МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Сумський державний університет

Факультет електроніки та інформаційних технологій

Кафедра інформаційних технологій

«До захисту допущено»

В.о. завідувача кафедри

_____ Світлана ВАЩЕНКО

____ _____ 2024 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня магістр

зі спеціальності 122 «Комп'ютерні науки»,

освітньо-професійної програми «Інформаційні технології проектування»

на тему: Вебсистема управління контентом з використанням Next.js

Здобувача групи __ІТ.м-33__ Днєстров Нікіта Сергійович_

(шифр групи) (прізвище, ім'я, по батькові)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

_____ Нікіта ДНЄСТРОВ

(підпис) (Ім'я та ПРІЗВИЩЕ здобувача)

Керівник професор кафедри ІТ, д.т.н., доцент Сергій ТИМЧУК _____

(посада, науковий ступінь, вчене звання, Ім'я та ПРІЗВИЩЕ) (підпис)

Суми – 2024

Сумський державний університет

Факультет електроніки та інформаційних технологій

Кафедра інформаційних технологій

Спеціальність 122 «Комп'ютерні науки»

Освітньо-професійна програма «Інформаційні технології проектування»

ЗАТВЕРДЖУЮ

В.о. завідувача кафедри ІТ

Світлана ВАЩЕНКО

«_____» _____ 2023 р.

ЗАВДАННЯ

на кваліфікаційну роботу магістра студентові

Днєстров Нікіта Сергійович

(прізвище, ім'я, по батькові)

1 Тема кваліфікаційної роботи Вебсистема управління контентом з використанням Next.js

затверджена наказом по університету від «08» листопада 2024 р. № 1249-VI

2 Термін здачі студентом кваліфікаційної роботи «_15_» ___грудня__ 2024 р.

3 Вхідні дані до кваліфікаційної роботи перелік вимог до вебсистеми управління контентом

4 Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) аналіз предметної області, постановка задачі та методи дослідження, проектування системи управління контентом, практична реалізація системи управління контентом

5 Перелік графічного матеріалу (з точним зазначенням обов'язкових слайдів презентації) Актуальність, постановка задачі, огляд фреймворків для побудови веб-застосунків, порівняльна таблиця фреймворків, задачі проєкту, функціональні вимоги до інформаційної системи ,

інструменти реалізації, структурно-функціональна модель,діаграма системи CMS ,моделювання варіантів використання ,діаграма послідовностей, практична реалізація, демонстрація інформаційної системи, тестування,висновки,

6. Консультанти випускної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

Дата видачі завдання _____.

Керівник _____

(підпис)

Завдання прийняв до виконання _____

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Аналіз та планування	19.08.24 – 23.08.24	
2	Розробка інфраструктури	26.08.24 – 06.09.24	
3	Розробка клієнтської частини	09.09.24 – 18.10.24	
4	Тестування та оптимізація	21.10.24 – 01.11.24	
6	Впровадження застосунку	04.11.24 – 11.11.24	

Магістрант _____ **Нікіта Днєстров**

Керівник роботи _____ **д.т.н., доц. Сергій ТИМЧУК**

АНОТАЦІЯ

Кваліфікаційна робота магістра на тему «Вебсистема управління контентом з використанням Next.js» є актуальною в умовах стрімкого розвитку електронної комерції де стає важливою розробка ефективних систем управління контентом, які можуть забезпечити зручність та швидкість адміністрування веб ресурсів. Робота складається зі вступу, чотирьох розділів, висновків, списку використаних джерел на 41 найменування, та додатків, загальним обсягом 82 сторінки, з яких 44 сторінки становить основний текст, 5 сторінок списку використаних джерел, 25 сторінок додатків.

Метою дослідження є підвищення ефективності процесу управління контентом у веб системах через створення та аналіз веб застосунку на основі сучасних технологій, таких як Next.js. Робота включає аналіз існуючих фреймворків для створення сучасних веб застосунків, розробку концептуальної моделі, проектування та реалізацію веб застосунків. Особлива увага приділяється SEO оптимізації та швидкості роботи таких веб проектів.

Одним із ключових результатів є проведення тестування за допомогою інструменту Lighthouse.

Щодо впровадження, розроблена підсистема може бути використана як основа створення клієнтських веб застосунків та адміністративних панелей.

Ключові слова: Next.js, серверний рендеринг, SSR, статична генерація, SSG, SEO оптимізація, REST API

ВСТУП

У сучасному світі електронна комерція займає важливе місце у сфері бізнесу та обслуговування клієнтів. Щороку кількість онлайн-платформ для продажу зростає, оскільки вони надають можливість швидкого доступу до товарів та послуг для користувачів. Це робить актуальним питання створення ефективних систем керування контентом (CMS), які дозволяють адмініструвати інтернет-магазини та забезпечувати зручний інтерфейс для кінцевих споживачів. У цьому контексті розробка веб-додатків на основі сучасних фреймворків і технологій стає ключовою складовою успішної реалізації таких проектів.

Актуальність дослідження

У сучасних умовах стрімкого розвитку електронної комерції важливою стає розробка ефективних систем управління контентом, які можуть забезпечити зручність та швидкість адміністрування веб ресурсів. Використання сучасних технологій, таких як Next.js, відкриває нові можливості для створення веб систем, що поєднують високу продуктивність, швидкий рендеринг та простоту інтеграції з бекендом. Такий підхід дозволяє підвищити якість обслуговування користувачів, зменшити час завантаження сторінок та покращити загальний користувацький досвід. Актуальність дослідження полягає у необхідності впровадження нових підходів до створення веб систем управління контентом, що забезпечують ефективну підтримку адміністраторів та інтерактивний досвід для кінцевих споживачів. [28]

Тема дослідження

Тема мого дослідження полягає у вдосконаленні системи управління контентом для онлайн-магазину взуття за допомогою сучасних технологій, таких як Next.js. Дослідження фокусується на створенні продуктивного, масштабованого та зручного у використанні вебзастосунку, який забезпечує ефективну підтримку адміністрування товарів і взаємодію користувачів з онлайн-магазином.

Мета дослідження

Метою дослідження є підвищення ефективності процесу управління контентом у веб системах через створення та аналіз веб застосунку на основі сучасних технологій, таких як Next.js.

Об’єкт дослідження

Об'єктом дослідження є процес управління контентом у веб системах, що забезпечує створення, редагування, відображення та зберігання інформації в контексті електронної комерції.

Предмет дослідження

Предметом дослідження є використання сучасних фреймворків та технологій, таких як Next.js, для вдосконалення процесу управління контентом у веб системах.

Гіпотеза дослідження

Гіпотеза дослідження полягає в тому, що використання Next.js у поєднанні з Node.js і SQLite для створення системи управління контентом дозволяє значно підвищити продуктивність, гнучкість та зручність у користуванні веб застосунку.

Задачі дослідження

Дослідження вимагає виконання таких задач:

- Дослідження технологій на яких базується фреймворк Next.js
- Порівняння Next.js з іншими фреймворками фреймворками. Визначити його переваги та недоліки
- Планування архітектури проекту із використанням Next.js
- Практична

Наукова новизна

Набув подальшого розвитку метод проектування вебсистеми управління контентом із застосуванням Next.js, який на відміну від існуючих поєднує серверний і клієнтський рендерінг, що дозволило суттєво підвищити швидкість завантаження сторінок та SEO-оптимізації.

Практичне значення

Практичне значення роботи полягає у створенні системи, яка дозволяє адміністраторам зручно управляти контентом, додаючи, редагуючи та видаляючи товари, а користувачам отримувати інтерактивний досвід перегляду і покупки товарів. Розробка цієї системи може бути використана як шаблон для створення інших подібних веб застосунків.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Ідентифікація проблеми

Дев'яності роки стали початком розвитку такого напряму як веб розробка. Почали з’являтися перші сайти які створювалися за допомогою базових технологій, як HTML і CSS. Це були статичні сайти які просто відображали необхідну інформацію, в яких була відсутність динамічної взаємодії з боку користувача. З розвитком інтернету компанії почали розглядати його як ефективний спосіб підвищити прибутки та залучити нових клієнтів. Тому з’явилася потреба у розробці більш складних веб додатків щоб задовольнити потреби бізнесу та надати більш широкі та корисні послуги для користувачів. [18]

Статичні сторінки довгий час були стандартом і будь яка зміна наповнення такої сторінки вимагала внесення змін з боку розробника. Підтримка таких застосунків була досить складною і чим більшим був сайт тим складніше його було підтримувати. З розвитком технологій, а саме таких як CGI, PHP та ASP, стало можливим робити перші динамічні сайти. Вони могли зв’язуватись з базами даних та відображати дані в залежності від запитів користувачів. [7] Але досі проблема виконання динамічної логіки на стороні клієнта не була вирішена.

З початком 2000 років, JavaScript стає дуже популярним із за його імплементації у популярні браузери. JavaScript надав можливість динамічно маніпулювати DOM системою та дозволив робити більш інтерактивні веб сторінки. Відсутність стандартів призвела до того що браузери різному імплементували JavaScript і тому вигляд однієї кодовою бази міг відрізнятися при запуску у двох різних браузерах. Цю проблему вирішила нова бібліотека, яка швидко завоювала популярність під назвою JQuery. Простота використання для маніпуляцій з DOM елементами та AJAX запитами, а також кросбраузерність зробила цю бібліотеку дефакто стандартом в індустрії для створення

динамічних веб застосунків. Але з новими можливостями з’являються нові потреби, тому з'явилась необхідність в чомусь більшому ніж просто бібліотека.[34]

AngularJS представлений компанією Goggle у 2010 році, став першим фреймворком для створення веб сайтів. Цей фреймворк представив ідею односторінкових додатків (SPA), яка полягає в тому що з серверу надсилається пустий HTML документ а увесь вміст сайту вже створювався за допомогою JavaScript на стороні браузера. Такий підхід став новим стандартом у розробці. Окрім цього SPA реалізував принцип двостороннього зв’язування даних, що дозволило спростити розробку складних застосунків, тому що кодова реалізація повністю відповідала вмісту сайту. Цей фреймворк відрізнявся складністю навчання і тому у Angualr почали з’являтися конкуренти які також почали реалізувати SPA підхід до розробки веб додатків.[21]

Одним з таких конкурентів стала розробка компанії Facebook у 2013 році під назвою React. Окрім реалізації SPA, React представив уявлення про додаток у вигляді компонентів, а також систему віртуального DOM що оптимізувати затримки у динамічному оновленні вмісту сайту та пришвидшило його рендеринг. Після громіздкого Angular, простий та гнучкий React дуже швидко зайняв лідерство у світі фреймворків для створення веб сайтів і досі займає перше місце за використанням. [7]

Іншим фреймворком який заслуговує уваги став Vue.js. Він був створений паралельно з React у 2014 році колишніми співробітниками Google. Цей фреймворк став спробою об’єднати найкращі риси Angular та React. Простота та легкість у використанні із інтуїтивним синтаксисом виділила Vue.js у окрему нішу для швидкої розробки менш складних веб додатків для менш досвідчених команд.[3]

Незважаючи на революційність SPA, у такого підходу до рендерингу був значний мінус. Чим більшим ставав проект тим більшою ставало навантаження на браузер. Команди розробки повинні були прикладати великих зусиль для оптимізації робити таких застосунків. Пустий HTML документ на початковому рендерингу погіршив SEO можливості у просуванні таких сайтів у пошукових системах. Так з’явилася ідея серверного рендерингу (SSR) яка дозволяла формувати статичний вміст на стороні

серверу. Обробка динамічного вмісту зі сторони клієнта стала можливою завдяки ідеї під назвою Hydration. Гідрація це клієнтський код який відправляється зі сторони серверу паралельно із статично згенерованою сторінкою HTML. Це дозволило послабити навантаження на браузер, покращити можливості у SEO оптимізації, а також зберегти можливості у створенні динамічних компонентів застосунку. Об’єднання цих підходів назвали мета-фреймворком і першою технологією яка це реалізувала був Next.js який вперше представили у 2016 році. [23]

Таким чином індустрія розробки веб додатків пройшла значний шлях від статичних веб сайтів, до високо оптимізованих динамічних веб застосунків. Ця еволюція супроводжувалася створенням нових підходів, таких як компонентний підхід і мета-фреймворки, які дозволяють розробникам створювати все більш продуктивні та інтерактивні рішення. Тепер інтеграція з базами даних, динамічний користувацький інтерфейс, оптимізація під різні браузери та пристрої тепер є невід'ємною частиною будь якого сучасного веб сайту. [2]

Зважаючи на швидку еволюцію веб технологій вимагає постійних змін у підходах до проектування таких систем. Архітектура повинна забезпечувати масштабованість, ефективність та зручність у використанні. На сьогоднішній день сформувалося три провідних мета-фреймворки: Next.js, Nuxt.js та Astro, між якими необхідно зробити вибір, щоб визначити найкраще рішення для розробки нашого додатку

Переваги Next.js

Next.js має безліч переваг, що зробило його дефакто лідером серед сучасних мета-фреймворків. По перше, ця технологія містить вбудовану підтримку SSR та Hydration. Що дозволить створення оптимізованих та динамічних веб сторінок. Реалізація цих підходів дозволяє клієнту отримати найкращій користувацький досвід завдяки швидкому завантаженню сторінок, що є дуже важливим для залучення користувачів та зниження показника відмов.[24]

Next.js був створений на базі React. Це дозволяє розробникам які з ним знайомі, дуже швидко стати продуктивними під час розробки. Людина яка знає React може

продовжити використовувати усі свої знання та бібліотеки і одночасно мати переваги Next.js.

Однією з важливих функцій Next.js також є так звані API-routes. Це функції які виконуються на стороні серверу під час запиту на сторінку. У цих API-routes можна робити запити до баз даних, або віддалених серверів до того як сторінка буде відправлена до клієнта. Після цього сторінка наповнюється даними з таких функцій за допомогою статичної генерації (SSG). Це забезпечує блискавичну швидкість завантаження. Користувач одразу отримує вже готову HTML сторінку з даними які були динамічно згенеровані на сервері. Це зменшує навантаження зі сторони клієнта а також дозволяє більш ефективно використовувати серверні ресурси. Завдяки цій функції Next.js є одним з найкращих виборів, як для створення контентних сайтів, блогів і сторінок з постійним або рідко змінюваним контентом, так і для сторінок які вимагають активних дій від користувача та динамічне відображення результатів таких дій. [22]

Next.js має велику спільноту розробників, які створюють бібліотеки та плагіни для цього фреймворку. Зазвичай вони мають чудову якість і детальну документацію. Вони можуть вирішувати проблеми з якими розробники часто зустрічаються. Вибір досить великий і може включати в себе як бібліотеки з готовими компонентами та стилізацією, так і бібліотека для реалізації окремого функціоналу, як наприклад авторизація або валідація введених даних.[8]

Таким чином Next.js досить часто обирають для створення комплексних а також високо оптимізованих веб застосунків. Цей фреймворк не відрізняються складністю, тому навіть невеликі або менш досвідченні команди можуть досить швидко розробити продукт який буде стояти на одному місці з іншими сучасними веб застосунками над якими працювали сотні спеціалістів.[1]

Недоліки Next.js

Не зважаючи на велику кількість функцій та їх високу якість, Next.js також має деякі недоліки, які необхідно враховувати перед тим як починати працювати над проектом.

React як база Next.js є як і плюсом так і мінусом. Синтаксис JSX, увправління станом через хуки, або virtual-DOM модель рендерингу можуть бути складними для засвоєння у обмежений час, як для новачків, так і для тих хто звик до роботи в інших фреймворках. Також в Next.js звісно не вийде використати ті бібліотеки які популярні серед інших фреймворків і тому може бути витрачений час на пошук альтернативи або створення власної бібліотеки для вирішення специфічних проблем.

Через те що Next.js не накладає обмежень у архітектурі може виникнути проблема у підтримці великих проектів у разі зростання кількості сторінок. Це вимагає акуратного планування та реалізації архітектури проекту. Налаштування маршрутизації та взаємодія складних компонентів вимагають додаткового досвіду та знань щоб не втратити рішень з оптимізації які пропонує Next.js “із коробки”. Це може стати проблемою для тих розробників, які звикли до строгих вимог до архітектури у фреймворках як Spring Boot у світі Java, де архітектура є чітко визначеною і реалізується через шаблони.[11]

Next.js також вимагає високо оптимізованої інфраструктури у випадках великих навантажень. SSR не дивлячись на всі свої плюси в оптимізації затримок та SEO, може збільшити час відповіді у випадку навантаженості серверу. Такі проблеми також можуть виникнути у випадку поганого налаштування серверу яка не заблокує DDOS атаку. У таких випадках щоб забезпечити стабільність роботи веб сайту потрібно імплементувати додаткові рішення для кешування та балансування навантаження, що може призвести до збільшення витрат на підтримку та створення більш складної серверної інфраструктури.

Через строгу архітектуру компонентів та системи маршрутизації, деякі специфічні потреби додатку у цих питаннях можуть потребувати створення комплексних власних рішень, або використання специфічних бібліотек. Це також може підвищити вартість та складність розробки у випадку такої необхідності.[35]

Навпаки у випадках потреби в створенні маленького сайту Next.js може бути занадто складним інтрументом. Для реалізації чогось простого від розробника може вимагатись розуміння усіх аспектів фреймворку. Така вимога може бути занадто

надмірною для початкових розробників, або простих проектів які повинні бути створеними у маленький проміжок часу. У таких випадках значно краще використовувати інші рішення які на цьому спеціалізуються, або просто використати простий HTML та CSS якщо динамічний контент не є потребою.[25]

У випадках необхідності в міграції з іншого фреймворку, Next.js показує себе дуже погано. Навіть якщо проект спочатку був написаний лише з використанням React, щоб додати Next.js можно витрати багато зусиль та часу. У будь яких випадках розробникам доведеться робити дуже значні зміни у кодову базу для підтримки маршрутизації, рендерингу та налаштування, що зазвичай Next.js робить сам під час ініціалізації проекту. [25]

Зрештою, Next.js має велику екосистему бібліотек та захоплених розробників які їх підтримують, але це не означає що ваша команда не зіткнеться з проблемою для якої немає готового рішення. Тому у таких випадках важливо мати досвідчену команду для розробки рішення спеціально під проблему з якою зіткнулися.[19]

Переваги Nuxt.js

Якщо Next.js створювався на основі React, то Nuxt.js ставив ті самі задачі але на основі Vue.js. Цей мета-фреймворк також використовується для розробки сучасних веб додатків і поєднує SSR із SPA підходами для створення оптимізованих та динамічних веб сторінок.

Одна із найважливіших переваг це інтеграція з Vue.js. Розробники які вже стали ефективними у використанні Vue.js, можуть продовжити його використати із додатковими перевагами серверного рендерингу. Схожий синтаксис та структура проекту полегшує адаптацію розробників та зменшує необхідний час для залучення нових розробників.

У Nuxt.js маршрутизація побудована на файловій структурі проекту. Це дозволяє одразу використовувати цю функцію без необхідності у додатковому налаштуванні проекта як це зроблено у багатьох інших фреймворках.

Інтеграція з Vue.js відкриває можливості для використання її екосистеми модулів.

У свободному доступі є значна кількість модулів для реалізації таких поширених функцій як: авторизація та аутентифікація, увправління станом, робота за асинхронними запитами, тощо.

Nuxt.js дозволяє розробникам також створювати прогресивні веб додатки (PWA). Це додатки які мають змогу працювати навіть тоді коли у користувача відсутній інтернет. Така функція покращує користувацький досвід та залучає користувачів з обмеженим доступом до інтернету. Цей мета-фреймворк надає зручний API для реалізування кешування статичного контенту, та збереження динамічних функцій для взаємодії з ним. [6]

Якщо є необхідність у створенні блогу, документації або будь якого іншого веб сайту де контент рідко змінюється, то Nuxt.js пропонує функцію статичної генерації. Такий підхід дозволяє попередньо генерувати сторінки під час компіляції проекту для розміщення на CDN. Завдяки цьому користувач може швидко отримувати кешований зміст сайту з відмінною швидкістю.[29]

Nuxt.js також пропонує ефективні інструменти для розробника. Такі функції як гаряче перезавантаження та автоматичне збіркування проекту дає змогу зручно тестувати застосунок у реальному часі, що прискорює процес розробки. Відмінна документація також дозволяє швидко розібратися з будь яким аспектом фреймворку без зайвих проблем.[5]

Загалом, Nuxt.js дуже схожий на Next.js та пропонує схожі можливості. Команди які спеціалізуються на розробці з використанням Vue.js можуть віддати перевагу цьому мета-фреймворку задля швидкої реалізації необхідного функціоналу з перевагами серверного рендерингу, замість того щоб розбиратися в незнайомій технології.

Недоліки Nuxt.js.

Одним із найважливіших недоліків Nuxt.js є невелика спільнота розробників у порівнянні з іншими фреймворками. Кожен рік ця спільнота зростає але не тими самими темпами що і, наприклад, Next.js. Це призводить до випадків коли деякі проблеми для яких зазвичай існує стандартизоване рішення, буде вимагати мануальної реалізації.

Незважаючи на те що більшість популярних модулів мають високий рівень якості, шанс не знайти рішення під конкретну задачу все ще досить великий.

Також у порівнянні з іншими фреймворками, Vue.js працює повільніше тому зі зростом комплексності проекту можуть виникнути проблеми за оптимізацією. У випадках великої кількості динамічного контенту, або виконанні комплексних API запросів можуть виникати затримки або навіть блокування сайту. Оптимізація таких кейсів може потребувати ручних виправлень та кастомізації фреймворку під потреби проекту, або посилення серверних ресурсів для пом'якшення проблем з оптимізацією. Такі дії неодмінно збільшать витрату на час та вартість розробки [17]

Залежність від Vue.js є великої проблемою для розробників які з ним не знайомі. Шанс мати розробника знайомого саме з цим фреймворком набагато нижчий ніж розробника який знайомий із Angular або React. Специфічний синтаксис та методолія архітектури також значно відрізняється від інших через що процес навчання та залучення нових розробників може займати тривалий час. [30]

Обмежена кастомізація є великою проблемою для комплексних та унікальних проектів із специфічними вимогами. Такі кейси можуть вимагати навіть зміни у вихідному коду фреймворка, якщо необхідно змінити підхід для маршрутизації або стратегій рендерингу окремих сторінок, що значно підвищить складність розробки та підтримки великих проектів.

Час збірки проекту також може відштовхнути від вибору цього фреймворку. Зі збільшенням кількості сторінок, непропорційно підвищується час на компіляцію. Навіть якщо ці сторінки не мають "важкої" логіки або невелику кількість контенту, час на компіляцію буде зростати лише через те що у кодову базу була додана нова сторінка. Підтримка комплексних проектів буде значно ускладнюватися особливо для великих команд які звикли працювати у швидках середовищах.

Таким чином, хоча Nuxt.js є чудовим мета-фреймворком для створення маленьках або середніх за величиною веб додатків із використанням Vue.js, його недоліки, такі як складність налаштування у великих проектах, слабкіша оптимізація, обмежена

спільнота, залежність від Vue.js, можуть впливати на вибір розробників. Це важливо враховувати під час вибору цього фреймворку, та потребує обережного аналізу щодо того чи підходить цей фреймворк для вашого проекту.[10]

Переваги Astro.

Astro — це досить молодий фреймворк для створення сучасних веб додатків. Його особлива функція це його підхід до гідрації статичного вмісту JavaScript-ом за допомогою так званої архітектури "островів" (Island Architecture). Цей підхід за допомогою особливого алгоритму дозволяє ефективно ідентифікувати інтерактивні компоненти, додаючи JavaScript лише до них а не для всієї сторінки. Це покращує користувацький досвід та сприяє швидкому завантаженню сторінок, що особливо помітно на мобільних пристроях. [12]

Друга особливість це агностичність до фреймворків. Із Astro можна використати велику кількість популярних фреймворків для рендерингу таких як React, Angular та Vue.js та багатьох інших. Таким чином команді розробки не потрібно робити компроміс у виборі фреймворку та дозволяє починати розробку із використанням тієї технології яка є привабливою для всіх членів команди. Також це дозволяє комбінувати ці фреймворки та окремо їх використовувати для різних сторінок, що дозволяє залучати розробників з різними заннями то здібностями.

Статична генерація у поєднанні із архітектурою островів робить можливою швидку розробку простих веб сайтів з малою кількістю динамічних елементів та забезпечує відмінну швидкість завантаження для користувачів з мінімальним навантаженням на основний сервер.[38]

Мінімалістичний підхід до архітектури та вибіркове впровадження JavaScript робить можливим максимальну оптимізацію розміру завантаження файлів під час запиту користувача, що також покращує час першого завантаження та кешування. Зазвичай Astro самотужки визначає, які компоненти вимагають JavaScript та завантажує лише коли він необхідний, але API Astro також надає зручні методи для того щоб підключати JavaScript лише в необхідних місцях, що покращує досвід розробки.

Зручний і зрозумілий синтаксис, також дозволяє легко розібратися у використанні цього фреймворку. Інтуїтивна структура проекту та велика кількість прикладів з якісною документацією дозволяє розробникам швидко вирішити більшість проблем які могли б виникнути під час використання. Також є додаткові плагіни розроблені спільнотою для допомоги у вирішенні часто виникаючі проблеми.

Оптимізації перелічені вище також допомагають знизити навантаження на сервер. У випадках додатків з великим трафіком кешування, CDN, генерація статичного контенту дозволяють використовувати менше ресурсів серверної інфраструктури та часу на її підтримку. Така економічність забезпечує більш стабільну роботу і дозволяє легко масштабуватись.[39]

Об’єднання архітектури “островів”, статична генерація, вибіркове підключення JavaScript і простота у використанні робить Astro дуже перспективним інструментом для створення високо оптимізованих веб додатків. Такий фреймворк є відмінним вибором для невеликих проектів для яких продуктивність і швидкість є найбільш важливими.

Недоліки Astro.

Astro має багато переваг, але також і перелік недоліків які обмежують його вибір і деяких випадках.

Відносна молодість фреймворку не дозволяє бути впевненим у його стабільності у всіх можливих випадках. Зараз цим фреймворком активно користуються ентузіасти, але досі невідомо чи можна його використовувати для дійсно великих проектів. Спільнота розробників активно зростає, але кількість готових прикладів досі мала. Також кількість плагінів, бібліотек та інших готових рішень значно менша ніж у конкурентів. Це буде проблемою для тих кто хочу використовувати найкращі практики для більшості специфічних завдань. Тому більшість розробників надають перевагу більш перевіреним фреймворкам як Next.js або Nuxt.js. Активна розробка Astro може призвести до необхідності в значних оновленнях готової кодової бази у випадках

оновлення версій. Звичні методи та функції можуть значно змінитися, нові алгоритми імплементовані, що робе підтримку довгострочних проектів непередбачуваною.[26]

Не дивлячись на те що Astro підтримує можливість використовувати будь який фреймворк для рендерингу, таких як React, Vue, Svelte тощо, при роботі в такому середовищі можуть виникнути проблеми в розумінні та оновленні коду з одночасним використанням різних фреймворків для рендерингу.[36]

Острівна архітектура має незаперечні переваги в оптимізації, але такий підхід обмежує динамічність при взаємодії із компонентами додатку. Astro досить агресивно може обмежувати використання JavaScript, що може унеможливити або ускладнити реалізацію деякого функціоналу. Проекти з потребою із комплексною клієнтською логікою ускладнює створення необхідного користувацького досвіду. Складні односторінкові додатки (SPA), що потребують активної взаємодії з користувачем, може вимагати імплементацію неочевидних рішень які можуть ускладнити розробку.

Astro чудово підходить для статичних сайтів, але при необхідності динамічного контенту з серверу, що вимагає динамічного рендерингу може виникнути зникнення численних вбудованих оптимізацій для завантаження сторінок.

Розробники які звикли працювати з іншими мета фреймворками, як Next.js та Nuxt.js, можуть зіткнутись із складнощами при освоєнні цієї технології. Незважаючи на простий синтаксис і дружелюбність до новачків, розробники які спеціалізуються на інших мета фреймворках будуть більш продуктивними з ними які нічим не уступають Astro в функціональності, хоч і менш агресивно оптимізовані.

Astro це перспективний фреймворк з цікавими підходами до оптимізації застосунку, але проблеми з обмеженою екосистемою, проблеми зі створенням інтерактивного контенту, обмежені перспективи використання у великих проектах важливо мати на увазі під час вибору Astro як основної технології перед початком розробки проекту.

Порівняння Astro, Next.js та Nuxt.js: Вибір оптимального підходу.

Astro, Next.js та Nuxt.js це основні популярні мета фреймворки які можна розглядати для створення сучасних та оптимізованих веб додатків. Кожен з них має свої недоліки та переваги, які важливо враховувати в залежності від вимог проекту. Їх порівняння допоможе нам визначити який з них є найкращим варіантом для нашого проекту. [31]

Astro відрізняється своєю унікальною архітектурою “островів”. Генерація статичного HTML із вибірковим завантаженням JavaScript забезпечує високу продуктивність та прискорення часу завантаження сторінок. Це робить цей фреймворк чудовим вибором де швидкість завантаження та SEO оптимізація є критичною. Але певні обмеження у динамічному рендерингу та інтерактивності, створює обмеження для складних клієнто орієнтованих додатків.

Next.js на сьогодняшній день є найпопулярнішим мета0фреймворком для створення сучасних веб додатків. Його основними перевагами є підтримка серверного рендерингу, статична генерація контенту, основа у вигляді React та підтримка серверних функцій під назвою API routes що дозволяє завчасно завантажити дані з бази даних перед статичною генерацією. Усі ці аспекти дозволяють розробникам знайти баланс між оптимізацією та функціональністю фінального продукту. Завдяки гнучкій архітектурі та велиої спільноти розробників, Next.js став дуже популярни як серед новачків, так і серед ветеранів індустрії. Однак залежність від React може відлякати тих розробників які спеціалізуються на інших фреймворків для рендерингу. [14]

Альтернативою і конкурентом Next.js на Vue.js є Nuxt.js. Це також багатофункціональний мета-фреймворк із підтримкою серверного рендерингу та статичної генерації. Головною перевагою щодо Next.js є більша простота використання завдяки використанню Vue.js для рендерингу. Незважаючи на присутню SEO оптимізацію та пришвидшення завантаження статично згенерованих сторінок, Nuxt.js значно програє у комплексних та великих проектах. Продуктивність Next.js вважається більш стабільною та прогнозованою, а екосистема та спільнота досі залишається значно меншою, що створює обмежений доступ до допоміжних бібліотек.

Після порівняння трьох фреймворків можно впевнено сказати, що Next.js є найбільш універсальним вибором для створення масштабованих, складних та сучасних веб додатків. Широкий спектр потреб покривається присутністтю SSR, SSG, підтримкою API-routes та їх бездоганною реалізацією. Astro, у свою чергу, є чудовим рішенням для сайтів, орієнтованих на контент, де головний акцент робиться на швидкості завантаження та мінімізації використання JavaScript. Його обмеження у динамічності роблять його менш придатним для SPA-додатків із великою інтерактивністю. Nuxt.js забезпечує простоту та зручність для команд, що використовують Vue.js, і пропонує підтримку модулів, але може поступатися Next.js за продуктивністю та масштабованістю у великих проєктах.

Враховуючи всі переваги, недоліки та потреби проекту, Next.js буде найкращим вибором. У проекті де пріоритетом є створення швидких статичних сайтів із мінімальним навантаженням JavaScript, Next.js є відмінним рішенням.

1.2 Аналіз методів проектування

З розвитком веб технологій з'явилися нові підходи до створення веб додатків, які полегшували роботу розробників та підвищували продуктивність систем. Усе починалося з простих статичних сайтів у 90-их роках, але з часом виникли потреби в динамічних системах з якими користувачі могли б активно взаємодіяти. З підвищенням можливостей розробників, підвищилися вимоги до фінального продукту, що призвело до створення високофункціональних мета-фреймворків для забезпечення ефективності та масштабованості без компромісів з користувацьким досвідом. [40]

Одним із таких сучасних рішень є Next.js — мета-фреймворк на основі React, який підтримує серверний рендеринг та статичну генерацію сторінок. Цей фреймворк пропонує розробникам зручні можливості використовувати сучасні методи оптимізації для створення швидких та SEO-оптимізованих веб застосунків.

Традиційні веб додатки на основі React використовували підхід односторінкових додатків, що ускладнювало можливості у SEO оптимізації та створювало умови для зменшеня бистродії за рахунок постійного збільшення ваги JavaScript файлів. Відповіддю на це стала поява Next.js який завдяки об'єднання переваг клієнтських та серверних додатків значно вплинув на покращення досвіду користувачів. [27]

Однією з основних переваг Next.js є його здатність гнучко адаптуватися до різних типів проектів Він забезпечує легке налаштування та масштабування архітектури під зростаючі потреби проекту. У Next.js передбачено створення API-роутів, що дозволяє заздалегідь виконати складну логіку підготувати динамічний контент у вигляді статично згенерованої сторінки. Таке рішення спрощує створення додатків, що обробляють великі обсяги даних або вимагають інтерактивності, зберігаючи при цьому високу продуктивність та оптимізацію.

Компанія Vercel яка розробила та підтримує Next.js також надає велику кількість інструментів для легкого деплою та масштабування серверної інфраструктури спеціалізованої від додатки розроблені на цьому мета фреймворку. Вони надають послуги з інтеграції з сучасними хмарними рішенням та автоматизують процес розгортання кодових баз на серверах. Такі можливості можуть значно знизити витрати на власну інфраструктуру а також дозволяє не витрачати необхідний на розробку такої самої інфраструктури час. Vercel має демократичну цінову політику та є вигідним рішенням як для маленьких команд з маленьким бюджетом так і корпорацій яким важлива прогнозованість та продуктивність. [28]

У порівнянні з іншими методами проектування, Next.js пропонує рішення які дозволяють спростити керування станом додатку а також гнучку компонентну структуру. Розбиття коду (code splitting) та динамічний імпорт модулів дозволяє значно зменшити обсяг JavaScript необхідного для обробки клієнтської логіки, що сприяє оптимізації швидкості завантаження та рендерингу без необхідності йти на компроміс із користувацьким досвідом.

Але важливо зазначити, що використання Next.js потребує акуратного планування та вибору оптимальних підходів до розробки. Розробники мають розбиратись у специфічних методах інтеграцій з іншими сервісами, та відповідально ставитись до використання існуючих бібліотек, щоб не втратити оптимізаційні переваги Next.js. У цьому сенсі проектування додатків з використанням Next.js вимагає врахування архітектурних особливостей і правильного налаштування серверного та клієнтського рендерингу для досягнення найвищої продуктивності.

Таким чином, аналіз методів проектування з використанням Next.js показує, що цей мета-фреймворк пропонує збалансоване рішення між простотою використання та високою продуктивністю, дозволяючи створювати оптимізовані, SEO-дружні вебзастосунки. Його інтеграція з сучасними хмарними рішеннями та зручна підтримка серверного рендерингу роблять Next.js вибором номер один для багатьох проєктів, орієнтованих на масштабованість і швидке розгортання.

2 ПОСТАНОВКА ЗАДАЧІ

2.1 Мета та задачі дослідження

Метою даного дослідження є поліпшення процесів управління контентом шляхом створення вебсистеми з використанням фреймворка Next.js. Ефективність веб системи управління контентом є ключовим фактором у сучасному інтернеті. Основа у вигляді Next.js дозволяє використовувати сучасні підходи до оптимізації веб сторінок та забезпечення динамічного користувацького досвіду. Висока продуктивність та проста інтеграція з бекенд сервісами, сприяє удосконаленню користувацького досвіду та оптимізацію адміністрування контенту.

Об’єктом дослідження є процес підвищення ефективності управління контентом веб системи шляхом впровадження сучасних технологій рендерингу та автоматизації, зокрема використання серверного рендерингу, статичної генерації сторінок та інтеграції з API. Цей процес включає аналіз сучасних підходів до розробки веб систем, вибір найефективніших методів і засобів реалізації, а також впровадження технологій, які дозволять зробити управління контентом більш зручним і гнучким.

Предметом дослідження є інструменти та методи реалізації вебсистеми управління контентом із використанням фреймворка Next.js. Особливу увагу приділено особливостям застосування серверного рендерингу та статичної генерації сторінок для підвищення швидкості завантаження контенту, покращення SEO-оптимізації та забезпечення високої продуктивності системи. Також у межах предмета досліджуються підходи до інтеграції бекенд-функціоналу з API-роутами та інші можливості, що надає Next.js для управління контентом.

Задачі дослідження випливають із мети та предмета дослідження і включають:

1. Проведення аналізу сучасних методів проектування веб систем управління контентом з акцентом на продуктивність та зручність використання.

2. Вивчення можливостей фреймворка Next.js для поліпшення процесу рендерингу сторінок, SEO-оптимізації та інтеграції з API.

3. Розробка прототипу веб системи управління контентом, що забезпечує використання серверного рендерингу та статичної генерації сторінок.

4. Дослідження можливостей підвищення продуктивності системи шляхом оптимізації роботи з API-роутами та інших серверних рішень.

5. Проведення тестування створеної системи для оцінки її ефективності з точки зору швидкості завантаження сторінок, стабільності роботи та зручності використання.

6. Підготовка рекомендацій щодо впровадження подібних систем для управління контентом у комерційних проектах з акцентом на використання Next.js.

Завданням дослідження також є оцінка результатів реалізації та розробка висновків щодо можливостей і обмежень застосування фреймворка Next.js для веб систем управління контентом. Для цього планується дослідити вплив технологій рендерингу та генерації сторінок на продуктивність системи та користувацький досвід, що дозволить обґрунтувати ефективність вибору Next.js як інструменту для створення таких систем.[33]

Виконання поставлених задач дозволяє провести дослідження ефективності використання таких сучасних підходів як серверний рендеринг та статична генерація та їх вплив на продуктивність і досвід користувачів та амінінстраторів.

2.2 Вибір інструментів та методів реалізації

Для розробки системи управління контентом було обрано такі технології:

1. Next.js для фронтенд частини
2. Node.js для бекенд API серверу
3. SQLite як база даних

Такий вибір дозволяє належним чином розподілити функції між відображенням, агрегацією та зберіганням даних. Простота налаштування та мінімальна залежність від зовнішніх систем забезпечує гнучкість яка дуже важлива на початкових етапах розробки. [15]

Фреймворк Next.js було обрано за функцію серверного рендерингу, статичну генерацію сторінок та підтримку API-роутів. Завдяки цьому можно створити оптимізований веб додаток з можливістю подальшого розширення з високою швидкістю завантаження сторінок і забезпечувати SEO-оптимізацію. Підтримка серверного рендерингу в Next.js дозволяє миттєво присилати згенеровані сторінки для користувача. Це також допомагає індексації пошуковими системами, що є критичним для комерційних проектів. Статична генерація сторінок Next.js є важливим аспектом для контентних сайтів оскільки вона дозволяє створювати сторінки, які вимагають мінімального навантаження на сервер.[20]

Вибір Node.js для бекенду обумовлений тим що це популярне середовище для виконання JavaScript на сервері. Це дозволить використовувати однакову мову програмування як для бекенду так і для фронтенду. Це знижує складність коду та спрощує роботу в обох середовищах. Node.js добре підходить для створення серверів з великим одночасним навантаженням. Слабкою стороною є складні обчислення на процесорі, але у нашому кейсі набагато важливіша можливість швидко обробляти одночасні запроси до серверу. Для проекту зі створення системи управлінням контенту, Node.js виконує роль обробника запитів від користувачів та адміністраторів. Бекенд забезпечує доступ до бази даних та виконує CRUD-операції і не вимагає важких вичислень за допомогою процесора. API для роботи з файловою системою та обробка http запитів дозволяє створити простий та масштабований бекенд для системи управління контентом який буде обробляти запити адмін панелі та клієнтського застосунку.

SQLite як база даних на даному етапі є гарним варіантом оскільки така реалізація реляційної бази даних дуже добре підходить на початкових етапах розробки проекту. SQLite дозволяє уникнути залежності від зовнішнього сервера бази даних, що знижує складність розгортання та експлуатації системи. Замість того щоб робити помилку передчасної оптимізації (premature optimization), краще обрати простий варіант на початкових етапах з можливістю розширення або міграції на більш багатофункціональні бази даних як PostgreSQL або MySQL. На даному етапі важливіше створити стабільну архітектуру і обробка великих обсягів даних не є пріоритетом. Незважаючи на "легкість" цієї бази даних SQL залишається ефективним інструментом для зберігання та отримання даних. SQLite забезпечує достатню продуктивність для невеликих та середніх проектів, де кількість одночасних користувачів і навантаження на базу даних є помірними. Як зазначав відомий інженер програмного забезпечення Роберт Мартін, навіть текстовий файл може бути рішенням для зберігання даних на ранніх етапах, якщо він задовольняє поточні вимоги, і лише з появою складніших потреб слід обирати відповідні системи баз даних.[18]

Наступним важливим етапом планування архітектури є вибір фронтенд фреймворку. Як зазначалося раніше Next.js ідеально підходить під кейс менеджменту контенту та його відображення, а також дозволяє ефективну взаємодію з бекендом. Підтримка API-роутів дозволяє заздалегідь генерувати статичні сторінки з динамічним контентом. У кейсах коли відбувається велика кількість запитів на сервер використання API-роутів особливо важливе для зменшення затримки та забезпечення стабільності.

Такий вибір створює умови для легкого масштабування системи. Майбутнє зростання може бути легко адаптоване під нову серверну архітектуру або зміни у залежності від бази даних. Кодова база вже буде підготовлена до роботи з будь якою СУБД завдяки використанню ORM або об'єктно-реляційного мапінгу. Таким чином можна забезпечити можливе розширення проекту у майбутньому з мінімальними зусиллями.[37]

Отже, Next.js, Node.js та SQLite як основні інструменти для створення веб системи управління контентом є обґрунтованим рішенням. Важливими аспектами є гнучкість, легкість у розгортанні, оптимізованість під сьогоднішні потреби, та потенціал до розширення під зростаючі вимоги бізнесу. Це дає можливість уникнути передчасних оптимізацій та забезпечити максимальну ефективність розробки, орієнтуючись на поточні потреби та забезпечуючи потенціал для подальшого зростання.

3 МОДЕЛЮВАННЯ ТА ПРОЕКТУВАННЯ

3.1 Структурно-функціональне моделювання процесу

Структурно-функціональне моделювання процесу розробки веб системи управління контентом передбачає аналіз ключових компонентів архітектури та визначення їхньої взаємодії.

Однією із ключових частин системи це фронтенд. Ця частина складається з двох застосунків:

- Адміністративної панелі
- Клієнтського веб застосунку

Для розробки адміністративної панелі було використано Next.js і буде віконувати роль інструменту для менеджменту контенту який буде відображатися на боці клієнтського веб застосунку. Переваги Next.js дозволяють створити безпечний та прогнозований функціонал для редагування, додавання та видалення товарів.

Адміністратор має доступ до динамічних сторінок, які будуть автоматично оновлюватися під час взаємодії з бекендом через REST API. Серверний рендеринг забезпечить швидке завантаження та негайний доступ до найбільш актуального стану бази даних, а сами списку товарів, кількості замовлень та їх статус. Статична генерація сторінок дозволяє заздалегідь оптимізувати ті частини інтерфейсу які не мають динамічно змінюючихся елементів. Завдяки цим функціям фреймворку ми маємо можливість мінімізувати навантаження на серверну інфраструктуру. [41]

На базі Next.js також створено клієнтський веб застосунок. Він буде виконувати роль інтерфейсу взаємодії користувачів з бізнес логікою. Користувач буде мати змогу переглядати, шукати за назвою та фільтрувати необхідні товари, читати їх детальні описи та зображення, а також створити замовлення. Веб застосунок також передбачає функцію авторизації та автентифікації, що необхідно для створення зв’язку між

користувачем та окремим товаром. Функції Next.js дозволять максимізувати якість користувацького досвіду завдяки статичної генерації сторінок та серверного рендерингу, що також допоможе зменшити навантаження на серверну інфраструктуру. [22]

Для підтримки запитів від веб застосунків, була розроблена бекенд система на базі Node.js. Вона забезпечує підключення та управління базою даних а також забезпечує безперервну роботу API. Таке оточення для виконання коду є вигідним у парі із Next.js завдяки використання однієї мови програмування, JavaScript, що спрощує одночасну підтримку як фронтенд так і бекенд частин. Завдяки подієво-орієнтованої моделі обробки запитів Node.js має можливість ефективно обробляти асинхронні запити та не блокувати процес виконання коду паралельно зроблених запитів. Це особливо важливо для систем управління контентом, де часто виникають одночасні запити на додавання, редагування та видалення інформації.

База даних SQLite використана для зберігання даних про товари, замовлення, користувачів та адміністраторів. SQLite це вбудована база даних яка не потребує налаштування окремого сервера для її використання. Це особливо важливо на початку розробки коли створення стабільної архітектури важливіше за швидкість обробки запитів у базу даних. Завдяки своїй структурі SQLite дозволяє зберігати дані у вигляді звичайних файлів, що спрощує процес розгортання системи і зменшує витрати на інфраструктуру. SQLite також відкриває можливість замінити у майбутньому базу даних на більш багатофункціональну як наприклад PostgreSQL або MySQL. У поєднанні з Node.js така міграція буде максимально гладкою та безперешкодною. [13]

Функціональне моделювання передбачає взаємодію всіх компонентів системи. Адміністративна панель та клієнтський веб застосунки отримують дані завдяки REST API які відправляються на Node.js сервер. Сервер обробляє такі запити та робить запит до бази даних після чого результати запитів передаються назад до панелі у вигляді оновлених даних.

Навіть при великих обсягах контенту, серверний рендеринг Next.js забезпечує швидке завантаження сторінок, що особливо важливо для клієнтської частини. Бекенд на Node.js обробляє запити користувачів асинхронно, забезпечуючи стабільну роботу системи навіть при одночасному зверненні великої кількості користувачів. База даних SQLite дозволяє зберігати дані з мінімальними затримками, забезпечуючи швидкий доступ до інформації.[16]

За результатами структурно-функціонального моделювання, була побудована архітектура проекту, яка відповідає вимогам сучасних веб додатків ключовими аспектами яких є: швидкість завантаження, зручність використання, легкість у розробці та подальшому масштабуванні. Вибір Next.js як фронтенд-фреймворку, Node.js як серверної платформи та SQLite як бази даних забезпечує гнучкість, продуктивність та ефективність у роботі з контентом.

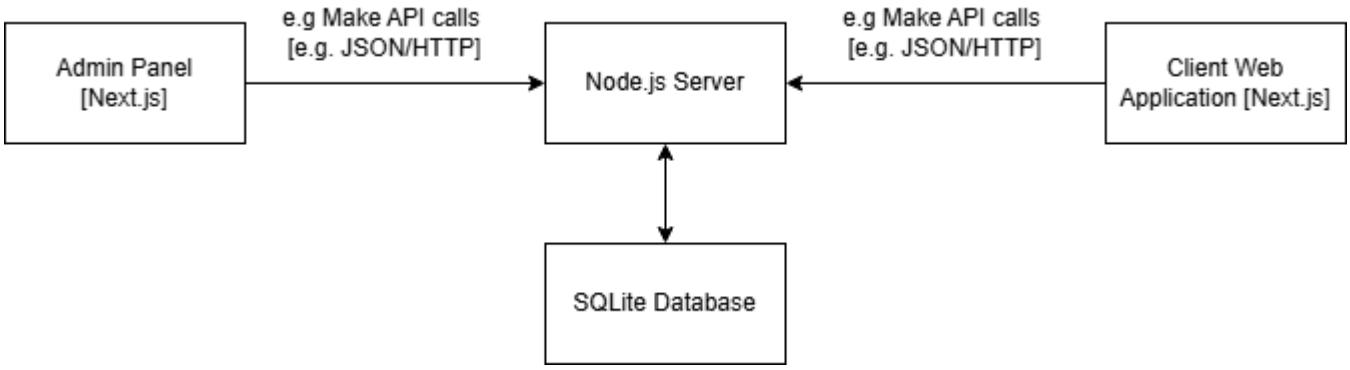

Рисунок 3.1 – Архітектура системи представлена в нотації C4

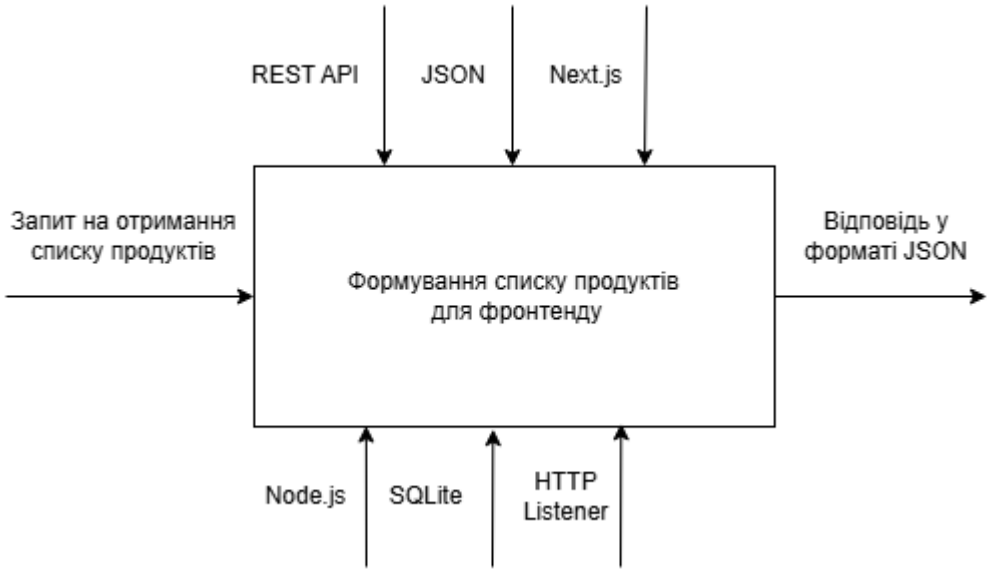

Рисунок 3.2 – Контекстна діаграма в нотації IDEF0

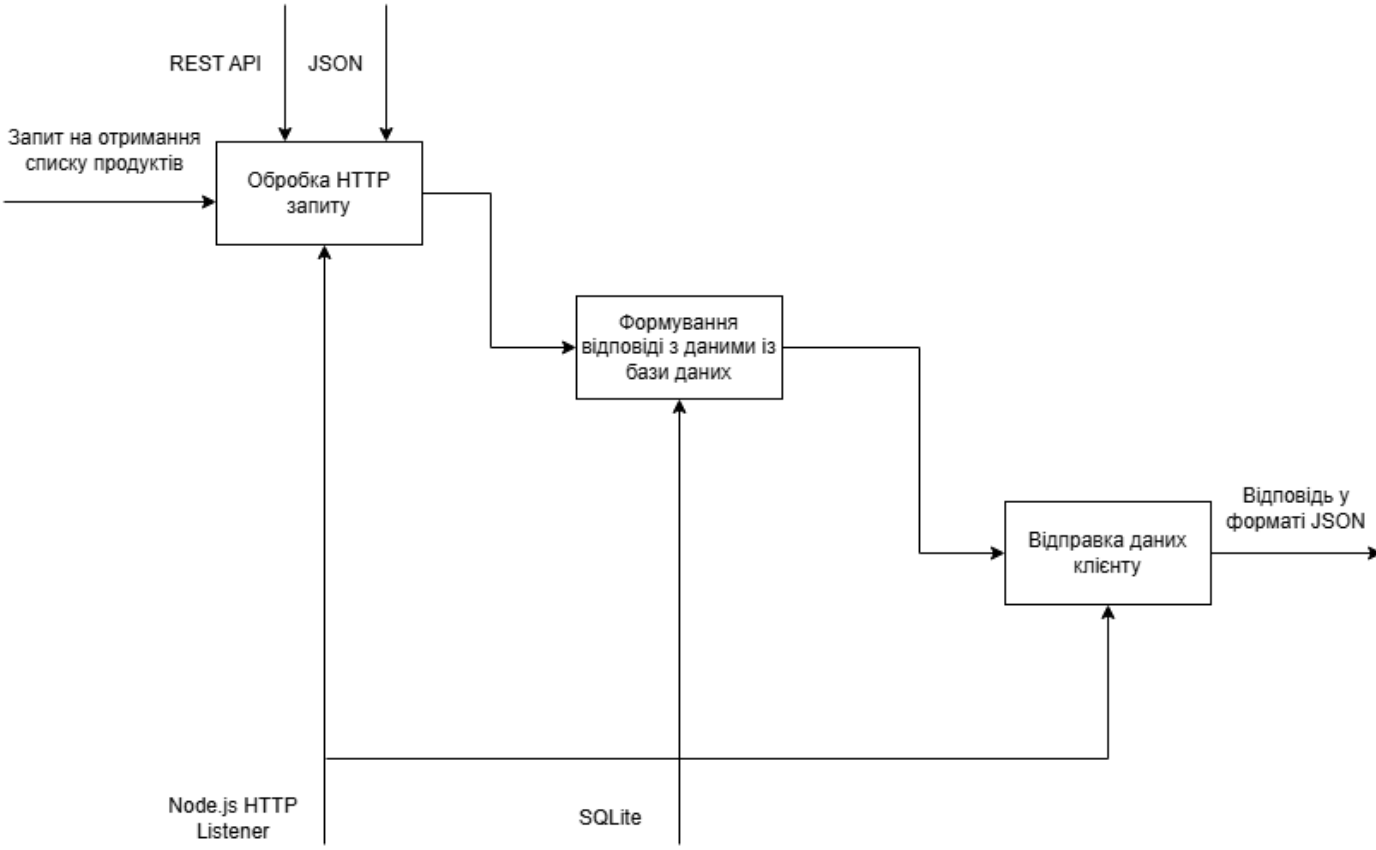

Рисунок 3.3 – Діаграма декомпозиції в нотації IDEF0 – Отримання списку продуктів для фронтенду

З контекстної діаграми, зображеної на рисунку 3.2 та 3.3, можна побачити, що основним засобом отримання даних для відображення списку продуктів є запит на

отримання списку продуктів. Клієнт повинен сформувати HTTP запит у форматі REST API до серверу та додати аргументи запиту у форматі JSON.

При цьому серед основних ресурсів та інструментів, які задіяні для виконання даного процесу є:

- Node.js HTTP Listener – потрібен для того щоб приймати та обробляти вхідні HTTP запити
- SQLite– потрібен для збереження, змінення та отримання даних

На виході нашого процесу клієнт отримує список продуктів у форматі JSON.

3.2 Моделювання варіантів використання

Важливим кроком створення архітектури проекту є моделювання варіантів використання систему яку потрібно розробити. Це дозволяє визначити яким чином користувач має змогу взаємодіяти із додатком, та прогнозувати які типа даних будуть необхідними та дає уяву про те яка бізнес логіка буде необхідною задля забезпечення ефективної інтеграції між усіма частинами проекту. У межах даного проекту, що базується на технологіях Next.js, Node.js та SQLite, розглянемо основні варіанти використання системи управління контентом (рис. 3.1).

Сценарій 1: Авторизація адміністратора

Адміністратор при завантаженні веб сторінки адміністративної панелі зустрічається із формою авторизації. Після заповнення форми адміністратор має натиснути на кнопку “Логін”. Це сформує запит до серверу та у випадку співпадіння електронної адреси та паролю, у адміністратора з’являється доступ до подальших функцій панелі. Дані успішної авторизації будуть збережені у браузері і дозволить відвідувати сайт у подальшому без необхідності у повторній авторизації.

Сценарій 2: Авторизація користувача

Користувач повинен бути залогіненим щоб мати змогу формувати замовлення. Процес авторизації користувача не сильно відрізняється від авторизації адміністратора,

але відбувається в іншому застосунку і формує окремий запит до серверу. Після заповнення форми користувач має натиснути на кнопку “Логін”. Це сформує запит до серверу та у випадку співпадіння електронної адреси та паролю, у користувача також з’являється доступ до здатності формувати замовлення, або змінювати свої користувацькі дані. Успішна авторизація так само буде збережена у браузері.

Сценарій 3: Додавання нового товару через адміністративну панель

Ключовою функцією є створення нового товару. При заході на відповідну сторінку, адміністратор повинен заповнити форму, яка містить такі поля як назва товару, категорія, опис, колір, розміри та зображення товару. Коли всі поля будуть заповнені, адміністратор натискає кнопку “Зберегти” після чого веб додаток відправляє запит до бекенду за допомогою API. Сервер у свою чергу обробляє запит та виконує запит у базу даних. Цей сценарій покриває базову функціонал спроможності адміністраторів управляти контентом і тому є критично важливим.

Сценарій 4: Перегляд товарів на клієнтському веб сайті

Користувач при завантаженні веб сторінки клієнтського застосунку, зустрічається зі списком товарів. Застосунок надсилає API запит до серверу для отримання списку доступних товарів. Також на цій сторінці можливе зазначення необхідних фільтрів або ввести назву необхідного товару для модифікації запиту до серверу. Сервер обробляє запит і динамічно формує запит до бази даних. В залежності від присутності товарів із необхідними параметрами, сервер повертає знайдені дані.

Перед тим як відобразити товари, Next.js формує статичну сторінку з переліком даних з серверу. Це позитивно впливає на SEO та зменшує час відповіді. Динамічний рендерінг дозволяє, наприклад, під час обрання фільтру, не перезавантажувати сторінку. Це також добре впливає на користувацький досвід.

Сценарій 5: Перегляд існуючих замовлень через адміністративну панель

Маючи доступ до адміністративної панелі, адміністратор може переглянути список сформованих замовлень користувачами. Адміністратор може переглянути деталі такого замовлення, а також змінити статус вибраного замовлення. Під час завантаження

сторінки замовлень формується запит до серверу на список замовлень. Після обробки запиту сервером, на клієнт також відправляється список існуючих замовлень.

Інформації під час формування відповіді достатньо щоб відобразити повну інформацію про замовлення. На адміністративній панелі можливо динамічно сформувати компонент з детальними даними про замовлення.

Всередині компоненту деталей замовлення можливо змінити статус замовлення. Адміністратор має можливість вибрати один із можливих статусів, та натиснути кнопку “Зберегти”. Після цього буде сформовано новий запит до серверу. Після обробки такого запиту, сервер вносить оновлення у базу даних.

Сценарій 6: Створення нового замовлення на клієнтському веб сайті

Ще одним важливим функціоналом застосунку є можливість оформити замовлення на товар. На сторінці пошуку товарів, користувач має змогу вибрати товар. Після вибору користувач буде перенапрвленим на нову сторінку з деталями товару. На цій сторінці користувач моє змогу обрати товар, колір та розмір. Сторінка моє досить високу динамічність, бо в залежності від вибору користувача буде змінюватись ціна та зображення.

Після цього користувач має змогу додати товар до “корзини”. Після натискання кнопку покупки, користувач буде перенаправленим до сторінки “корзини”. Якщо користувач авторізований, він має змогу обрати кількість товарів які він хоче придбати, та підтвердити формування замовлення.

Сценарій 7: Оновлення користувацьких даних

Користувач також має змогу відвідати сторінку користувацьких даних. На цій сторінці присутня форма з заповненими користувацькими даними. Після натиснення на перемикач редагування, форма стане активною і з'явиться можливість відредагувати дані серед яких є ім’я, адреса та номер телефону.

Якщо користувач впевнений у тому що дані які він хоче змінити заповнені правильно він може натиснути кнопку “Зберегти” після чого буду сформований відповідний запит до серверу який внесе оновлення у базу даних (рис. 3.2).

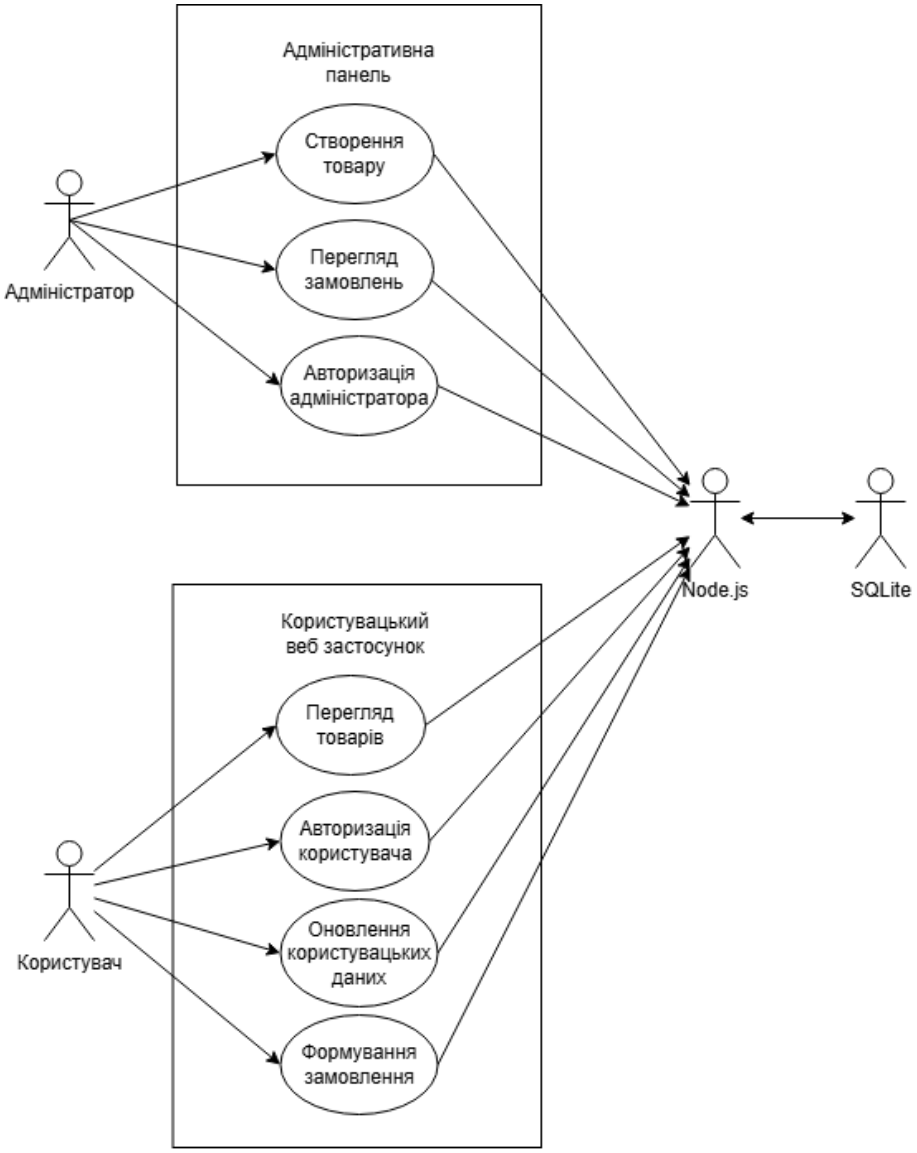

Рисунок 3.4 – Діаграма варіантів використання (зроблено автором)

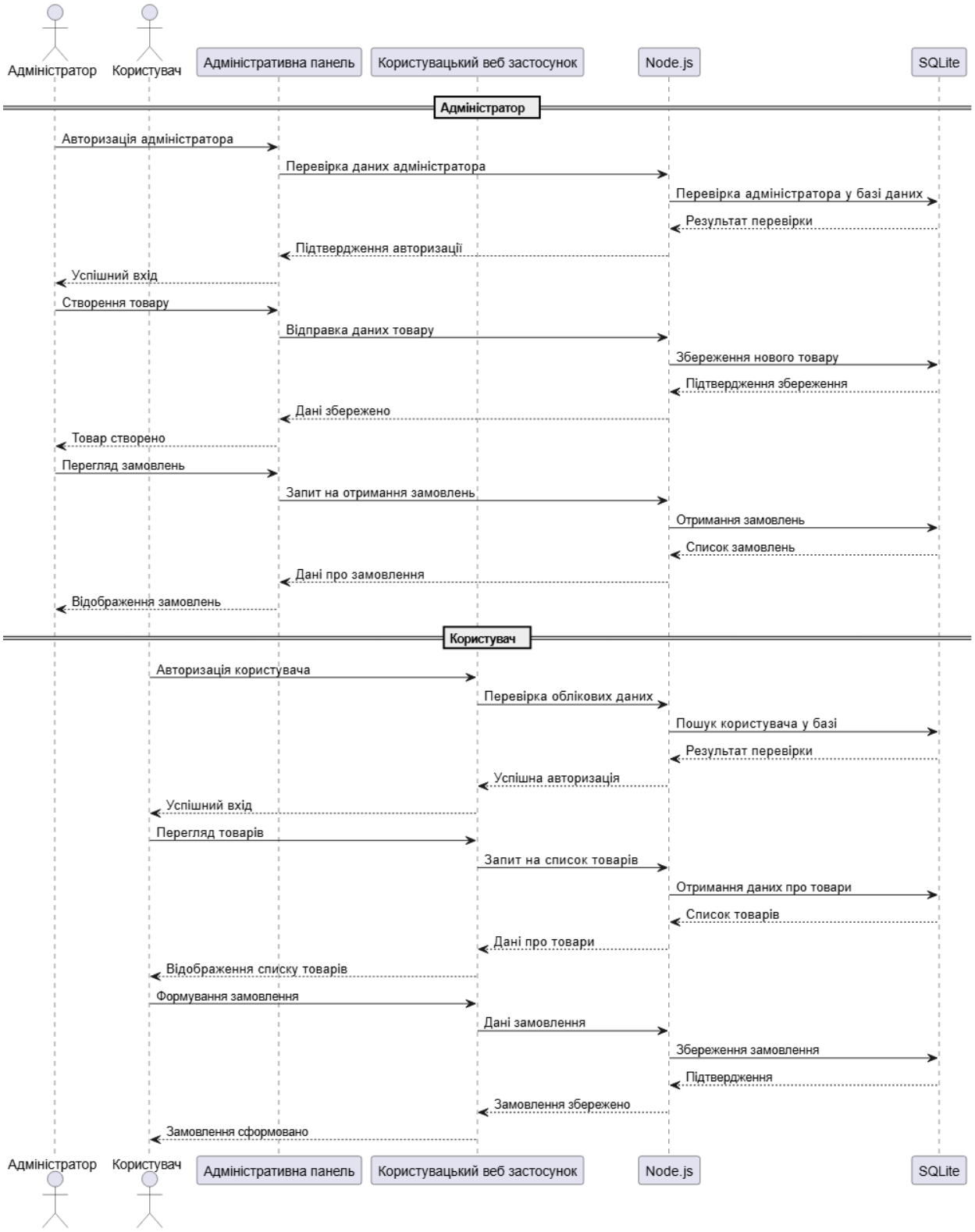

Рисунок 3.5 – Діаграма послідовності

4 ПРАКТИЧНА РЕАЛІЗАЦІЯ СИСТЕМИ УПРАВЛІННЯ КОНТЕНТОМ

4.1 Структурно-функціональне моделювання розробки

У процесі розробки проекту управління контентом було використано сучасні інструменти та технології для створення, тестування та розгортання застосунку. Робота розпочалася з налаштування робочого середовища.

Для управління версіями Node.js був обраний Node Version Manager (NVM), який забезпечує зручний спосіб встановлення та перемикання між різними версіями Node.js. Після встановлення NVM було завантажено останню стабільну версію Node.js, що гарантувало сумісність із обраними бібліотеками та фреймворками, такими як Next.js.

Для написання коду використовувався Visual Studio Code (VSCode). Цей редактор забезпечує гнучкість у роботі завдяки широкому вибору плагінів. Було встановлено такі розширення, як ESLint, Prettier, які дозволяють підтримувати код у чистому та стандартизованому вигляді. Використання цих інструментів сприяло покращенню якості коду та пришвидшило робочі процеси.

Проект складався з трьох основних компонентів:

1. Node.js сервер для бекенду системи, який реалізовував API для взаємодії з базою даних та забезпечував основний функціонал. Для зберігання даних використовувалася база даних SQLite у поєднанні з ORM Prisma. Це рішення обрано через його простоту, легкість у налаштуванні та відсутність зовнішніх залежностей. Використання Prisma забезпечило гнучкість у роботі з базою даних, зокрема при створенні моделей, міграцій та запитів.

2. Адміністративна панель, створена на основі Next.js, яка надавала адміністраторам зручний інтерфейс для управління контентом. Тут реалізовувалися функції додавання, редагування, видалення товарів, а також перегляд інформації про контент, збережений у системі.

3. Клієнтський веб застосунок, також створений за допомогою Next.js, який дозволяв користувачам переглядати товари, здійснювати пошук, фільтрувати та створювати замовлення. Завдяки серверному рендерингу (SSR) було забезпечено швидке завантаження сторінок та покращений користувацький досвід.

Після завершення локальної розробки всі компоненти проекту були розгорнуті на віртуальній машині в Google Cloud Platform (GCP). Для доступу до сервера та управління розгортанням використовувався інструмент Termius, що спростило роботу із сервером через SSH. Кожен компонент був налаштований таким чином, щоб забезпечити стабільність та ефективність роботи системи (рис. 4.1).

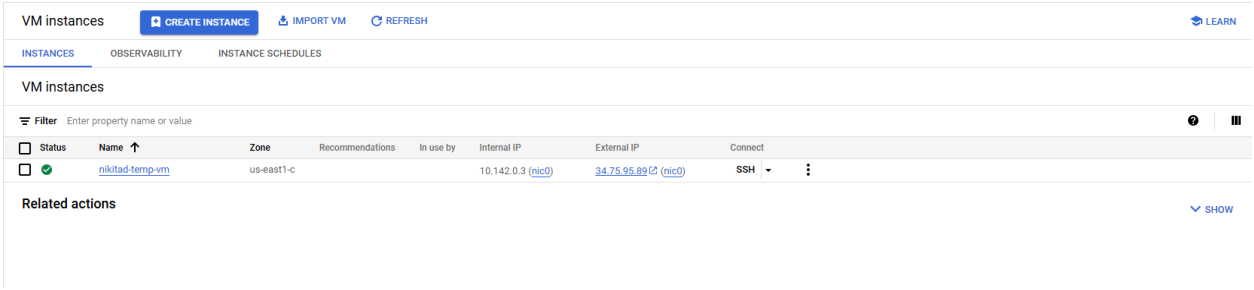

Рисунок 4.1 - Створена віртуальна машина для хостингу проекту

Цей підхід дозволив створити гнучку, продуктивну та масштабовану систему, що відповідає сучасним вимогам веб розробки.

4.2 Приклад використання адміністративної панелі

Для забезпечення зручного доступу до адміністративної панелі було виконано розгортання проекту на платформі Google Cloud Platform (GCP). Завдяки цьому рішенню адміністратори отримують можливість працювати з системою без необхідності налаштування локального середовища розробки або запуску проекту на власних пристроях.

Адміністративна панель доступна за адресою: http://34.75.95.89:3001/. Це дозволяє адміністраторам отримати прямий доступ до всіх функцій управління контентом, включаючи додавання, редагування та видалення товарів, без додаткових дій, таких як встановлення залежностей або запуск серверів локально.

Розгортання на платформі GCP також забезпечує стабільність роботи системи та її доступність у будь-який час, незалежно від операційної системи або географічного розташування користувача. Завдяки цьому рішенню адміністратори можуть зосередитися виключно на виконанні своїх задач, не переймаючись технічними аспектами роботи системи.

Для доступу до адміністративної панелі використовується проста система авторизації. Адміністратор вводить стандартний логін та пароль (рис. 4.2):

Email: admin@test.com

Пароль: 123

Після успішного введення цих даних користувач автоматично перенаправляється до основної сторінки адміністративної панелі.

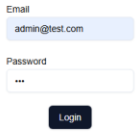

Рисунок 4.2 - Екран авторизації адміністратора

Після входу адміністратор потрапляє на сторінку, що містить таблицю з продуктами. У таблиці відображено ключову інформацію про товари: назва, опис, ціна, категорія тощо. Поруч із кожним товаром знаходиться кнопка "Редагувати", яка дозволяє змінювати дані про вибраний продукт (рис.4.3).

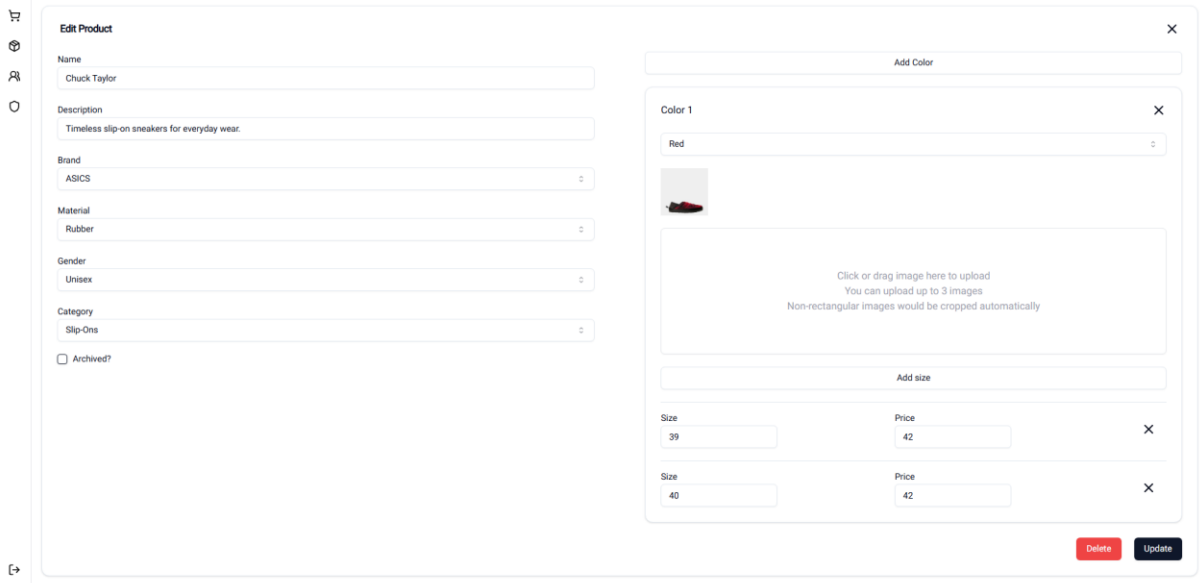

Рисунок 4.3 - Сторінка списку продуктів

Також на сторінці передбачено інпут для пошуку, що дозволяє швидко знаходити товари за їх назвою. Це значно полегшує навігацію для адміністратора, особливо у випадках, коли база даних містить велику кількість продуктів.

Сторінка редагування продукту в адміністративній панелі забезпечує зручний і структурований інтерфейс для внесення змін до даних товару.

Основні елементи сторінки:

1. Форма редагування:

a. Назва (Name): текстове поле для введення або редагування назви продукту.

b. Опис (Description): текстове поле для опису продукту.

c. Бренд (Brand): поле для введення бренду продукту.

d. Матеріал (Material): текстове поле для вказівки матеріалу товару.

e. Гендер (Gender): випадаючий список для вибору цільової аудиторії товару (наприклад, чоловічий, жіночий або унісекс).

f. Категорія (Category): поле для введення або вибору категорії товару.

g. Архівований (Archived): прапорець для позначення товару як архівованого (видалення з активного списку).

2. Робота з кольорами:

a. Додавання кольорів (Add Color): кнопка для створення нового варіанту кольору.

b. Для кожного кольору відображається окремий блок з полем для назви кольору, а також можливістю додавання зображень.

c. Додавання зображень: можливість завантаження до трьох зображень для кожного кольору шляхом перетягування або вибору з файлової системи.

d. Робота з розмірами:

3. У кожному кольоровому блоці знаходиться секція для управління розмірами.

a. Додавання розмірів (Add Size): кнопка для створення нового розміру.

b. Для кожного розміру є два поля:

c. Розмір (Size): числове поле для вказання розміру.

d. Ціна (Price): поле для вказання ціни відповідного розміру.

e. Кожен розмір можна видалити за допомогою кнопки "X".

Адміністратор може редагувати будь-які параметри продукту.

На сторінці також присутня кнопка видалити (Delete) при натисканні якої товар буде видалено.

Після внесення змін дані автоматично зберігаються після натискання кнопки "Оновити" (Update).

Ця сторінка дозволяє гнучко працювати з товарами, додаючи нові варіанти кольорів, розмірів і фотографій, що спрощує управління асортиментом (рис. 4.4).

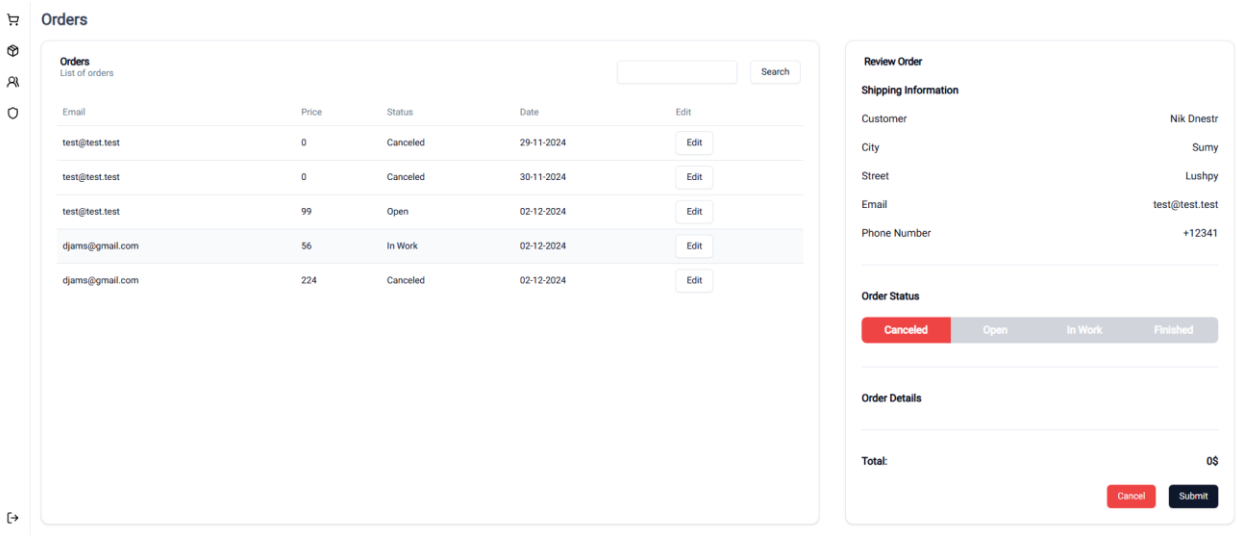

Рисунок 4.4 - Сторінка перегляду замовлень

Адміністратор використовуючи навігаційну панелі зліва може перейти на сторінку управління замовленнями (Orders). Ця сторінка забезпечує адміністратору доступ до перегляду, редагування та оновлення інформації про замовлення.

Кожен елемент списку має кнопку редагування (Edit) яка відобразить динамічну панель деталей замовлення. На цій панелі адміністратор має змогу змінити статус замовлення після натискання кнопки підтвердження (Submit).

4.3 Приклад використання користувацького веб застосунку

Клієнтський застосунок є публічним інтерфейсом для користувачів, що дозволяє переглядати та вибирати товари. Застосунок створений з акцентом на зручність використання, швидку навігацію та інтуїтивно зрозумілий інтерфейс (рис. 4.5).

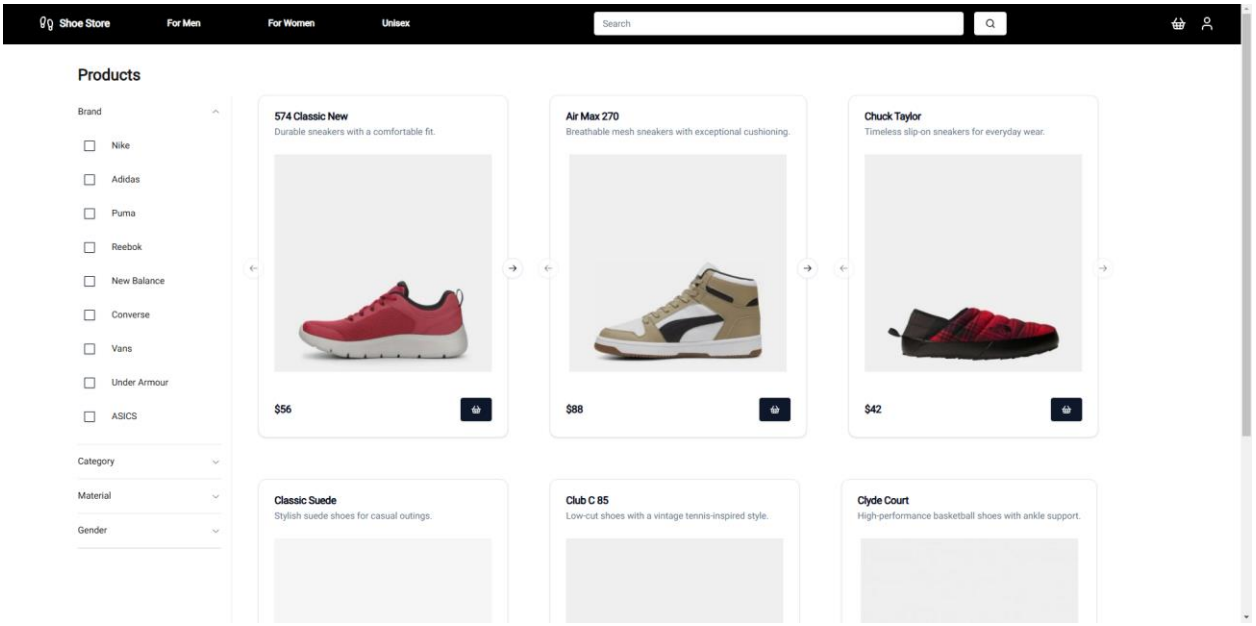

Рисунок 4.5 - Сторінка перегляду товарів користувачем

Ліва частина сторінки містить фільтри, які дозволяють користувачу сортувати товари за такими параметрами, як бренд, категорія, матеріал, стать тощо. У верхній частині сторінки розташована пошукова строка, яка дозволяє знайти товари за назвою. Товари відображаються у вигляді карток, які містять зображення, назву, короткий опис і ціну. При натисканні на кнопку з іконкою “корзини” користувач буде перенаправлений на сторінку детального опису товару (рис. 4.6).

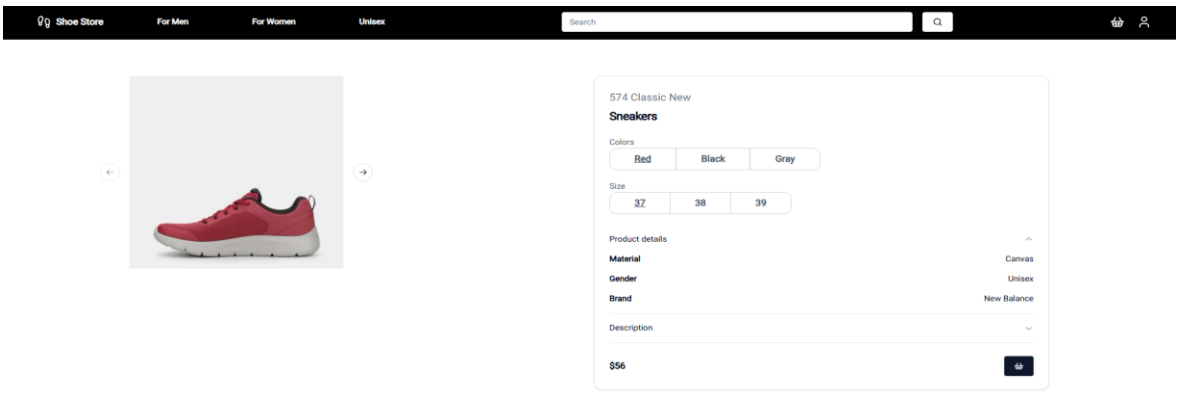

Рисунок 4.6 - Сторінка перегляду детального опису товару

Ліва частина сторінки відведена під демонстрацію великого зображення товару. Передбачена можливість перегортання, якщо для товару є кілька зображень. У правій частині розташовані кнопки для вибору кольорів товару. Наприклад, доступні кольори: червоний (Red), чорний (Black), сірий (Gray). Нижче користувач може обрати потрібний розмір серед доступних варіантів, таких як 37, 38, 39 та відображається актуальна ціна товару.

Після вибору кольору та розміру користувач може натиснути на кнопку для додавання вибраного товару в кошик.

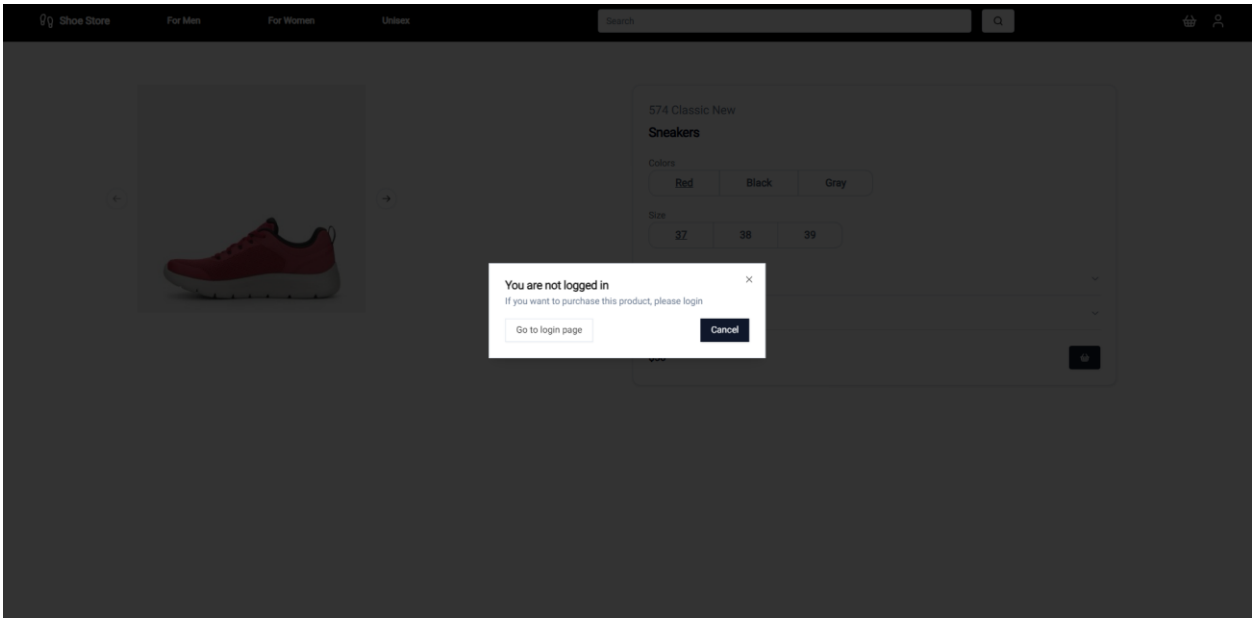

Рисунок 4.7 - Відображення екрану попередження неможливості додати товар у кошик через те що користувач не залогінений

Якщо користувач не авторизований у системі, під час спроби додати товар до кошика відображається спливаюче попередження. У цьому попередженні міститься повідомлення про те, що для придбання товару потрібно увійти до облікового запису.

Цей функціонал забезпечує інформування користувачів про необхідність авторизації та спрощує перехід до процедури входу в систему, покращуючи користувацький досвід (рис. 4.7).

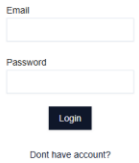

Рисунок 4.8 - Екран авторизації користувача

На цій сторінці (рис. 4.8) користувач може увійти до системи, ввівши свою електронну пошту та пароль у відповідні поля. Для тестування вже створено обліковий запис з такими даними:

Електронна пошта: djams@gmail.com

Пароль: 123

Крім того, для нових користувачів є можливість зареєструватися, якщо у них ще немає облікового запису. Посилання для реєстрації розміщене під кнопкою входу.

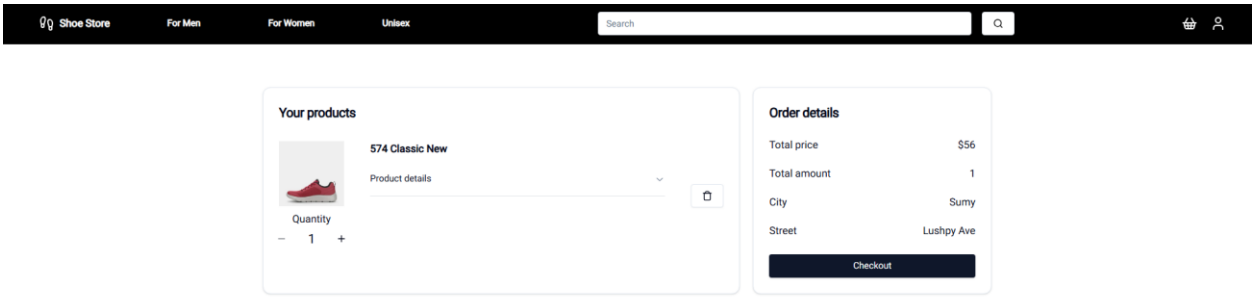

Рисунок 4.9 - Сторінка кошику користувача

Після авторизації користувач може додавати товари до кошика. На цій сторінці відображається список доданих товарів, їхня кількість та загальна сума замовлення. Для кожного товару доступні кнопки збільшення або зменшення кількості, а також кнопка для видалення з кошика (рис. 4.9).

Справа показані деталі замовлення: загальна ціна, кількість товарів, а також адреса доставки. Для підтвердження покупки користувач може натиснути кнопку "Checkout", щоб завершити оформлення замовлення.

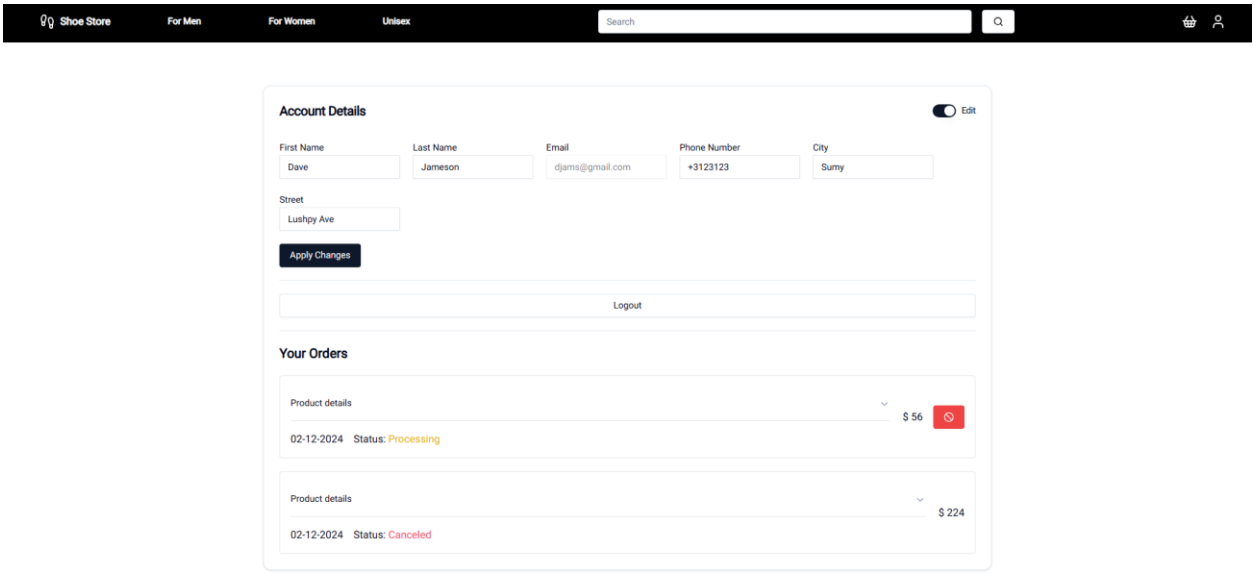

Рисунок 4.10 - Сторінка даних користувача

У застосунку також доступна сторінка профілю користувача (рис. 4.10). Щоб сюди потрапити треба натиснути на іконку користувача у верхньому правому куті навігаційної панелі. На цій сторінці користувач може переглядати та редагувати свої персональні дані, такі як ім'я, прізвище, адресу електронної пошти, номер телефону, місто та вулицю. Для внесення змін передбачений перемикач "Edit", після активації якого можна змінювати поля, а для збереження змін — кнопка "Apply Changes".

Також є кнопка "Logout" для виходу з акаунта. У нижній частині сторінки розташований список замовлень користувача з датою замовлення, статусом (наприклад, Processing або Canceled) і загальною сумою. У замовленнях є можливість переглянути деталі, зокрема товари, що входять у замовлення, та їхню кількість.

4.4 Результати тестування продуктивності системи

Одним із ключових аспектів розробки проекту управління контентом стала оптимізація продуктивності завдяки використанню можливостей, наданих фреймворком Next.js. Завдяки впровадженню технологій серверного рендерингу (SSR), статичної

генерації (SSG) та динамічного завантаження компонентів вдалося досягти швидкого завантаження сторінок і плавної роботи інтерфейсу.

Оптимізація завдяки Next.js:

SSR (Server-Side Rendering): Використання серверного рендерингу забезпечує швидке відображення сторінок для користувачів за рахунок їхнього попереднього рендерингу на сервері. Це особливо корисно для сторінок із динамічним контентом, таких як сторінки продуктів або замовлень.

SSG (Static Site Generation): Статична генерація дозволила заздалегідь створити сторінки, які не потребують частих змін. Це забезпечило ще більшу продуктивність для сторінок, доступних для великої кількості користувачів.

Динамічне завантаження: Розділення коду та завантаження тільки тих компонентів, які необхідні в конкретний момент, дозволило зменшити розмір початкового завантаження сторінки.

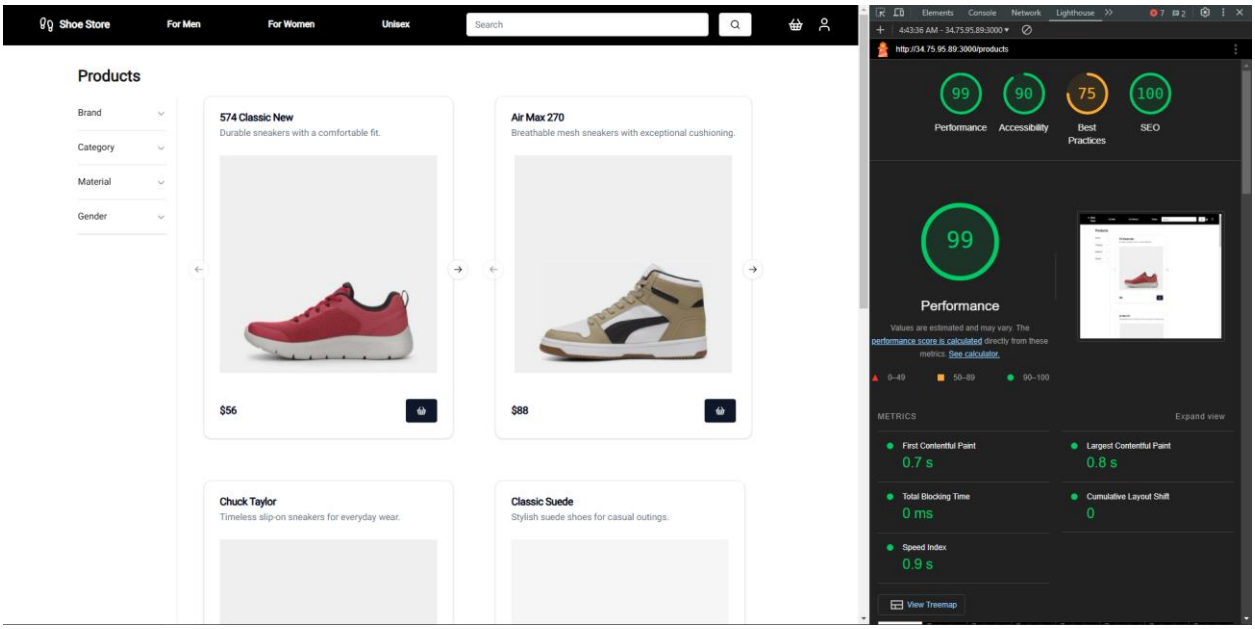

Рисунок 4.11 - Результати тестування за допомогою Lighthouse

Для оцінки продуктивності застосунку було проведено тестування за допомогою Google Lighthouse (рис. 4.11). Результати показали:

1. Performance – 99%:

a. First Contentful Paint (FCP): 0.7 сек – швидке відображення першого контенту.

b. Largest Contentful Paint (LCP): 0.8 сек – швидке завантаження основного елемента сторінки.

c. Total Blocking Time (TBT): 0 мс – взагалі немає блокування, що говорить про оптимізований JavaScript.

d. Speed Index: 0.9 сек – дуже швидке завантаження сторінки.

2. Accessibility – 90%: Достатньо високий рівень доступності, можливо, є кілька моментів, які потребують виправлення.

3. SEO – 100%: Повна відповідність сучасним вимогам SEO.

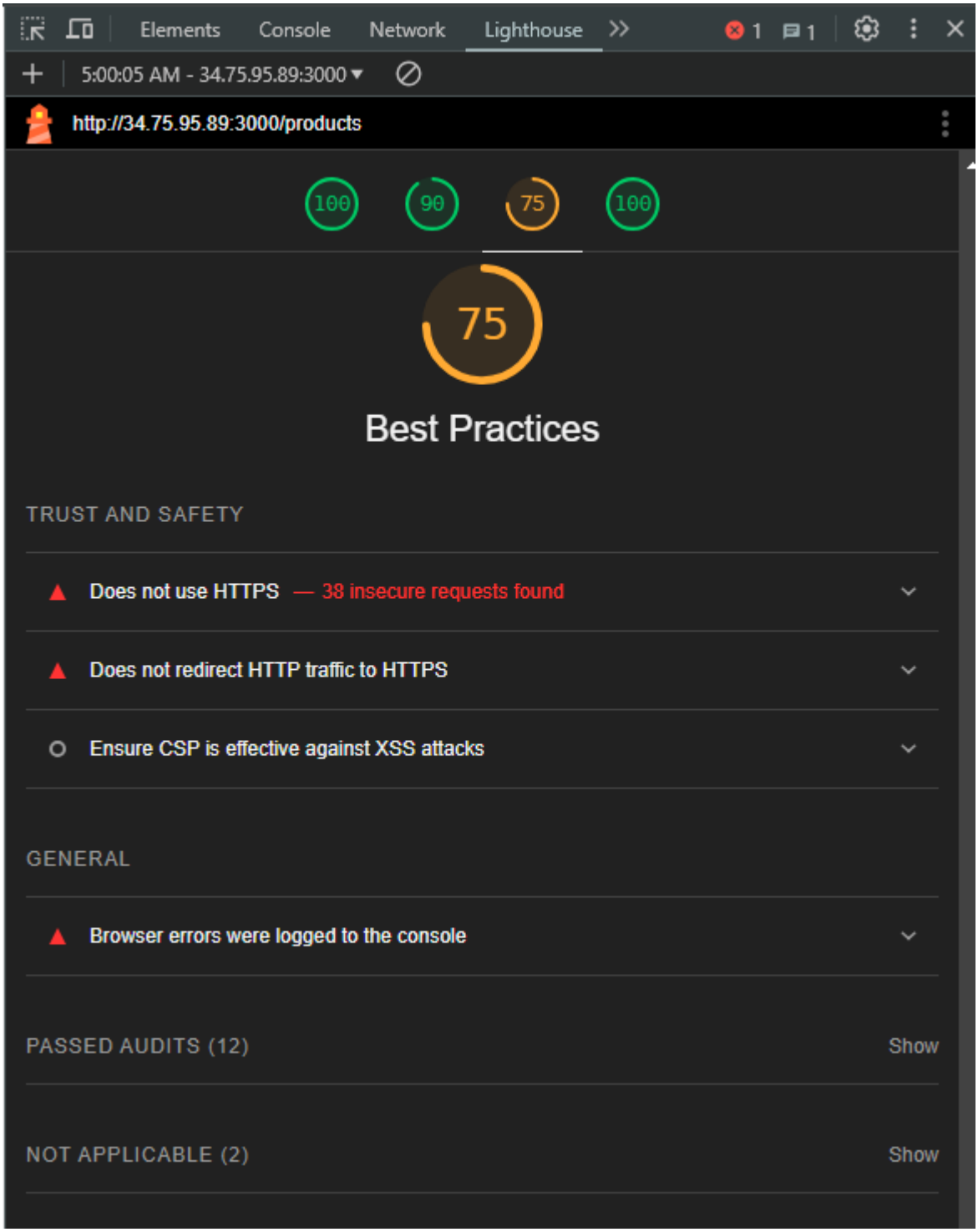

Рисунок 4.12 - Деталі тестування Best Practices

Єдиний показник, який вимагає уваги, це Best Practices (75%) (рис. 4.12). Lighthouse оцінює велику кількість показників і одним з них є хостинг. Через те що я не

налаштовував HTTPS який потребує окремого сертифікату, результати перевірки були менше ста відсотків. Але ця проблема не стосується дослідження тому не є критичною.

Завдяки використанню передових технологій фреймворка Next.js та ефективній архітектурі, система демонструє високу продуктивність і готовність до роботи під значним навантаженням. Високі показники Lighthouse підтверджують успішність реалізації технічних рішень, що використовувалися під час розробки.

ВИСНОВКИ

В рамках кваліфікаційної роботи було виявлено та чітко сформульовано завдання для розробки веб системи управління контентом на основі Next.js із підтримкою бекенд-функціоналу на Node.js та використанням бази даних SQLite. Було розроблено план виконання робіт, проведено детальний аналіз ризиків, пов'язаних із реалізацією та підтримкою системи, та створено стратегію для мінімізації можливих технічних і організаційних ризиків. Проведено дослідження сучасних підходів до розробки веб додатків та обрано найефективніші методи і технології для створення системи управління контентом.

У ході кваліфікаційної роботи було порівняно три основні мета-фреймворки: Next.js, Nuxt.js та Astro. За результатами аналізу встановлено, що Next.js є оптимальним вибором для розробки системи, яка вимагає швидкої генерації контенту, гнучкості у налаштуванні серверного рендерингу (SSR) та інтеграції з API. Завдяки використанню Next.js забезпечується висока продуктивність системи та її SEO-оптимізація, що є важливими факторами для сучасних комерційних веб застосунків.

Було також проведено аналіз методів інтеграції бекенд-функціоналу з використанням Node.js та бази даних SQLite. Вибір SQLite як СУБД обґрунтовано її легкістю у використанні та відсутністю зовнішніх залежностей, що дозволяє уникати передчасної оптимізації та спростити початкове розгортання системи. При збільшенні вимог до продуктивності система спроектована так, щоб дозволити легку міграцію на більш багато функціональні СУБД, такі як PostgreSQL або MySQL.

У результаті моделювання було створено концептуальну архітектурну модель системи, що включає адміністративну панель на Next.js, клієнтський веб сайт, серверну частину на Node.js та базу даних SQLite. Було розроблено діаграми варіантів використання, які ілюструють взаємодію між основними компонентами системи у таких сценаріях, як додавання, редагування, видалення та перегляд контенту. Розроблено

також структурно-функціональні моделі, що забезпечують розуміння взаємодії між компонентами та їхніми ролями у загальній архітектурі системи.

Під час виконання кваліфікаційної роботи було розроблено і впроваджено інструменти для забезпечення стабільності та гнучкості системи. Зокрема, використання Next.js для фронтенду та Node.js для бекенду дозволяє забезпечити швидке реагування системи на запити користувачів, що підвищує зручність використання. База даних SQLite використовується для зберігання інформації з можливістю масштабування в майбутньому.

У підсумку, виконання кваліфікаційної роботи дозволило досягти мети дослідження, зокрема підвищити ефективність процесу управління контентом у веб-системах. Це було реалізовано шляхом створення та аналізу веб-застосунку, побудованого на сучасних технологіях, таких як Next.js. Обрана архітектура забезпечує високий рівень продуктивності, гнучкість у розширенні функціоналу та готовність до майбутнього масштабування, що робить систему стійкою до змін та придатною для подальшого розвитку.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Srivastava, Shraddha, Harsh Shukla, Nisha Landge, Agam Srivastava and Devansh Jindal. “A Comprehensive Review of Next.js Technology: Advancements, Features, and Applications.” SSRN Electronic Journal. 2024. pp. 1-15.
2. Pavić, Filip and Ljiljana Brkić. “Methods of Improving and Optimizing React Web-applications.” 2021 44th International Convention on Information, Communication and Electronic Technology (MIPRO). 2021. pp. 1753-1758.
3. Kowalczyk, Karolina and Tomasz Szandała. “Enhancing SEO in Single-Page Web Applications in Contrast With Multi-Page Applications.” IEEE Access. 2024. Vol. 12. pp. 11597-11614.
4. Maurer F., Cohn M., Griffiths M., Highsmith J., Schwaber K., Kruchten Ph. B. Agile project management // Project Management, Planning and Control. 2021. – С. 200-214.
5. Al-Saqqa S., Sawalha S., Abdelnabi H. Agile Software Development: Methodologies and Trends // Int. J. Interact. Mob. Technol. 2020. Т. 14. С. 246-270.
6. Cák, Filip and Pavle Dakić. “Configuration Tool for CI/CD Pipelines and React Web Apps.” 2024 14th International Conference on Advanced Computer Information Technologies (ACIT). 2024. pp. 586-591.
7. Gomes, Francisco, Pamella Soares and Allysson Allex Araújo. “Examining the Performance Implications of Design Patterns in Front-end Web Development: A Preliminar Comparative Study with React and Vue.js.” Proceedings of the 29th Brazilian Symposium on Multimedia and the Web. 2023. pp. 1-10.
8. Bielak, Konrad, Bartłomiej Borek and Małgorzata Plechawska-Wójcik. “Web application performance analysis using Angular, React and Vue.js frameworks.” Journal of Computer Sciences Institute. 2022. pp. 1-12.

9. Skrzypiec, Sylwester and Małgorzata Plechawska-Wójcik. “Comparative analysis of Angular and React development frameworks.” Journal of Computer Sciences Institute. 2023. pp. 1-14.

10. Sharma, Tarun, Shruti Gupta and Uday Raj Singh. “Analyzing the difference between ReactJS and AngularJS.” 2023 International Conference on Computational Intelligence, Communication Technology and Networking (CICTN). 2023. pp. 37-42.

11. Mokoginta, Deyidi, Desfita Eka Putri and Fegie Yoanti Wattimena. “Developing Modern JavaScript Frameworks for Building Interactive Single-Page Applications.” International Journal Software Engineering and Computer Science (IJSECS). 2024. pp. 1-20.

12. Boczkowski, Krzysztof and Beata Pańczyk. “Comparison of the performance of tools for creating a SPA application interface - React and Vue.js.” Journal of Computer Sciences Institute. 2020. pp. 1-11.

13. Levlin, M., Annamari Soini and Dragos Truscan. “DOM benchmark comparison of the front-end JavaScript frameworks React, Angular, Vue, and Svelte.” 2020. pp. 1-17.

14. Gaffney, Kevin P., Martin Prammer, Laurence C. Brasfield, D. Richard Hipp, Dan R. Kennedy and Jignesh M. Patel. “SQLite: Past, Present, and Future.” Proc. VLDB Endow. 2022. Vol. 15. pp. 3535-3547.

15. Kabakuş, Abdullah Talha. “A Performance Comparison of SQLite and Firebase Databases from A Practical Perspective.” 2019. pp. 1-15.

16. Aurelia, Anastasia, Wasino Wasino, Desella Chandra and Tjibeng Jap. “Developing Website-Based Information System Applications to Map PT. XYZ’s Properties Using Next.JS Framework with Haversine Method.” International Journal of Application on Sciences, Technology and Engineering. 2023. pp. 1-20.

17. Kumar, Sanjay, Sandhya Umrao, Harshit Gupta and Kumud Saxena. “Project Management and Evaluation System Using Node JS.” 2023 3rd International Conference on Advance Computing and Innovative Technologies in Engineering (ICACITE). 2023. pp. 567-571.

18. Karim, Rezwana, Frank Tip, Alena Sochurkova and Koushik Sen. “Platform-Independent Dynamic Taint Analysis for JavaScript.” IEEE Transactions on Software Engineering. 2020. Vol. 46. pp. 1364-1379.

19. Scarsbrook, Joshua D., Mark Utting and Ryan Kok Leong Ko. “TypeScript’s Evolution: An Analysis of Feature Adoption Over Time.” 2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR). 2023. pp. 109-114.

20. Hoppe P. H. B., Silva Souza V. E. Support for Single Page Application Frameworks on FrameWeb // Proceedings of the 29th Brazilian Symposium on Multimedia and the Web. 2023. – C. 131-144.

21. Carvalho, Fernando, Fialho, Pedro. Enhancing SSR in Low-Thread Web Servers: A Comprehensive Approach for Progressive Server-Side Rendering with any Asynchronous API and Multiple Data Models // International Conference on Web Information Systems and Technologies. 2023.

22. Jartarghar, Harish, Salanke, Girish Rao, Kumar, Ashok A. R., Sharvani, G. S., Dalali, Shivakumar. React Apps with Server-Side Rendering: Next.js // Journal of Telecommunication, Electronic and Computer Engineering (JTEC). 2022.

23. Curie, D., Jaison, Joyce, Yadav, Jyoti, Fiona, Rex. Analysis on Web Frameworks // Journal of Physics: Conference Series. 2019. T. 1362.

24. Lanka, Surekha, Srivallop, Atikom, Pinto, Colin A. A specialized framework of web application for efficient data retrieval on social media tools // 2021 5th International Conference on Electronics, Communication and Aerospace Technology (ICECA). 2021. C. 161-166.

25. Amjad, Mahfida, Hossain, Md. Tutul, Hassan, Rakib, Rahman, Md. Abdur. Web Application Performance Analysis of ECommerce Sites in Bangladesh: An Empirical Study // International Journal of Information Engineering and Electronic Business. 2021.

26. Angkasa, Harta, Farell, Darren, Putra Wijaya, Erik Hendrawan, Achmad, Said, Fitrianah, Devi. Improving Universal Rendering Performance on NuxtJS-based Web

Application // 2023 11th International Conference on Cyber and IT Service Management (CITSM). 2023. С. 1-6.

27. Kishore, Pushkar, MahendraB., M. Evolution of Client-Side Rendering over Server-Side Rendering. 2020.

28. Iskandar, Taufan Fadhilah, Lubis, Muharman, Kusumasari, Tien Fabrianti, Lubis, Arif Ridho. Comparison between client-side and server-side rendering in the web development // IOP Conference Series: Materials Science and Engineering. 2020. Т. 801.

29. Weigle, Michele C., Nelson, Michael L., Alam, Sawood, Graham, Mark. Right HTML, Wrong JSON: Challenges in Replaying Archived Webpages Built with Client-Side Rendering // 2023 ACM/IEEE Joint Conference on Digital Libraries (JCDL). 2023. С. 82-92.

30. Li, Nian, Zhang, Bo. The Research on Single Page Application Front-end development Based on Vue // Journal of Physics: Conference Series. 2021. Т. 1883.

31. Rojas, Carlos. Building Progressive Web Applications with Vue.js: Reliable, Fast, and Engaging Apps with Vue.js. 2020.

32. Mishra, Sourabh. Angular for A Single Page Application. 2022.

33. Uehara, Minoru. Experiences with a Single-Page Application for Learning Programming // Advances on Broad-Band Wireless Computing, Communication and Applications. 2020. Т. 159. С. 55-66.

34. Sood, Reecha, Singh, Gurpreet, Chawla, Sunil K. Single Page Application: Architecture and Application. 2019.

35. Deshmukh, Smita R., Mane, Deepak, Retawade, Abhijeet. Building a Single Page Application Web Front-end for E-Learning site // 2019 3rd International Conference on Computing Methodologies and Communication (ICCMC). 2019. С. 985-987.

36. Molin, Eric. Comparison of Single-Page Application Frameworks: A method of how to compare Single-Page Application frameworks written in JavaScript. 2016.

37. Tong, Jason, Jikson, Ricky Rivaldo, Gunawan, Alexander Agung Santoso. Comparative Performance Analysis of Javascript Frontend Web Frameworks // 2023 3rd

International Conference on Electronic and Electrical Engineering and Intelligent System (ICE3IS). 2023. С. 81-86.

38. Mukhiya, Suresh Kumar, Hung, Hoang Khac. An Architectural Style for Single Page Scalable Modern Web Application. 2019.

39. Vepsäläinen, Juho, Hellas, Arto, Vuorimaa, Petri. Implications of Edge Computing for Static Site Generation // International Conference on Web Information Systems and Technologies. 2023.

40. Vepsäläinen, Juho, Vuorimaa, Petri. Bridging Static Site Generation with the Dynamic Web // International Conference on Web Engineering. 2022.

41. Challapalli, Sai Sri Nandan, Kaushik, Prakarsh, Suman, Shashikant, Shivahare, Basu Dev, Bibhu, Vimal, Gupta, Amar Deep. Web Development and performance comparison of Web Development Technologies in Node.js and Python // 2021 International Conference on Technological Advancements and Innovations (ICTAI). 2021. С. 303-307.

ДОДАТОК А

ПЛАНУВАННЯ РОБІТ

для розробки кваліфікаційної роботи магістра «Вебсистема управління контентом з використанням Next.js»

А.1 Ідентифікація мети ІТ-проекту

Метою даного ІТ-проекту є удосконалення веб системи управління контентом на основі сучасних технологій, таких як Next.js, з інтеграцією Node.js для бекенду та використанням SQLite як бази даних. Це дозволить створити ефективну, зручну та масштабовану систему, яка підтримуватиме роботу адміністративної панелі та клієнтського веб сайту для управління та відображення товарів.

Система повинна забезпечити:

- Швидке відображення та рендерінг сторінок завдяки використанню серверного рендерингу (SSR) Next.js.
- Інтерактивне управління контентом через адміністративну панель на Next.js.
- Надійне збереження та обробку даних за допомогою Node.js і SQLite.
- Гнучке оновлення та масштабування системи через інтеграцію з CI/CD інструментами, такими як GitHub Actions.

Результати деталізації методом SMART розміщені у табл. Б.1.

Таблиця Б.1 − Деталізація мети методом SMART

Specific (конкретна)	Метою даного ІТ-проекту є створення веб системи управління контентом на основі Next.js з інтеграцією Node.js для бекенду та використанням SQLite для забезпечення ефективного збереження та обробки даних.
Measurable (вимірювана)	Проект буде успішним, якщо забезпечить: ● Час відгуку вебсайту при серверному рендерингу не перевищуватиме 200 мс.

	● Можливість збереження та обробки даних із затримкою не більше 50 мс при запитах до бази даних. ● Автоматизацію процесів CI/CD через GitHub Actions із 95% успішних розгортань без збоїв.
Achievable (досяжна)	Завдання є досяжним, оскільки використовуються перевірені та популярні технології, такі як Next.js для фронтенду, Node.js для бекенду та SQLite для бази даних. Команда розробників має достатній досвід у роботі з цими інструментами та у створенні ефективних веб застосунків.
Relevant (реалістична)	Створення сучасної веб системи управління контентом є важливим для бізнесу, що прагне забезпечити зручність користувачам і швидкість доступу до даних. Проект відповідає сучасним вимогам до веб розробки та інтернет-комерції.
Time-framed (обмежена у часі)	Проект має бути завершений протягом трьох місяців, кінцевий термін виконання — 10 листопада 2024 року.

А.2 Планування змісту структури робіт ІТ-проєкту

Для виконання проекту були визначені основні етапи роботи, структуровані за методологією WBS (Work Breakdown Structure):

Аналіз і планування:

- Визначення вимог до підсистеми управління контентом.
- Аналіз існуючих фреймворків та інструментів, вибір оптимальної архітектури (Next.js, Node.js, SQLite).
- Визначення ризиків, пов'язаних з обмеженнями SQLite та іншими аспектами технологій, та створення плану їх мінімізації.

Розробка інфраструктури:

- Налаштування середовища для розробки та тестування (включаючи Docker для контейнеризації).
- Розробка бекенд-сервісів на Node.js для забезпечення API та обробки даних.
- Інтеграція бази даних SQLite для збереження та керування даними.
- Створення адміністративної панелі та клієнтського вебсайту на основі Next.js для управління та відображення контенту.

Тестування та оптимізація:

- Проведення тестування працездатності та стабільності веб системи.
- Оптимізація запитів до бази даних SQLite для забезпечення ефективності обробки даних.
- Тестування серверного рендеринга (SSR) та взаємодії клієнтського сайту з API для швидкого завантаження сторінок.
- Виправлення виявлених помилок та усунення потенційних вузьких місць у продуктивності.

Впровадження та документування:

- Підготовка та впровадження підсистеми у продуктивне середовище.

- Налаштування автоматизованих процесів CI/CD за допомогою GitHub Actions для забезпечення безперебійного розгортання.
- Документування інтерфейсів API, структури системи та основних сценаріїв використання.
- Передача підсистеми замовнику або зацікавленим сторонам із відповідною документацією для подальшої підтримки та розвитку.

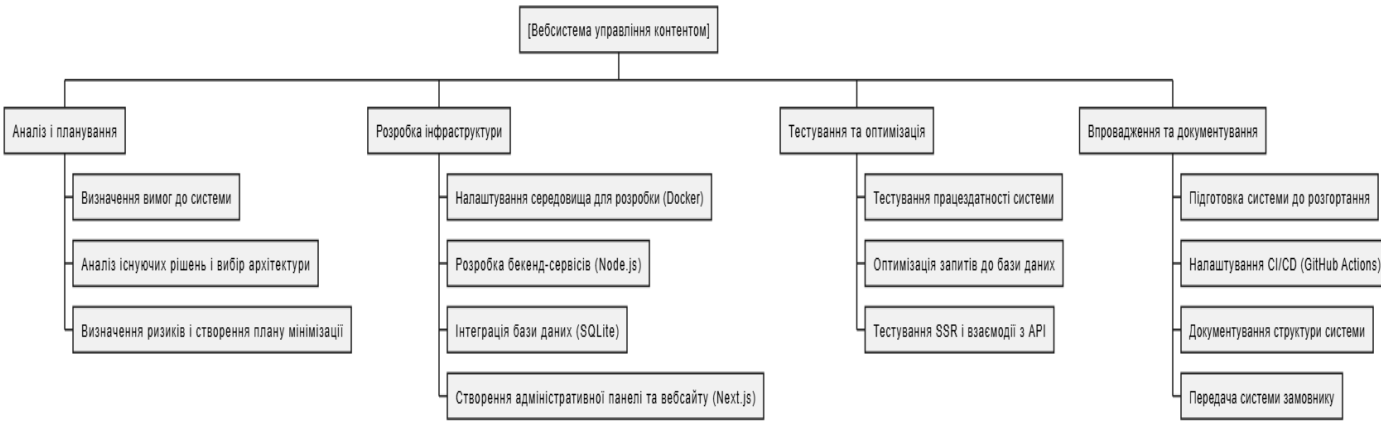

Рисунок А.1 – Структура WBS (зроблено автором)

Рисунок А.2 – Структура OBS (зроблено автором)

А.3 Побудова календарного графіка виконання ІТ-проекту

Проект реалізовувався протягом 19.08.2024 – 10.11.2024. Нижче наведено календарний графік робіт:

Етап	Опис робіт	Тривалість	Дати виконання
Аналіз та планування	Визначення вимог, аналіз архітектур	1 тиждень	19.08.24 – 23.08.24
Розробка інфраструктури	Налаштування середовища розробки, розробка бекенду на Node.js, інтеграція SQLite	2 тижні	26.08.24 – 06.09.24
Розробка клієнтської частини	Створення адміністративної панелі та клієнтського вебсайту на Next.js	6 тижнів	09.09.24 – 18.10.24
Тестування та оптимізація	Тестування функціональності, стабільності	2 тижні	21.10.24 – 01.11.24

	системи, оптимізація запитів до бази даних		
Впровадження застосунку	Розгортання системи	1 тиждень	04.11.24 – 11.11.24

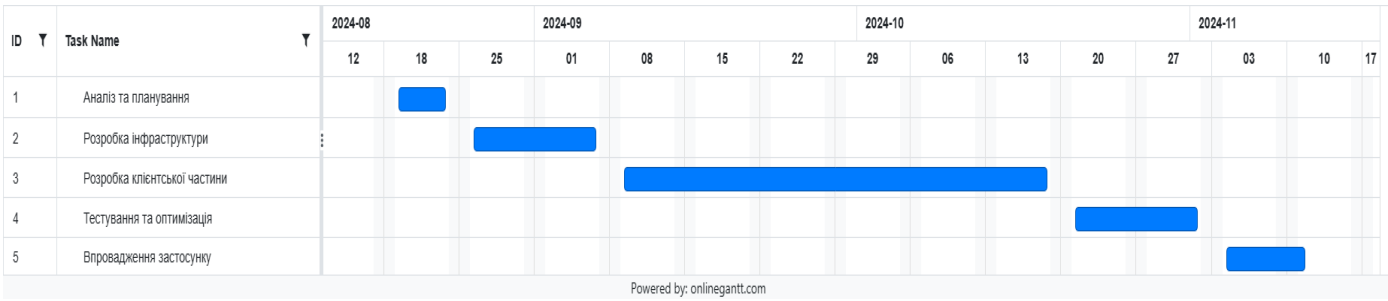

Рисунок А.3 – Діаграма Ганта (зроблено автором)

А.4 Планування ризиків проекту

Було визначено основні ризики та розроблено стратегії для їх мінімізації:

Технічні ризики:

- Ризик: Збої в роботі API на Node.js або проблеми з продуктивністю бази даних SQLite при високому навантаженні.
- Мінімізація: Впровадження кешування для зниження навантаження на базу даних, використання логування та моніторинг основних показників продуктивності.

Інфраструктурні ризики:

- Ризик: Некоректне налаштування середовища розробки або збої під час роботи контейнерів Docker.
- Мінімізація: Регулярне тестування контейнерів, автоматизація процесів розгортання, налаштування резервного копіювання для збереження стабільності системи.

Комунікаційні ризики:

- Ризик: Несвоєчасне виконання запитів між клієнтським сайтом і бекендом або затримки в обробці даних.
- Мінімізація: Використання асинхронних запитів для оптимізації обробки даних і зменшення часу відгуку, регулярне тестування на швидкодію.

Ризики впровадження:

- Ризик: Проблеми під час розгортання нових версій або конфлікти між компонентами (адмін-панель, клієнтський веб сайт, API).
- Мінімізація: Використання CI/CD для автоматизації розгортання та перевірки сумісності компонентів, налаштування відкату до попередніх версій у разі виникнення помилок.

Організаційні ризики:

- Ризик: Невідповідність термінам або затримки у виконанні окремих етапів проекту.

- Мінімізація: Регулярні зустрічі команди для обговорення статусу проекту та контроль виконання завдань відповідно до встановленого графіка.

ДОДАТОК Б

Б.1 Лістинг програмного коду

Посилання на повні репозиторії

1. https://github.com/nikita-dnestrov/uni-client
2. https://github.com/nikita-dnestrov/uni-admin
3. https://github.com/nikita-dnestrov/uni-server

Головна сторінка відображення товарів клієнтського застосунку

https://github.com/nikita-dnestrov/uni-client/blob/main/src/app/(app)/products/page.tsx

```
import { Separator } from "../../../components/ui/separator";
import { Accordion, AccordionContent, AccordionItem, AccordionTrigger } from "../../../components/ui/accordion";
import { API_BASE_URL, BRANDS, CATEGORIES, GENDERS, MATERIALS, SERVER_BASE_URL } from "../../../lib/const";
import { ProductCard } from "../components/ProductCard";
import { Pagination, PaginationContent, PaginationItem, PaginationLink } from "../../../components/ui/pagination";
import { redirect } from "next/navigation";
import axios from "axios";
import { Checkbox } from "../../../components/ui/checkbox";
import { Label } from "../../../components/ui/label";

type TFilters = {
  material: string[];
  gender: string[];
  category: string[];
  brand: string[];
  search: string[];
```

```
};

type TProduct = {
  id: string;
  name: string;
  description: string;
  colors: { sizes: { price: number }[]; images: { url: string }[] }[];
};

type TPagination = {
  currentPage: number;
  totalPages: number;
};

async function fetchProducts(searchParams: URLSearchParams) {
  const filters: Partial<TFilters> = {
    material: searchParams.get("material") ? [searchParams.get("material")!] : [],
    gender: searchParams.get("gender") ? [searchParams.get("gender")!] : [],
    category: searchParams.get("category") ? [searchParams.get("category")!] : [],
    brand: searchParams.get("brand") ? [searchParams.get("brand")!] : [],
    search: searchParams.get("search") ? [searchParams.get("search")!] : [],
  };

  const page = parseInt(searchParams.get("page") || "1", 10);

  //@ts-ignore
  const query = new URLSearchParams({ page: page.toString(), limit: "6", ...filters });
```

```
  const response = await axios.get(`${API_BASE_URL}/products?${query.toString()}`);
  return response.data;
 }

 export default async function Page({
  searchParams,
 }: {
  searchParams: {
   page: number | undefined;
   brand: string | undefined;
   material: string | undefined;
   category: string | undefined;
   gender: string | undefined;
   search: string | undefined;
  };
 }) {
  //@ts-ignore
  const data = await fetchProducts(new URLSearchParams(searchParams));
  const products: TProduct[] = data.data;
  const pagination: TPagination = {
   currentPage: data.currentPage,
   totalPages: data.totalPages,
  };

  const filterMapper =
   (type: "material" | "gender" | "category" | "brand") => (el: { label: string; value: string }, i: number) => {
```

```
            return (
              <div key={i} className="flex gap-4 py-2 items-center">
                <PaginationLink href={linkHrefGeneratorFilter(type, el.value)}>
                  <Checkbox checked={searchParams[type] === el.value ? true : false} id={`${el.label}`} />
                </PaginationLink>
                <Label htmlFor={`${el.label}`}>{el.label}</Label>
              </div>
            );
          };

        const productMapper = (el: TProduct) => {
          const { id, name, description, colors } = el;
          const price = colors[0].sizes[0].price;
          const images = colors[0].images.map((img) => `${SERVER_BASE_URL}${img.url}`);

          return <ProductCard key={id} href={`/products/${id}`} data={{ name, description, price, images }} />;
        };

        const renderPagination = () => {
          const pages = Array.from({ length: pagination.totalPages }, (_, i) => i + 1);
          return pages.map((page) => (
            <PaginationItem key={page}>
              <PaginationLink href={`?page=${page}`} className={`${pagination.currentPage === page ? "bg-slate-300" : ""}`}>
                {page}
```

```
      </PaginationLink>
    </PaginationItem>
  ));
};

const linkHrefGeneratorFilter = (key: string, value: string): string => {
  const searchParamsCopy = { ...searchParams, [key]: value };

  //@ts-ignore
  if (key in searchParams && searchParams[key] === value) {
    //@ts-ignore
    delete searchParamsCopy[key];
  }

  searchParamsCopy.page = 1;
  //@ts-ignore
  const urlQuery = new URLSearchParams(searchParamsCopy);

  return `?${urlQuery.toString()}`;
};

return (
  <>
    <div className="text-2xl font-bold ml-[100px] my-[15px]">Products</div>
    <div className="flex px-[100px]">
      <div className="flex w-1/6 h-screen">
        <div className="w-full">
          <Accordion defaultValue={Object.keys(searchParams)} type="multiple">
```

```
                <AccordionItem value="brand">
                  <AccordionTrigger>Brand</AccordionTrigger>

<AccordionContent>{BRANDS.map(filterMapper("brand"))}</AccordionContent>
                </AccordionItem>
                <AccordionItem value="category">
                  <AccordionTrigger>Category</AccordionTrigger>

<AccordionContent>{CATEGORIES.map(filterMapper("category"))}</AccordionContent>
                </AccordionItem>
                <AccordionItem value="material">
                  <AccordionTrigger>Material</AccordionTrigger>

<AccordionContent>{MATERIALS.map(filterMapper("material"))}</AccordionContent>
                </AccordionItem>
                <AccordionItem value="gender">
                  <AccordionTrigger>Gender</AccordionTrigger>

<AccordionContent>{GENDERS.map(filterMapper("gender"))}</AccordionContent>
                </AccordionItem>
              </Accordion>

              <Separator />
            </div>
            <Separator className="h-full ml-4" orientation="vertical" />
          </div>
          <div className="w-full px-10">
```

```
            <div className="flex flex-wrap gap-16 mb-10">{products.map(productMapper)}</div>
            <Pagination>
              <PaginationContent>{renderPagination()}</PaginationContent>
            </Pagination>
          </div>
        </div>
      </>
    );
  }
```

Сторінка відображення деталей товару клієнтського застосунку https://github.com/nikita-dnestrov/uni-client/blob/main/src/app/(app)/products/%5Bid%5D/page.tsx

```
import axios from "axios";
import { ClientProductContent } from "./components/ClientProductContent";

export default async function Page({ params }: { params: { id: string } }) {
  const productId = params.id;
  const getProduct = async () => {
    const response = await axios.get(`http://34.75.95.89:5000/api/products/${productId}`);

    return response.data;
  };

  const product = await getProduct();
```

```
  return (
    <div className="flex gap-4 pt-[50px] px-[10vw]">
      <ClientProductContent product={product} />
    </div>
  );
}
```

Компонент для відображення клієнтської логіки розглядання деталей товару https://github.com/nikita-dnestrov/uni-client/blob/main/src/app/(app)/products/%5Bid%5D/components/ClientProductContent.tsx

```
"use client";
import {
  Carousel,
  CarouselContent,
  CarouselItem,
  CarouselNext,
  CarouselPrevious,
} from "../../../../../components/ui/carousel";
import { SERVER_BASE_URL } from "../../../../../lib/const";
import {
  Card,
  CardContent,
  CardDescription,
  CardFooter,
  CardHeader,
  CardTitle,
} from "../../../../../components/ui/card";
import { Button } from "../../../../../components/ui/button";
```

```
import { ShoppingBasket } from "lucide-react";
import { cn } from "../../../../../lib/utils";
import { Separator } from "../../../../../components/ui/separator";
import { useCallback, useEffect, useState } from "react";
import { Accordion, AccordionContent, AccordionItem, AccordionTrigger } from
"../../../../../components/ui/accordion";
import { Badge } from "../../../../../components/ui/badge";
import { productPageApiService } from "../api";
import { LoginStateDialog } from "./LoginStateDialog";
import { useRouter } from "next/navigation";

export const ClientProductContent = ({ product }: { product: any }) => {
  const [chosenColor, setChosenColor] = useState(product.colors[0].id);
  const [chosenSize, setChosenSize] = useState(product.colors[0].sizes[0].id);

  const [loginStateDialog, setLoginStateDialog] = useState(false);

  const router = useRouter();

  useEffect(() => {
    // console.log(product);
    console.log(chosenSize);
  }, [chosenColor]);

  const handleColorChoice = (id: string) => {
    setChosenColor(id);
    setChosenSize(product.colors.find((el: any) => el.id === id).sizes[0].id);
  };
```

```
const handleSizeChoice = (id: string) => {
  setChosenSize(id);
};

const addProductToCart = useCallback(async () => {
  const userId = localStorage.getItem("userId");
  const token = localStorage.getItem("token");

  if (userId && token) {
    await productPageApiService.addProductToCart(
      {
        userId,
        productId: product.id,
        colorId: chosenColor,
        sizeId: chosenSize,
        quantity: 1,
      },
      token
    );
    router.push("/cart");
  } else {
    setLoginStateDialog(true);
  }
}, [chosenColor, chosenSize]);

return (
  <>
```

```
            <div className="w-1/2  ">
              <Carousel className="w-[350px]">
                <CarouselContent>
                  {product.colors
                    .find((el: any) => el.id === chosenColor)
                    .images.map((el: any) => {
                      return (
                        <CarouselItem key={el}>
                          <img className="object-cover select-none"
src={`${SERVER_BASE_URL}${el.url}`} alt="img" />
                        </CarouselItem>
                      );
                    })}
                </CarouselContent>
                <CarouselPrevious />
                <CarouselNext />
              </Carousel>
            </div>
            <div className="w-1/2">
              <Card className="w-full">
                <CardHeader>
                  <CardDescription className="text-lg">{product.name}</CardDescription>
                  <CardTitle className="text-xl">{product.category}</CardTitle>
                </CardHeader>
                <CardContent className="flex flex-col gap-5">
                  <div>
                    <div className="text-sm text-gray-600">Colors</div>
```

```
                <div className="flex w-fit h-[40px] items-center rounded-xl overflow-hidden border border-slate-300">
                  {product.colors.map((color: any, i: number) => (
                   <>
                    <div
                     key={color.id}
                     className={cn(
                      "flex-1 text-center py-2 font-bold text-slate-600 hover:underline cursor-pointer transition-opacity border-r border-slate-300 px-10 last:border-none",
                       chosenColor === color.id ? "underline" : "hover:underline"
                     )}
                     onClick={() => handleColorChoice(color.id)}
                    >
                     {color.color}
                    </div>
                    {product.colors.length !== i + 1 && <Separator orientation="vertical" />}
                   </>
                  ))}
                 </div>
                </div>
                <div>
                 <div className="text-sm text-gray-600">Size</div>
                 <div className="flex w-fit h-[40px] items-center rounded-xl overflow-hidden border border-slate-300 ">
                  {product.colors
                   .find((el: any) => {
                    return el.id === chosenColor;
                   })
```

```
                .sizes.map((size: any, i: number) => (
                  <>
                    <div
                      key={size.id}
                      className={cn(
                        "flex-1 text-center py-2 font-bold text-slate-600 hover:underline cursor-pointer transition-opacity border-r border-slate-300 px-10 last:border-none",
                        chosenSize === size.id ? "underline" : "hover:underline"
                      )}
                      onClick={() => handleSizeChoice(size.id)}
                    >
                      {size.size}
                    </div>
                  </>
                ))}
              </div>
            </div>
            <Accordion type="single" collapsible>
              <AccordionItem value="item-2">
                <AccordionTrigger>Product details</AccordionTrigger>
                <AccordionContent className="flex flex-col gap-4">
                  <div className="flex justify-between">
                    <div className="font-semibold">Material</div>
                    <div>{product.material}</div>
                  </div>
                  <div className="flex justify-between">
                    <div className="font-semibold">Gender</div>
                    <div>{product.gender}</div>
```

```
                </div>
                <div className="flex justify-between">
                  <div className="font-semibold">Brand</div>
                  <div>{product.brand}</div>
                </div>
              </AccordionContent>
            </AccordionItem>
            <AccordionItem value="item-3">
              <AccordionTrigger>Description</AccordionTrigger>
              <AccordionContent>{product.description}</AccordionContent>
            </AccordionItem>
          </Accordion>
        </CardContent>
        <CardFooter className="flex justify-between">
          <CardTitle>
            $
            {
              product.colors.find((el: any) => el.id === chosenColor).sizes.find((el: any)
=> el.id === chosenSize)
                .price
            }
          </CardTitle>
          <Button onClick={addProductToCart} type="button" className="rounded">
            <ShoppingBasket size={28} />
          </Button>
        </CardFooter>
      </Card>
```

```
            <LoginStateDialog isOpen={loginStateDialog} onClose={() =>
setLoginStateDialog(false)} />
          </div>
        </>
      );
    };
```

Сторінка для авторизації

https://github.com/nikita-dnestrov/uni-client/blob/main/src/app/(auth)/welcome/page.tsx

```
    "use client";

    import { useState } from "react";
    import { LoginForm } from "./components/LoginForm";
    import { RegisterForm } from "./components/RegisterForm";

    export default function Page() {
      const [view, setView] = useState("login");

      return (
        <div className="w-screen h-screen flex justify-center items-center">
          {view === "login" ? (
            <LoginForm onViewChange={() => setView("register")} />
          ) : (
            <RegisterForm onViewChange={() => setView("login")} />
          )}
        </div>
      );
```

}

Компонент для відображення форми заповнення для логіну

https://github.com/nikita-dnestrov/uni-client/blob/main/src/app/(auth)/welcome/components/LoginForm.tsx

```
"use client";

import { SubmitHandler, useForm } from "react-hook-form";
import { Label } from "../../../../components/ui/label";
import { Input } from "../../../../components/ui/input";
import { Button } from "../../../../components/ui/button";
import { useCallback } from "react";
import { welcomePageApiService } from "../api";
import { useRouter } from "next/navigation";

type TProps = {
  onViewChange: () => void;
};

export function LoginForm({ onViewChange }: TProps) {
  type Inputs = {
    email: string;
    password: string;
  };
  const router = useRouter();

  const {
    register,
    handleSubmit,
```

```
    watch,
    formState: { errors },
  } = useForm<Inputs>();

  const onSubmit: SubmitHandler<Inputs> = useCallback(async (data) => {
    const response = await welcomePageApiService.login(data);
    localStorage.setItem("token", response.token);
    localStorage.setItem("userId", response.userId);

    router.push("/products");
  }, []);

  return (
    <form onSubmit={handleSubmit(onSubmit)} className="flex gap-5 flex-col items-center">
      <div>
        <Label htmlFor="email">Email {errors.email && <span className="text-red-500">*</span>}</Label>
        <Input {...register("email", { required: true })} />
      </div>
      <div>
        <Label htmlFor="password">Password {errors.password && <span className="text-red-500">*</span>}</Label>
        <Input type="password" {...register("password", { required: true })} />
      </div>
      <Button className="w-fit" variant="default" type="submit">
        Login
      </Button>
```

```
          <Button className="w-fit" variant="link" type="button"
onClick={onViewChange}>
            Dont have account?
          </Button>
        </form>
      );
    }
```

Компонент для відображення форми заповнення для реєстрації https://github.com/nikita-dnestrov/uni-client/blob/main/src/app/(auth)/welcome/components/RegisterForm.tsx

```
    "use client";

    import { useForm, SubmitHandler } from "react-hook-form";
    import { Label } from "../../../../components/ui/label";
    import { Input } from "../../../../components/ui/input";
    import { Button } from "../../../../components/ui/button";
    import { welcomePageApiService } from "../api";
    import { useCallback } from "react";
    import { useRouter } from "next/navigation";

    type TProps = {
      onViewChange: () => void;
    };

    export function RegisterForm({ onViewChange }: TProps) {
      type Inputs = {
        email: string;
```

```
    password: string;
    firstName: string;
    lastName: string;
    phoneNumber: string;
  };
  const router = useRouter();

  const {
    register,
    handleSubmit,
    watch,
    formState: { errors },
  } = useForm<Inputs>();

  const onSubmit: SubmitHandler<Inputs> = useCallback(async (data) => {
    const { firstName, lastName, ...rest } = data;
    await welcomePageApiService.register({ ...rest, name: `${firstName} ${lastName}`
});
    const response = await welcomePageApiService.login(data);
    localStorage.setItem("token", response.token);
    localStorage.setItem("userId", response.userId);
    router.push("/products");
  }, []);

  return (
    <form onSubmit={handleSubmit(onSubmit)} className="flex gap-5 flex-col items-
center">
      <div>
```

```
            <Label htmlFor="firstName">First Name {errors.firstName && <span
className="text-red-500">*</span>}</Label>

            <Input {...register("firstName", { required: true })} />
          </div>
          <div>
            <Label htmlFor="lastName">Last Name {errors.lastName && <span
className="text-red-500">*</span>}</Label>
            <Input {...register("lastName", { required: true })} />
          </div>
          <div>
            <Label htmlFor="email">Email {errors.email && <span className="text-red-
500">*</span>}</Label>
            <Input {...register("email", { required: true })} />
          </div>
          <div>
            <Label htmlFor="phoneNumber">
              Phone Number {errors.phoneNumber && <span className="text-red-
500">*</span>}
            </Label>
            <Input {...register("phoneNumber", { required: true })} />
          </div>
          <div>
            <Label htmlFor="password">Password {errors.password && <span
className="text-red-500">*</span>}</Label>
            <Input type="password" {...register("password", { required: true })} />
          </div>
          <Button variant="default" type="submit">
```

```
        Register
      </Button>
      <Button variant="link" type="button" onClick={onViewChange}>
        Already have an account?
      </Button>
    </form>
  );
}
```