх конкретних потреб.

3. Навчальні ресурси: Shopify пропонує обширну документацію, навчальні посібники та підтримку клієнтів, щоб допомогти користувачам орієнтуватися на платформі та подолати будь-які навчальні проблеми.

4. Резервне копіювання даних і міграція: регулярне резервне копіювання даних і використання інструментів міграції сторонніх розробників може полегшити процес переміщення онлайн-магазину з Shopify на іншу платформу, якщо це необхідно.

Підсумовуючи Shopify пропонує численні переваги, зокрема простоту використання, параметри налаштування, інтеграцію додатків, зручність

використання мобільних пристроїв і надійну безпеку. Однак компаніям слід враховувати вартість, обмежений контроль, криву навчання та потенційні проблеми, пов’язані з переносимістю даних. Використовуючи економічно ефективні варіанти, використовуючи функції налаштування, доступ до навчальних ресурсів і забезпечуючи можливості резервного копіювання та міграції даних, компанії можуть пом’якшити обмеження та максимально використати переваги, які пропонує Shopify. Drupal — це система керування вмістом (CMS), яка широко використовується підприємствами, організаціями та окремими особами для створення та керування веб-сайтами. Як і Shopify, Drupal пропонує ряд переваг, включаючи налаштування, навчальні ресурси, а також параметри резервного копіювання та міграції даних. Однією з ключових переваг Drupal є його гнучкість і можливості налаштування. Маючи понад 44 000 доступних модулів, користувачі можуть адаптувати свій сайт Drupal відповідно до своїх потреб[7].

1.1.4 Next.js + Medusa

Next.js — це сучасний фреймворк на основі React, який допомагає розробникам створювати швидкі веб-сайти та додатки. Він пропонує такі функції, як серверний рендеринг (SSR), статичну генерацію сторінок (SSG) та оптимізацію для SEO, роблячи його ідеальним вибором для створення як статичних, так і динамічних веб-проектів.

Medusa Admin — це механізм безголової комерції з відкритим кодом, який дозволяє розробникам створювати гнучкі рішення для електронної комерції. Він надає повний набір функцій для управління продуктами, замовленнями та клієнтами які показані на рис. 1.3.

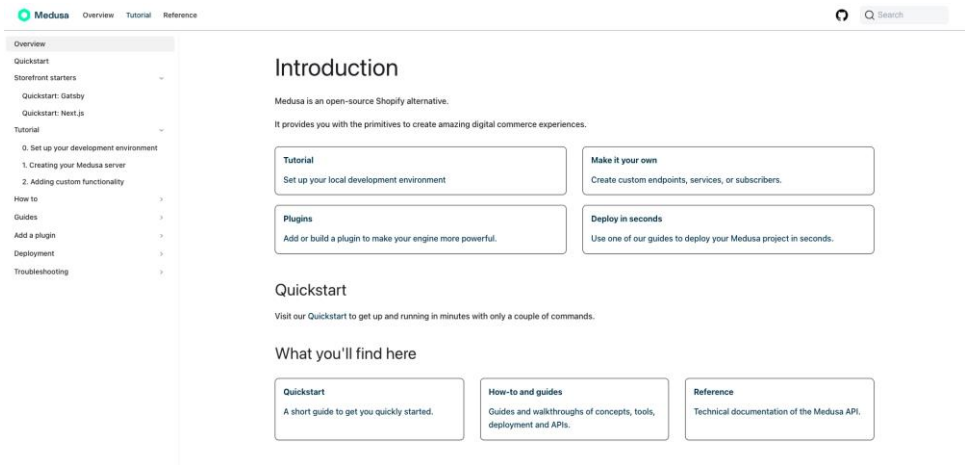


Рис. 1.3. Інтерфейс застосунку Medusa

Хоча Medusa Admin раніше використовувався з Gatsby, він також може бути ефективно інтегрований з Next.js, що дозволяє створювати більш динамічні та інтерактивні веб-додатки[7].

Для створення веб-сайту електронної комерції з використанням Next.js та Medusa Admin, розробники повинні спочатку налаштувати середовище Next.js, встановивши необхідні залежності та створивши базову структуру проекту. Після

цього вони можуть інтегрувати Medusa Admin, використовуючи API Medusa для управління даними про продукти, замовлення та клієнтів[7].

Переваги та обмеження:

Переваги використання Next.js включають покращену продуктивність веб-сайту, оптимізацію для SEO та гнучкість у виборі між статичною генерацією та серверним рендерингом. Однак, враховуючи його складність, може знадобитися час для освоєння фреймворку, особливо для новачків.З іншого боку, Medusa Admin додає гнучкість у управлінні електронною комерцією, але вимагає розуміння концепцій безголової комерції та роботи з API.

Висновок

Комбінація Next.js та Medusa Admin є потужним рішенням для створення сучасних, швидких та гнучких веб-сайтів електронної комерції. Це поєднання дозволяє розробникам створювати високопродуктивні веб-додатки, які забезпечують виняткову взаємодію з користувачем, в той же час надаючи гнучкість у управлінні контентом та продуктами[7].

1.1.5 Contentful + Gatsby.js

Contentful — це безголова CMS, яка надає гнучку та масштабовану платформу для керування вмістом на різних каналах і пристроях[7]. Він пропонує ряд функцій, включаючи моделювання вмісту, редагування форматованого тексту, керування ресурсами та локалізацію, які можна побачити на рис. 1.4.

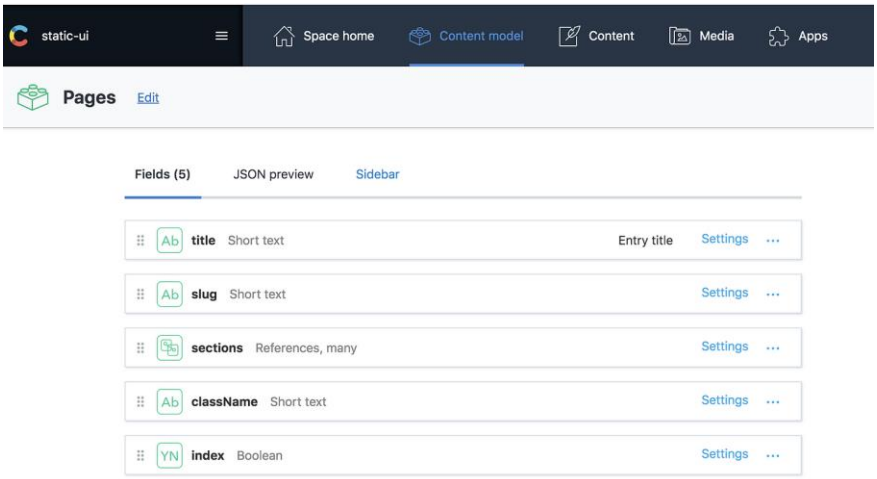


Рис. 1.4. Інтерфейс застосунку Contentful

За допомогою Contentful розробники можуть створювати структури контенту, які відповідають їхнім конкретним потребам, а редактори контенту можуть легко керувати вмістом і публікувати його без технічних знань. Це робить його ідеальним вибором для створення сучасних веб-сайтів і програм, які потребують динамічного та персоналізованого вмісту.

Gatsby — це генератор статичних сайтів, який дозволяє розробникам створювати швидкі динамічні веб-сайти за допомогою сучасних веб-технологій[23]. Він пропонує ряд функцій, включаючи інтуїтивно зрозуміле середовище розробки, автоматичне поділ коду та оптимізоване завантаження зображень. За допомогою Gatsby розробники можуть створювати високопродуктивні веб-сайти, які оптимізовані для пошукових систем і забезпечують чудову взаємодію з користувачем. Крім того, використання Gatsby React і GraphQL дозволяє легко створювати динамічні та інтерактивні інтерфейси користувача. При спільному використанні Contentful і Gatsby забезпечують потужне та ефективне рішення для створення сучасних веб-сайтів і програм. Використовуючи плагін gatsby-source-contentful Gatsby, розробники можуть легко отримувати вміст із Contentful і використовувати його для створення динамічних сторінок і компонентів. Це дозволяє бездоганно інтегрувати керування вмістом і розробку веб-сайту, а зміни вмісту автоматично відображаються на створеному веб-сайті. Крім того, поєднання Gatsby і

Contentful забезпечує покращену оптимізацію продуктивності та швидший час створення. Використовуючи сильні сторони обох платформ, розробники можуть створювати високопродуктивні веб-сайти, якими легко керувати та масштабувати[8].

Поєднання Gatsby і Contentful забезпечує чудове рішення для створення швидких і масштабованих веб-сайтів. Щоб почати, потрібно налаштувати Contentful і Gatsby і з'єднати їх один з одним. Зробивши це, можна почати створювати вміст у Contentful і з легкістю ним керувати. Інтеграція Gatsby з Contentful забезпечує підвищення оптимізації продуктивності, пришвидшення часу створення та значно покращує загальну взаємодію з користувачем. Дотримуючись кроків, наданих обома платформами, можна легко налаштувати веб-сайт, яким буде швидко та легко керувати[8].

Однією з ключових переваг використання Contentful with Gatsby є можливість легко створювати вміст і керувати ним. За допомогою Contentful можна створювати різні типи вмісту, зокрема текст, зображення, відео тощо. Також можете легко керувати своїм вмістом, вносячи оновлення та зміни за потреби. Інтеграція з Gatsby дозволяє динамічно отримувати вміст із Contentful і використовувати його для створення свого веб-сайту. Це означає, що можна легко оновити свій веб-сайт новим вмістом без необхідності вручну оновлювати код веб-сайту[9].

Інтеграція Contentful із Gatsby є відносно простою, і є багато доступних ресурсів, які допоможуть почати. Можете використовувати хук Gatsby useStaticQuery, щоб запитувати різні типи даних Contentful і використовувати їх на своєму веб-сайті. Крім того, плагін gatsby-source-contentful пропонує повну підтримку функцій локалізації Contentful. Це означає, що можна створювати веб-сайти різними мовами та легко керувати вмістом для кожної мови. Дотримуючись наявних ресурсів і використовуючи доступні інструменти, можна створити швидкий, масштабований і простий в управлінні веб-сайт за допомогою Contentful і Gatsby[9].

Переваги та обмеження

Переваги:

1. Підвищення ефективності: інтеграція Contentful + Gatsby дозволяє оптимізувати процес керування вмістом. Завдяки Contentful як безголовій CMS творці контенту можуть легко керувати вмістом і оновлювати його без участі розробників. Це економить час і ресурси, що сприяє підвищенню ефективності.

2. Покращена масштабованість: Contentful + Gatsby дозволяє створювати високомасштабовані веб-сайти або програми. Генерація статичних сайтів Gatsby та гнучка модель контенту Contentful дозволяють легко масштабувати вміст і трафік. Це забезпечує безперебійну роботу користувача навіть за підвищеного попиту.

Недоліки:

1. Крива навчання: Використання Contentful + Gatsby може вимагати деякого навчання, особливо для тих, хто вперше знайомиться з безголовою CMS або створенням статичного сайту. Розробникам може знадобитися ознайомитися з залученими інструментами та фреймворками, що може сповільнити початковий процес розробки.

2. Технічна складність: інтеграція Contentful із Gatsby вимагає технічних знань. Розробникам потрібно правильно налаштувати інтеграцію, налаштувати веб-хуки та керувати синхронізацією між двома системами. Ця складність може бути складною для менш досвідчених розробників або команд з обмеженими технічними ресурсами.

Вирішення недоліків:

1. Освіта та навчання. Забезпечення відповідного навчання та ресурсів може допомогти розробникам подолати криву навчання, пов’язану з Contentful + Gatsby. Пропонування навчальних посібників, документації та підтримки може допомогти пришвидшити процес адаптації та зменшити початкові труднощі.

2. Співпраця та підтримка: заохочення співпраці між розробниками та творцями вмісту може допомогти вирішити технічні складнощі. Регулярне спілкування та підтримка досвідчених розробників можуть допомогти у вирішенні проблем та оптимізації інтеграції.

Підсумовучи Інтеграція Contentful + Gatsby пропонує кілька переваг, зокрема підвищену ефективність і покращену масштабованість. Однак він також має обмеження, такі як крива навчання та технічна складність. Завдяки правильному навчанню та підтримці ці обмеження можна подолати, що робить Contentful + Gatsby потужною комбінацією для створення ефективних і масштабованих веб-сайтів або програм[9].

1.2 Підсумок між оглянутими технологіями

Коли справа доходить до вибору найкращої технології для створення веб-сайту електронної комерції, доступно кілька варіантів, зокрема WordPress, Drupal, Shopify, Next.js + Medusa та Contentful + Gatsby.js. Кожна з цих технологій має свої плюси та мінуси, і порівняльна таблиця може допомогти зрозуміти їхні особливості та функції. Наприклад, Next.js + Medusa пропонує найшвидший інтерфейс для веб-сайтів електронної комерції, тоді як Drupal надає вихідний плагін для завантаження даних у Gatsby. З іншого боку, Shopify — це комплексна платформа електронної комерції, яка пропонує ряд функцій, включаючи хостинг, обробку платежів і безпеку[10].

Таблиця 1.1. - Сильні сторони та недоліки технологій

Технологія	Плюси	Мінуси	Переваги
WordPress	Легкість використання, велика кількість плагінів, велика спільнота	Може бути повільним, потребує регулярних оновлень, можливі проблеми з безпекою	Ідеальний для блогів та невеликих сайтів
Drupal	Гнучкість, масштабованість, висока безпека	Високий поріг входу, потребує спеціалізованих знань	Добре підходить для великих і складних проектів
Shopify	Легкість використання, інтеграція з платіжними системами, готові шаблони	Місячна плата, обмеження в налаштуванні	Ідеальний для електронної комерції
Next.js + Medusa	Висока швидкість завантаження, гнучкість, сучасні технології	Потребує певних технічних знань	Найкращий вибір для сучасних веб-сайтів та інтернет-магазинів
Gatsby.js + Contentful	Гнучкість у управлінні контентом, висока швидкість завантаження	Потребує певних технічних знань, вартість за користування Contentful	Ідеальний для сайтів з частими оновленнями контенту

Щоб підкреслити переваги використання Next.js + Medusa над іншими технологіями. Наприклад, Next.js + Medusa пропонує готовий до виробництва стартовий пакет для Medusa, який дозволяє розробникам створювати неймовірні можливості цифрової комерції. Contentful + Gatsby.js, з іншого боку, забезпечує підвищення оптимізації продуктивності та пришвидшення часу створення. Однак, інтегрувавши Contentful до Medusa, розробники можуть отримати вигоду від потужних функцій у своєму магазині електронної комерції, включаючи детальну інформацію про CMS продукту та просте у використанні керування вмістом [11].

Загалом Next.js + Medusa є найкращим вибором для створення веб-сайту електронної комерції завдяки численним перевагам. Next.js — це безкоштовний фреймворк із відкритим вихідним кодом на основі React, який спрощує використання та налаштування . Крім того, Next.js + Medusa пропонує тему Next для вмісту Drupal , що полегшує інтеграцію даних Drupal у Next. Крім того, підключення Next.js + Medusa до нового або існуючого сайту Drupal займає всього кілька кроків [11], що робить його зручним вибором для розробників. Нарешті, підключивши програму Next до сервера Medusa, розробники можуть легко створити інтерфейс електронної комерції Next.

1.2.1 Переваги Next.js + Medusa перед іншими технологіями

Однією з головних переваг використання Next.js + Medusa перед іншими технологіями, такими як WordPress, Drupal, Shopify і Contentful + Gatsby.js, є його висока продуктивність і швидкість. Next.js використовує генератор статичних сайтів та динамічних, який попередньо створює сторінки або динамічно їх завантажує, що забезпечує блискавичне завантаження та покращує взаємодію з користувачем. Крім того, Medusa є високопродуктивною платформою електронної комерції, оптимізованою для швидкості та масштабованості [12]. У поєднанні Next.js + Medusa створюють потужну та ефективну платформу, яка може легко обробляти великі обсяги трафіку.

Ще однією перевагою Next.js + Medusa є його масштабованість і гнучкість[37]. Next.js — це дуже гнучка платформа, яку можна легко налаштувати відповідно до конкретних потреб і вимог. Medusa, з іншого боку, створена для високої масштабованості, що дозволяє підприємствам рости та розширюватися, не турбуючись про технічні обмеження[12]. Разом Next.js + Medusa забезпечують надійну та гнучку платформу, яка може адаптуватися до мінливих потреб компаній і клієнтів.

Next.js + Medusa також пропонують повну інтеграцію з кількома платформами, що полегшує керування та оптимізацію різних аспектів бізнесу. Плагіни next-source-drupal і next-source-shopify забезпечують плавну інтеграцію з Drupal і Shopify відповідно. Крім того, Contentful можна інтегрувати з Medusa, щоб оптимізувати керування вмістом і покращити загальну взаємодію з користувачем. Ця комплексна інтеграція дозволяє підприємствам керувати всіма аспектами своєї діяльності з єдиної платформи, підвищуючи ефективність і продуктивність. Підсумовуючи, Next.js + Medusa пропонує численні переваги перед іншими технологіями, включаючи високу продуктивність і швидкість, масштабованість і гнучкість, а також повну інтеграцію з кількома платформами. Ці переваги роблять Next.js + Medusa найкращим вибором

для компаній, яким потрібна ефективна, настроювана та комплексна платформа для керування своїми операціями[12].

Переваги та обмеження

Переваги:

1. Покращена продуктивність і швидкість завантаження: Next.js + Medusa забезпечує статичний та динамічні підходи до створення сайтів, що призводить до швидшого часу завантаження та покращення загальної продуктивності. Це може сприяти покращенню взаємодії з користувачем і вищим коефіцієнтам конверсії.

2. Покращена безпека: Next.js + Medusa дотримується найкращих практик безпеки, таких як автоматичні оновлення безпеки та захист від поширених уразливостей. Це гарантує, що веб-сайт або інтернет-магазин буде менш схильним до атак або витоку даних.

Недоліки:

1. Крива навчання: Next.js і Medusa можуть мати крутішу криву навчання порівняно з іншими технологіями, такими як WordPress або Shopify. Розробникам може знадобитися деякий час, щоб зрозуміти концепції та найкращі практики, пов’язані з цими технологіями.

2. Обмежені можливості налаштування: хоча Next.js + Medusa пропонує широкий спектр параметрів налаштування, він може бути не таким гнучким, як інші платформи, такі як WordPress або Drupal. Це може обмежити певні розширені функції або унікальний дизайн.

Вирішення недоліків:

1. Навчальні ресурси: для Next.js і Medusa доступні численні онлайн-посібники, документація та спільноти. Розробники можуть використовувати ці ресурси для навчання та подолання початкової кривої навчання.

2. Екосистема плагінів: хоча Next.js + Medusa може мати обмежені можливості налаштування, існує розгалужена екосистема плагінів, яка може надавати додаткові

функції. Розробники можуть використовувати ці плагіни для вдосконалення своїх веб-сайтів або онлайн-магазинів.

Підсумовуючи Next.js + Medusa пропонує покращену продуктивність і підвищену безпеку, що є суттєвою перевагою для будь-якого веб-сайту чи інтернет-магазину. Однак вона може мати крутішу криву навчання та обмежені можливості налаштування порівняно з іншими технологіями. Щоб подолати ці обмеження, розробники можуть використовувати навчальні ресурси та розгалужену екосистему плагінів. Зрештою, вибір між WordPress, Drupal, Shopify, Next.js + Medusa або Contentful + Gatsby.js залежить від конкретних потреб і вимог проекту[12].

1.3 Огляд екосистеми Next.js та Medusa

1.3.1 Що таке Next.js і як він працює з екосистемою Medusa

Next.js — це платформа веб-розробки з відкритим вихідним кодом, яка спрощує процес створення веб-додатків, зокрема веб-сайтів електронної комерції. Це популярний фреймворк React для візуалізації на стороні сервера, який працює в екосистемі Medusa, щоб пришвидшити процес створення повноцінного веб-сайту електронної комерції. Next.js можна використовувати для створення повноцінного веб-сайту електронної комерції з Medusa, надаючи такі переваги, як покращена продуктивність і SEO. Будучи серверним набором інструментів для електронної комерції, Medusa може працювати з будь-яким стеком інтерфейсних технологій, включаючи Next.js [12].

Насправді Next.js — це технологія, яка використовується Medusa для живлення різних модулів. Ця технологія допомагає розробникам швидше додавати важливі функції до веб-сайту для покупок, роблячи створення веб-сайтів або цифрових платформ легшим і швидшим [12-13]. Вітрини підключаються до серверної частини Medusa за допомогою API Store REST. Для роботи вітрини магазину Next.js потрібен бекенд Medusa, який використовується для надання комерційних функцій клієнтам. Бекенд Medusa — це механізм безголової комерції, тоді як Next.js — це вітрина, яка працює з екосистемою Medusa. Крім того, Next.js забезпечує повну екосистему для створення додатків легше та з меншими зусиллями, одночасно вирішуючи фундаментальні проблеми, які зазвичай виникають у веб-розробці. Він пропонує вбудовані рішення для різноманітних поширених проблем, з якими стикаються під час веб-розробки, з деякими фрагментами коду та угодами, які вже доступні для використання. Таким чином, Next.js можна використовувати з екосистемою Medusa для створення безперебійної онлайн-платформи для роздрібної торгівлі [12-13].

1.3.2 Інструменти оптимізації веб-сайту за допомогою Next.js та екосистеми Medusa

Оптимізація функціональності веб-сайту має важливе значення для створення бездоганної взаємодії з користувачем. Для сторінок з великою кількістю інтерактивних елементів рекомендований гібридний підхід для оптимізації. Next.js, популярний фреймворк React, пропонує кілька інструментів для оптимізації функціональності веб-сайту. За замовчуванням Next.js використовує рендеринг на стороні сервера (SSR) для статичних сторінок, який відображає їх майже миттєво . Крім того, Next.js Commerce — це набір модулів, які допомагають швидко запускати продуктивні проекти електронної комерції, що може допомогти оптимізувати функціональність веб-сайту. Next.js також забезпечує вбудовану інтеграцію CSS і SaaS, яка може допомогти оптимізувати функціональність веб-сайту. Нарешті, Next.js підтримує TypeScript з автоматичною конфігурацією та компіляцією, що може додатково допомогти в оптимізації функціональності веб-сайту. Іншим зручним інструментом для прискорення завантаження сторінки є Next.js, який може генерувати сторінки HTML на сервері, що ще більше покращує швидкість завантаження [14]. За допомогою цих інструментів, наданих Next.js та екосистемою Medusa, функціональність веб-сайту можна оптимізувати, щоб створити бездоганний досвід для користувачів.

1.3.3 Плагіни Next.js і екосистеми Medusa для оптимізації веб-сайту

Щоб оптимізувати функціональність веб-сайту для компаній електронної комерції, Medusa пропонує початковий шаблон Next.js, який поєднує модулі Medusa для комерційної системи з найновішими функціями Next.js 14 для ефективної вітрини. Архітектура Medusa складається з основного рівня API, плагінів, адаптерів зберігання та інтерфейсної інтеграції. За допомогою цього бекенда можно легко інтегрувати Next.js у свій веб-сайт або програму електронної комерції [14]. Next.js — це фреймворк React, який розширює свої можливості додатковими функціями, включаючи рендеринг на стороні сервера та генерацію статичного веб-сайту.

Використовуючи маршрути API Next.js і модуль продукту Medusa, може створювати логіку персоналізації в реальному часі, яка демонструється в демонстрації [14]. Крім того, веб-сайти та програми, створені з використанням фреймворку Next.js, добре масштабуються, мають чудові можливості інтеграції та реагують на потреби користувачів. Підприємства електронної комерції, які бажають ще більше розширити свою функціональність, можуть додавати огляди продуктів на свій сервер Medusa Next.js storefront і Gatsby Admin. Загалом, використовуючи екосистему Medusa та фреймворк Next.js, компанії електронної комерції можуть оптимізувати функціональність своїх веб-сайтів, щоб краще обслуговувати своїх клієнтів[14].

1.3.4 Бібліотеки Next.js і екосистеми Medusa для оптимізації веб-сайту

Під час оптимізації функціональності веб-сайту за допомогою Next.js і екосистеми Medusa є кілька доступних бібліотек для використання. Одним із таких ресурсів є початковий шаблон Medusa Next.js, який поєднує модулі Medusa для комерційного серверу з найновішими функціями Next.js 14 для ефективного використання вітрини. Архітектура Medusa складається з базового рівня API, плагінів, адаптерів зберігання та інтерфейсної інтеграції, що забезпечує простий у користуванні серверний механізм, який забезпечує бездоганну інтеграцію з іншими ресурсами. Крім того, Next.js — це фреймворк React, який розширює свої можливості додатковими функціями, такими як рендеринг на стороні сервера та генерація статичних веб-сайтів, що робить його ідеальним для створення адаптивних, масштабованих і добре інтегрованих веб-сайтів і програм. Щоб покращити бізнес електронної комерції, можна додавати огляди продукту до свого сервера Medusa Next.js storefront і Gatsby Admin, про які можна дізнатися через різні онлайн-ресурси. Можна також навчитися будувати логіку персоналізації в реальному часі за допомогою маршрутів API Next.js і модуля продукту Medusa[15]. Використовуючи ці доступні бібліотеки, веб-розробники можуть оптимізувати функціональність веб-сайту та створити кращий досвід користувача для своїх клієнтів[15].

Висновки до розділу

У цьому розділі було проведено глибокий теоретичний аналіз сучасних технологій для розробки веб-додатків електронної комерції. Основний акцент був зроблений на вивченні функціональних та технічних аспектів різних технологій, зокрема Medusa.js та інших пов'язаних інструментів та платформ.

Важливими критеріями вибору технологій стали їх масштабованість, гнучкість, безпека та здатність до інтеграції з іншими інструментами та сервісами. Було визначено, що Medusa.js відповідає цим критеріям, пропонуючи сучасний підхід до створення веб-додатків, які легко масштабуються та адаптуються до змінних вимог ринку.

Також у розділі було розглянуто важливість ефективної взаємодії між бекендом та фронтендом, з акцентом на важливість створення зручного та інтуїтивно зрозумілого користувацького інтерфейсу. Висвітлено роль сучасних фронтенд-фреймворків, зокрема Next.js, у створенні динамічних та відповідальних веб-інтерфейсів.

Крім того, було підкреслено значення контейнеризації та використання Docker та Docker Compose як засобів забезпечення стандартизації та портативності розробки. Це сприяє легкості розгортання та сумісності між різними середовищами.

У підсумку, перший розділ закладає фундамент для подальшого вибору та використання технологій у проекті, демонструючи важливість комплексного підходу до аналізу та вибору інструментів, що забезпечують оптимальну продуктивність, безпеку та гнучкість[15-16].

РОЗДІЛ 2. ПРОЕКТУВАННЯ АРХІТЕКТУРИ ВЕБ-САЙТУ

2.1 Розробка архітектури веб-сайту за допомогою Nextjs і Medusa Admin

Next.js - це сучасний фреймворк для створення веб-сайтів з використанням React.js, який дозволяє створювати швидкі, оптимізовані сайти з гарною продуктивністю та SEO показниками. Medusa Admin - це відкрита панель управління адміністратора на основі React.js, яка дозволяє створювати налаштовані адмін-панелі для керування сайтом, продуктами, користувачами та іншими компонентами. Комбінація цих двох інструментів дає можливість розробити сучасний, швидкий та зручний веб-сайт з максимальною продуктивністю. Розуміння мети сайту та цільової аудиторії є критичним кроком у процесі розробки архітектури веб-сайту. Необхідно застосувати основні цілі та функції, які очікуєте від свого сайту, а також врахувати потреби та інтереси цільової аудиторії. Це допоможе створити ефективний та зручний сайт, який відповідатиме потребам користувачів і допоможе підвищити конверсію та продажі. Особливо це актуально для сайтів з адмін-панеллю, продуктами та платіжною системою, де коректна робота та налаштування всіх компонентів має вирішальне значення для успіху сайту[16].

Планування структури та верстки веб-сайту включає в себе розробку логічної структури сайту, визначення основних розділів, сторінок та елементів, а також створення чіткої та зрозумілої верстки як покащано на рис. 2.1.

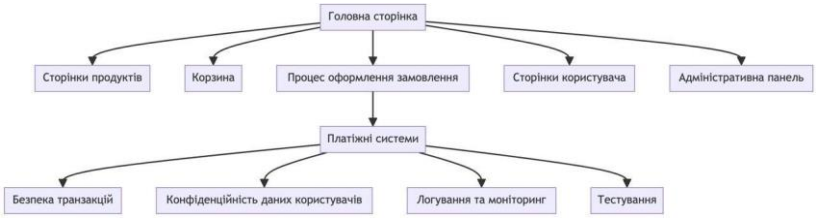


Рис. 2.1. Діаграма схеми сайту

Процес успішно виконує важливу роль у забезпеченні цієї зручності користувачів та покращеної продуктивності сайту. Особливу увагу слід приділити адмін-панелі, оскільки вона є основним інструментом для управління сайтом,

продуктами та платіжною системою. Важливо забезпечити належний рівень безпеки, щоб захистити дані користувачів і транзакцій,а також розробити зручний та інтуїтивно зрозумілий інтер'єр[16].

2.2 Впровадження продуктів та системи оплати з адмінпанеллю

Налаштування бази даних товарів і адмінпанелі є одним із ключових етапів створення веб-сайту з адмін-панеллю, продуктами та системою оплати. Перш ніж перейти до розробки функціональної платіжної системи, необхідно створити структуру бази даних, яка буде забезпечувати зберігання інформації про товари, їх характеристики, ціни та наявність. Також необхідно розробити адмін-панель для керування цією інформацією, що дозволяє легко додавати, редагувати або видаляти товари та їх характеристики.

- Вибір бази даних, що відповідає потребам проекту

- Розробка схеми бази даних для зберігання товарів, характеристики та ціни

- Створення адмін-панелі для керування інформацією про товари.

Інтеграція платіжної системи та забезпечення безпеки транзакцій є комерційним аспектом при створенні веб-сайту з продуктами та оплатою. При виборі платіжного провайдера слід забезпечити його надійність, швидкість обробки транзакцій та наявність інших опцій оплати для користувачів, виключно того, важливо забезпечити безпеку транзакцій та дотримання вимог законодавства щодо обробки персональних даних[16].

Тестування та оптимізація веб-сайту для продуктивності та взаємодії з користувачем є необхідними етапами перед запуском веб-сайту. Протестування повинно включати перевірку функціональності сайту, коректності роботи бази даних і платіжної системи, а також оптимізацію швидкості завантаження сторінок і відгуків на дії користувача. Усі ці заходи сприяють підвищенню задоволеності користувачів

та конверсії веб-сайту. Перевірка функціонального сайту, бази даних та платіжної системи. Оптимізація швидкості.

Переваги та обмеження

Переваги:

1. Висока продуктивність: Next.js — це генератор статичних та динамічних сайтів, що означає, що він може генерувати статичні файли HTML/CSS/JS під час створення, так й підтягувати всю інформацію динамчіно, що забезпечує високопродуктивні сайти, які завантажуються швидше.

2. Зручність для SEO: статичні сайти часто є більш зручними для SEO, оскільки вони забезпечують кращу індексацію пошуковими системами, що може покращити видимість веб-сайту.

3. Medusa Admin: використання Medusa Admin для панелі адміністратора забезпечує зручний інтерфейс для керування продуктами, замовленнями та клієнтами. Він також підтримує різноманітних постачальників платежів, що робить його комплексним рішенням для веб-сайтів електронної комерції.

4. Масштабованість: через статично-динамічну природу Next.js веб-сайти можуть обробляти великий трафік, не вимагаючи додаткових ресурсів сервера.

Обмеження:

1. Крива навчання: Next.js і Medusa Admin вимагають хорошого розуміння JavaScript і React, що може стати перешкодою для новачків.

2. Початковий час створення: спочатку створення сайтів Next.js може зайняти більше часу, особливо для великих веб-сайтів.

Вирішення недоліків:

1. Для початківців є численні ресурси, навчальні посібники та підтримка спільноти, щоб вивчити Next.js і Medusa Admin.

2. Щоб скоротити час початкової збірки, подумайте про впровадження поступових збірок або використання служби збірки, яка підтримує розпаралелювання.

Підсумовучи дизайн архітектури веб-сайту з Next.js і Medusa Admin пропонує численні переваги, включаючи високу продуктивність, зручність для пошукових систем, повну панель адміністратора та масштабованість. Однак це створює обмеження, коли справа доходить до його кривої навчання та початкового часу створення. Ці обмеження можна подолати за допомогою відповідних рішень, що робить його надійним вибором для створення веб-сайтів електронної комерції[16-17].

Висновки до розділу

У цьому розділі магістерської дисертації детально розглянуто процес проектування архітектури веб-сайту, використовуючи фреймворк Next.js та адміністративну панель Medusa Admin. Аналіз показав, що інтеграція цих двох потужних інструментів сприяє створенню сучасного веб-сайту, який відрізняється високою продуктивністю, ефективною оптимізацією для пошукових систем та наявністю інтегрованої адміністративної панелі.

У роботі було підкреслено важливість врахування цілей сайту та потреб цільової аудиторії на етапі проектування. Також було ретельно розглянуто питання планування структури сайту, верстки, налаштування бази даних продуктів, інтеграції платіжних систем та забезпечення безпеки транзакцій.

Серед ключових переваг обраної архітектури особливо відзначено її високу продуктивність, зручність у пошуковій оптимізації, наявність інтегрованої адміністративної панелі Medusa та масштабованість. Водночас було відмічено, що крива навчання фреймворків та час, необхідний для початкової розробки, можуть створювати певні обмеження, однак ці недоліки можна ефективно подолати за допомогою відповідних заходів.

Висновок цього розділу підкреслює, що архітектура на базі Next.js та Medusa Admin є оптимальним вибором для створення сучасних, високопродуктивних веб-

сайтів електронної комерції з ефективними можливостями адміністрування. Ця комбінація технологій забезпечує зручність розробки та впровадження ключового функціоналу, необхідного для успішного ведення бізнесу в сучасному цифровому світі[16-18].

РОЗДІЛ 3. НАЛАШТУВАННЯ MEDUSA.JS У СЕРЕДОВИЩІ РОЗРОБКИ ТА ОГЛЯД

3.1 Встановлення Medusa за допомогою create-medusa-app | Medusa

Medusa - це набір інструментів для розробників для створення цифрових комерційних додатків. У своїй найпростішій формі Medusa є бекендом на Node.js з основним API, плагінами та модулями, встановленими через npm.

create-medusa-app - це команда, яка спрощує створення екосистеми Medusa. Вона встановлює бекенд Medusa та адміністративну панель, а також необхідні конфігурації для запуску бекенду[3].

Перш ніж встановити та використовувати Medusa, потрібно встановити наступні інструменти на комп'ютері:

- Node.js v16+
- Git
- PostgreSQL. Сервер PostgreSQL також повинен працювати під час процесу встановлення.

Крок 1: Запуск create-medusa-app

У терміналі запустіть наступну команду: **npx create-medusa-app@latest**

Додаткові опції команди:

- **--repo-url <url>**: URL-адреса репозиторію для створення проекту. За замовчуванням використовується https://github.com/medusajs/medusa-starter-default.
- **--seed**: Прапорець, який вказує, чи слід заповнювати базу даних демонстраційними даними. За замовчуванням заповнення вимкнено.
- **--no-boilerplate**: Прапорець, який видаляє всі файли, додані для покращення досвіду використання (файли в src/admin, src/api тощо). Це корисно, якщо потрібно створити чистий проект.

- **--no-browser**: Вимикає відкриття браузера в кінці створення проекту та показує лише повідомлення про успіх.
- **--skip-db**: Пропускає створення бази даних, запуск міграцій та заповнення, а також відкриття браузера. Корисно, якщо потрібно вказати URL бази даних пізніше в конфігураціях.
- **--db-url <url>**: Пропускає створення бази даних і встановлює URL бази даних на наданий URL. Видає помилку, якщо не вдається підключитися до бази даних. Все одно запускає міграції та відкриває адміністративну панель після створення проекту. Корисно, якщо вже є створена база даних, локально або віддалено.
- **--no-migrations**: Пропускає запуск міграцій, створення адміністратора та заповнення. Якщо використовується, очікується, що передається параметр **--db-url** з URL бази даних, яка має всі необхідні міграції. В іншому випадку можуть виникнути неочікувані помилки. Корисно лише у поєднанні з **--db-url**.
- **--directory-path <path>**: Дозволяє вказати шлях до батьківської директорії для створення директорії нового проекту.
- **--with-nextjs-starter**: Встановлює стартовий магазин Next.js у директорії **<PROJECT_NAME>-storefront**, де **<PROJECT_NAME>** - назва проекту, яка вводиться на першому питанні. Якщо директорія **<PROJECT_NAME>-storefront** вже існує, в кінці **<PROJECT_NAME>-storefront** додаються випадкові символи.

Приклад: Підключення до бази даних PostgreSQL Vercel,
Додайте до кінця URL-адреси підключення **?sslmode=require**. Наприклад:

npx create-medusa-app@latest --db-url "postgres://default:<password>@<host-region>.postgres.vercel-storage.com:5432/verceldb?sslmode=require"

Приклад: Підключення до бази даних Supabase.

Якщо потрібно підключитися до бази даних Supabase, необхідно використовувати опцію **--db-url** зі значенням, яке є URL-адресою підключення до бази даних Supabase. Наприклад:

npx create-medusa-app@latest --db-url postgresql://postrges
<password>@<host>.supabase.co:5432/postrges

Ця команда встановить Medusa, використовуючи базу даних Supabase, яку вказали, забезпечуючи підключення до неї через вказаний URL. Якщо база даних вже має необхідні міграції і не потрібно, щоб команда запускала міграції, можна передати опцію **--no-migrations[19]**.

Крок 2: Вкажіть назву проекту

Спочатку буде запропоновано ввести назву проекту, яка використовуватиметься для створення директорії з бекендом Medusa. Можна використовувати назву за замовчуванням **my-medusa-store** або ввести іншу назву проекту.

Крок 3: Введіть адміністративний емейл

Далі потрібно ввести адміністративний емейл для адміністратора. Цей адміністративний емейл буде використовуватись пізніше для входу в адміністративну панель. Можна використовувати емейл за замовчуванням admin@medusa-test.com або ввести будь-який інший емейл.

(Необов'язково) Крок 4: Встановлення стартового магазину Next.js

Якщо не використовувалась опція **--with-nextjs-starter**, запитають, чи потрібно встановити стартовий магазин Next.js разом з бекендом Medusa. Це встановить магазин у директорії **<PROJECT_NAME>-storefront**, де **<PROJECT_NAME>** - назва проекту, яку було вказано на кроці 2.

Якщо потрібно встановити магазин, наберіть **y** терміналі **--with-nextjs-starter**, та натисніть Enter. Якщо ні, можна використовувати значення за замовчуванням **N** та просто натиснути Enter.

Примітка: Завжди можна встановити магазин пізніше. Medusa - це безголовий бекенд, тому він працює без магазину за замовчуванням. Також можна підключити будь-який магазин до нього. Стартовий магазин Next.js - це хороший варіант для використання, але також можна побудувати свій власний магазин пізніше[19].

(Необов'язково) Крок 5: Вкажіть облікові дані PostgreSQL

Примітка: Це не застосовується, якщо було вкиористано опцію **--db-url**. У цьому випадку команда не вдасться, якщо вона не зможе підключитися до наданого URL-адреси підключення. За замовчуванням ця команда спробує використовувати облікові дані PostgreSQL за замовчуванням для підключення до сервера PostgreSQL. Якщо вони не працюють, потрібно буде ввести базу даних PostgreSQL та пароль. Якщо вони працюють, можна перейти до наступного кроку. Ці облікові дані будуть використовуватися для створення бази даних під час цього налаштування та налаштування бекенду Medusa для підключення до цієї бази даних[19].

Після виконання вищезазначених кроків розпочнеться налаштування проекту, яке включає:

- Створення директорії проекту. Назва директорії буде назвою проекту, якубуло введенно на кроці 1.
- Створення бази даних проекту, якщо не передані опції **--db-url** та **--skip-db**.
- Встановлення залежностей у директорії проекту.
- Збірка проекту.
- Запуск міграцій для міграції схеми Medusa у базу даних проекту, якщо не передані опції **--skip-db** або **--no-migrations**.
- Створення адміністратора, якщо не передані опції **--skip-db** або **--no-migrations**.
- Заповнення бази даних демонстраційними даними, якщо не передані опції **--skip-db** або **--no-migrations**.

Крок 6: Увійдіть в адміністративну панель

Як тільки проект буде готовий, запуститься бекенд Medusa, а адміністративна панель автоматично відкриється у браузері за замовчуванням. Після чого буде запропоновано ввести пароль для адміністративної електронної пошти, яку було вказано раніше, а також додаткову інформацію про обліковий запис.
Після входу в систему можна розпочати використання Medusa.

3.2 Основи розробки з Medusa

Medusa — це набір інструментів, які розробники можуть використовувати для створення додатків цифрової комерції. Незалежно від того, чи потрібно запропонувати унікальний досвід клієнтам, створити потужні засоби автоматизації або створити різноманітні комерційні програми, такі як ринки, Medusa надасть усі необхідні інструменти. Інші платформи електронної комерції пропонують обмежений набір функцій, доступних через API. Medusa відрізняється тим, що надає підприємствам і розробникам будівельні блоки для створення комерційних функцій. Це означає, що можна розширити свій комерційний API відповідно до своїх потреб[19].

Будівельні блоки Medusa поставляються як пакети NPM таких типів:

- Комерційні модулі, які є ізольованою комерційною логікою для різних доменів. Наприклад, модуль інвентаризації.
- Основний пакет, який відповідає за оркестрування різних комерційних модулів і надання REST API.

Основним пакетом є пакет NPM @medusajs/medusa. Це сервер Node.js, створений за допомогою Express та інших інструментів, які пропонують функції для керування подіями, кешуванням, чергами завдань тощо.

Основний пакет має дві основні цілі:

1. Оркеструвати комерційні модулі

Коли створюється комерційна програма за допомогою Medusa, зазвичай виконується взаємодія з більш ніж одним комерційним модулем. Базовий пакет керує зв'язками між модулями та переадресацією викликів до модулів у потрібний час під час виконання бізнес-логіки. Наприклад, уявіть собі модуль інвентаризації, який містить легку логіку для збільшення та зменшення рівнів запасів для одиниці зберігання (SKU). У комерційній програмі зазвичай потрібно пов'язати рівні запасів із певним продуктом. Medusa пропонує як модуль Inventory, так і модуль Product, і основний пакет створює зв'язки між цими модулями та виконує відповідну бізнес-логіку[19]. Отже, основний пакет містить код, подібний до того як показано на рис. 3.1:

```
import type {
MedusaRequest,
MedusaResponse,
} from "@medusajs/medusa"
export const POST = async (
    req: MedusaRequest,
    res: MedusaResponse
) => {
 // ...

 // пов'язати продукт з елементом інвентаризації
 const product = await productService.create(data)
 const inventoryItem = await inventoryService.create(
 inventoryData
 )
 await productVariantInventoryService.associate(
 product.id,
inventoryItem.id
 )

 // ...
}
```

Рис. 3.1. Код пакету

2. Відкриття REST API

Метою оркестрування модулів є надання API, який можуть використовувати клієнтські програми, як-от веб-сайти чи програми. За замовчуванням основний пакет Medusa надає REST API, який пропонує комерційні функції, подібні до тих, що пропонують інші платформи. Основний пакет також містить логіку, яка дозволяє

розробникам розширювати та додавати власні маршрути API, серед інших доступних налаштувань[20].

3.3. Приклади використання Medusa.

Розробники можуть налаштувати основний пакет і вручну вибрати комерційні модулі (рис. 3.2), які вони хочуть використовувати. Це дає їм велику гнучкість у виборі функцій, які вони хочуть надати у своєму магазині електронної комерції, одночасно використовуючи потужну архітектуру в основному пакеті.

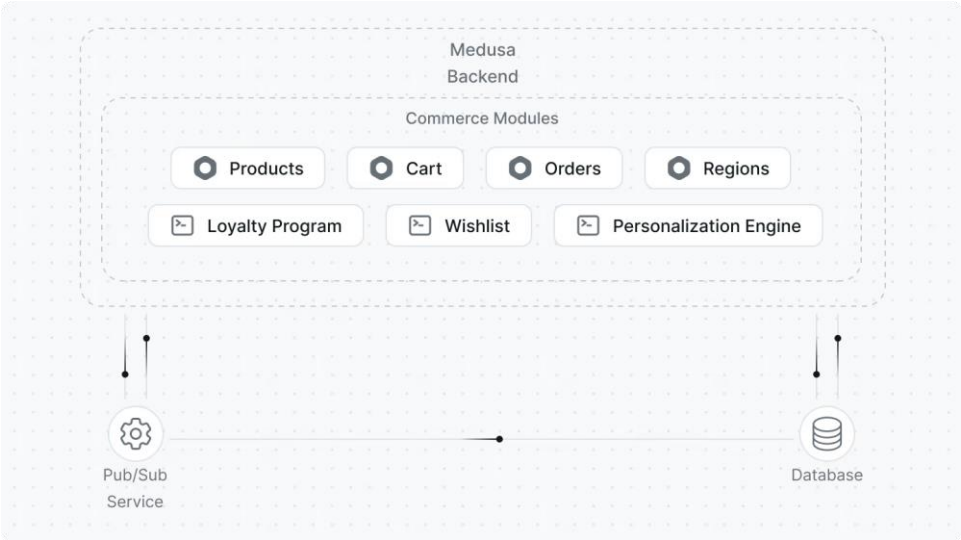


Рис. 3.2. Комерційні модулі

На зображенні представлено схему модулів комерції бекенду Medusa. Переклад компонентів:

- Medusa Backend (Медуза Бекенд)
- Commerce Modules (Комерційні Модулі):
  - Products (Продукти)
  - Cart (Кошик)
  - Orders (Замовлення)
  - Regions (Регіони)
  - Loyalty Program (Програма Лояльності)

    - Wishlist (Список Бажань)
    - Personalization Engine (Двигун Персоналізації)
- Pub/Sub Service (Сервіс Pub/Sub)
- Database (База Даних)

Ця схема ілюструє різні компоненти, які можна використовувати при розробці електронної комерційної платформи з використанням Medusa. Розробники можуть змінювати та налаштовувати модулі, які пропонує Medusa, відповідно до свого сценарію використання. Вони також можуть створювати власні модулі для впровадження нових функцій. Усі ці модулі стають будівельними блоками, які формують їхню систему електронної комерції. Комерційні модулі Medusa можна використовувати окремо від основного пакета та в більшій екосистемі. Наприклад, можна використовувати модуль Medusa’s Cart у блозі (рис. 3.3), щоб дозволити читачам купувати товари[20].

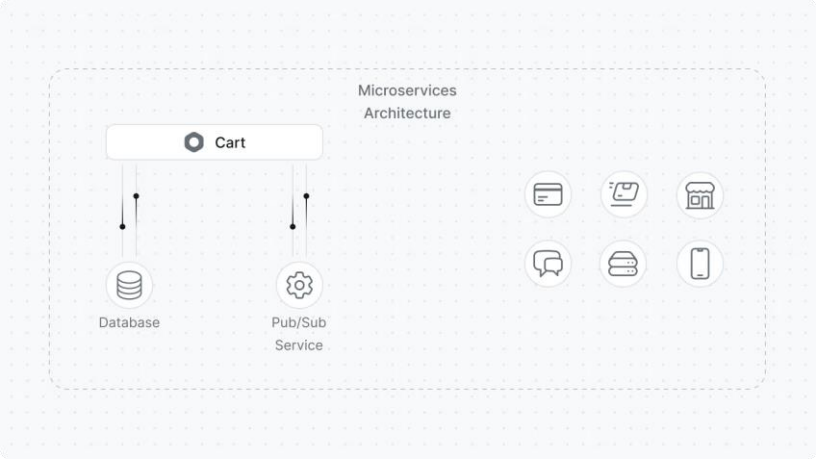


Рис. 3.3. Архітектура мікросервісів

На зображенні представлено схему мікросервісної архітектури. Ось переклад компонентів:

- Microservices Architecture (Мікросервісна Архітектура)
- Cart (Кошик)
- Database (База Даних)
- Pub/Sub Service (Сервіс Pub/Sub)

Розробники можуть скористатися модулями Medusa, які надають основні функції електронної комерції, зберігаючи екосистему електронної комерції за своїм вибором. Модулі Commerce можна встановити у системі як пакети NPM.

Вертикальні платформи електронної комерції

Платформа вертикальної електронної комерції – це платформа, яка надає спеціалізовані функції та функції для певного типу бізнес-сектору. Наприклад, платформа для фармацевтики. Розробники можуть використовувати Medusa для створення вертикальної платформи електронної комерції, оскільки вона забезпечує сходинки, які усувають потребу винаходити колесо для базових функцій електронної комерції, але є достатньо настроюваними, щоб їх можна було змінювати відповідно до конкретного випадку використання[20].

Встановлені API

Оскільки комерційні модулі Medusa є пакетами NPM, їх можна встановити та використовувати в будь-якому проекті JavaScript.
Установивши модуль у свій проект і надавши його API на основі фреймворку, який використовується, можна отримати REST API електронної комерції прямо зі свого зовнішнього фреймворку без необхідності створювати окремий проект[20].

Розробники можуть використовувати набір інструментів Medusa для створення своєї системи електронної комерції як показано на рис. 3.4. За допомогою команди create-medusa-app розробники можуть налаштувати сервер Medusa, адміністратор Medusa та вітрину.

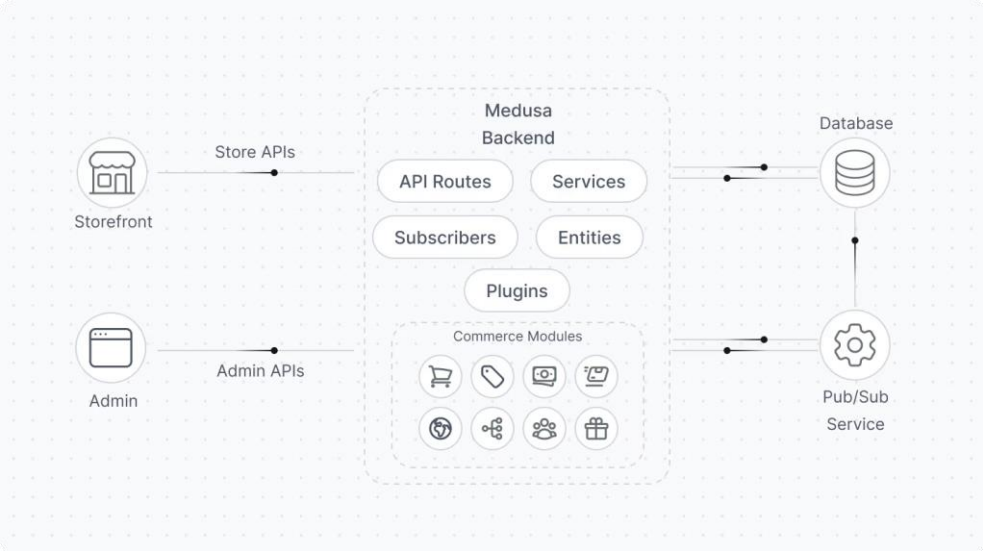


Рис. 3.4. Повноцінна система електронної комерції

На зображенні схема архітектури системи з різними компонентами і зв'язками між ними. Переклад назв компонентів та елементів системи, що зазначені на діаграмі:

- Storefront — Вітрина
- Store APIs — API магазину
- Admin — Адміністрування
- Admin APIs — API адміністрування
- Medusa Backend — Бекенд Medusa
- API Routes — Маршрути API
- Services — Сервіси
- Subscribers — Підписники
- Entities — Сутності
- Plugins — Плагіни
- Commerce Modules — Комерційні модулі
- Database — База даних
- Pub/Sub Service — Сервіс публікації/підписки

Розробники все ще можуть скористатися можливостями налаштування, які надає Medusa. Це включає створення таких ресурсів, як API-маршрути та служби, створення плагінів, інтеграцію сторонніх служб, створення спеціальної вітрини тощо.

3.4 Структура директорій бекенду Medusa

Структура директорій бекенду Medusa є ключовим елементом для розуміння та ефективної роботи з цією платформою. Вона відіграє важливу роль у організації коду та ресурсів, забезпечуючи чітке розмежування функціональності та спрощуючи процес розробки. Нижче наведено детальний опис кореневих файлів та директорій, які формують основу проекту бекенду Medusa[21].

Кореневі файли:

Кореневі файли - це набір критично важливих файлів, які знаходяться в корені бекенду Medusa. Вони відіграють ключову роль у налаштуванні, конфігурації та управлінні проектом. Кожен файл має свою специфічну роль, наприклад, .babelrc.js задіяний у процесі транспіляції коду, .env містить змінні середовища для розробки, а .gitignore визначає, які файли слід ігнорувати системою контролю версій Git. Файли tsconfig.json і tsconfig.server.json налаштовують TypeScript для різних частин проекту, в той час як README.md надає важливу інформацію про проект. Всі ці файли разом формують фундамент для ефективної розробки та підтримки проекту на Medusa[21].

- .babelrc.js: Визначає конфігурації Babel, які використовуються при виконанні команди build, що транспілює файли з директорії src в dist.
- .env: Містить значення змінних середовища. Зазвичай використовується тільки під час розробки. У продакшні змінні середовища слід визначати в залежності від хостинг-провайдера.
- .env.template: Надає приклад того, які змінні можуть бути включені в .env.
- .gitignore: Вказує файли, які не слід комітити в Git-репозиторій.
- .yarnrc.yml: Забезпечує встановлення залежностей завжди в node-modules. Це забезпечує сумісність з pnpm.
- index.js: Визначає вхідний файл, корисний при запуску бекенду Medusa за допомогою менеджера процесів, наприклад, pm2.
- medusa-config.js: Визначає конфігурації бекенду Medusa, включаючи конфігурації бази даних, використовувані плагіни, модулі та інше.

- package.json: Оскільки бекенд Medusa є пакетом NPM, цей файл визначає його інформацію та залежності.
- README.md: Надає загальну інформацію про бекенд Medusa.
- tsconfig.admin.json: Визначає конфігурації TypeScript для транспіляції файлів налаштувань адміністратора.
- tsconfig.json: Визначає загальні конфігурації TypeScript для транспіляції файлів з src в dist.
- tsconfig.server.json: Визначає конфігурації TypeScript для транспіляції файлів налаштувань бекенду Medusa.
- tsconfig.spec.json: Визначає конфігурації TypeScript для тестових файлів.
- yarn.lock або package-lock.json: Автоматично генерований файл, який зберігає поточні версії всіх встановлених залежностей.

Кореневі директорії:

Кореневі директорії - це ключові папки, які знаходяться в корені бекенду Medusa, що відіграють важливу роль в організації та управлінні проектом. Кожна директорія має своє призначення: .cache зберігає кешовані файли для побудови адміністраторських активів, build містить файли, готові до використання в браузері, data використовується для заповнення демонстраційними даними, а dist - для зберігання транспільованих налаштувань. Папка node_modules важлива для зберігання всіх залежностей проекту, src містить основні налаштування бекенду та адміністратора, а uploads - для зберігання завантажених файлів[20-21]. Ці директорії разом створюють структуру та основу для ефективної роботи з бекендом Medusa, дозволяючи легко навігувати та управляти різними аспектами проекту.

- .cache: Містить усі кешовані файли, пов'язані з побудовою активів адміністратора Medusa.
- build: Містить побудовані файли, які використовуються для обслуговування адміністратора у браузері.
- data: Містить JSON-файл, який використовується для заповнення бекенду Medusa демонстраційними даними.

- dist: Містить транспільовані налаштування бекенду Medusa.
- node_modules: Містить усі встановлені залежності у проекті.
- src: Містить усі налаштування бекенду та адміністратора Medusa.
- uploads: Містить усі файли, завантажені в бекенд Medusa.

Піддиректорії src:

Файли, розташовані в директорії src бекенду Medusa, є важливими елементами, які включають налаштування та конфігурації для бекенду та адміністратора. Ці файли є основою для розробки, оскільки вони містять весь код, який визначає функціональність бекенду Medusa. Після розробки ці файли потрібно транспілювати в директорію dist для того, щоб вони могли бути ефективно використані під час роботи бекенду. Ця процедура транспіляції забезпечує, що код оптимізовано для продакшну, забезпечуючи вищу продуктивність та стабільність. Структура директорії src дозволяє легко навігувати та організувати код, що сприяє чіткому розмежуванню різних аспектів бекенду, таких як адміністраторські налаштування, API-маршрути, міграції, моделі, репозиторії, сервіси та підписники. Ця організація спрощує розробку та налаштування, забезпечуючи ефективність та зрозумілість процесу роботи з бекендом Medusa[20].

- admin: Містить усі налаштування адміністратора Medusa.
- api: Містить усі користувацькі API-маршрути.
- loaders: Містить скрипти, які запускаються при старті бекенду Medusa.
- migrations: Містить усі скрипти міграцій.
- models: Містить усі користувацькі сутності.
- repositories: Містить усі користувацькі репозиторії.
- services: Містить усі користувацькі сервіси.
- subscribers: Містить усі користувацькі підписники.

Ця структура директорій допомагає організувати та управляти різними аспектами бекенду Medusa, забезпечуючи чітке розмежування функціональності та спрощуючи розробку та налаштування.

3.5 Огляд інтерфейсу адмін панелі Medusa

Адміністративна панель Medusa є центральним елементом для управління та моніторингу електронної комерції, який забезпечує інтуїтивно зрозумілий інтерфейс для виконання різноманітних задач. Вона розроблена з метою забезпечення зручності та ефективності для адміністраторів магазинів, менеджерів продуктів та інших користувачів, які відповідають за управління комерційними операціями[20].

Основні компоненти інтерфейсу:

- **Головне меню**: Розташоване зліва, головне меню надає швидкий доступ до основних розділів, таких як продукти, замовлення, клієнти, налаштування тощо.
- **Робочий стіл**: Центральна частина інтерфейсу, де відображаються деталі та опції для кожного вибраного розділу з головного меню.
- **Панель пошуку**: Дозволяє швидко знаходити продукти, замовлення, клієнтів та інші елементи в системі.
- **Панель сповіщень**: Інформує про важливі події, оновлення або системні повідомлення.
- **Налаштування користувача**: Розділ, де можна змінити особисті налаштування, управління обліковим записом або вийти з системи.

Функціональні можливості:

- **Управління продуктами**: Дозволяє створювати, редагувати та видаляти продукти, керувати запасами та категоріями.
- **Обробка замовлень**: Перегляд та управління замовленнями, включаючи обробку повернень та відправлень.
- **Клієнтське управління**: Управління інформацією про клієнтів, включаючи історію замовлень та персональні дані.
- **Фінансові звіти**: Моніторинг фінансових показників, включаючи продажі, доходи та інші ключові метрики.

- **Налаштування магазину**: Конфігурація основних параметрів магазину, включаючи платіжні системи, доставку, податки та інші налаштування.

Інтуїтивно зрозумілий дизайн

Інтерфейс адміністративної панелі Medusa розроблений з акцентом на інтуїтивність та зручність використання. Він забезпечує чітке розмежування розділів та функцій, що дозволяє користувачам легко орієнтуватися та ефективно виконувати необхідні задачі. Завдяки гнучкості налаштувань, адміністратори можуть адаптувати робочий простір до своїх потреб, оптимізуючи процеси управління та аналітики.

Цей огляд інтерфейсу адміністративної панелі Medusa демонструє її ключові особливості та функціональні можливості, які роблять її ефективним інструментом для управління всіма аспектами електронної комерції[20].

Доступ до адміністратора Medusa

Для доступу до адміністративної панелі необхідно використовувати URL-адресу розгорнутого веб-сайту. Сторінка входу з'являється після відкриття цієї URL-адреси, де користувачі повинні ввести свою електронну пошту та пароль для входу в систему. У разі виникнення проблем із доступом до облікового запису, рекомендується звернутися до технічного персоналу, який відповідає за розгортання серверної частини та адміністратора Medusa[21].

Сторінка входу

Сторінка входу є важливим елементом безпеки, оскільки вона контролює доступ до управлінського інтерфейсу магазину. Вона вимагає від користувачів ввести вірні облікові дані для доступу до адміністративної панелі як показано на рис. 3.5:

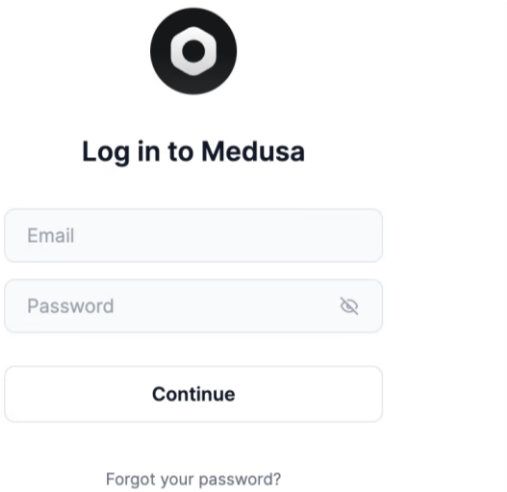


Рис. 3.5. Сторінка входу

Відновлення доступу

У випадку забутих паролів або необхідності зміни облікових даних, більшість систем Medusa надають можливість відновлення пароля через електронну пошту. Це забезпечує додатковий рівень безпеки та гнучкість у управлінні обліковими записами[21].

Безпека облікових даних

Безпека облікових даних є критично важливою для забезпечення захисту інформації та управління магазином. Використання складних паролів та їх регулярне оновлення є ключовими аспектами забезпечення безпеки облікових записів.

Після входу в адміністративну панель Medusa користувач направляється на головну сторінку інтерфейсу (рис. 3.6), де відображається центральний робочий стіл. Цей робочий стіл забезпечує швидкий доступ до ключових функцій та інформаційних панелей, які важливі для щоденного управління магазином. На цій сторінці користувачі можуть переглядати оглядові панелі, які відображають актуальну статистику продажів, замовлень, запасів продуктів та інші важливі метрики, що допомагають у прийнятті управлінських рішень[21].

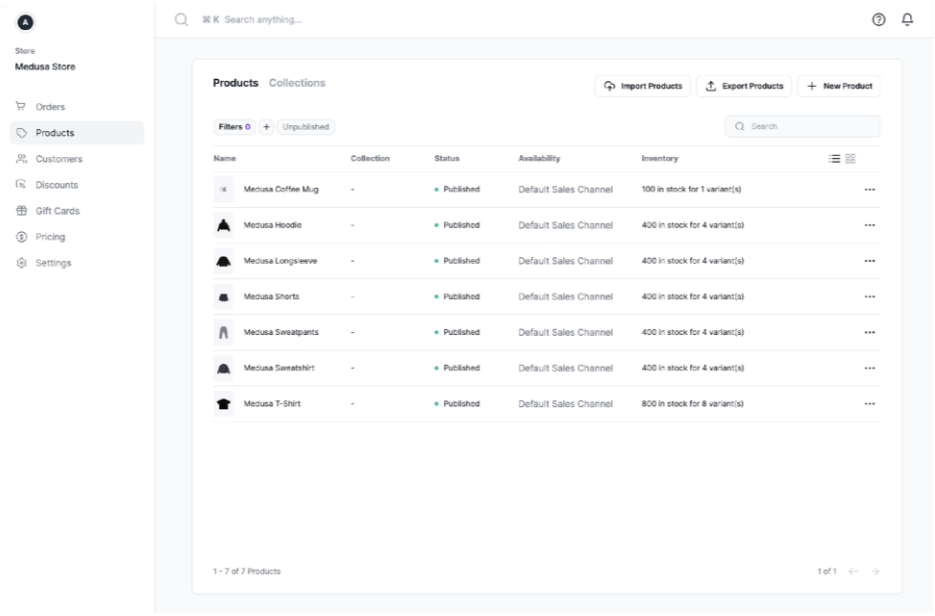


Рис. 3.6. Головна сторінка адміністративної панелі

З лівого боку розташоване навігаційне меню яке показано на рис. 3.7, яке дозволяє користувачам легко переходити між різними розділами адміністративної панелі, такими як управління продуктами, замовленнями, клієнтами, налаштуваннями магазину та іншими. Кожен розділ містить спеціалізовані інструменти та опції, призначені для конкретних аспектів управління магазином[21].

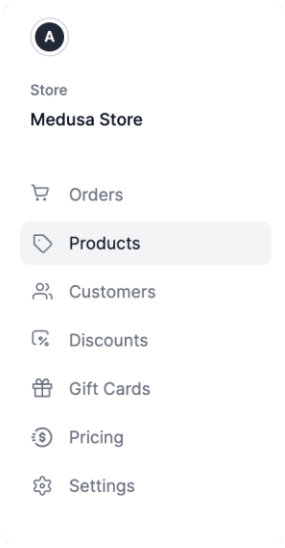


Рис. 3.7. Меню бічної панелі

У верхній частині інтерфейсу розташована панель пошуку (рис. 3.8), яка дозволяє користувачам швидко знаходити конкретні продукти, замовлення або

клієнтів. Ця функція значно спрощує навігацію та управління великою кількістю даних[21]. Відкрити вікно пошуку можна за допомогою наступних команд:

Windows і Linux: CTRL + K

ОС Mac: ⌘ + K

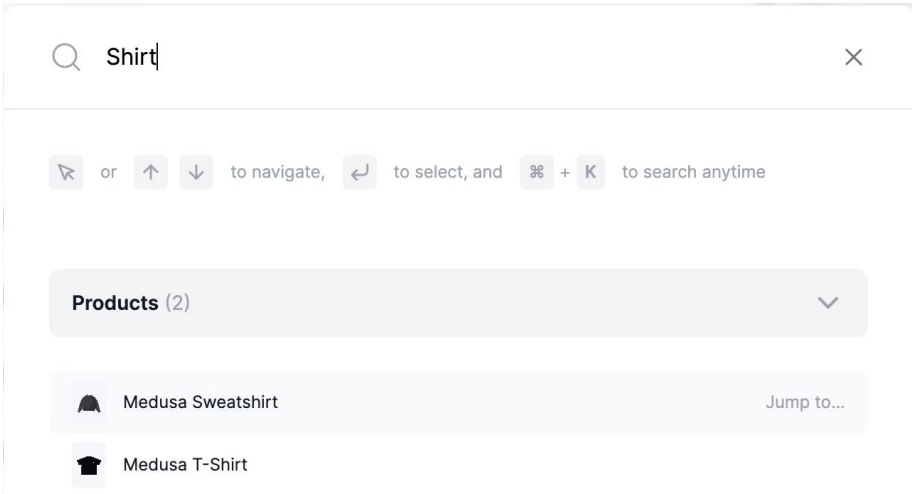


Рис. 3.8. Панель пошуку

Користувач може вибрати результат за допомогою миші. Або використовувати стрілки вгору та вниз на клавіатурі для переходу між результатами, а потім обрати результат, натиснувши клавішу Enter.У верхньому лівому куті бічної панелі можна знайти значок аватара[21]. Натиснувши цей значок, відкриється спадне меню яке показане на рис. 3.9, в якому можна отримати доступ до налаштувань або вийти.

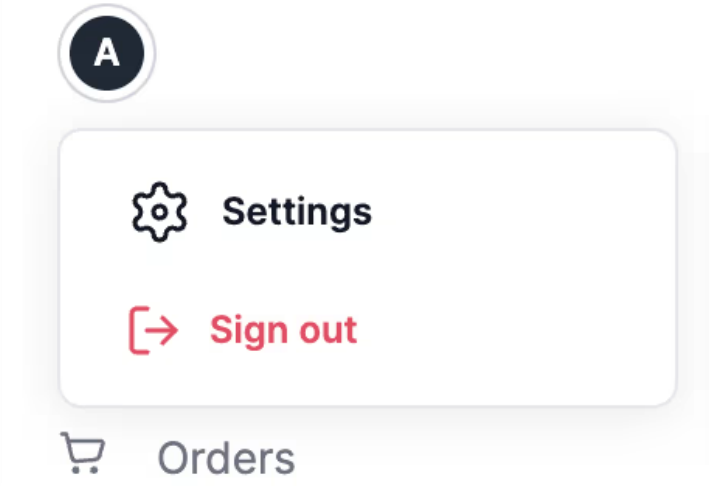


Рис. 3.9. Панель швидких дій

3.6 Огляд меню бічної панелі

3.6.1 Огляд замовлення

Коли клієнти купують один або декілька продуктів, їх замовлення з'являється на сторінці "Замовлення". На цій сторінці можна переглянути інформацію про замовлення, здійснити оплату, створити відправлення та виконання, зареєструвати повернення або обмін замовлення та більше.

Домен "Замовлення" також містить чернетки замовлень. Чернетки замовлень - це замовлення, які створюються з адміністративної панелі Medusa. Після заповнення замовлення та позначення його як оплаченого, воно перетворюється на замовлення, яке з'являється у списку замовлень[22].

Перегляд списку замовлень

Список доступних замовлень у магазині електронної комерції можна переглянути, натиснувши на "Замовлення" у бічному меню.

У списку можна побачити деталі замовлення, такі як ID, дата, клієнт, статус виконання та оплати, загальна сума та країна доставки, як показано на рис. 3.10:

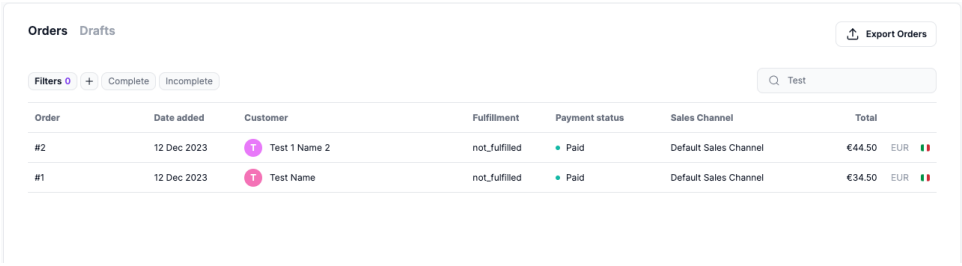


Рис. 3.10. Панель замовлень

Перегляд списку чернеток замовлень

Список чернеток замовлень на сторінці "Замовлення" можна переглянути, натиснувши на сірий заголовок "Чернетки" поруч із заголовком "Замовлення".

У списку можна побачити деталі чернетки замовлення, такі як ID, ID замовлення, до якого належить чернетка, дата, клієнт та статус чернетки замовлення[22].

3.6.2 Огляд продуктів

Продукти є товарами, які продаються у магазині. Вони можуть бути як фізичними, так і цифровими, а також можуть бути простими продуктами або продуктами з варіантами.Можна створювати продукти та редагувати їхні деталі. Це включає базову інформацію, канали продажів, управління запасами, управління цінами та багато іншого. Частиною домену продуктів є категорії. Категорії дозволяють організовувати продукти у більш ніж одну категорію. Також можна управляти ієрархією категорій. Наприклад, категорія "Дім" з вкладеною категорією формату креслення. Іншою частиною домену є колекції. Колекції використовуються для розділення продуктів за маркетинговими або продажними цілями[22].

Перегляд списку продуктів

Список доступних продуктів у магазині електронної комерції можна переглянути, натиснувши на "Продукти" у бічному меню.

У списку можна побачити деталі продукту, такі як назва, колекція, запаси та статус продукту, як показано на рис. 3.11:

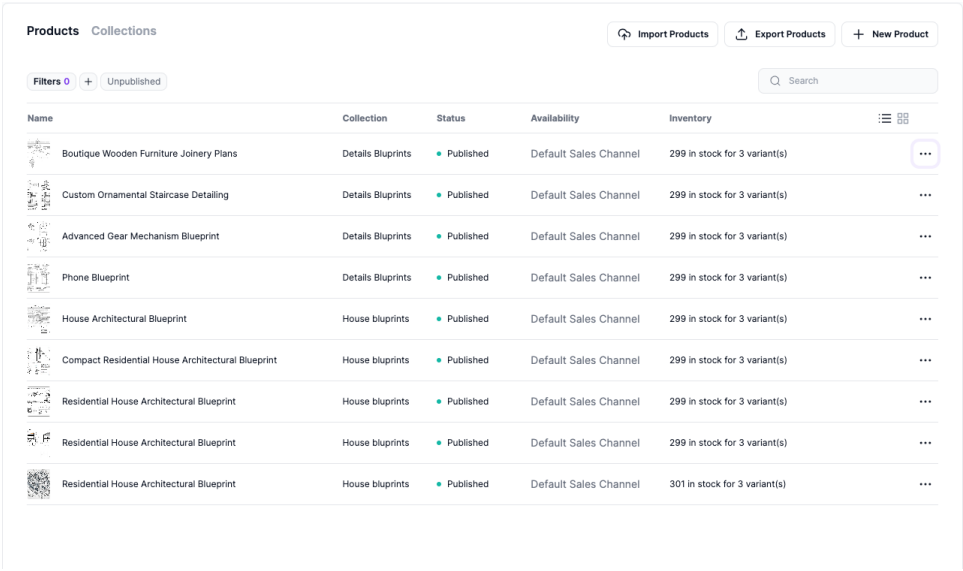


Рис. 3.11. Список продуктів

Щоб переглянути продукти у вигляді списку як на рис. 3.11, який є стандартним виглядом, треба натиснути на відповідну іконку у верхньому правому куті списку продуктів. Щоб переглянути продукти у вигляді сітки як на рис. 3.12,

треба натиснути на відповідну іконку у верхньому правому куті списку продуктів[21].

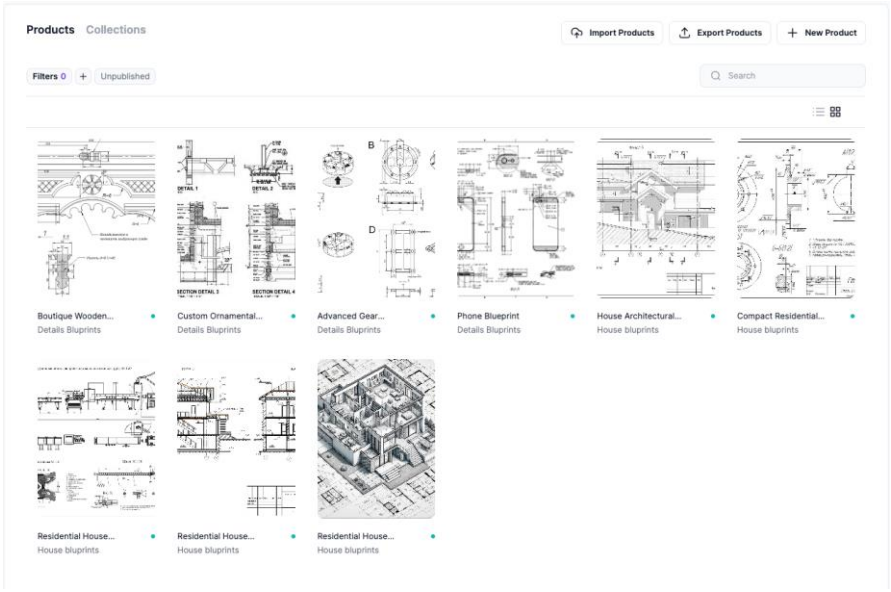


Рис. 3.12. Список продуктів у вигляді сітки

Список колекцій на сторінці "Продукти" можна переглянути, натиснувши на сірий заголовок "Колекції" поруч із заголовком "Продукти". У списку можна побачити деталі колекції, такі як назва, ідентифікатор та кількість продуктів у колекції[22].

3.6.3 Огляд клієнтів

Список зареєстрованих клієнтів магазину можна знайти на сторінці "Клієнти". Для кожного з цих клієнтів можна переглянути деталі, такі як їхні замовлення або особиста інформація. Частиною домену "Клієнти" також є групи клієнтів. Групи клієнтів дозволяють об'єднувати набір клієнтів для бізнес-цілей. Наприклад, можна створити групу VIP-клієнтів для клієнтів, яким необхідно надати спеціальні знижки[22].

Перегляд списку клієнтів

Список доступних клієнтів у магазині електронної комерції можна переглянути, натиснувши на "Клієнти" у бічному меню.

У списку можна побачити деталі клієнта, такі як ім'я клієнта, електронна адреса та кількість замовлень, як покаано на рис. 3.13:

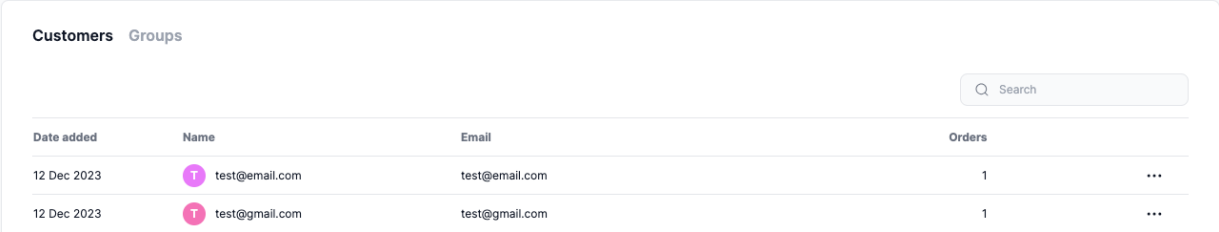


Рис. 3.13. Список клієнтів

Перегляд списку груп клієнтів

Список груп клієнтів на сторінці "Групи клієнтів" можна переглянути, натиснувши на сірий заголовок "Групи" поруч із заголовком "Клієнти".

У списку можна побачити деталі групи, такі як назва групи та кількість клієнтів у групі.

3.6.4 Огляд знижок

У Medusa можна створювати знижки різних типів, які клієнти можуть використовувати під час оформлення замовлення:

- **Відсоткові знижки** для зменшення загальної суми замовлення на певний відсоток.
- **Фіксовані знижки** для зменшення загальної суми замовлення на фіксовану суму.
- **Безкоштовна доставка** для анулювання вартості доставки замовлення та зроблення її безкоштовною.

Знижки представлені у вигляді коду, який клієнти можуть використовувати. Можна встановити параметри для знижки, такі як дата початку та закінчення.

Також можна вказати умови для знижок, наприклад, застосування їх до певних продуктів або варіантів доставки[22].

Перегляд списку знижок

Список доступних знижок у магазині електронної комерції можна переглянути, натиснувши на "Знижки" у бічному меню.

У списку можна побачити деталі знижки, такі як код, опис, сума та статус, як покаано на рис. 3.14:

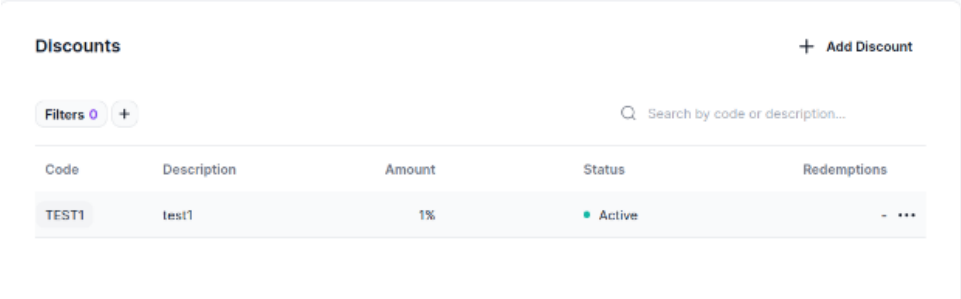


Рис. 3.14. Список знижок

3.6.5 Огляд цінових списків

Цінові списки (рис. 3.15) в Medusa є важливим інструментом для гнучкого управління цінами продуктів, дозволяючи торговцям адаптувати ціноутворення до різних ринкових умов та потреб клієнтів.

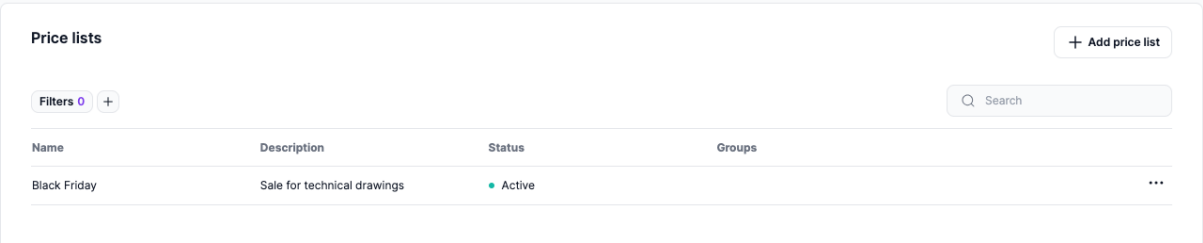


Рис. 3.15. Список цін

Створення та налаштування цінових списків

Цінові списки дозволяють встановлювати спеціальні ціни для окремих продуктів або груп продуктів. Торговці можуть створювати цінові списки для проведення акцій, сезонних розпродажів або надання ексклюзивних знижок певним групам клієнтів[22].

- **Період дії**: Встановлення дат початку та закінчення дозволяє контролювати часові рамки, протягом яких діють спеціальні ціни.
- **Цільові групи клієнтів**: Можливість застосування цінових списків до певних груп клієнтів, наприклад, VIP-клієнтів або постійних покупців.
- **Гнучкість ціноутворення**: Встановлення різних цін для різних продуктів або категорій, що дозволяє оптимізувати стратегію ціноутворення.

Управління ціновими списками

Управління ціновими списками включає перегляд, редагування та видалення існуючих списків. Торговці можуть легко оновлювати умови та параметри цінових списків, реагуючи на зміни ринкових умов або стратегій бізнесу.

Моніторинг та аналітика

Цінові списки можуть бути важливим інструментом для моніторингу реакції ринку та поведінки клієнтів. Аналізуючи ефективність різних цінових стратегій, торговці можуть краще розуміти попит та вподобання клієнтів, а також оптимізувати свої продажі та маркетингові кампанії[23].

Інтеграція з іншими функціями

Цінові списки в Medusa можуть бути інтегровані з іншими функціями платформи, такими як управління продуктами, замовленнями та знижками, що забезпечує єдиний підхід до управління цінами та продажами. Цей розширений огляд цінових списків у Medusa підкреслює їх важливість як інструменту для гнучкого та ефективного управління ціноутворенням в сфері електронної комерції. Він демонструє, як торговці можуть використовувати цінові списки для адаптації своїх цінових стратегій до різних потреб та умов ринку[23].

3.6.6 Огляд налаштувань

У налаштуваннях Medusa можна управляти різними аспектами інтернет-магазину. Доступні такі категорії, як показано на рис. 3.16:

- **Регіони**: Дозволяє вибрати та налаштувати географічні ринки, на яких буде представлений магазин, включаючи специфічні податкові та валютні налаштування.
- **Деталі магазину**: Тут можна ввести та оновити основну інформацію про бізнес, таку як назва, адреса, контактні дані.
- **Причини повернення**: Категорія для керування можливими причинами повернення товарів, що допомагає організувати процес повернень.
- **Особиста інформація**: Розділ для управління особистими даними власника магазину, включаючи контактну інформацію та налаштування безпеки.
- **Налаштування податків**: Місце для конфігурації податків за різними регіонами та продуктами, що включає ставки податків та правила їх застосування.
- **Управління API ключами**: Створення та управління ключами API для інтеграції з зовнішніми сервісами та дозволами доступу.
- **Валюти**: Визначення та налаштування валют, які будуть використовуватись у магазині, та їх курси обміну.
- **Доставка**: Налаштування профілів доставки, включаючи постачальників, тарифи та правила доставки.
- **Команда**: Управління обліковими записами користувачів магазину, включаючи дозволи та ролі.
- **Канали продажів**: Конфігурація доступності продуктів у різних продажних каналах, таких як онлайн-маркетплейси або фізичні магазини

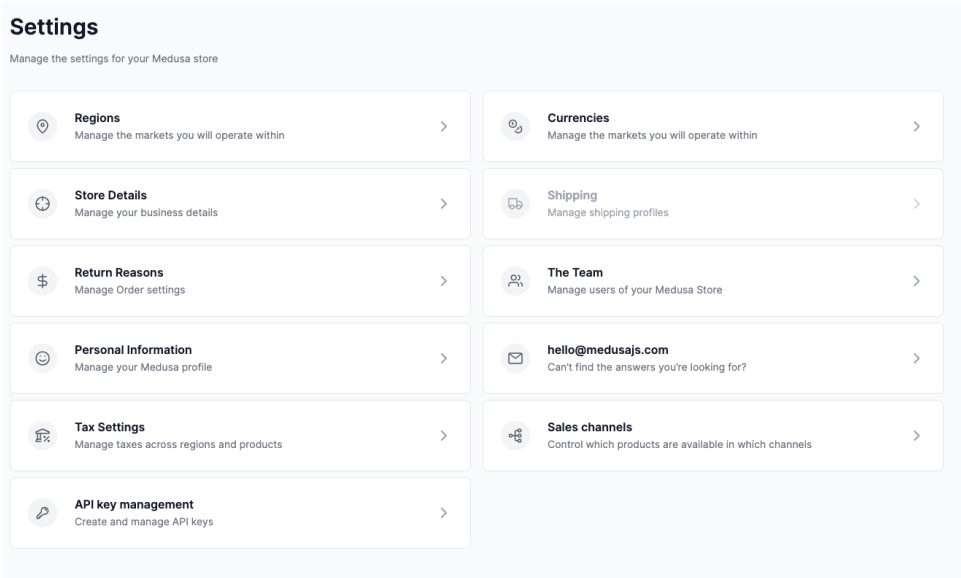


Рис. 3.16. Налаштування адмін панелі

Висновки до розділу

У даному розділі магістерської дисертації представлено всебічний аналіз процедури налаштування та використання платформи Medusa.js для створення додатків електронної комерції. Розглянуто детально процес інсталяції Medusa за допомогою команди create-medusa-app, висвітлено особливості та варіативність застосування цієї команди з різними параметрами для адаптації під конкретні потреби розробки[23].

Далі описано ключові аспекти розробки на базі Medusa, включаючи аналіз її архітектури, основних компонентів та можливостей їх розширення. Виділено комерційні модулі як фундаментальні елементи для формування бізнес-логіки майбутнього додатку.

Презентовано типові сценарії використання Medusa з ілюстративними схемами, що демонструють її застосування як комплексного рішення для електронної комерції, а також використання окремих модулів чи API для інтеграції з іншими системами.

Описано структуру файлів та директорій у проектах на базі Medusa, надаючи зрозуміле уявлення про організацію коду, включаючи роль ключових піддиректорій та значення основних файлів конфігурації.

Розділ завершується детальним оглядом інтерфейсу адміністративної панелі Medusa, де послідовно розглянуто розташування та функції основних елементів панелі керування, включаючи меню, робочий стіл, панелі та списки об'єктів[23].

Таким чином, наданий у цьому розділі матеріал дає комплексне розуміння процесу налаштування та розробки з використанням Medusa, що дозволяє ефективно створювати сучасні та функціонально насичені додатки електронної комерції.

РОЗДІЛ 4. СТВОРЕННЯ ВЕБ-САЙТУ ТА РОЗМІЩЕННЯ НА ХОСТИНГУ

4.1 Інтеграція фронтенду та бекенду

У сучасному швидкоплинному світі компаніям необхідно забезпечити безперебійний і ефективний досвід онлайн-покупок для своїх клієнтів. Цього можна досягти лише шляхом інтеграції надійних і гнучких серверних рішень із ефективними інтерфейсними фреймворками розробки. У цьому есе ми обговоримо переваги інтеграції інтерфейсу Next.js і бекенда Medusa.js для ефективної розробки в електронній комерції. Спочатку ми висвітлимо аргументи на користь цієї інтеграції, а потім розглянемо контраргументи, щоб забезпечити всебічне розуміння теми[24]. Next.js забезпечує плавну та ефективну розробку інтерфейсу. Це пов’язано з декількома факторами, такими як рендеринг на стороні сервера, легка маршрутизація та обробка API, а також ефективне розділення та оптимізація коду. Візуалізація на стороні сервера забезпечує швидше завантаження сторінок і покращення пошукової оптимізації, а легка маршрутизація та обробка API спрощують керування складними програмами. Ефективне поділ і оптимізація коду дозволяють розробникам зосередитися на створенні високоякісного коду, не турбуючись про проблеми з продуктивністю[24].

Medusa.js надає надійне та гнучке серверне рішення. Це пов’язано з кількома функціями, такими як настроювана платформа електронної комерції, проста інтеграція з різними платіжними шлюзами та проста інтеграція з різними постачальниками послуг доставки. Ці функції полегшують розробникам створення налаштованої платформи електронної комерції, яка відповідає конкретним потребам їхнього бізнесу.

Інтеграція інтерфейсу Next.js із сервером Medusa.js може забезпечити бездоганний досвід електронної комерції. Це пов’язано з декількома факторами, такими як можливість обмінюватися даними через API, швидкий і чутливий досвід електронної комерції та легке налаштування відповідно до конкретних потреб бізнесу. Обмін даними через API забезпечує плавний потік даних між інтерфейсом і

сервером, що забезпечує кращу взаємодію з користувачем. Швидкий і чуйний досвід електронної комерції гарантує, що клієнти можуть робити покупки без будь-яких затримок або проблем. Легке налаштування відповідно до конкретних бізнес-потреб гарантує, що платформа відповідає унікальним вимогам кожного бізнесу[24].

Інтеграція двох різних фреймворків може призвести до проблем із сумісністю. Різні фреймворки можуть мати різні залежності та версії пакетів, стандарти кодування та практики. Інтеграція двох різних фреймворків може потребувати додаткового часу та ресурсів на розробку. Ці проблеми можуть призвести до повільнішого та складнішого процесу розробки. Саме через це було обрано готове рішення яке можна кастомізувати під себе Вибір Next.js Starter Template для розробки фронтенду був обумовлений низкою переваг, які він пропонує. Стартовий шаблон Next.js не тільки спрощує процес розробки завдяки вбудованим функціям та оптимізаціям, але й забезпечує міцний фундамент для розширення та налаштування інтерфейсу згідно з унікальними вимогами проекту[25].

4.1.1 Встановлення та налаштування Next.js Starter Storefront

Оскільки у розділі 3 було встановлено та налаштовано серверну частину Medusa, залишається встановити та налаштувати Next.js Starter Storefront з вже встановленою серверною частиною Medusa. Це включає декілька ключових етапів:

1. **Створення нового проекту Next.js:**

Використання команди npx create-next-app дозволяє створити новий проект Next.js. Аргумент -e https://github.com/medusajs/nextjs-starter-medusa вказує на шаблон Medusa Starter, який буде використовуватися для створення проекту. my-medusa-storefront в кінці команди є назвою новоствореного проекту, яку можна змінити на будь-яку іншу назву за бажанням[25].

2. Перехід до директорії проекту:

Виконання команди cd my-medusa-storefront переносить користувача в директорію, де знаходиться новостворений проект.

3. Перейменування файлу змінних середовища:

Файл .env.template містить шаблон змінних середовища, які потрібні для налаштування проекту. Команда mv .env.template .env.local перейменовує цей файл, роблячи його активним для використання в проекті.

4. Налаштування змінних середовища:

У файлі .env.local потрібно вказати значення змінних, які відповідають конкретному середовищу розробки. Особлива увага приділяється змінній NEXT_PUBLIC_MEDUSA_BACKEND_URL, яка має містити URL-адресу, за якою запущено Medusa backend.

5. Запуск локального сервера Next.js:

Команда npm run dev запускає локальний сервер для розробки. Стандартно сервер запускається на порту 3000, але це можна змінити в налаштуваннях проекту.

6. Перевірка інтеграції з Medusa Backend:

Після запуску Next.js Starter Storefront важливо переконатися, що він взаємодіє з Medusa backend. Medusa backend має бути запущений і доступний за вказаною URL-адресою.

7. Додаткові налаштування:

- Інтеграція з пошуковими системами: Якщо потрібно інтегрувати пошукову систему (наприклад, MeiliSearch або Algolia), потрібно налаштувати відповідні змінні середовища та конфігурації. В нашому випадку використовується MeiliSearch.
- Інтеграція платіжних систем: Для використання платіжних систем, таких як Stripe або PayPal, потрібно встановити та активувати їх на Medusa backend, а також налаштувати відповідні змінні середовища у Next.js проекті.
- Персоналізація: Можливість персоналізації Next.js Starter Storefront полягає у редагуванні файлів у директоріях src/app, src/modules та src/styles. Це дозволяє налаштувати сторінки, компоненти та стилі відповідно до індивідуальних потреб та вимог.

4.1.2 Встановлення та налаштування плагіну MeiliSearch

MeiliSearch — це надшвидка пошукова система з відкритим кодом, створена на Rust. Вона має широкий набір функцій, включаючи захист від друкарських помилок, фільтрацію та сортування. MeiliSearch також забезпечує приємний досвід для розробників, оскільки вона надзвичайно інтуїтивно зрозуміла і зручна для новачків. Таким чином, навіть якщо користувач новачок в екосистемі пошукових систем, документація MeiliSearch є достатньо ресурсною, щоб кожен міг її пройти та зрозуміти. Завдяки гнучкій системі плагінів Medusa можна додати пошукову систему до серверної частини та вітрини Medusa за допомогою MeiliSearch лише за кілька кроків[25].

Щоб використовувати Meilisearch Cloud, спочатку потрібно створити проект. Проекти діють як контейнери для індексів, завдань, платежів та іншої інформації, пов'язаної з Meilisearch Cloud. Для створення проекту потрібно натиснути на кнопку «Новий проект» у верхньому меню яке продеманстровано на рис. 4.1[25].

Рис. 4.1 Верхнє меню

Після цього потрібно дати назву проекту <назва проекту>, обрати найближчий регіон та потім натиснути «Створити проект», як показано на рис.4.2:

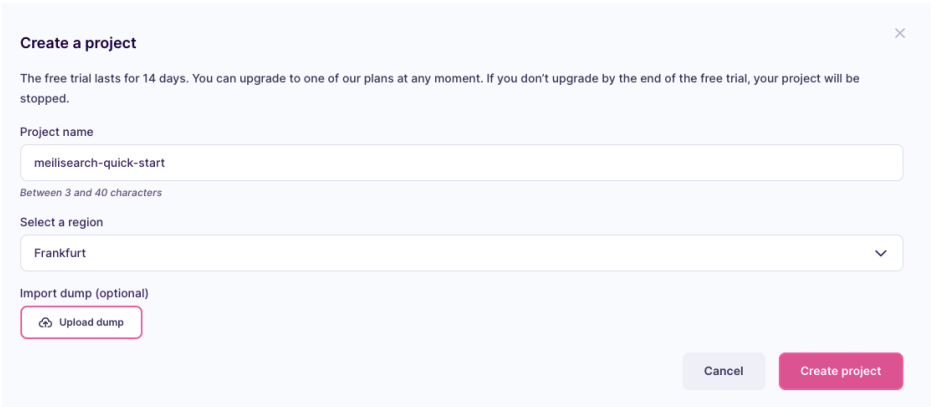


Рис. 4.2. Модальне вікно ствоерння проекту

Створення проекту може зайняти кілька хвилин. Перевірити статус можна у списку проектів, який виділено на рис. 4.3. Коли проект буде готовий, треба натиснути на його назву, щоб перейти на сторінку огляду проекту:

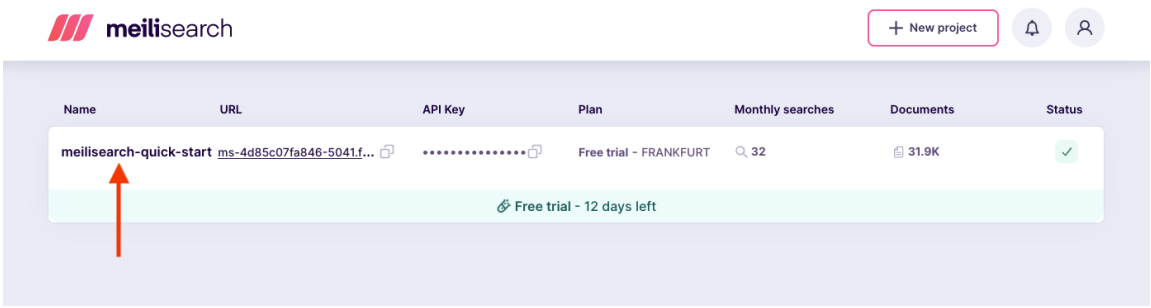


Рис. 4.3. Список проектів

Після створення проекту потрібно проіндексувати дані, які будуть використовуватись у пошуку. Meilisearch зберігає та обробляє дані, які додаються до нього в індексах. Один проект може містити декілька індексів. Натиснувши у верхній панелі на кнопку “Indexes” в новому вікні, яке продемонстровано на рис. 4.4, буде запропоновано створити ім’я індексу:

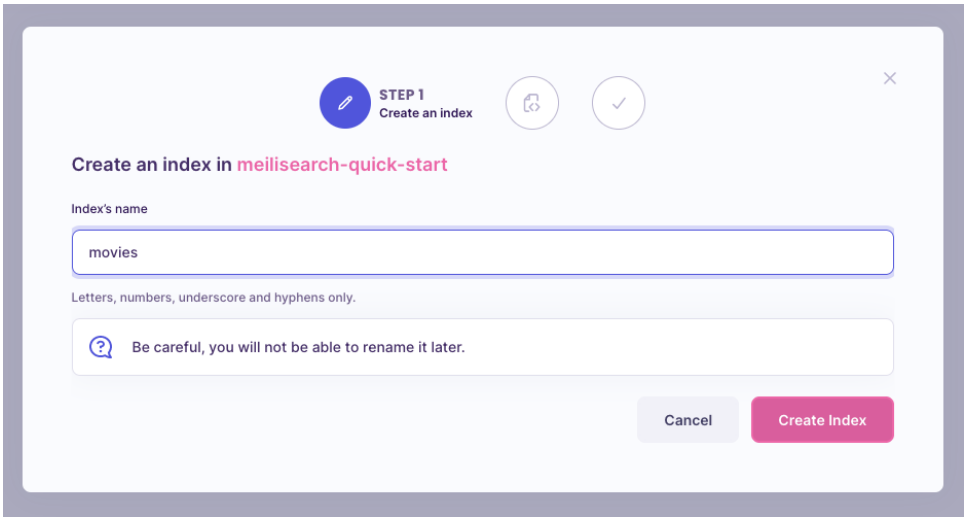


Рис. 4.4. Модальне вікно створення індексу

Останнім кроком у створенні індексу є додавання даних до нього у вигляді json файлу, Meilisearch Cloud проіндексує документи, це може зайняти деякий час. Коли

це буде зроблено, на вкладці «Налаштування» можна перейти до огляду індексу[25]. Коли всі дані завантажено й оброблено, останнім кроком є виконання кількох тестових пошуків, щоб підтвердити, що Meilisearch працює належним чином. Після налаштування Meilisearch потрібно встановити плагін на бекенді командою yarn add medusa-plugin-meilisearch та додати змінні до середовища серверної частини Medusa, які показані на рис. 4.5:

```
MEILISEARCH_HOST=<YOUR_MEILISEARCH_HOST>
MEILISEARCH_API_KEY=<YOUR_MASTER_KEY>
```

Рис. 4.5. Змінні серверної частини

Де:

<YOUR_MEILISEARCH_HOST> — хост екземпляра MeiliSearch. За замовчуванням, якщо MeiliSearch встановлено локально, хостом є http://127.0.0.1:7700.

<YOUR_MASTER_KEY> — головний ключ екземпляра MeiliSearch.

Після цього у medusa-config.js потрібно додати наступний елемент у масив плагінів, рис 4.6:

```
const plugins = [
  // ...
  {
    resolve: `medusa-plugin-meilisearch`,
    options: {
      config: {
        host: process.env.MEILISEARCH_HOST,
        apiKey: process.env.MEILISEARCH_API_KEY,
      },
      settings: {
        indexName: {
          indexSettings: {
            searchableAttributes,
            displayedAttributes,
          },
          primaryKey,
          transformer: (product) => ({
            id: product.id,
            // other attributes...
          }),
        },
      },
    },
  },
];
```

Рис. 4.6. Масив плагінів

Де:

- **indexName**: назва індексу для створення в MeiliSearch. Наприклад, продукти. Його значенням є об’єкт, що містить такі властивості:
- **indexSettings**: об’єкт, який містить такі властивості:
  - **searchableAttributes**: масив рядків, що вказує на атрибути сутності продукту, за якими можна шукати.
  - **displayedAttributes**: масив рядків, що вказує на атрибути об’єкта продукту, які мають відображатися в результатах пошуку.
- **primaryKey**: додатковий рядок, що вказує, яка властивість діє як первинний ключ документа. Він використовується для забезпечення виконання унікальних документів в індексі. Значенням за умовчанням є id.

- **transformer**: додаткова функція, яка приймає продукт як параметр і повертає об’єкт для індексації. Це дає більше контролю над тим, що індексується. Наприклад, можна додати деталі, пов’язані з варіантами чи спеціальними зв’язками, або відфільтрувати певні продукти.

Використовуючи цю структуру параметрів індексу, можна додати більше одного індексу.

4.1.3 Встановлення та налаштування сервісу AWS S3

S3, або Amazon Simple Storage Service, є об'єктним сховищем, що надається Amazon Web Services (AWS). Воно використовується для зберігання та відтворення будь-якої кількості даних у будь-якому місці в Інтернеті. S3 пропонує масштабованість, безпеку даних, високу доступність та простоту управління. У контексті Medusa, яка є гнучкою та відкритою платформою для електронної комерції, плагін S3 використовується для інтеграції з Amazon S3 як рішення для зберігання. Це може включати зберігання зображень продуктів, медіафайлів, документації та інших об'єктів, пов'язаних з електронною комерцією[25-27].

Інтеграція S3 з Medusa backend дозволяє використовувати переваги надійного та масштабованого сховища Amazon для управління активами електронної комерції. Це забезпечує ефективне управління великими обсягами даних, оптимізує швидкість завантаження сторінок та покращує загальний досвід користувача[27].

Встановлення плагіна S3 на Medusa backend включає додавання плагіна до конфігурації Medusa та налаштування відповідних параметрів для забезпечення з'єднання з обліковим записом Amazon S3. Це дозволяє Medusa автоматично зберігати та отримувати дані з S3, забезпечуючи ефективне та безпечне управління активами [27]. Для підключення S3 для початку потрібно в консолі AWS знайти S3 у полі пошуку вгорі та створити сегмент як показано на рис. 4.7:

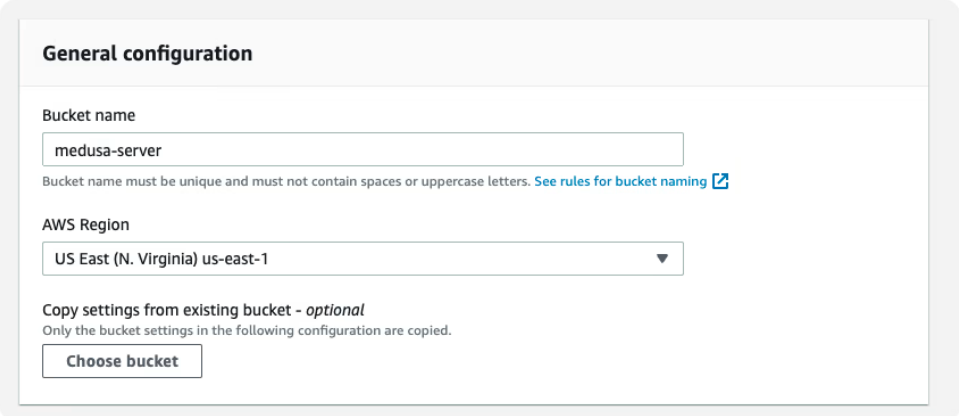


Рис. 4.7. Створення сегменту

Після чого розблокувати публічний доступ та перейти на сторінку сегменту. Далі треба налаштувати політику “bucket” та в аканутіі додати політику AmazonS3FullAccessCopy це потрібно для налаштування середовища в якому будуть зберігатись файли та для того, щоб отримати ключі доступу для інтеграції плагіну у Medusa[27]. Для того щоб встановити плагін потрібно виповнити команду yarn add medusa-file-s3 та у файлі medusa-config.js налаштувати плагін як показано на рис. 4.8:

```
const plugins = [
  // ...
  {
    resolve: `medusa-file-s3`,
    options: {
        s3_url: process.env.S3_URL,
        bucket: process.env.S3_BUCKET,
        region: process.env.S3_REGION,
        access_key_id: process.env.S3_ACCESS_KEY_ID,
        secret_access_key: process.env.S3_SECRET_ACCESS_KEY,
        cache_control: process.env.S3_CACHE_CONTROL,
        // optional
        download_file_duration:
          process.env.S3_DOWNLOAD_FILE_DURATION,
        prefix: process.env.S3_PREFIX,
    },
  },
]
```

Рис. 4.8. Налаштування плагіну

Де:

- **s3_url**(обов’язково) – це рядок, що вказує URL-адресу сегмента. Він знаходиться у формі https://<BUCKET_NAME>.s3.<REGION>.amazonaws.com, де <BUCKET_NAME>— назва сегмента, а — <REGION>регіон, у якому створено сегмент.
- **bucket**(обов’язково) — це рядок, що вказує на назву створеного сегмента.
- **region**(обов’язково) – це рядок, що вказує код регіону сегмента. Наприклад, us-east-1.
- **access_key_id**(обов’язково) – це рядок, що вказує ідентифікатор ключа доступу, який був створений для користувача.
- **secret_access_key**(обов’язково) – це рядок із зазначенням секретного ключа доступу, який був створений для користувача.
- **cache_control(**за замовчуванням: max-age=31536000) — це рядок, що вказує значення для кешування об’єктів у веб-браузері або мережі CDN. Наприклад: кешувати об’єкт на 10 годин,max-age=36000
- **download_file_duration**(необов’язковий) – це рядок, що вказує термін дії URL-адреси для завантаження.
- **prefix**(необов’язковий) – це рядок, що вказує на префікс, який слід застосувати до збережених імен файлів[28].

Обов’язково треба встановити ці значення у змінних середовища які продемонстровано на рис. 4.9:

```
S3_URL=<YOUR_BUCKET_URL>
S3_BUCKET=<YOUR_BUCKET_NAME>
S3_REGION=<YOUR_BUCKET_REGION>
S3_ACCESS_KEY_ID=<YOUR_ACCESS_KEY_ID>
S3_SECRET_ACCESS_KEY=<YOUR_SECRET_ACCESS_KEY>
S3_CACHE_CONTROL=<YOUR_CACHE_CONTROL>
S3_DOWNLOAD_FILE_DURATION=<YOUR_DOWNLOAD_FILE_DURATION_AMOUNT>
S3_PREFIX=<YOUR_BUCKET_PREFIX>
```

Рис. 4.9. Значення змінних

4.2 Розгортання веб-проекту у Docker контейнерах та на хостингу
4.2.1 Розгортання проекту у Docker контейнерах

Для розгортання проекту з Medusa JS, Next.js storefront, MeiliSearch, використовуючи Docker і docker-compose, можна дотримуватися наступних кроків:

- Підготовка Dockerfile для кожного сервісу, рис. 4.10 - 4.11,
- Створення docker-compose.yml,
- Розгортання проекту.

1. Підготовка Dockerfile для кожного сервісу:
Створення Dockerfile для Medusa JS

- **Відкриття Текстового Редактора:**
  - Відкрити текстовий редактор для створення нового файлу.
- **Створення Dockerfile:**
  - Створити новий файл без розширення і назвати його **Dockerfile**. Це стандартна практика для файлів конфігурації Docker.
- **Визначення Базового Образу:**
  - На початку **Dockerfile** вказати базовий образ. Для Medusa JS, який використовує Node.js, використовується офіційний образ Node.js.
  - Приклад: **FROM node:latest**. Це вказує Docker використовувати останню версію образу Node.js.
- **Встановлення Робочої Директорії:**
  - Встановити робочу директорію всередині контейнера. Це директорія, де будуть зберігатися файли проекту.
  - Приклад: **WORKDIR /app**. Це створює та використовує директорію **/app** у контейнері.
- **Копіювання Файлів Проекту:**
  - Спочатку копіювати **package.json** та **package-lock.json** (якщо він існує) у робочу директорію контейнера. Це оптимізує процес встановлення залежностей.

- Приклад: **COPY package*.json ./**. Це копіює обидва файли **package.json** та **package-lock.json** у робочу директорію.

- **Встановлення Залежностей:**
  - Виконати команду **npm install** для встановлення залежностей, визначених у **package.json**.
  - Приклад: **RUN npm install**. Це встановлює всі необхідні залежності.
- **Копіювання Решти Файлів Проекту:**
  - Після встановлення залежностей, скопіювати всі інші файли проекту у робочу директорію контейнера.
  - Приклад: **COPY . .**. Це копіює всі файли з поточної директорії на локальному комп'ютері у робочу директорію контейнера.
- **Вказівка Команди для Запуску:**
  - Вказати команду, яка буде виконуватися при запуску контейнера. Для Medusa JS це зазвичай команда **npm start**.
  - Приклад: **CMD ["npm", "start"]**. Це вказує Docker запускати Medusa JS, використовуючи команду **npm start**.

Приклад Завершеного Dockerfile для Medusa JS,[28]

```
1 FROM node:latest
2 WORKDIR /app
3 COPY package*.json ./
4 RUN npm install
5 COPY . .
6 CMD ["npm", "start"]
```

Рис. 4.10. Завершений Dockerfile

Збереження та Використання Dockerfile

- Зберегти файл **Dockerfile** у кореневій директорії Medusa JS.
- Для створення образу Docker використовувати цей **Dockerfile**. Команда для створення образу зазвичай виглядає як **docker build -t medusa-js-app .**, виконана у директорії, де знаходиться **Dockerfile**

Створення Dockerfile для Next.js Storefront

- **Відкриття Текстового Редактора:**
  - Відкрити текстовий редактор для створення нового файлу.
- **Створення Dockerfile:**
  - Створити новий файл без розширення і назвати його **Dockerfile**.
- **Визначення Базового Образу:**
  - На початку **Dockerfile** вказати базовий образ. Для Next.js зазвичай використовується офіційний образ Node.js.
  - Приклад: **FROM node:latest**
- **Встановлення Робочої Директорії:**
  - Встановити робочу директорію всередині контейнера.
  - Приклад: **WORKDIR /app**
- **Копіювання Файлів Проекту:**
  - Спочатку копіювати **package.json** та **package-lock.json** у робочу директорію.
  - Приклад: **COPY package*.json ./**
- **Встановлення Залежностей:**
  - Виконати команду **npm install** для встановлення залежностей.
  - Приклад: **RUN npm install**
- **Копіювання Решти Файлів Проекту:**
  - Після встановлення залежностей, скопіювати всі інші файли проекту.
  - Приклад: **COPY . .**
- **Побудова Проекту:**
  - Виконати команду **npm run build** для побудови проекту.
  - Приклад: **RUN npm run build**
- **Вказівка Команди для Запуску:**
  - Вказати команду для запуску контейнера.
  - Приклад: **CMD ["npm", "start"]**

Приклад Завершеного Dockerfile для Next.js Storefront[28]

```
FROM node:latest
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
RUN npm run build
CMD ["npm", "start"]
```

Рис. 4.11. Завершений Dockerfile

Збереження та Використання Dockerfile

- Зберегти файл **Dockerfile** у кореневій директорії Next.js Storefront.
- Для створення образу Docker використовувати цей **Dockerfile**. Команда для створення образу зазвичай виглядає як **docker build -t nextjs-storefront .**, виконана у директорії, де знаходиться **Dockerfile**.

2. Розгортання MeiliSearch з використанням офіційного Docker образу

Для розгортання MeiliSearch з використанням Docker, не потрібно створювати власний **Dockerfile**, оскільки можна використовувати офіційний Docker образ MeiliSearch.

- **Запуск MeiliSearch з Docker:**
    - Використовувати команду **docker run** для запуску MeiliSearch з офіційного образу. Команда створить і запустить контейнер з MeiliSearch.

docker run -it --rm -p 7700:7700 getmeili/meilisearch:latest

Ця команда робить наступне:

- **-it** запускає контейнер у інтерактивному режимі.
- **--rm** автоматично видаляє контейнер після його зупинки.
- **-p 7700:7700** відображає порт 7700 з контейнера на порт 7700 хост-системи.
- **getmeili/meilisearch:latest** вказує на використання останньої версії MeiliSearch.

- **Перевірка Роботи MeiliSearch:**
    - Після запуску контейнера, MeiliSearch буде доступний на порту 7700 хост-системи. Можна перевірити його, відкривши http://localhost:7700 у веб-браузері.
- **Використання MeiliSearch:**
    - Тепер можна використовувати MeiliSearch для індексації та пошуку даних. Для взаємодії з MeiliSearch можна використовувати HTTP запити або одну з бібліотек клієнтів, доступних для різних мов програмування.

Якщо використовується **docker-compose** для управління багатоконтейнерними Docker додатками, можна інтегрувати MeiliSearch у **docker-compose.yml** файл як показано на рис. 4.12:

```
version: '3'
services:
  meilisearch:
    image: getmeili/meilisearch:latest
    ports:
      - "7700:7700"
```

Рис. 4.12. Інтеграція MeiliSearch у docker-compose.yml

Цей фрагмент **docker-compose.yml** файлу створює сервіс **meilisearch**, який використовує останню версію MeiliSearch і відображає порт 7700. Використання офіційного Docker образу MeiliSearch спрощує процес розгортання та забезпечує швидке налаштування пошукового сервера[28].

Створення docker-compose.yml:

- **Створення Файлу docker-compose.yml:**
    - Відкрити текстовий редактор і створити новий файл.
    - Зберегти файл як **docker-compose.yml** у кореневій директорії проекту.
- **Визначення Версії docker-compose:**
    - На початку файлу вказати версію docker-compose. Наприклад, **version: '3.8'**.

- **Конфігурація Сервісів:**
    - Визначити кожен сервіс (Medusa JS, Next.js, MeiliSearch, PostgreSQL) у секції **services**.
- **Medusa JS:**
    - Вказати шлях до директорії з **Dockerfile** для Medusa JS у параметрі **build**.
    - Відкрити порти для зовнішнього доступу.
    - Встановити змінні середовища, наприклад, **DATABASE_URL**.
    - Вказати залежності від інших сервісів, наприклад, **depends_on: - postgres**.
- **Next.js:**
    - Аналогічно вказати шлях до **Dockerfile** для Next.js.
    - Відкрити порти.
    - Вказати залежність від Medusa JS.
- **MeiliSearch:**
    - Використати офіційний образ MeiliSearch.
    - Відкрити порти.
    - Визначити томи для зберігання даних.
- **PostgreSQL:**
    - Використати офіційний образ PostgreSQL.
    - Встановити змінні середовища для бази даних.
    - Визначити томи для зберігання даних бази.
- **Визначення Томів:**
    - В секції **volumes** визначити томи, які будуть використовуватися сервісами.

Приклад Завершеного docker-compose.yml на рис. 4.13:

```yaml
version: '3.8'
services:
  medusa:
    build: ./path_to_medusa
    ports:
      - "9000:9000"
    environment:
      - DATABASE_URL=postgres://...
    depends_on:
      - postgres

  nextjs:
    build: ./path_to_nextjs
    ports:
      - "3000:3000"
    depends_on:
      - medusa

  meilisearch:
    image: getmeili/meilisearch:latest
    ports:
      - "7700:7700"
    volumes:
      - meilisearch_data:/data

  postgres:
    image: postgres:latest
    environment:
      POSTGRES_DB: medusadb
      POSTGRES_USER: user
      POSTGRES_PASSWORD: password
    volumes:
      - postgres_data:/var/lib/postgresql/data

volumes:
  meilisearch_data:
  postgres_data:
```

Рис. 4.13. docker-compose.yml файл

Цей файл **docker-compose.yml** описує мульти-контейнерну конфігурацію, яка включає Medusa JS, Next.js, MeiliSearch та PostgreSQL, з відповідними налаштуваннями портів, залежностей та зберігання даних[28].

3. Розгортання проекту

Після успішного тестування та виправлення помилок, наступним кроком є розгортання сервісів на виробничому сервері. Ось детальні кроки для цього процесу:

- **Вибір Хостингу:**
  - Вибрати хостинговий сервіс, який підтримує Docker та docker-compose. Це може бути хмарний провайдер (наприклад, AWS, Google Cloud, Azure) або власний сервер.
- **Перенесення Файлів на Сервер:**

- Завантажити файли проекту, включаючи **Dockerfile** та **docker-compose.yml**, на виробничий сервер. Це можна зробити через Git, FTP або інші методи передачі файлів.

- **Налаштування Сервера:**
  - Встановити та налаштувати Docker та docker-compose на сервері, якщо вони ще не встановлені.
  - Переконатися, що сервер має достатньо ресурсів (CPU, пам'ять, дисковий простір) для запуску всіх контейнерів.
- **Налаштування Безпеки:**
  - Налаштувати брандмауер та правила безпеки для сервера. Відкрити лише необхідні порти (наприклад, 80, 443 для веб-трафіку, та інші порти, які використовуються сервісами).
  - Налаштувати SSL/TLS для захисту HTTP трафіку. Це можна зробити, використовуючи Let's Encrypt для отримання безкоштовних SSL-сертифікатів.
- **Резервне Копіювання:**
  - Налаштувати регулярне резервне копіювання важливих даних. Це може включати бази даних, конфігураційні файли та інші важливі дані.
  - Використовувати засоби хмарного зберігання або інші рішення для зберігання резервних копій.
- **Запуск Контейнерів:**
  - Запустити контейнери за допомогою команди **docker-compose up -d** на сервері. Це створить та запустить всі визначені в **docker-compose.yml** сервіси.
- **Моніторинг та Управління:**
  - Налаштувати систему моніторингу для відстеження стану сервера та контейнерів. Можна використовувати інструменти, як Prometheus, Grafana або інші рішення моніторингу.
  - Регулярно перевіряти логи системи та контейнерів для виявлення та вирішення потенційних проблем.

- **Оновлення та Підтримка:**
    - Регулярно оновлювати всі компоненти системи, включаючи Docker образи, для забезпечення безпеки та стабільності.
    - Відслідковувати оновлення залежностей та вносити необхідні зміни в конфігурацію.

Розгортання на виробничому сервері вимагає ретельного планування та уваги до деталей, особливо щодо безпеки та резервного копіювання. Важливо регулярно перевіряти та підтримувати інфраструктуру після розгортання.

4.3 Налаштування Nginx проксі-сервера

Nginx є одним з найпопулярніших веб-серверів та обернених проксі-серверів, відомим своєю високою продуктивністю, надійністю та гнучкістю. Цей легкий сервер використовується для різноманітних завдань: від обслуговування статичного контенту та медіа-стрімінгу до балансування навантаження та кешування для веб-додатків. Його асинхронна архітектура дозволяє ефективно обробляти тисячі одночасних запитів, що робить його ідеальним вибором для високонавантажених веб-сайтів та додатків. Nginx також використовується як обернений проксі-сервер для перенаправлення та управління веб-трафіком, а також для SSL/TLS термінації, забезпечуючи безпеку з'єднань[28].

Завдяки своїй гнучкій системі конфігурації, Nginx може бути налаштований для вирішення широкого спектру завдань, включаючи захист від DDoS-атак та оптимізацію продуктивності через компресію та інші техніки. Його модульна структура дозволяє розширювати функціональність за допомогою різних модулів, що робить Nginx вибором номер один для багатьох системних адміністраторів та розробників веб-додатків[29].

Для налаштування Nginx як оберненого проксі-сервера, що управляє трафіком до контейнерів, та налаштування SSL/TLS, потрібно виконати наступні кроки:

Налаштування Nginx

- **Встановлення Nginx:**
  - Встановити Nginx на сервер. Це можна зробити за допомогою менеджера пакетів системи, наприклад, **apt** для Ubuntu (**sudo apt install nginx**).
- **Конфігурація Nginx:**
  - Створити або модифікувати конфігураційні файли Nginx, які знаходяться у директорії **/etc/nginx/sites-available/**. Зазвичай для кожного сайту або сервісу створюється окремий конфігураційний файл.
  - Наприклад, для перенаправлення запитів до Medusa та Next.js, можна налаштувати конфігурацію таким чином:

На рис. 4.14, всі запити до **example.com/api/** перенаправляються на Medusa, а інші запити - на Next.js[29].

```
server {
    listen 80;
    server_name example.com;

    location /api/ {
        proxy_pass http://medusa-container:9000;
    }

    location / {
        proxy_pass http://nextjs-container:3000;
    }
}
```

Рис. 4.14. Перенаправлення запитів

- **Активація Конфігурації:**
  - Створити символічне посилання на файл конфігурації у директорії **/etc/nginx/sites-enabled/**.
  - Перезавантажити Nginx для застосування змін (**sudo systemctl reload nginx**).

Налаштування SSL/TLS з Let's Encrypt

- **Встановлення Certbot:**

- Встановити Certbot, клієнт Let's Encrypt, який автоматизує процес отримання та встановлення SSL-сертифікатів.
- Для Ubuntu це можна зробити за допомогою команди **sudo apt install certbot python3-certbot-nginx**.

- **Отримання SSL-Сертифіката:**
  - Запустити Certbot для автоматичного отримання та налаштування SSL-сертифіката для Nginx. Наприклад, **sudo certbot --nginx -d example.com**.
  - Слідувати інструкціям Certbot для завершення процесу.
- Автоматичне Оновлення Сертифікатів:
  - Certbot автоматично створює задачу cron або systemd timer для оновлення сертифікатів. Переконатися, що ця задача активна, щоб сертифікати автоматично оновлювалися перед закінченням їх терміну дії.

Висновки до розділу

У даному розділі дисертації детально розглянуто процес створення та розміщення веб-сайту електронної комерції на базі платформи Medusa JS. Викладено інтеграцію фронтенду, реалізованого на Next.js, та бекенду на Medusa JS, що дозволило ефективно поєднати функціональність та переваги обох технологій для створення гнучкої та масштабованої системи[30].

Далі представлено процес контейнеризації додатку за допомогою Docker та Docker Compose, що сприяло спрощенню розгортання та управління компонентами проекту. Такий підхід забезпечив легкість масштабування та гнучкість управління ресурсами[30].

Окрім того, висвітлено налаштування Nginx як оберненого проксі-сервера, що відіграло ключову роль у оптимізації маршрутизації трафіку, забезпеченні безпеки через SSL-шифрування та підвищенні загальної продуктивності веб-додатку.

Таким чином, у розділі надано комплексний огляд використання сучасних фреймворків та інструментів для створення ефективної інфраструктури веб-сайту

електронної комерції, демонструючи їх важливість для успішного розгортання та функціонування проекту[29-30].

ВИСНОВКИ

Ця робота представляє собою глибоке та всеосяжне дослідження, орієнтоване на розробку веб-сайту електронної комерції з використанням передових технологій. Ретельно продумана структура роботи включає широкий спектр аспектів - від теоретичного аналізу та вибору технологій до практичної реалізації та оцінки ефективності запроваджених рішень. Особлива увага приділяється детальному розгляду Medusa.js, як головного інструменту для бекенд-розробки, та його інтеграції з фронтенд-фреймворками, зокрема Next.js.

Значний обсяг роботи присвячено тонкощам вибору та інтеграції технологій, з акцентом на їхній взаємодії, що забезпечує високу продуктивність та зручність користування. Розглядається спектр інструментів та практик, що забезпечують гнучкість та масштабованість рішень, а також такі ключові аспекти, як безпека, стабільність, та забезпечення якісного користувацького досвіду.

Важливою частиною роботи є аналіз процесу розробки, включаючи використання контейнеризації за допомогою Docker для забезпечення однорідності середовищ розробки та виробництва. Обговорюються методики тестування, практики безпеки та стратегії оптимізації продуктивності, зокрема використання Nginx для управління веб-трафіком та забезпечення ефективного кешування.

В контексті розгортання та підтримки, робота надає детальний огляд різних стратегій та підходів для забезпечення стабільної та надійної роботи веб-сайту в продакшн-середовищі. Розглядається, як важливо адаптувати сайт під поточні та майбутні потреби користувачів та реагувати на зміни ринкових умов.

Значний внесок цієї роботи полягає у всебічному розумінні того, як сучасні технології можуть бути інтегровані для створення ефективних, безпечних та користувацьки привабливих рішень в області електронної комерції. Робота демонструє, що успіх проекту залежить не лише від технологій, а й від уважного

підходу до процесу розробки, включаючи планування, тестування, моніторинг та адаптацію.

У підсумку, ця робота являє собою цінний внесок у розвиток та використання передових технологій у сфері електронної комерції. Вона висвітлює ключові аспекти, що мають бути враховані при розробці веб-сайтів, і надає важливі напрямки для майбутніх досліджень та розробок в цій швидко розвиваючійся галузі.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Огляд платформи Medusa JS для електронної комерції. URL: superbwebsitebuilders.com/uk/oglyad-cms-wordpress/ (дата звернення: September 10, 2023).
2. Розробка інтерфейсу веб-магазину з використанням Next.js. URL: hostiq.ua/wiki/ukr/wordpress-guide/ (дата звернення: September 10, 2023).
3. Оптимізація SEO для електронної комерції на Medusa JS. URL: seo-evolution.com.ua (дата звернення: September 10, 2023).
4. Аналіз продуктивності інтернет-магазинів на Medusa JS. URL: infosoft.ua/ua/perevahy-i-nedoliky-wordpress (дата звернення: September 10, 2023).
5. Вибір CMS для електронної комерції: Medusa JS проти WordPress. URL: esfirum.com/blog/pros-and-cons-cms-wordpress/ (дата звернення: September 11, 2023).
6. Порівняльний аналіз CMS для електронної комерції. URL: atriples.com.ua (дата звернення: September 13, 2023).
7. Інтеграція Medusa JS з іншими CMS. URL: uk.wikipedia.org/wiki/Drupal (дата звернення: September 13, 2023).
8. Початкова настройка Medusa JS для новачків. URL: original.drupal.ua/begin (дата звернення: September 13, 2023).
9. Використання Medusa JS як безголового CMS. URL: www.weareaccess.co.uk (дата звернення: September 16, 2023).
10. Конфігурація Medusa JS для генерації статичних сайтів. URL: www.gatsbyjs.com/plugins/gatsby-theme-drupal/ (дата звернення: September 17, 2023).
11. Інтеграція Medusa JS з Gatsby для створення магазинів. URL: medusajs.com/blog/gatsby-ecommerce/ (дата звернення: September 18, 2023).
12. Оптимізація Medusa JS для використання з Drupal. URL: www.optasy.com (дата звернення: September 20, 2023).

13. Налаштування адміністративного інтерфейсу Medusa JS. URL: github.com/medusajs/nextjs-theme-medusa-admin (дата звернення: September 20, 2023).
14. Покращення UI плагіна Medusa JS. URL: medium.com (дата звернення: September 20, 2023).
15. Додавання панелі управління Medusa JS. URL: dev.to/dailydevtips1/adding-the-medusa-admin-dashboard-5bh9 (дата звернення: September 20, 2023).
16. Створення експрес-проекту з Medusa JS і Gatsby. URL: github.com/medusajs/medusa-express-gatsby (дата звернення: September 20, 2023).
17. Використання Medusa JS з Contentful. URL: www.contentful.com (дата звернення: September 21, 2023).
18. Інтеграція Medusa JS і Contentful. URL: www.gatsbyjs.com/solutions/contentful/ (дата звернення: September 21, 2023).
19. Огляд можливостей Next.js. URL: dou.ua/lenta/articles/nextjs-guide/ (дата звернення: September 26, 2023).
20. Швидкий початок роботи з Gatsby і Contentful. URL: www.gatsbyjs.com (дата звернення: September 26, 2023).
21. JAMstack: створення блогу з Gatsby і Contentful. URL: dou.ua/lenta/articles/creating-blog-with-jamstack/ (дата звернення: September 26, 2023).
22. Корисні поради по роботі з Contentful і Gatsby. URL: www.gatsbyjs.com (дата звернення: September 26, 2023).
23. Інтеграція WordPress з Medusa JS і Next.js. URL: github.com/websupergirl/hacktoberfest2022-frontend (дата звернення: October 10, 2023).
24. Джерело даних Drupal для Gatsby. URL: www.gatsbyjs.com/plugins/gatsby-source-drupal/ (дата звернення: October 12, 2023).
25. Готовий стартовий набір Gatsby для Medusa JS. URL: github.com/medusajs/gatsby-starter-medusa (дата звернення: October 13, 2023).

26. Швидкий старт з GatsbyJS і Contentful. URL: www.contentful.com (дата звернення: October 13, 2023).
27. Використання Drupal як безголового CMS з Gatsby.js. URL: www.weareaccess.co.uk (дата звернення: October 20, 2023).
28. Використання даних з Drupal у Gatsby. URL: www.gatsbyjs.com (дата звернення: October 23, 2023).
29. Створення інтернет-магазину з Next.js і Medusa JS. URL: medusajs.com/blog/nextjs-ecommerce/ (дата звернення: October 25, 2023).
30. Конфігурація Nginx і Docker Compose. URL: www.optasy.com (дата звернення: October 25, 2023).

Додаток А

SUMMURY

SUMMURY

This report comprehensively explores the development and deployment of an e-commerce website leveraging contemporary technologies like Medusa JS, Next.js, Docker, and Nginx. It covers key aspects of the project, from platform selection to server environment setup and service integration. A significant focus is placed on ensuring seamless front-end and back-end interaction using flexible and scalable solutions.

A crucial part of the work is the adoption of containerization with Docker, enhancing project deployment and management. Additionally, configuring Nginx as a reverse proxy server is meticulously detailed, optimizing load processes and security measures. The integration of auxiliary services such as AWS S3 for data storage and MeiliSearch for search functionality is also explored.

The report delves into an in-depth analysis and application of best practices in web application development, showcasing both technical proficiency and a strategic approach to solving e-commerce challenges. This work exemplifies a holistic approach to modern web development, ensuring the resilience, scalability, and efficiency of the project. It demonstrates a well-rounded expertise in web technologies and a keen understanding of the e-commerce domain, providing valuable insights into contemporary web development practices.

Furthermore, the document discusses the strategic choices made in the tech stack, emphasizing the benefits and trade-offs of each technology. It details the integration process of various APIs and third-party services, highlighting the importance of creating a robust and user-friendly e-commerce platform. The report also addresses challenges faced during development, offering solutions and workarounds that contribute to the knowledge base of web development.

In conclusion, this report is a comprehensive guide that exemplifies modern web development practices for e-commerce platforms. It serves as a valuable resource for developers and businesses alike, looking to understand and implement advanced web technologies in creating efficient, scalable, and user-centric e-commerce solutions.

Додаток Б

Частковий програмний код веб-сайту з функціональною адміністративноюпанеллю

page.tsx

```
import { getCollectionsList } from "@lib/data"
import FeaturedProducts from "@modules/home/components/featured-
products"
import Hero from "@modules/home/components/hero"
import SkeletonHomepageProducts from
"@modules/skeletons/components/skeleton-homepage-products"
import { Metadata } from "next"
import { Suspense } from "react"

export const metadata: Metadata = {
  title: "Ukrgiprotrans",
  description:
    "A leading provider of premium engineering and architectural
drawings. Specializing in detailed, high-quality designs for a range
of projects. Explore our extensive collection of blueprints perfect
for construction, renovation, and development. Trusted by
professionals for accuracy and excellence.",
  openGraph: {
    images: '/og-image.png',
  },
}

export default async function Home() {
  const { collections, count } = await getCollectionsList(0, 3)

  return (
    <>
      <Hero />
      <Suspense fallback={<SkeletonHomepageProducts count={count} />}>
        <FeaturedProducts collections={collections} />
      </Suspense>
    </>
  )
```

```
}
```

global.css

```
@import "tailwindcss/base";
@import "tailwindcss/components";
@import "tailwindcss/utilities";

@layer utilities {

  /* Chrome, Safari and Opera */
  .no-scrollbar::-webkit-scrollbar {
    display: none;
  }

  .no-scrollbar::-webkit-scrollbar-track {
    background-color: transparent;
  }

  .no-scrollbar {
    -ms-overflow-style: none;
    /* IE and Edge */
    scrollbar-width: none;
    /* Firefox */
  }

  input:focus~label,
  input:not(:placeholder-shown)~label {
    @apply -translate-y-2 text-xsmall-regular;
  }

  input:focus~label {
    @apply left-0;
  }
```

```
  input:-webkit-autofill,
  input:-webkit-autofill:hover,
  input:-webkit-autofill:focus,
  textarea:-webkit-autofill,
  textarea:-webkit-autofill:hover,
  textarea:-webkit-autofill:focus,
  select:-webkit-autofill,
  select:-webkit-autofill:hover,
  select:-webkit-autofill:focus {
    border: 1px solid #212121;
    -webkit-text-fill-color: #212121;
    -webkit-box-shadow: 0 0 0px 1000px #fff inset;
    transition: background-color 5000s ease-in-out 0s;
  }

  input[type="search"]::-webkit-search-decoration,
  input[type="search"]::-webkit-search-cancel-button,
  input[type="search"]::-webkit-search-results-button,
  input[type="search"]::-webkit-search-results-decoration {
    -webkit-appearance: none;
  }
}

@layer components {
  .content-container {
    @apply max-w-[1440px] w-full mx-auto px-4;
  }

  .contrast-btn {
    @apply px-4 py-2 border border-black rounded-full hover:bg-black
hover:text-white transition-colors duration-200 ease-in;
  }

  .text-xsmall-regular {
    @apply text-[10px] leading-4 font-normal;
```

```
}

.text-small-regular {
  @apply text-xs leading-5 font-normal;
}

.text-small-semi {
  @apply text-xs leading-5 font-semibold;
}

.text-base-regular {
  @apply text-sm leading-6 font-normal;
}

.text-base-semi {
  @apply text-sm leading-6 font-semibold;
}

.text-large-regular {
  @apply text-base leading-6 font-normal;
}

.text-large-semi {
  @apply text-base leading-6 font-semibold;
}

.text-xl-regular {
  @apply text-2xl leading-[36px] font-normal;
}

.text-xl-semi {
  @apply text-2xl leading-[36px] font-semibold;
}

.text-2xl-regular {
```

```
    @apply text-[30px] leading-[48px] font-normal;
  }

  .text-2xl-semi {
    @apply text-[30px] leading-[48px] font-semibold;
  }

  .text-3xl-regular {
    @apply text-[32px] leading-[44px] font-normal;
  }

  .text-3xl-semi {
    @apply text-[32px] leading-[44px] font-semibold;
  }
}
```

src/app/(main)/products/[handle]/page.tsx

```
import { getProductByHandle } from "@lib/data"
import ProductTemplate from "@modules/products/templates"
import SkeletonProductPage from
"@modules/skeletons/templates/skeleton-product-page"
import { Metadata } from "next"
import { notFound } from "next/navigation"

type Props = {
  params: { handle: string }
}

export async function generateMetadata({ params }: Props):
Promise<Metadata> {
  const data = await getProductByHandle(params.handle)

  const product = data.products[0]

  if (!product) {
```

```
    notFound()
  }

  return {
    title: `${product.title} | Ukrgiprotrans`,
    description: `${product.title}`,
    openGraph: {
      title: `${product.title} | Ukrgiprotrans`,
      description: `${product.title}`,
      images: product.thumbnail ? [product.thumbnail] : [],
    },
  }
}

export default async function ProductPage({ params }: Props) {
  const { products } = await
getProductByHandle(params.handle).catch((err) => {
    notFound()
  })

  return <ProductTemplate product={products[0]} />
}
```

src/app/(main)/collections/[handle]/page.tsx

```
import { getCollectionByHandle } from "@lib/data"
import CollectionTemplate from "@modules/collections/templates"
import { Metadata } from "next"
import { notFound } from "next/navigation"

type Props = {
  params: { handle: string }
}
```

```
export async function generateMetadata({ params }: Props):
Promise<Metadata> {
  const { collections } = await getCollectionByHandle(params.handle)

  const collection = collections[0]

  if (!collection) {
    notFound()
  }

  return {
    title: `${collection.title} | Ukrgiprotrans`,
    description: `${collection.title} collection`,
  }
}

export default async function CollectionPage({ params }: Props) {
  const { collections } = await getCollectionByHandle(params.handle)

  const collection = collections[0]

  return <CollectionTemplate collection={collection} />
}
```