Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мережних технологій факультету інформатики

ПОБУДОВА БАГАТОРІВНЕВОГО ВЕБ-ЗАСТОСУВАННЯ НА ХМАРНІЙ ПЛАТФОРМІ (ФУНКЦІОНАЛЬНІСТЬ - ЗА ВИБОРОМ СТУДЕНТА)

Текстова частина до кваліфікаційної роботи
за спеціальністю “Комп’ютерні науки”

Керівник кваліфікаційної роботи
ст. викладач, к.н. Черкасов Д.І.
_____*(підпис)*
“__”_____2023 р.
Виконав студент
Бучок Богдан Сергійович
“__”_____2023 р.

Київ 2023

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мережних технологій факультету інформатики

ЗАТВЕРДЖУЮ

Зав.кафедри

Гороховський С.С.

(підпис) _____

«___»_____ 2023 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на дипломну роботу

Студента Бучка Богдана Сергійовича факультету інформатики 4 курсу

ТЕМА: ПОБУДОВА БАГАТОРІВНЕВОГО ВЕБ-ЗАСТОСУВАННЯ НА ХМАРНІЙ ПЛАТФОРМ (ФУНКЦІОНАЛЬНІСТЬ - ЗА ВИБОРОМ СТУДЕНТА)

Вихідні дані: багаторівневе веб-застосування на хмарній платформі, яке забезпечує функціональність електронної комерції для продажу товарів

Індивідуальне завдання:

Вступ

1. Визначення основних понять у предметній області
2. Аналіз вимог до веб-застосування
3. Аналіз та вибір технологій, які будуть використані для розробки.
4. Опис поглибленого алгоритму для реалізації.
5. Тестування розробленого веб-застосування

Висновки

Список використаних ресурсів

Додатки

Дата видачі «___» _____ 2023 р.

Керівник _____ Завдання отримав _____

(підпис) (підпис)

Календарний план виконання роботи:

Номер	Назва етапу	Термін виконання	Примітка
1.	Отримання теми кваліфікаційної роботи	12.12.2023	
2.	Аналіз основних визначень та теоретичної частини	12.01.2023	
3.	Огляд аналогічних магазинів	19.01.2024	
4.	Аналіз актуальності питання	27.01.2024	
5.	Постановка завдання	01.02.2024	
6.	Аналіз технічного завдання	11.02.2024	
7.	Аналіз вимог до застосунку	21.02.2024	
8.	Вибір технологій розробки	25.02.2024	
9.	Створення ODM для бази даних	01.03.2024	
10.	Розробка серверної частини	07.03.2024	
11.	Розробка клієнтської частини	17.03.2024	
12.	Розгортання додатку на хмарній платформі	03.04.2024	
13.	Забезпечення	07.04.2024	

	веб-доступності інтерфейсу		
14.	Оптимізація продуктивності	17.04.2024	
15.	Автоматичне та ручне тестування	27.04.2024	
16.	Перегляд змісту роботи керівником	19.05.2024	
17.	Внесення змін та правок відповідно до зауважень керівника	21.05.2024	

Студент: Бучок Б.С.

Керівник: Черкасов Д.І.

«___»_____

ЗМІСТ

ВСТУП

Це дослідження присвячене створенню продуктивного, багаторівневого веб-застосування на хмарній платформі, яке забезпечує функціональність електронної комерції для продажу товарів.

Статистика свідчить про зростання роздрібних продажів електронної комерції в США та в усьому світі на майже 11% в період з 2023 по 2027 роки [31], а отже потреба в ефективних веб-застосуваннях для електронної комерції зростає та стає все більш актуальною.

Для розгортання додатку буде використана хмарна платформа. У епоху цифрових технологій розробка веб-додатків з використанням хмарних технологій є наріжним каменем індустрії веб технологій, адже вони надають зручний спосіб розгортання, масштабування та керування веб-додатками. Їх поширеність продовжує зростати. У 2021 році 42% підприємств ЄС використовували хмарні обчислення. Це вдвічі більше, ніж у 2016 році. [19]

Також у досліджені розглянуто практики та технології для створення доступного (accessibility rich) користувацького інтерфейсу для надання комфортного сервісу для людей з обмеженими фізичними можливостями (наприклад: вадами слуху або зору, руховими обмеженнями, т.д). Велику потребу в адаптивних технологіях створює Російсько-українська війна. А також, те, що цільова аудиторія - люди, яким вже виповнилося тридцять років, причому найбільш активні користувачі відносяться до вікової групи від 40 до 70 років.

Крім того, навіть найуспішніші українські комерційні застосунки не мають фундаментальних рішень, що сприяють їх доступності. Враховуючи важливість цієї

проблеми, активне дослідження технологій, які сприятимуть розвитку цифрової інклюзії в Україні, є необхідним.

Результати дослідження можуть слугувати відмінною основою для подальшого вивчення та вдосконалення процесу розробки продуктивних та доступних веб-додатків на хмарних платформах.

Також, дослідження дозволить краще зрозуміти можливості та виклики, пов’язані з використанням сучасних веб-технологій та хмарних платформ.

Мета дослідження: розробка продуктивного багаторівневого веб-застосування на хмарній платформі, яке забезпечує функціональність електронної комерції для продажу товарів.

Задача включає в себе розробку систем для відображення, фільтрації за характеристиками, пошуку, рекомендацій та купівлі товарів, а також можливості реєстрації та управління власним обліковим записом користувача.

Додатково, робота полягає в реалізації інтуїтивно зрозумілого та динамічного користувацького інтерфейсу. Особливу увагу приділяється його адаптації для людей з будь-якими обмеженими фізичними можливостями.

Об’єктом дослідження є багаторівневе веб-застосування, створене на хмарній платформі.

Ця кваліфікаційна робота складається з п’яти розділів та висновку.

У першому розділі надається визначення основним поняттям, таким як: Багаторівневий веб-застосунок, Хмарна Платформа, RESTful API, Веб-доступність (Accessibility)

У другому розділі виконується аналіз вимог до веб-застосування

У третьому розділі обираються технології, які будуть використані для розробки.

У четвертому розділі описується поглиблений алгоритм для реалізації.

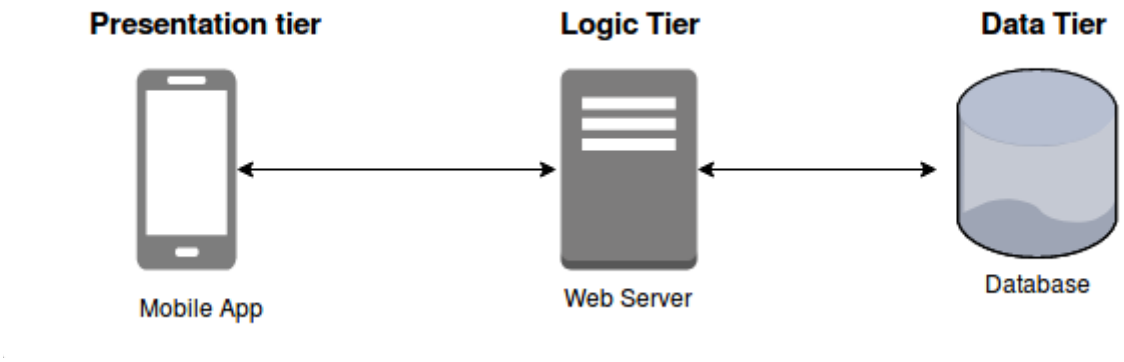
У п’ятому розділі проводиться тестування розробленого веб-застосування та описуються рекомендації щодо подальшого вдосконалення

У висновку подається підсумок проведеного дослідження.

РОЗДІЛ 1. ВИЗНАЧЕННЯ ТА АНАЛІЗ ОСНОВНИХ ПОНЯТЬ

1.1 Багаторівневий веб-застосунок

Багаторівневий веб-застосунок (рис. 1) складений з двох або більше рівнів. Кожен з рівнів виконує свою визначену функцію. Така архітектура сприяє легкій масштабованості, безпеці та гнучкості в розробці застосунку.



Однорівневі, на противагу, тісно інтегрують всі рівні системи в одному застосунку. Ця архітектура переважала, коли комп’ютери мали малу обчислювальну потужність. Однак зі зростанням складності програмних систем і збільшенням вимог користувачів стали очевидними обмеження однорівневої архітектури. [1]

1.2 Хмарна платформа

Хмарні платформи, які є підмножиною хмарних технологій, надають обчислювальні ресурси серверів, через мережу, на підставі плати за використання. Вони дозволяють швидко масштабувати ресурси без необхідності власного фізичного обладнання та обслуговування власних дата-центрів. Також, забезпечують розробників всіма необхідними інструментами для створення, тестування, управління, моніторингу та розгортання додатків.

Основні моделі сервісів хмарних платформ включають: **IaaS (**інфраструктура як послуга), **PaaS** (платформа як послуга), **SaaS** (програмне забезпечення як послуга) [23]

Приклади популярних хмарних платформ: Cloudflare, Vercel, Amazon Web Services, Microsoft Azure, Google Cloud Platform

Види моделей хмарних платформ:

- **Serverful**: Це традиційна модель, в якій розробники мають повний контроль над серверами та їх конфігурацією. Вони відповідають за управління, масштабування та обслуговування серверів. Це може бути часо-затратним, але надає більше гнучкості.
    - Плюси:
        - Повний контроль над сервером і його налаштуванням.
    - Мінуси:
        - Потрібен час і досвід для керування, масштабування та обслуговування серверів.
        - Може бути дорогим з точки зору ресурсів і робочої сили.
- **Serverless** - модель, у якій хмарний провайдер відповідає за управління та обслуговування серверів
    - Плюси:
        - Розробники можуть зосередитися на написанні коду, не турбуючись про керування сервером.

        - Ресурси автоматично масштабуються відповідно до потреб.
    - Мінуси:
        - Менше контролю над сервером і його конфігурацією.
        - Можливість збільшення затримки через час, потрібний для запуску нового екземпляра функції (також відомий як «холодний запуск»).
- **Managed**: - це також модель, у якій хмарний провайдер відповідає за управління та обслуговування серверів. Однак, є певні особливості: можна вибирати додаткові послуги, такі як: бази даних, кешування, пошук тощо, а також доступна більш гнучка конфігурація порівняно з serverless [34].

Кожна з цих архітектур має свої переваги та недоліки, тому вибір залежить від конкретних потреб та цілей проекту.

1.3 RESTful API:

Це архітектурний стиль для побудови веб-сервісів, який дозволяє взаємодіяти з додатками за допомогою стандартних HTTP-запитів. REST API є гнучким та підтримують передачу даних у різних форматах, включаючи XML, HTML, звичайний текст, JSON тощо. Також, це популярний вибір для розробки багаторівневих веб-застосунків через свою простоту імплементації.

Існує кілька інших архітектурних стилів, кожен має свою унікальну філософію дизайну та потенційні сфери застосування. Однак, варто врахувати, що кожен з них може мати певні обмеження при реалізації проекту:

- **SOAP** - незважаючи на високий рівень надійності, SOAP може бути надлишковим через свою складність, потреби більше ресурсів та неможливості інтеграції з JSON.
- **GraphQL** - цей стиль може бути надмірним для більш простих програм. Він також вимагає більшої обробки на стороні сервера.
- **gRPC** - має обмежену підтримку в браузерах.
- **WebSocket** - є недоречним, якщо програма не вимагає обробки даних у реальному часі.
- **Webhook** - є не найкращим вибором, якщо потрібен синхронний зв’язок або негайна відповідь.

1.4 Веб-доступність (accessibility)

Веб-доступність означає, що веб-сайти, інструменти та технології розроблені таким чином, щоб ними могли користуватися люди з обмеженими можливостями. Зокрема, можуть сприймати, розуміти, орієнтуватися в Інтернеті та взаємодіяти з ним, а також

що вони можуть робити свій внесок у Веб [22].

Веб-доступність охоплює всі обмеження, які впливають на доступ до Інтернету, зокрема: слухові, когнітивні, неврологічні, рухові, мовні та візуальні.

Веб-доступність також приносить користь людям без обмежених можливостей, наприклад:

- люди похилого віку, здібності яких змінюються внаслідок старіння
- люди з «тимчасовими обмеженнями», наприклад такими як перелом руки

1.5 Веб Робітники (Web Workers)

У веб-додатках існують три типи працівників:

Веб-робітники (Dedicated Workers) - це фонові потоки на веб-сторінці, які можуть бути запущені головним. Кожен з працівників працює у власному, ізольованому потоці, що дозволяє виконувати задачі з великим навантаженням та довгим часом виконання без блокування основного потоку і, в свою чергу, інтерфейсу користувача.

Спільний робітник (Shared Workers) - це тип веб-робітника, який може бути спільно використаний у кількома контекстами одночасно. Він може обслуговувати різні вікна, iframe або навіть інших робітників. Це робить його надзвичайно гнучким і ефективним інструментом для веб-розробки.

Сервісний робітник (Service Worker) - це тип веб робітника із життєвим циклом, який повністю відокремлений від сторінки. Вони діють як проксі-сервери між веб-додатком, браузером і мережею (якщо доступно). Це дозволяє їм перехоплювати та змінювати навігацію та запити ресурсів, що дозволяє їм контролювати сайт. Також він може кешувати ресурси для завершення роботи в офлайні.

Кожен тип воркера має власні випадки використання та може значно покращити продуктивність веб-програми, дозволяючи виконувати завдання без сповільнення основного потоку.

ВИСНОВОК

Отже, в даному розділі було розглянуто ключові поняття, які є важливими для розуміння та розробки багаторівневих веб-застосунків. Було надано визначення таким поняттям, як “Багаторівневий веб-застосунок”, “Хмарна платформа”, “RESTful API”, “Веб-доступність” та “Веб Робітники (Web Workers)”.

Багаторівневий веб-застосунок було описано як систему, що складається з двох або більше рівнів, кожен з яких виконує свою визначену функцію. Така архітектура сприяє легкій масштабованості, безпеці та гнучкості в розробці застосунку.

Хмарна платформа була визначена як підмножина хмарних технологій, що надають обчислювальні ресурси серверів через мережу на підставі плати за використання. Вони дозволяють швидко масштабувати ресурси без необхідності власного фізичного обладнання та обслуговування власних дата-центрів.

RESTful API було описано як архітектурний стиль для побудови веб-сервісів, який дозволяє взаємодіяти з додатками за допомогою стандартних HTTP-запитів.

Веб-доступність була визначена як концепція, за якої веб-сайти, інструменти та технології розроблені та розроблені таким чином, щоб ними могли користуватися люди з обмеженими можливостями.

Було розглянуто також три типи веб-робітників: веб робітник **(Web worker або Dedicated Worker)**, спільний робітник **(Shared Workers)** та сервісний працівник **(Service Worker)**. Їх застосування може значно покращити продуктивність веб-застосунку.

РОЗДІЛ 2: АНАЛІЗ ВИМОГ ДОДАТКУ

Для визначення інструментів розробки застосунку важливо проаналізувати такі ключові аспекти як:

2.1 Функціональність системи

Визначення основних функцій і можливостей, які повинен підтримувати додаток:

- Перегляд каталогу товарів з пагінацією
- Фільтрування асортименту за унікальними характеристиками товарів. Сортувати за ціною або рейтингом.
- Здійснити **пошук** товарів за ключовими словами, наприклад: “для котів”
- Можливість переглянути рекомендації схожих товарів
- Переглянути індивідуальну сторінку кожного з товарів (його опис, характеристики, ціну, відгуки, рекомендації)
- Додати товар до кошику та здійснити покупку
- Додати товар до бажаних та перегляд бажаних товарів
- Створення та управління особистим обліковим записом користувача

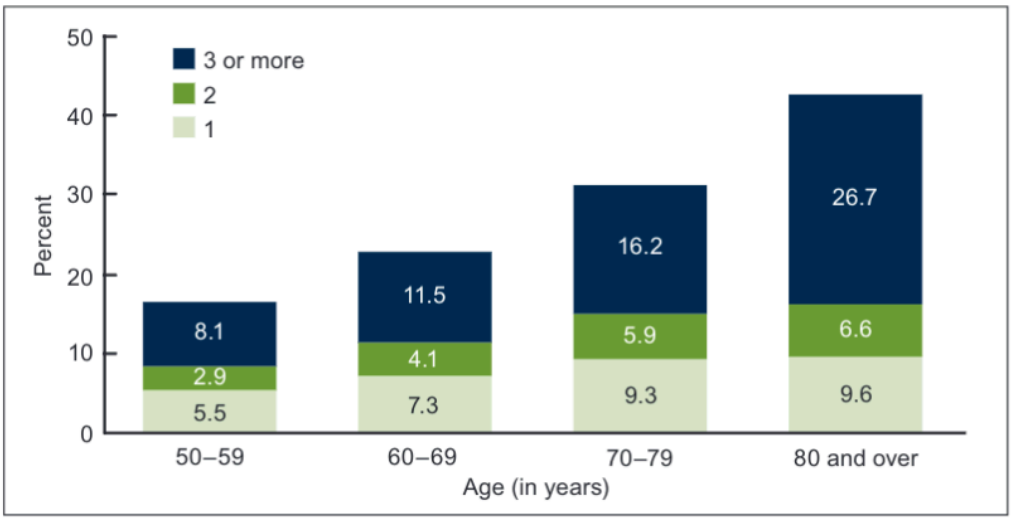
2.2 Цільова аудиторія

Для успішної адаптації нашого додатку до потреб користувачів, важливо визначити цільову аудиторію. Асортимент представлений застосунку - це товари для садівництва, господарства та домашніх улюбленців. Отже, цільова аудиторія - люди, яким вже виповнилося тридцять років, причому найбільш активні користувачі відносяться до вікової групи від 40 до 70 років, оскільки саме користувачі такого віку найбільше цікавиться подібними товарами.

За офіційною статистикою США багато людей набувають більше фізичних обмежень із віком. Причому, бачимо, що відсоток людей, що мають кілька фізичних

обмежень сильно збільшується з віком, результуючи в 8.1% у віці від 50 до 59 років та в 11.5% від 60 до 69 років [24].

Figure 1. Percentage of adults aged 50 years and over, by age and number of physical limitations: United States, 2001–2007



Також, якщо говорити про статистику, то інформаційний бюлетень Всесвітньої організації охорони здоров'я «Інвалідність і здоров'я» стверджує, що «понад мільярд людей, приблизно 15% населення світу, кожен шостий, мають певну форму інвалідності», а «від 110 до 190 мільйонів дорослих мають значні труднощі у життєдіяльності» [32].

Враховуючи це, ми повинні забезпечити високу доступність користувацького інтерфейсу для людей, які можуть мати обмежені певні фізичні можливості (або навіть їх комбінацію).

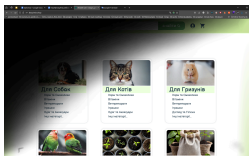
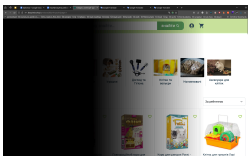
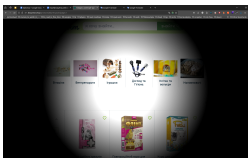
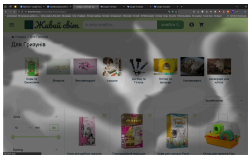
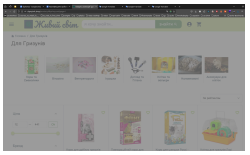
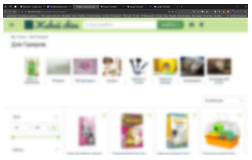
Застосунок повинен бути інтуїтивно зрозумілим, з легкою навігацією та чіткою логікою користування, щоб кожен користувач міг з легкістю знайти потрібний товар.

2.3 Адаптивність у користувацькому інтерфейсі

Важливо чітко визначити та проаналізувати **потреби людей**, які мають будь-які слухові, когнітивні, неврологічні, рухові, мовні або візуальні порушення або обмеження.

2.3.1 Вади зору

Вади зору включають короткозорість, далекозорість, часткове або повне блокування зору, астигматизм (розмитість), кольорова сліпота, “подвійне бачення”, “хмарність” та інші. Використовуючи розширення No Coffee для Firefox Mozilla, можна змоделювати ці порушення зору [27]. Це допоможе краще зрозуміти, що саме потрібно реалізувати для покращення доступності та зручності використання додатку для людей з візуальними порушеннями

 	 	
Периферійне блокування бачення	Центральне блокування бачення	“Подвійне бачення”
 		
“Хмарність”	Втрата контрастності	Розмитість

Отже, для покращення досвіду користування необхідно:

Для розмитості, втрати контрастності та “Хмарності”:

- Використання чітких шрифтів великого розміру та контрастних кольорів для тексту та фону. (**в різних освітленнях та в старості, коли сприйняття втрачається**)
- Передання стану додатку за допомогою різних способів(кольору, тексту, компонентів). Кольори можуть передавати значення (наприклад, червони для помилки), але покладання виключно на колір є недостатнім для користувачів з

вадами зору.

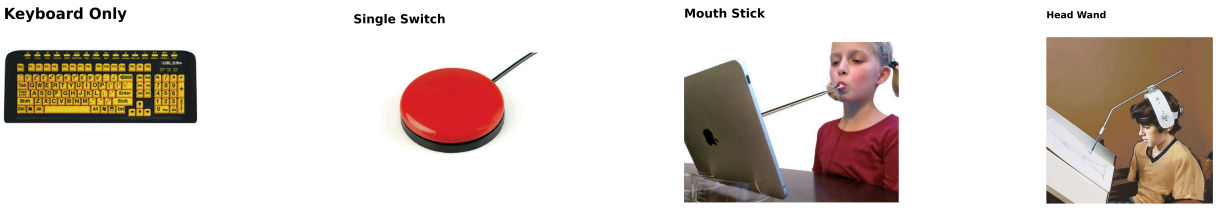
- Забезпечення адаптивності користувацького інтерфейсу задля можливості збільшення елементів

Для часткового, периферійного або центрального блокування зору:

- Розміщення елементів, які відображають стан додатку у передбачуваних місцях або їх повторення (наприклад, повторення візуального компоненти завантаження по центру сторінки та зверху сторінки)

2.3.2 Рухові обмеження чи повне блокування зору

Приклади того, як люди з руховими порушеннями можуть взаємодіяти з веб-застосунками:



Для людей, що використовують клавіатуру чи одну кнопку для навігації: (а також для людей, що використовують читач екрану за відсутності зору):

- Забезпечити сумісність з читачами екрану (наприклад: Orca, JAWS, NVDA, VoiceOver і т.д.).
    - Створення альтернативний текст для зображень та елементів, що можна сфокусувати
    - Використання семантичного html для надання

Також, варто зазначати, що розробка сумісності з читачами екрану корисно для людей, що мають дислексію або проблеми з читанням.

- Створити кнопку швидкої навігації до головного вмісту сторінки
- Створити гарячі клавіші для навігації, фокусування та взаємодії з додатком
- Надати можливість зручно навігуватись до всіх інтерактивних елементів за допомогою клавіатури (створення унікальної мапи навігації для кожної сторінки, дотримання стандартів WAI-ARIA для створення доступних компонентів (акордеон, сітка, кнопки, модальні вікна та ін.) користувацького інтерфейсу [26])

2.3.3 Когнітивні обмеження

Когнітивні порушення стосуються широкого спектру інвалідності, від людей з розумовими вадами, які мають найбільш обмежені можливості, до всіх нас, коли ми старіємо і маємо труднощі з мисленням і запам'ятовуванням. [25]

Необхідно реалізувати:

- Легко зрозумілий вміст, наприклад текст, написаний з використанням простих стандартів мови.
- Концентрація уваги на важливому контенті.
- Зведення до мінімуму відволікаючих факторів, таких як непотрібний вміст або реклама.
- Послідовний макет веб-сторінки та навігація.
- Знайомі елементи, такі як підкреслені посилання, сині, коли вони не відвідані, і фіолетові, коли вони відвідані.
- Поділ процесів на логічні, важливі кроки з індикаторами прогресу.
- Автентифікація веб-сайту максимально проста без шкоди для безпеки.
- Спрощення заповнення форм, наприклад, з чіткими повідомленнями про помилки та простим відновленням помилок. [25]

2.4 Прийняття до уваги аспектів електронної комерції як основного призначення платформи

Створення платформи для електронної комерції вимагає врахування конкурентоздатності з іншими веб-магазинами. Це надзвичайно важливо, адже успіх бізнесу в значній мірі залежить від того, наскільки якісним є сервіс.

Кожен з нижче проаналізованих аспектів допомагає привернути та утримати клієнтів, що, в свою чергу, веде до збільшення продажів та прибутку.

2.4.1 Доступність

Доступність допомагає виділитись серед конкурентів, адже забезпечує:

- Оптимізацію SEO, оскільки пошукові системи, віддають перевагу сайтам, які:
    - застосовують семантичний HTML
    - оптимізовані для мобільних пристроїв.
- Можливість користування для тих, хто має повільне підключення до Інтернету
- Легкість користування тим, хто не володіє достатніми навичками користування вебом або технологіями узагалі.
- Покращення суспільного іміджу платформи, адже турбота про доступність демонструє добру етику та мораль [25].

2.4.2 Продуктивність

Продуктивність є критичним компонентом як для оптимізації SEO, так і для забезпечення високоякісного користувацького досвіду, що безпосередньо впливає на кількість трафіку на платформі.

Отже, для досягнення конкурентної переваги серед інших ресурсів, необхідно визначити та проаналізувати вимоги до метрик продуктивності застосунку

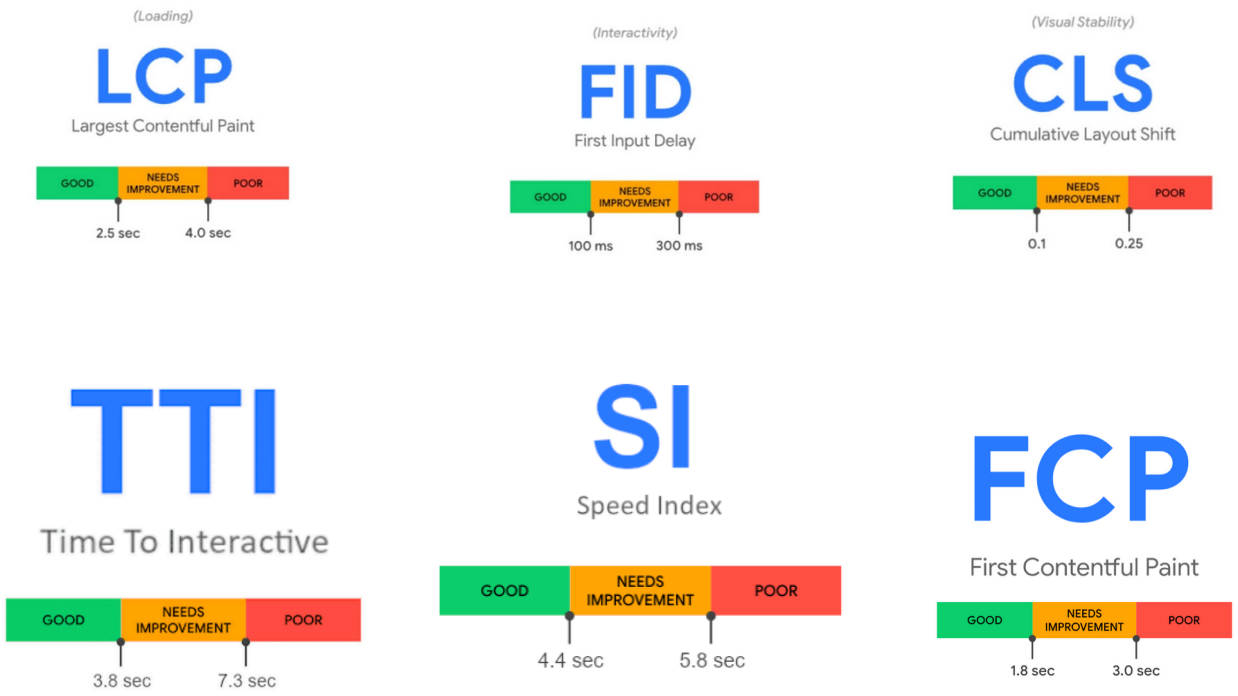
Ключові метрики, на які потрібно звернути увагу:

- **Page Load Time** - час, необхідний для завантаження всієї вмісту сторінки

  Відвідувачі веб-сайтів очікують, що сторінка завантажується менше ніж за три секунди. Кожне затримання в часі завантаження може вплинути на конверсію до 20% [3].

- **Core Web Vitals -** набір метрик, які Google вважає ключовими метриками для розуміння якості взаємодії користувачів з веб-сторінкою. Основні з них:
  - **Largest Contentful Paint (LCP)** - час для завантаження найбільшого ресурса на сторінці
  - **First Input Delay (FID) -** час від момента взаємодії з додатком до його відповіді
  - **Cumulative Layout Shift (CLS) -** метрика, що вимірює стабільність макету сторінки під час її завантаження
  - **Time to Interactive (TTI)**: Це час, який потрібен сторінці, щоб з нею була можливість взаємодіяти
  - **First Contentful Paint (FCP)**: час, перш ніж, користувач бачить перший вміст, що буде відображений на сторінці
  - **Speed Index (SI)**: Це метрика, яка показує, як швидко вміст сторінки візуально відображається.

    На наведеному нижче рисунку представлено показники метрик, які класифіковані як хороші, ті, які потребують покращення, та погані.



Оптимізація метрик вимагатиму детального аналізу та вдосконалення різних аспектів застосунку, однак це суттєво посприяє поліпшенню користувацького досвіду й підвищує сайт у ранжуванні пошуку Google.

2.4.3 SEO

Для того, аби краще ранжуватись у пошуку за інші веб-магазини необхідно реалізувати наступне:

- Використання ключових слів у заголовках, мета-тегах, URL та контенті
- **SSR** (рендеринг сторінки на стороні сервера)**:**
    - Оскільки, весь контент буде видимим для пошукових ботів вони зможуть краще індексувати його
    - Server-side-rendered content завантажується швидше, ніж client-side-rendered, що також впливає на ранжування

2.5 Забезпечення легкої масштабованості:

Потрібно також врахувати майбутнє розширення функціоналу та зростання кількості користувачів. Для досягнення легкої масштабованості у майбутньому, потрібно зосередитися на декількох ключових аспектах:

2.5.1 Кешування

Кешування може зменшити навантаження на базу даних та покращити час відгуку. Використання кешування на рівні додатку або бази даних може допомогти зменшити час відгуку та збільшити продуктивність.

2.5.2 Балансування

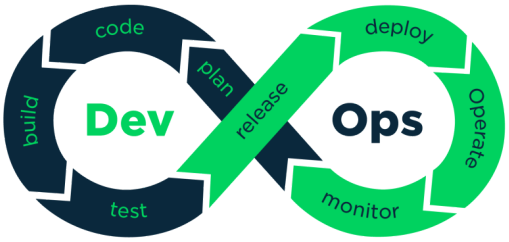
Використання балансувальників навантаження може допомогти розподілити навантаження між різними серверами та покращити доступність застосування.

2.6 Ефективні практики для створення веб-додатків

Такі практики допомагають масштабованості, надають впевненість в коректності коду у codebase, надають інструменти для швидкого реагування на помилки

2.6.1 CI/CD

Неперервні інтеграція та розгортання (CI/CD) дозволяють автоматизувати процеси збірки, тестування та розгортання. Це прискорює цикл розробки, зменшує ймовірність помилок та забезпечує швидке та надійне оновлення застосування.



2.6.2 Лінтування

Лінтування – це процес аналізу коду на наявність можливих помилок та потенційних проблем.

- Допомагає підвищити загальну якість коду та зробити його зручнішим для обслуговування, економлячи час на розробку
- Встановлює стиль кодування, гарантуючи, консистентність коду у всьому проєкті
- Допомагає дотримуватися найкращих практик розробки. Якщо не дотримуватись цих практик, лінтер видасть помилку.

2.6.3 Форматування

Форматування коду забезпечує:

- Читабельність
- Допомагає розробникам зосередитися на написанні коду, а не на його форматуванні, знижуючи час на розробку.
- Забезпечує **консистентність** стилю кодування в команді розробників, незалежно від особистих уподобань кожного розробника.

2.6.4 Моніторинг

Автоматизовані системи моніторингу допомагають відстежувати стан додатку та виявляти проблеми.

- Необхідно реалізувати перевірку додатку перед його розгортанням в production середовище на його продуктивність та коректність.

- Потрібно впровадити систему логування помилок та запитів, що закінчились невдачею. Це допоможе попередити про потенційні проблеми та вузькі місця, перш ніж вони зможуть вплинути на користувачів.

2.7 Безпека

Використання наступних практик необхідне для забезпечення безпеки веб-застосунку та цілісності даних користувачів.

- **Захист даних користувачів**
  Важливо забезпечити конфіденційність, цілісність та доступність даних користувачів. Важливо реалізувати хешування паролів користувачів, використання HTTPS, а також регулярного резервного копіювання бази даних.
- **Захист від хакерських атак**
  Веб-застосунки часто стають мішенями для хакерських атак, таких як SQL Injection, Cross-Site Scripting (XSS) та Cross-Site Request Forgery (CSRF). Для захисту від таких атак потрібно валідацію вхідних даних як на front-end, так і на backend та інші техніки безпеки.
- **DevSecOps**: Інтеграція безпеки в життєвий цикл додатку включає в себе використання автоматизованих інструментів для виявлення та виправлення проблем безпеки на ранніх стадіях розробки.
  - Автоматичне виявлення та блокування відомих вразливостей у 3th party залежностях (за допомогою Dependabot на GitHub, Dependabot is Software Composition Analysis (SCA) tool [28])
  - Виявлення та блокування витоку конфіденційної інформації в репозиторії github (secret management)

ВИСНОВОК

У цьому розділі було проаналізовано ключові аспекти, які слід врахувати при розробці мобільного додатку для продажу товарів для садівництва, господарства та домашніх улюбленців.

Було визначено всю функціональність для платформи електронної комерції. Виконано аналіз сучасних практик розробки, таких як CI/CD, лінтування, форматування та моніторинг, що допоможуть провести реалізацію роботи швидше і якісніше.

Було визначено цільову аудиторії та потреби користувачів, що можуть мати фізичні обмеження. Визначено, що інтерфейс додатку повинен бути доступним для людей з вадами зору, слуху, руховими обмеженнями та когнітивними порушеннями.

Додатково було проаналізовано вимоги до того, як забезпечити надійний захист від загроз у безпеці, продуктивність та легку масштабованість додатку у майбутньому.

Досягнення поставлених вимог допоможе створити успішний мобільний додаток, який буде затребуваний цільовою аудиторією

РОЗДІЛ 3: ВИБІР ТЕХНОЛОГІЙ

3.1 Технологічний стек

Комбінація технологій, що використовуються для створення додатків, називається стеком технологій. Для проєкту було обрано MERN (MongoDB, Express, React, Node.js) стек

Чому був обраний саме MERN стек:

- Розробка веб-додатку виконується єдиною мовою програмування **JavaScript**. Це забезпечує уніфікацію коду та спрощення написання програми. Особливо, це корисно, якщо над додатком працює одна людина, адже перехід від написання від серверу до клієнта непомітний та безшовний, що покращує продуктивність.

  Також, варто зазначити, що JavaScript є найпопулярнішою мовою програмування серед розробників по всьому світу, з часткою використання близько 65% [7]. JavaScript є ключовою технологією у веб-розробці сьогодення. Його популярність забезпечує широкою підтримкою, великою спільнотою та екосистемою.

- React дозволяє створювати динамічні **SPA** (Single Page Applications) з швидким відгуком без необхідності перезавантаження сторінки. На відміну від LAMP (не підтримує взагалі) та Ruby on Rails (частково підтримує, але необхідні розширення фреймфо для цього) **.**

- Досвід розробки на цьому стеку також надає можливість легко писати нативні додатки для мобільних пристроїв, замінивши React на **React Native**. React Native - це фреймворк для створення нативних мобільних додатків за допомогою JavaScript та React. Він дозволяє розробникам використовувати

один і той же код для Android та iOS, що значно спрощує процес розробки та підтримки мобільних застосунків. [5]

Переваги MERN над іншими стеками технологій, такими як: MEAN, LAMP, MEVN, Ruby on Rails, Django, т.д.

- **MEAN** використовує Angular для фронтенду, який є більш складним для вивчення порівняно з React.
- **LAMP** вимагає знання різних мов програмування, таких як PHP, що може ускладнити розробку.
- **MEVN** аналогічний до MERN, але використовує Vue.js замість React, який має меншу спільноту та екосистему порівняно з React.
- **Ruby on Rails** та **Django** пропонують швидку розробку, але вони використовують Ruby та Python відповідно, що вимагає від розробників знання додаткових мов.

3.2 База даних

База даних — це структуроване сховище даних, яке використовується для зберігання, організації та управління інформацією. У якості бази даних використовується MongoDB Atlas.

MongoDB Atlas - це database-as-a-service platform (**DBaaS**), набір хмарних баз даних і служб даних, призначених для прискорення та спрощення роботи з даними.

Сервіс дозволяє легко налаштовувати, експлуатувати та масштабувати розгортання баз даних MongoDB у хмарі. Він є інтуїтивно зрозумілий та високоавтоматизований.

Чому був обраний саме MongoDB Atlas:

- Завдяки NoSQL (документо-орієнтованій) моделі, MongoDB Atlas дозволяє:

    - Високу швидкість читання та запису (шардування, відсутність JOIN операцій, денормалізація, т.д.), що є важливим для e-commerce застосунків, де швидкість відгуку є критичною.
    - Використовувати гнучкі моделі даних, які надають можливість легко змінювати та зберігати різноманітні типи даних. Це забезпечує адаптивність до змін, підвищує швидкість розробки, забезпечує гнучкість у відповіді на будь-які вимоги та загалом спрощує роботу з даними.
    - Легко масштабуватись горизонтально (шардування), що дешевше та легше при масштабуванні. А також, це забезпечує кращу швидкість, більшу надійність та ефективне розподілення навантаження, адже дані зберігаються на багатьох серверах.

- Уніфікований API Atlas Mongodb запитів спрощує роботу з даними, забезпечуючи безперебійну інтеграцію незалежно від обраної хмарної платформи.
- Використовує BSON для зберігання документів. BSON (Binary JSON) є бінарною версією JSON. BSON оптимізований для швидкості сканування та вилучення даних, що робить його швидшим для обробки та передачі мережею.

  Крім того, BSON є розширенням JSON, що дозволяє легко перетворювати BSON на JSON та навпаки. Це дозволяє працювати з даними у форматі JSON на бекенді, а MongoDB автоматично перетворює їх у BSON для зберігання та обробки, що значно спрощує розробку

- Забезпечує кілька рівнів безпеки. Вони включають надійний контроль доступу, мережеву ізоляцію за допомогою Amazon VPC та VPC Peering, білі списки IP-адрес, шифрування даних під час передачі за допомогою TLS/SSL та додаткове шифрування основної файлової системи. [8]

- MongoDB Atlas надає інструменти для моніторингу продуктивності бази даних в реальному часі, що дозволяє швидко реагувати на аномалії.

3.3 Технології розробки бекенду

Бекенд — це серверна частина програмного забезпечення, яка основну бізнес-логіку додатку, обробляє запити від користувачів, відповідає за взаємодію з базами даних, зовнішніми сервісами та іншими ресурсами.

Бекенд взаємодіє з фронтендом, передаючи дані через API (Application Programming Interface) для подальшого відображення користувачеві на стороні клієнта.

Вибрані технології включають:

3.3.1 Node.js

Надає середовище виконання для виконання JavaScript на стороні сервера, відоме своїм неблокуючим вводом/виводом та архітектурою, керованою подіями. Це дозволяє серверу обробляти інші запити, поки очікує відповіді на попередній запит [9].

Чому був обраний саме Node.js:

- Використовує потужний двигун V8 від Google Chrome для компіляції JavaScript коду в оптимізований машинний код, що дозволяє швидко виконувати скрипти [9].
- Підтримує модульну структуру, завдяки чому розробники можуть легко інтегрувати різноманітні бібліотеки та інструменти та масштабувати застосунки.
- Велика та активна спільнота сприяє створенню великої кількості модулів та пакетів для node.js.
- Полегшує розробку крос-платформних додатків, які можуть працювати на різних операційних системах, заощаджуючи час і ресурси при розробці.

3.3.2 Express.js

Express.js — популярний веб-фреймворк для Node.js, відомий своєю простотою та мінімалізмом. Виступає проміжним програмним забезпеченням, яке полегшує створення надійного API і обробляє запити та відповіді між базою даних і фронтендом [10].

Чому був обраний саме Express.js:

- Express.js надає базовий набір функцій без зайвих навантажень, що робить його швидким і ефективним у розробці.
- Підтримка проміжного програмного забезпечення: пропонує широкий спектр вбудованих функцій проміжного програмного забезпечення та можливість створювати власні [10].
- Має потужну систему маршрутизації для обробки різних HTTP-запитів для різних URL-адрес [10].
- Має вбудовану підтримку популярних баз даних і можливість інтегрувати інші модулі бази даних Node.js.

3.4 Технології розробки фронтенду

Фронтенд — це сегмент програмного забезпечення, який відображає інтерфейс та взаємодіє з користувачем. Відповідає за візуальний аспект програми, її дизайн, розміщення елементів, кольори, шрифти та загальний вигляд. А також, взаємодіє безпосередньо з користувачем, обробляючи його дії та реагуючи на події, такі як кліки мишкою або натискання клавіш.

Вибрані технології включають:

3.4.1 React.js

Пропонує архітектуру на основі компонентів, що надає динамічний користувацький досвід та дозволяє перевикористовувати одні й ті самі компоненти для створення інтерфейсу користувача.

Чому був обраний саме React.js:

- Найпопулярніший front-end бібліотека для JavaScript. React має велику спільноту розробників та багатий набір готових компонентів, що сприяє швидкій розробці. Близько 35.9% фронт-енд розробників надають перевагу React [11].
- Використовує віртуальну об’єктну модель документа (Virtual DOM) - полегшена копія справжньої DOM, яка є структурою, яку браузери використовують для відтворення веб-сторінок [12].

  Коли стається зміна стану додатку, React спочатку оновлює віртуальну DOM. Порівнявши зміни внесені у Virtual DOM з справжнім, він оновлює лише ті частини DOM, які дійсно були змінені, замість оновлення всього дерева елементів. Це серйозно оптимізує процес та прискорює його.

- JSX або JavaScript XML є розширенням синтаксису для JavaScript, яке використовується в бібліотеці React для опису структури інтерфейсу користувача. Воно дозволяє писати HTML-подібний код прямо в JavaScript-файлах [12]. Це полегшує візуалізацію та роботу з інтерфейсом користувача в коді JavaScript, що може призвести до швидшого часу розробки

3.4.2 Next.js

Next.js — це фреймворк для React, який спрощує веб-розробку. Він розширює можливості React таким ключовими функціоналом як SSR, SSG (static site generation), ISR (incremental static generation), автоматична оптимізація, автоматична конфігурація react та інші [30].

Чому був обраний саме Next.js:

- Легкість налаштування та використання
- Завдяки SSR (Server Side Rendering), застосунок може знаходитись у топі пошуку за основними ключовими словами. Це робить Next.js ідеальним вибором для веб-сайтів, які залежать від SEO.
- Вбудована оптимізація:
    - зображень
    - динамічного імпорту компонентів
    - кешування
    - передзавантаження контенту.
    - розбиття коду (code splitting), next.js автоматично розбиває код на різні пакети, які завантажуються за потреби
    - шрифтів

- Next.js має вбудовану маршрутизацію на основі файлової системи, тоді як у React.js вам знадобиться використовувати додаткову бібліотеку, як-от React Router.
- Підтримує бажані методи стилів, включаючи модулі CSS, Tailwind CSS і CSS-in-JS1.
- Next.js має дуже велику та активну спільноту, багато ресурсів для навчання, включаючи офіційну документацію, що робить його доступним для розробників різного рівня.

Однак, використання Next.js має свою ціну. На відміну від React, які є набором статичних файлів (HTML, CSS, JavaScript) після процесу побудови і не потребують віртуального приватного сервера (VPS), додаток Next.js потребує серверного середовища для рендерингу на його сервера.

3.4.3 Технології для стилізації

Sass (Syntactically Awesome Style Sheets) modules

Дозволяє впорядковувати каскадні таблиці стилів та зробити їх більш читабельними за допомогою змінних, функцій та міхінів (частинка коду, яку можна вставити куди завгодно) що робить код більш зручним для підтримки [14].

Чому був використаний SASS:

Крім того, використання css препроцессора, такого як Sass також надає наступні переваги:

- Sass **дозволяє укладати селектори** всередину інших селекторів. Це робить код більш організованим та читабельним порівняно з css
- Sass **дозволяє імпортувати** інші Sass файли в основний файл, що дозволяє розбити код на декілька файлів
    - CSS також має @import, однак в CSS, директива @import мусить бути лише вказана на початку документа, а також css завантажує кожен імпортований файл окремо, на відміну від sass, котрий оптимізовує процес завантаження, об’єднуючи його для всіх файлів одразу
- **Sass** має велику спільноту розробників та багато ресурсів для навчання.

MUI

MUI - це потужна бібліотека компонентів спеціально розроблена для React. MUI

надає широкий спектр готових компонентів для створення користувацького інтерфейсу, включаючи кнопки, карточки, діалогові вікна, меню, іконки та багато іншого.

Проект спочатку використовував bootstrap і react-bootstrap, але потім через ефективність, продуктивність і доступність бібліотека була змінена на mui.

Чому був обраний саме MUI:

- Має дуже велику бібліотеку готових компонентів, порівнюючи з іншими React орієнтованими бібліотеками компонентів
- MUI включає доступність і зручність використання в дизайні своїх компонентів за замовчуванням.

  Наприклад, модальні вікна реалізують Tab Trapping (це техніка, яка дозволяє користувачам клавіатури навігуватись лише в межах випадаючих меню або модальних вікон, перехоплюючи подію tab, щоб вона циклічно перемикалася в межах визначеного контексту), коли з bootstrap - це потрібно реалізовувати самотужки
- MUI регулярно оновлюється.
- MUI має велику спільноту користувачів, які надають зовнішню підтримку та приклади. Він також пропонує розширену підтримку щодо проблем і помилок.
- MUI надає більш прямолінійній та гнучкий процес кастомізації та налаштування дизайну компонентів, ніж Bootstrap та React-Bootstrap, що дозволяє створювати унікальний вигляд додатків, зберігаючи швидкість розробки.
- Дозволяє легко створювати адаптивний інтерфейс, який виглядатиме професійно, як на мобільних телефонах, так і на великих моніторах.
- MUI і Next.js добре працюють разом завдяки підтримці серверного рендерингу, що поліпшує продуктивність та SEO. Натомість, компоненти

Bootstrap, що використовують jQuery та Javascript не мають вбудованої підтримки серверного рендерингу, що може призвести до проблем зі стилями при використанні з Next.js.

- Material UI (MUI) зазвичай вважається більш легким, ніж Bootstrap. Bootstrap відомий тим, що він більш об’ємний, з мінімальним розміром файлу 49 КБ у JavaScript і розміром файлу CSS 137 КБ. Його залежності, такі як jQuery, також можуть вплинути на загальний розмір фронтенду, потенційно впливаючи на продуктивність.

  З іншого боку, інтерфейс Material UI оптимізовано для продуктивності та пропонує такі функції, як відкладене завантаження та поділ коду, щоб зменшити початковий розмір пакета та скоротити час завантаження [29].

- Bootstrap був розроблений до появи React і не використовує його потенціал. Це може призвести до проблем інтегрування Bootstrap з React. Зазвичай для використання Bootstrap використовують спеціальні проміжну бібліотеку, таку як як **react-bootstrap**, що огортає Bootstrap-компоненти в React-компоненти.

  Однак, використання проміжних бібліотек, таких як react-bootstrap, може додатково збільшити розмір додатку, що зробить його повільнішим. Крім того, такі “бібліотеки-обгортки” не йдуть нога в ногу з останніми оновленнями Bootstrap.

3.4.3 Технологія для авторизації

NextAuth.js

Обрана за простоту інтеграції з Next.js та налаштування, бібліотека підтримує різні провайдери автентифікації та ефективно обробляє сеанси і токени [15].

Також NextAuth.js був обраний через те, що:

- Легко інтегрується з OAuth службами Google, Facebook, Apple, GitHub та іншими. Він підтримує OAuth 1.0, 1.0A, 2.0 та OpenID Connect [15].

- NextAuth.js розроблений з урахуванням безпеки за замовчуванням і заохочує найкращі практики забезпечення даних користувачів.
- Має вбудовану підтримку MongoDB та інших популярних баз даних для зберігання інформації користувачів, але може працювати без неї, за допомогою JWT токенів

3.5 Хмарна платформа

У якості хмарної платформи був обраний **Vercel,** що пропонує Platform as a Service (PaaS). Він відомий своїм дружнім до розробників підходом та надлегкою інтеграцією з Next.js.

Ця хмарна платформа сильно полегшує роботу з інфраструктурою застосунку і надає дуже зручний сервіс. Vercel використовує безсерверні функцій, що дозволить сфокусувати увагу на головну логіку додатку, а не створення та підтримки сервера.

Чому був обраний саме Vercel:

- Завдяки глибокій інтеграції з Next.js, Vercel може надати оптимальну продуктивність та надійність для next.js проекту
- Vercel має сильну та активну спільноту розробників, яка постійно вносить покращенням платформи.

- Швидкість та продуктивність: Vercel використовує Edge Runtime, який є легшим за Node.js, для дуже швидкого часу відгуку та низької затримки в продакшині.

- Vercel надає автоматично для додатку:
    - Кордонні обчислення для зменшення затримок [4].
    - При підключенні git репозиторію, створюється конвеєр CI/CD для безперервної інтеграції та розгортання.
    - SSL сертифікат для безпечного з’єднання завдяки HTTPS
    - Автоматичне масштабування, що означає, що додаток може легко обробляти збільшення навантаження без необхідності вручну збільшувати кількість серверів.

Хоч, звісно не можна не зауважити, негативні сторони Vercel:

- Гнучкість та можливість детальної конфігурації - не є особливостями даної платформи, тому при розробці додатків з специфічними потребами вона не буде найкращим вибором.
- Vercel оптимізовано для статичних сайтів і безсерверних функцій, які можуть не підходити для проектів, що вимагають традиційного бекенда або великої обробки на стороні сервера. Традиційні бекенд-додатки, наприклад на Express.js і Node.js часто вимагають більш складних операцій на стороні сервера, які можуть не повністю підтримуватися Vercel. [33]
- Vercel пропонує щедрий безкоштовний рівень, витрати можуть швидко зрости для сайтів із високим трафіком

ВИСНОВОК

У цьому розділі було **окреслено технології**, які будуть використані для розробки веб-додатку. Також, проведено **порівняльний аналіз** інструментів з їхніми аналогами та обґрунтовано вибір тої чи іншої бібліотеки або фреймворку.

Було обрано **MERN** стек (MongoDB, Express, React, Node.js) через його простоту, універсальність, продуктивність та велику спільноту розробників.

Для бази даних було обрана **MongoDB Atlas (DBaaS)** завдяки її гнучкості, масштабованості та високій швидкості читання та запису.

Для розробки бекенду використовуються **Node.js** та **Express.js**. Node.js забезпечує швидке та ефективне виконання JavaScript на стороні сервера, а Express.js надає простий та елегантний інтерфейс для створення API.

На фронтенді використовуються **React.js** та **Next.js**. React.js пропонує компонентну архітектуру, яка полегшує створення та обслуговування інтерфейсу користувача, а Next.js надає додаткові функції, такі як серверний рендеринг, генерація статичних сайтів та автоматична оптимізація продуктивності.

Для стилізації використовується **SASS** та **MUI**. SASS робить CSS більш організованим та читабельним, а MUI надає широкий спектр готових компонентів для створення інтерфейсу користувача.

NextAuth.js використовується для авторизації користувачів. Ця бібліотека пропонує просте налаштування та інтеграцію з різними провайдерами автентифікації.

Vercel використовується як хмарна платформа. Ця платформа пропонує безсерверні функції, автоматичне масштабування та глибоку інтеграцію з Next.js.

РОЗДІЛ 4: ПОГЛИБЛЕНИЙ АЛГОРИТМ РЕАЛІЗАЦІЇ БАГАТОРІВНЕВОГО ВЕБ-ЗАСТОСУНКУ

Алгоритм реалізації багаторівневого веб-додатку складається з низки методичних кроків:

4.1 Налаштування середовища розробки

Налаштування середовища розробки, забезпечення наявності всіх інструментів та Visual Studio Code як IDE.

Використовувались розширення VS Code. Вони можуть значно покращити досвід кодування у середовищі розробки.

Prettier - автоматичний форматувальник коду, який забезпечує узгоджений та уніфікований стиль в режимі реального часу.
Console Ninja - відображає логи безпосередньо у редакторі.
EsLint - автоматичний лінтер, аналізує код на предмет помилок, порушення стилів програмування та ймовірних проблем в режимі реального часу.
Import Cost - відображає в редакторі розмір імпортованого пакета. Це допомагає краще контролювати розмір фронт-енд частини, що впливає на Page Load Time.
Toggle Quotes - дозволяє швидко переходити між різними типами лапок, що дуже допомагає з JSX та js string templates.
Peek Hidden Files - дозволяє швидко увімкнути або вимкнути видимість певних файлів у структурі додатку

4.2 База даних

4.2.1 ODM

MongoDB - це база даних, яка не має схем для визначення сутностей. ми можемо використовувати бібліотеку **mongoose** - ODM (Object Relational Mapping)**,** що надає такий функціонал.

```
import { Schema, model } from "mongoose";
import bcrypt from "bcryptjs";
import validator from "validator";

const userSchema = new Schema(
  {
    firstName: {
      type: String,
      required: [true, "Ім'я є обов'язовим полем"],
    },
    secondName: {
      type: String,
      required: false,
    },
    email: {
      type: String,
      required: [true, "Пошта є обов'язовим полем"],
      unique: [true, "Користувач з цієї поштою вже зареєстрований"],
    },
    password: {
      type: String,
      required: [true, "Пароль є обов'язовим полем"],
    },
    image: {
      type: String,
    },
    isAdmin: {
      type: Boolean,
      default: false,
    },
    likedProducts: [
      {
        type: Schema.Types.ObjectId,
        ref: "Product",
        required: false,
      },
    ],
    cart: [
      {
        product: {
          type: Schema.Types.ObjectId,
          ref: "Product",
          required: function () {
            return this.quantity != null;
          },
        },
        quantity: {
          type: Number,
          required: function () {
            return this.product != null;
          },
        },
      },
    ],
  },
  { timestamps: true }
);
```

```
import { Schema, model } from "mongoose";

//todo make certain fields required true
const productSchema = new Schema(
  {
    name: {
      type: String,
      required: false,
    },
    category: [
      {
        type: Schema.Types.ObjectId,
        ref: "category",
        required: false,
      },
    ],
    price: {
      type: Number,
      required: false,
    },
    characteristics: {
      type: Map,
      of: [String],
      required: false,
    },
    description: {
      type: String,
      required: false,
    },
    images: {
      type: [String],
      required: false,
    },
    left: {
      type: Number,
      required: false,
    },
    starRating: {
      type: Number,
      required: false,
    },
  },
  { timestamps: false }
);

export default model("Product", productSchema);
```

```
userSchema.statics.signIn = async function (email, password) {
  if (!email || !password) {
    throw Error("Email and password fields cannot be blank");
  }

  const user = await this.findOne({ email });

  if (!user) {
    throw Error("Incorrect email");
  }

  const match = await bcrypt.compare(password, user.password);
  if (!match) {
    throw Error("Incorrect password");
  }

  return user;
};

userSchema.statics.signUp = async function (user) {
  const { firstName, secondName, email, password, localStorageCartJson } = user;

  if (!email || !password) {
    throw Error("Email and password fields cannot be blank");
  }

  if (!validator.isEmail(email)) {
    throw Error("Email is not valid");
  }

  if (!validator.isStrongPassword(password)) {
    throw Error("Password is not strong enough");
  }

  const isUserExists = await this.findOne({ email });
  if (isUserExists) {
    throw Error("Email already in use");
  }

  const hash = await bcrypt.hash(password, 10);

  const min = 0;
  const max = 4;
  const randomImageIdx = Math.floor(Math.random() * (max - min + 1)) + min;
  const defaultUserImage = `https://storage.googleapis.com/live_world/users/user${randomImageIdx}.j

  const newUser = await this.create({
    firstName: firstName,
    secondName: secondName,
    email: email,
    password: hash,
    image: defaultUserImage,
    cart: localStorageCartJson,
  });

  {
    let newUserDoc = newUser._doc;
    let { firstName, secondName, email, image, likedProducts, cart, _id } =
      newUserDoc;

    return { firstName, secondName, email, image, likedProducts, cart, _id };
  }
};
```

```
import { Schema, model } from "mongoose";

const categorySchema = new Schema({
  name: {
    type: String,
    required: true,
  },
  order: {
    type: Number,
    required: true,
  },
  path: {
    type: String,
    required: true,
  },
  imagePath: {
    type: String,
    required: true,
  },
  filters: {
    type: [String],
    required: false,
  },
});

export default model("category", categorySchema);

```

Визначаємо три схеми: користувач, продукт і категорія.

У схемі користувача є поля для імені, прізвища, електронної пошти, пароля, зображення, статусу адміністратора, продуктів, що сподобалися, і кошика для покупок.

Також потрібно зробити два статичні методи автентифікації користувача: signIn і signUp. Метод signIn перевіряє, чи збігаються надані адреса електронної пошти та пароль з користувачем у базі даних, тоді як метод signUp створює нового користувача після підтвердження електронної пошти та пароля, а потім ще й

перевірки, чи електронна адреса вже використовується. При створенні нового користувача пароль хешується, для цього використовується бібліотека bcryptjs.

Також, застосовується бібліотека validator, яка перевіряє коректність введених даних. Наприклад, перевіряє чи змінна email є дійсно електронною поштою.

У схемі продукту є поля для назви продукту, категорії, ціни, характеристик, опису, зображень, кількості, що залишилася, і рейтингу. Поле категорії є посиланням на модель категорії.

Схема категорії має поля для назви категорії, порядку, шляху, картинки та спеціалізованих фільтрів для цієї категорії.

Потім створюються та експортуються моделі на основі визначених схем. Моделі в **mongoose** - це об’єкти, які надають інтерфейс (вбудовані методи Mongoose для взаємодії з MongoDB) для взаємодії з базою даних. Вони представляють колекції документів, які можна створювати, читати, оновлювати та видаляти.

4.2.2 Наповнення даними

Була зібрана інформації товарів з сайтів виробників за допомогою бібліотеки Puppeteer. Puppeteer - це бібліотека Node.js, яка надає високорівневий API для автоматичного керування Chrome/Chromium через протокол DevTools [41]

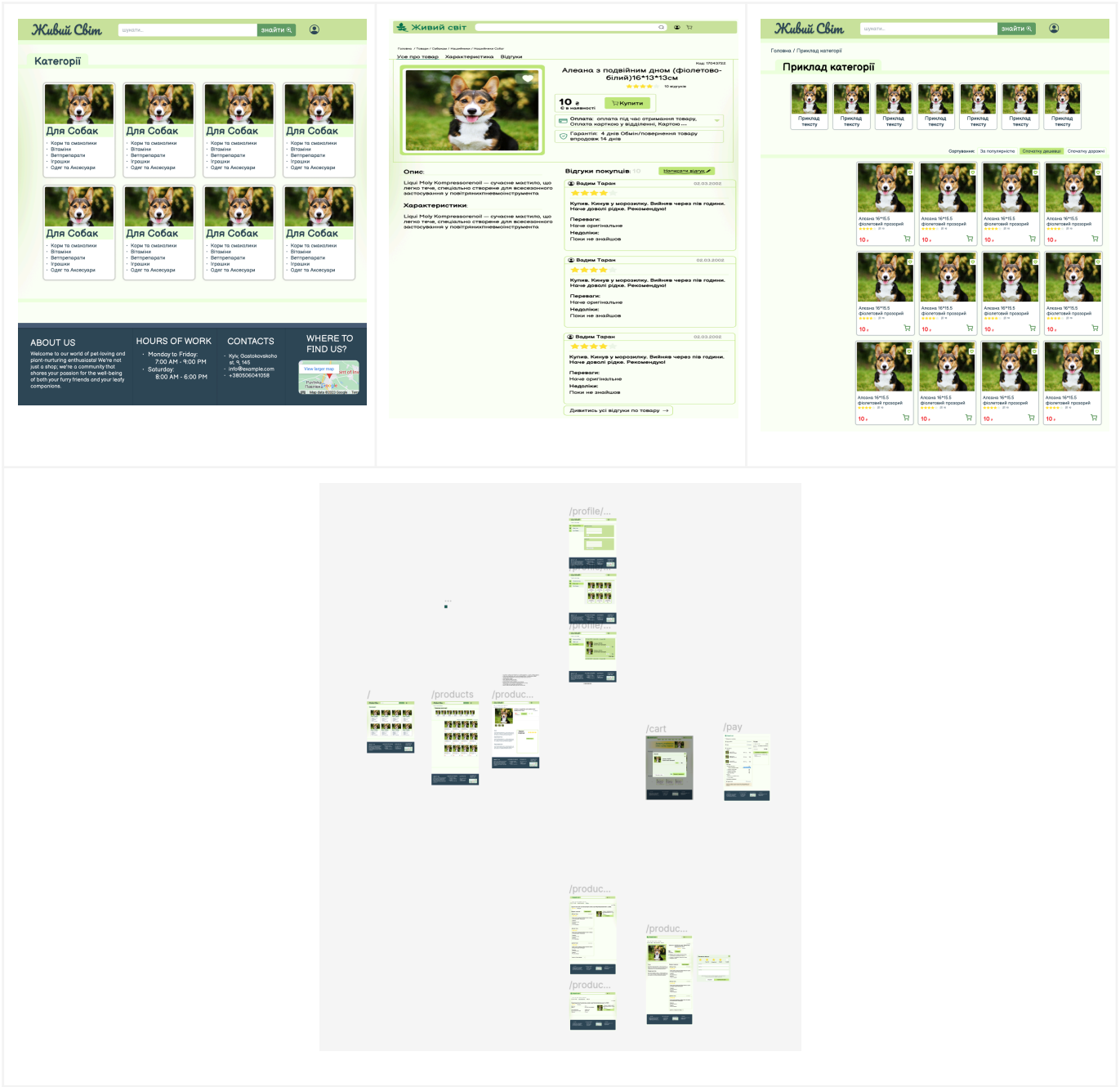
Нижче представлений скрипт, що агрегував дані з сайтів виробників.

```
import { getCategory } from "./scrapeUtils/getCategory.js";
import { getCharacteristics } from "./scrapeUtils/getCharacteristics.js";
import { getDescription } from "./scrapeUtils/getDescription.js";
import { getImages } from "./scrapeUtils/getImages.js";
import { getPrice } from "./scrapeUtils/getPrice.js";
import { getTitle } from "./scrapeUtils/getTitle.js";
import { randomUserAgent } from "../globals/randomUserAgent.js";
import { flash, red, terminator } from "../globals/variables.js";

export async function scrapeSearch(page, productsInfo, brand) {
  const products = [];
  for (const { entry, url } of productsInfo) {
    const imagesUrls = new Set();
    page.on("response", (res) => {
      if (
        res.request().resourceType() === "image" &&
        res.url().includes("/goods/images/big")
      ) {
        imagesUrls.add(res.url());
      }
    });

    await page.setUserAgent(randomUserAgent);
    await page.goto(url, {
      waitUntil: "domcontentloaded",
    });

    const product = { entry: entry, left: 0, brand: brand };

    product.name = await getTitle(page);
    product.price = await getPrice(page);
    product.description = await getDescription(page);
    product.characteristics = await getCharacteristics(page);
    product.category = await getCategory(page, product);
    product.images = await getImages(page, product, imagesUrls, brand);

    console.log(product, `\n${terminator}`);
    products.push(product);
  }

  return products;
}
```

4.3 Фронтенд, UI та UX

Спершу було змодельовано ui для всіх головних компонентів та сторінок.

За допомогою Figma я розробив початковий дизайн-документ для того, аби визначитись зі загальним стилем та кольоровою палітрою інтерфейсу.



Надалі було внесено багато стилістичних правок у початковий дизайн, але структура залишилась незмінною.

Потім було почато розробку інтерфейсу за допомогою React, використовуючи Sass для стилізації, та MUI для скаладних для розробки самотужки компонентів (Price Slider, Modals, Popovers, Drawers та інших

Була створена структуру фронтенду. Визначив, яка функціональність буде належати до тої чи іншої директорії проєкту.

Головні складові структури фронтенду:

- comps: загальні компоненти
- features: Компоненти, хуки та інше для головного функціоналу додатку
- hooks: загальні хуки
- pages: сторінки (API-маршути)
- public: асети
- styles: стилі scss, mixins, global css
- utils: допоміжні функції
- package.json - налаштування проєкту
- .env.local - змінні середовища

```
front
│   .env.local
│   package.json
├────comps
├────features
├────hooks
├────pages
├────public
├────styles
└────utils
…
```

За допомогою, JSX були створені загальні компоненти, які використовуються у всьому проєкті. Вони розміщені у папці root/comps, де root коренева папка фронтенд частини додатку.
Як приклад, компонент ImageFallback та InputField.

```jsx
import Image from "next/image";
import { useEffect, useState } from "react";

export default function ImageFallback({ src, fallbackSrc, ...rest }) {
  const [imgSrc, set_imgSrc] = useState(src);

  useEffect(() => {
    set_imgSrc(src);
  }, [src]);

  return (
    <Image
      {...rest}
      src={imgSrc}
      onLoad={(result) => {
        if (result.naturalWidth === 0) {
          set_imgSrc(fallbackSrc);
        }
      }}
      onError={() => {
        set_imgSrc(fallbackSrc);
      }}
    />
  );
}

```

```jsx
import { useId } from "react";
import s from "./input_field.module.scss";

const InputField = ({
  type,
  value,
  onChange,
  label,
  placeholder,
  error,
  disabled,
}) => {
  const id = useId();

  return (
    <div className={`form-floating ${s.input_container}`}>
      <input
        type={type}
        id={id}
        className={`form-control ${error ? "is-invalid" : ""}`}
        value={value}
        onChange={onChange}
        placeholder={placeholder}
        disabled={disabled}
      />
      <label className={`form-label ${s.label}`} htmlFor={id}>
        {label}
      </label>
    </div>
  );
};

export default InputField;

```

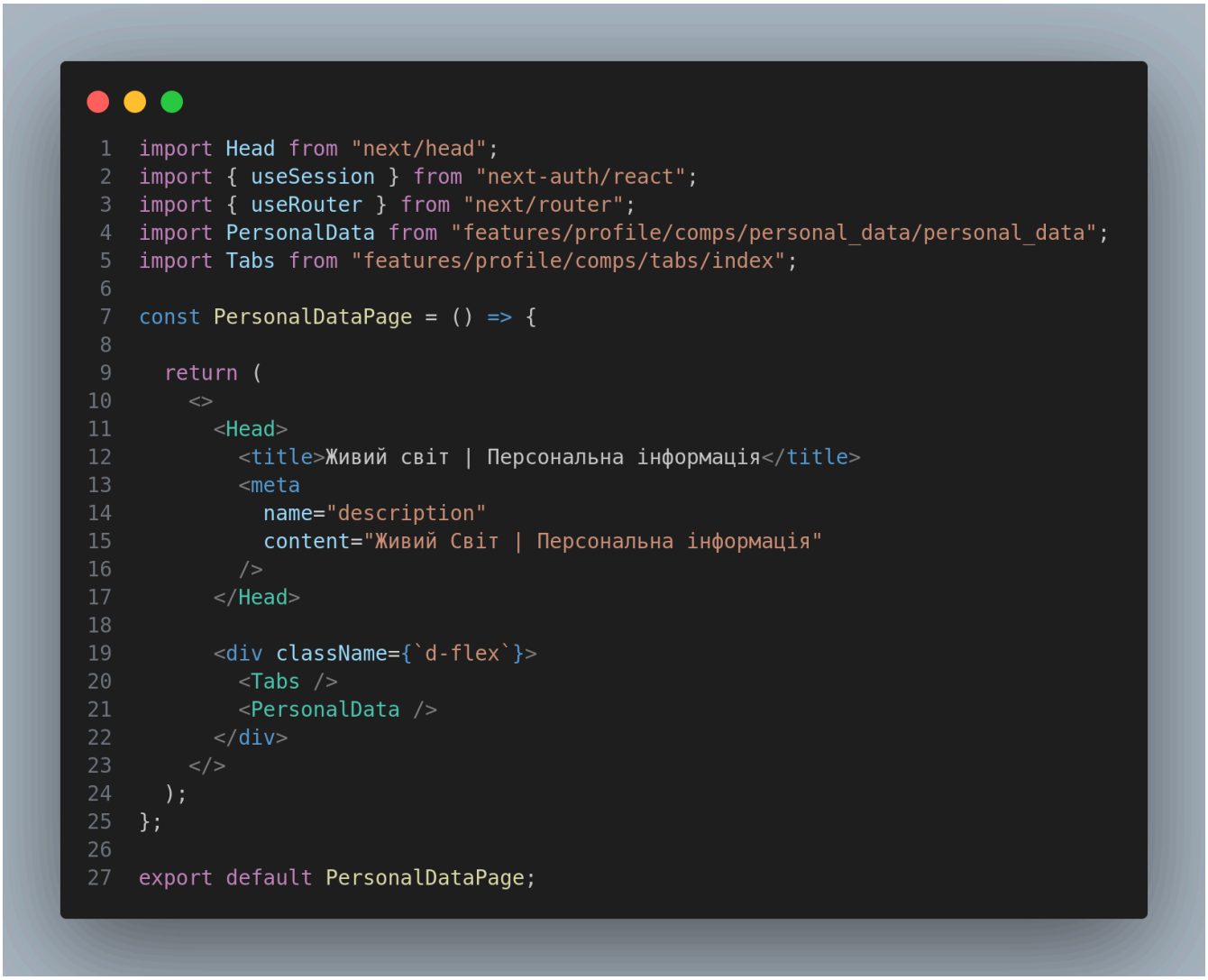
Загальні компоненти дозволяють повторно використовувати код, що зменшує дублювання та спрощує супровід додатку. Також, вони забезпечують уніфікацію інтерфейсу та масштабованість (використання у інших загальних компонентах).

Також, були написані компоненти, хуки та інші елементи, які є специфічними для основного функціоналу додатку. Наприклад, розглянемо компоненти для відображення карток категорії та продукту.

```jsx
import Link from "next/link";
import s from "./listing_card.module.scss";

import { slugify } from "@bbuukk/slugtrans/slugify";
import { transliterate } from "@bbuukk/slugtrans/transliterate";
import ProductFigure from "./comps/figure";
import ProductRating from "./comps/rating";
import BuyInfo from "./comps/buy_info";
import LikeButton from "./comps/like_btn";

const ListingProductCard = ({ product, like, isLiked }) => {
  const productUrl = (activeTab) =>
    `/product/${slugify(transliterate(product.name))}/${
      product._id
    }/${activeTab}`;

  return (
    <article className={`${s.card} `}>
      <LikeButton isLiked={false} />
      <ProductFigure product={product} productUrl={productUrl} />
      <ProductRating product={product} productUrl={productUrl} />
      <BuyInfo product={product} />
    </article>
  );
};

export default ListingProductCard;
```

```jsx
const Card = ({ category, subcategories }) => {
  const dispatch = useDispatch();
  const categoryPathSlug = (path) => {
    return `/products/${slugify(transliterate(path))}/page=1`;
  };

  return (
    <div className={`${s.cat_card}`}>
      <Link
        href={categoryPathSlug(category.path)}
        onClick={() => dispatch(startLoading())}
      >
        <ImageFallback
          src={category.imagePath}
          fallbackSrc={"/assets/goods_placeholder.svg"}
          alt="основна категорія"
          width={300}
          height={150}
          priority
        />
        <Image />
        <h2 className={`${s.naming} `}>{category.name}</h2>
      </Link>

      <ul className={`${s.subcat_list}`}>
        {subcategories
          .sort((a, b) => a.order - b.order)
          .map(({ _id, path, name }, index) => {
            return (
              <li key={_id}>
                <Link
                  href={categoryPathSlug(path)}
                  onClick={() => dispatch(startLoading())}
                >
                  {index == 4 ? `${name}` : `${name}`}
                </Link>
              </li>
            );
          })}
      </ul>
    </div>
  );
};

export default Card;
```

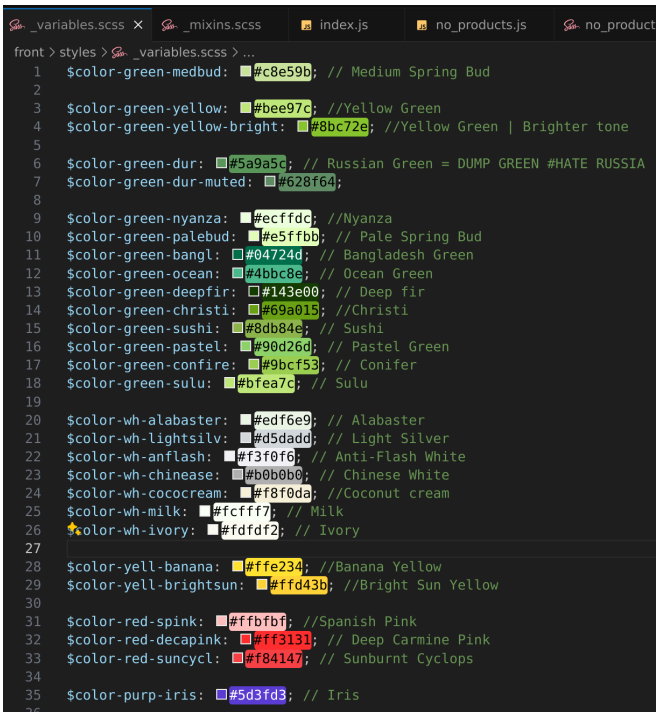
Завдяки Next.js я зміг легко визначити API-маршрути. У Next.js кожен файл у директорії pages автоматично стає доступним як маршрут. Тобто, компонент, який експортується з файлу /profile/personal_data.js автоматично відображається за запитом до браузера zhyvyisvit.shop/profile/personal_data

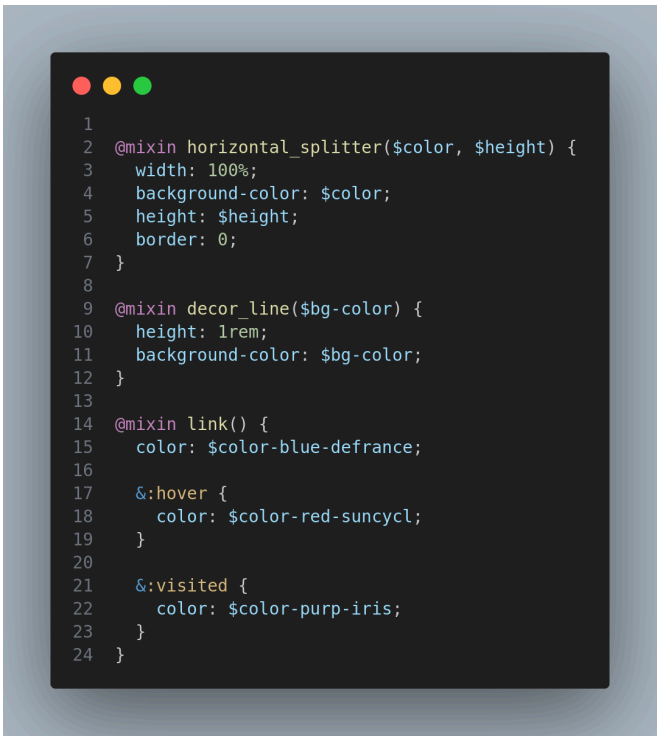


```jsx
import Head from "next/head";
import { useSession } from "next-auth/react";
import { useRouter } from "next/router";
import PersonalData from "features/profile/comps/personal_data/personal_data";
import Tabs from "features/profile/comps/tabs/index";

const PersonalDataPage = () => {

  return (
    <>
      <Head>
        <title>Живий світ | Персональна інформація</title>
        <meta
          name="description"
          content="Живий Світ | Персональна інформація"
        />
      </Head>

      <div className={`d-flex`}>
        <Tabs />
        <PersonalData />
      </div>
    </>
  );
};

export default PersonalDataPage;
```

Кожна сторінка має свої унікальні компоненти, які відповідають за відображення вмісту та інтерактивність на ній, також зазначені метадані про неї.

Після цього я перейшов до стилізації за допомогою Sass. Використовувались змінні кольору, які відповідають кольоровій палітрі та стилю, визначеним у Figma та міксини для більш ефективної роботи зі стилями.





Підсумовуючи, архітектура фронт-енду полягає в тому, що спочатку запит потрапляє на сервер Next.js. Потім, фреймворк виконує функцію getServerSideProps (SRR) або getStaticProps (SSG) у компоненті сторінки, отримує необхідні дані та рендерить сторінку на стороні сервера. У випадку SSR сторінка рендериться кожен раз, коли надходить запит, SSG же полягає в створенні сторінки один раз, лише під час будування проекту. Потім, клієнт отримує відтворену сторінку (разом із отриманими даними), а React.js робить сторінку інтерактивною, прикріплюючи слухачі подій до неї (що називається гідруванням у React).

Асети та Google Cloud CDN

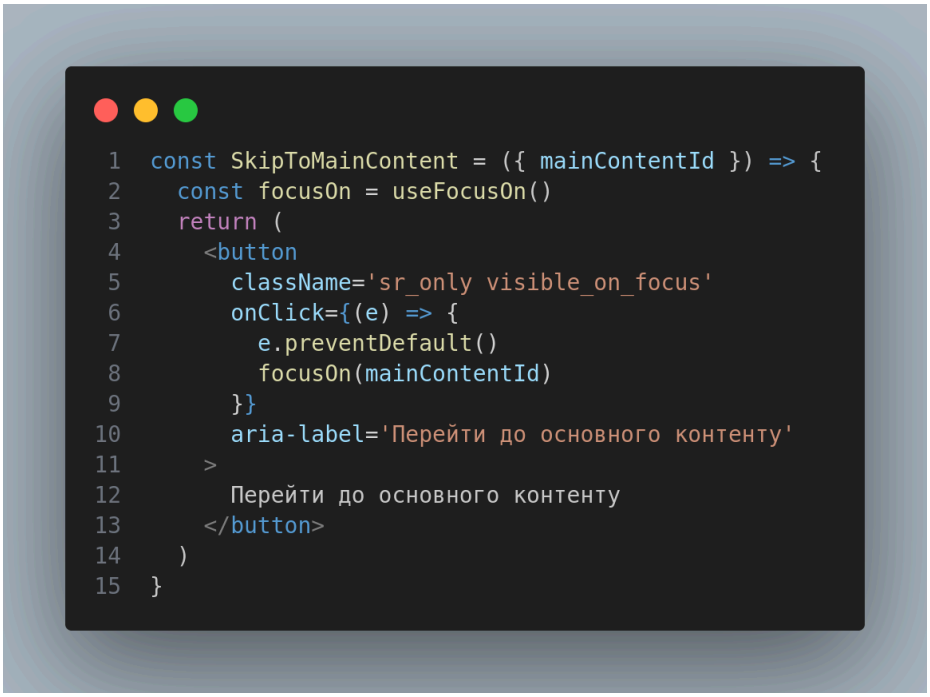
Також, варто зазначити, що у додатку було використано Google Cloud CDN. У сервісі Cloud Storage була створена папка (bucket) з усіма потрібними асетами для товарів, продуктів, зображень користувачів за замовчуванням.

4.3.1 Веб-доступність

Скорочення шляху навігації

Для поліпшення навігації на веб-сторінці за допомогою клавіш та кнопок було розроблено два компонента: **TabIndexButton** та **SkipToMainContent**:

Компонент **SkipToMainContent** - знаходить другим у порядку табування та дозволяє здійснити швидку навігацію до головного вмісту на сторінці, що позначено атрибутом id, що дорівнює main_content.



```
const SkipToMainContent = ({ mainContentId }) => {
  const focusOn = useFocusOn()
  return (
    <button
      className='sr_only visible_on_focus'
      onClick={(e) => {
        e.preventDefault()
        focusOn(mainContentId)
      }}
      aria-label='Перейти до основного контенту'
    >
      Перейти до основного контенту
    </button>
  )
}
```

Компонент **TabIndexButton** - діє як оболонка для дочірніх елементів, які можуть отримати фокус.

Основна ідея полягає в тому, що дочірні елементи не можуть отримати фокус до тих пір, поки на **TabIndexButton** не натиснуто клавішу Enter. Після цього користувач може взаємодіяти з дочірніми елементами за допомогою клавіш. Однак, якщо користувач натисне клавішу Escape або покине останній дочірній елемент, фокусування знову блокується та повернеться на **TabIndexButton**.

Цей підхід дозволяє скоротити шлях до потрібних елементів на сторінці за

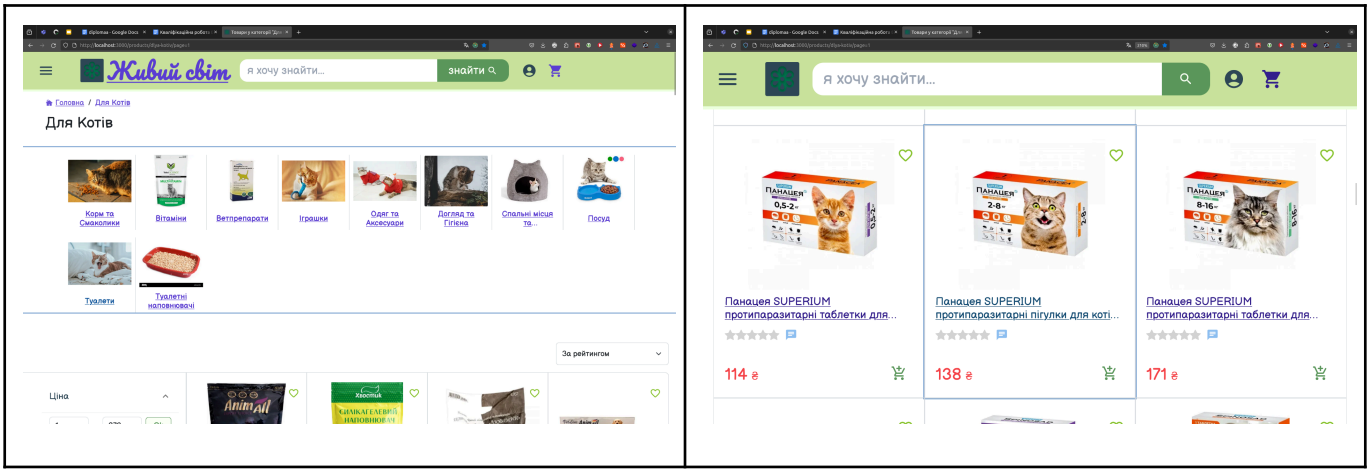
допомогою клавіш або кнопок, що значно підвищує зручність навігації, роблячи її більш інтуїтивно зрозумілою та ефективною.

TabIndexButton вимикає табуляцію дочірніх для всіх елементів за допомогою звичайного запиту елементів, що можуть отримувати фокус та виставленню значення атрибуту tabIndex до -1, що виключає їх з порядку фокусування.

```
const TabIndexButton = ({ children, ...props }) => {
  const ref = useRef()
  const [isTabbable, setIsTabbable] = useState(false)

  useEffect(() => {
    toggleTabbability(false)
  }, [children])

  const toggleTabbability = (tabbable) => {
    if (ref.current) {
      setIsTabbable(tabbable)

      const tabbableElements = ref.current.querySelectorAll(
        'button, [href], input, select, textarea, [tabindex]:not([tabindex="-1"])',
      )

      if (!tabbable) {
        tabbableElements.forEach((el) => el.setAttribute('tabindex', '-1'))
      } else {
        tabbableElements.forEach((el) => el.setAttribute('tabindex', '0'))
      }
    }
  }

  const handleKeyDown = (event) => {
    if (event.key === 'Enter') {
      if (document.activeElement === ref.current) {
        toggleTabbability(!isTabbable)
      }
    }

    if (event.key === 'Escape') {
      toggleTabbability(false)
      ref.current.focus()
    }
  }

  const handleBlur = (event) => {
    if (!event.currentTarget.contains(event.relatedTarget)) {
      toggleTabbability(false)
      ref.current.focus()
    }
  }

  return (
    <div
      className={`${s.focus_in_btn}`}
      ref={ref}
      tabIndex={0}
      role={props.role || 'button'}
      onKeyDown={handleKeyDown}
      onClick={handleKeyDown}
      onBlur={handleBlur}
      {...props}
    >
      {children}
    </div>
  )
}
```

Це особливо корисно наприклад у сітці продуктів та підкатегорій, а також фільтрах, де присутні багато checkbox, за якими можна вибрати потрібно опцію фільтра. Ми можемо швидко пройти всі checkbox у фільтрах, якщо нас цей фільтр не цікавить.

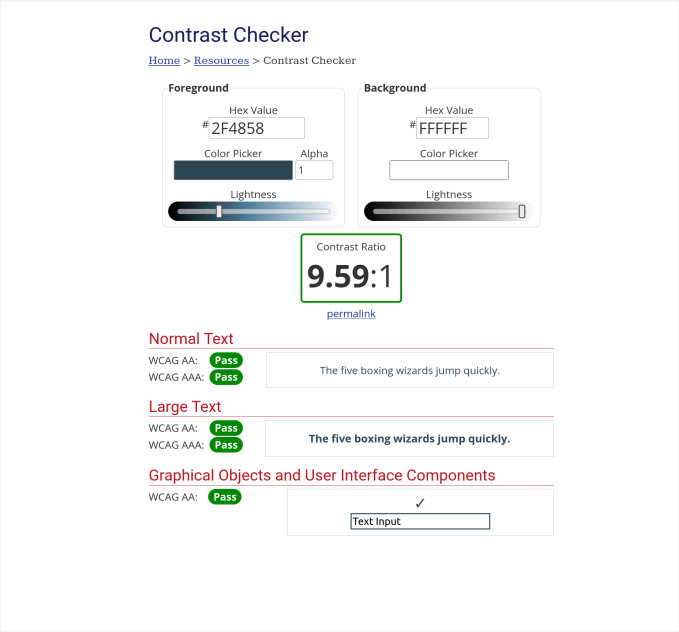
Можемо пропустити кнопку для для додавання до кошику та списку бажаних, оцінки товару, якщо нас не цікавить товар. Можемо пропустити всі підкатегорії, якщо хочемо залишитись в поточній, а не траверсувати усі, що є, для того, щоб перейти далі.

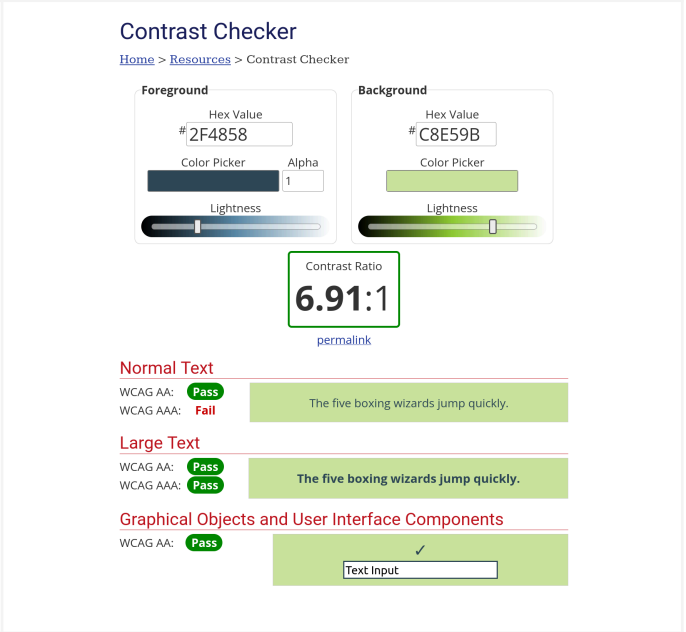


Кольоровий контраст

Контраст кольорів - це відношення між яскравістю тексту та його фону. Це важливо для забезпечення читабельності тексту для всіх користувачів, особливо для тих, хто має проблеми зі зором або кольоровим сприйняттям. Також це допомагає людям, краще орієнтуватись у інтерфейсі при яскравому освітленні (наприклад, при сонячній погоді).

Web Content Accessibility Guidelines (**WCAG**) рекомендують контрастність кольорів принаймні 4.5:1 для звичайного тексту та 3:1 для великого тексту [39]. Тестування від WebAIM [38], показує, що співвідношення контрасту основних кольорів додатку більш ніж відповідає стандартам.





Динамічні стилі для доступності

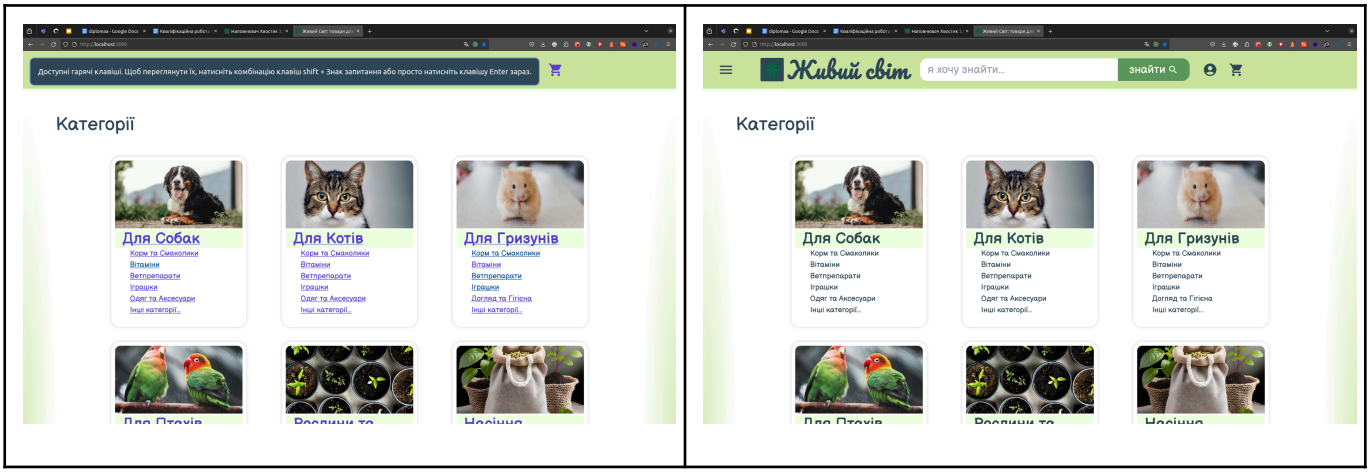
Було розроблені стилі спеціально для людей, що використовують клавіатуру.

На кожній сторінці встановлюється детектор того, чи людина використовує Tab для навігації і якщо це так, то до body елементу додається клас, що визначає нові стилі для деяких елементів, що відрізняються від стилів за замовчуванням

```
useOnUserTabbing(() => {
    if (typeof document !== 'undefined') {
      document.body.classList.add('user-is-tabbing')
    }
  })
```

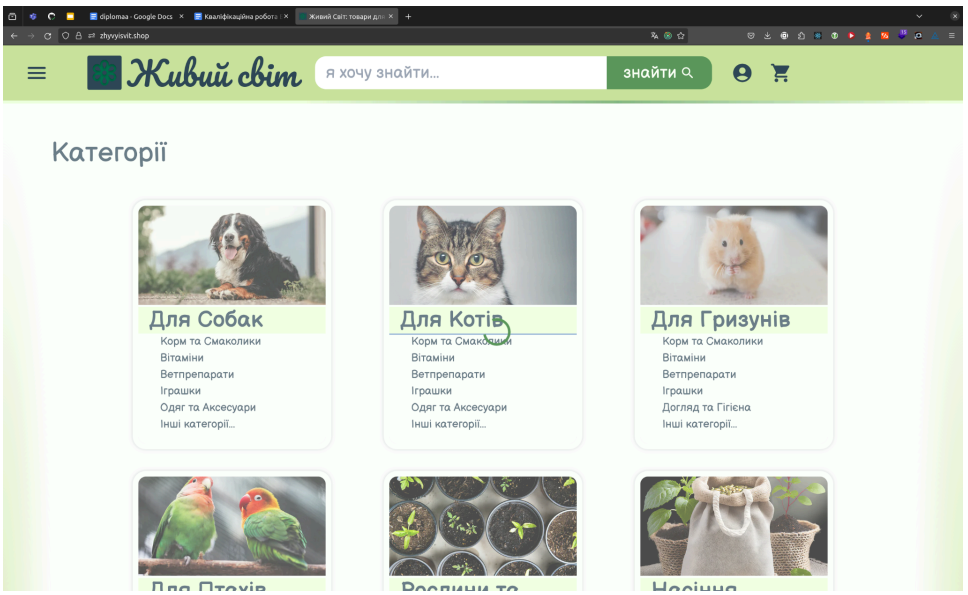
```
body.user-is-tabbing {
  button:focus,
  a:focus,
  input:focus,
  select:focus,
  div:focus,
  nav:focus,
  article:focus,
  section:focus {
    border: none;
    outline: none;
    box-shadow: 0 0 0 2px #719ece;
  }

  a {
    text-decoration: underline;
    color: darken($color: $color-blue-defrance, $amount: 20%);

    &:visited {
      color: $color-purp-iris;
    }
  }
}
```

Порівняння стилів за замовчуванням та адаптивних. Можемо бачити, що при використанні Tab, кожен сфокусований елемент буде мати box-shadow, а також посилання будуть мати більш звичний колір, де невідвідане посилання буде мати синій, а відвідане - фіолетовий.



Елементи, що відображають стан додатку

Було реалізовано повторення елементів відображення стану додатку у декількох передбачуваних місцях. Наприклад, стан завантаження відображається кількома компонентами: кругом по центру, лінією зверху, тоді як головний вміст сторінки перекривається напів-прозорим шаром, що чітко сигналізує про завантаження та не дозволяє з нею взаємодіяти під час завантажень.)

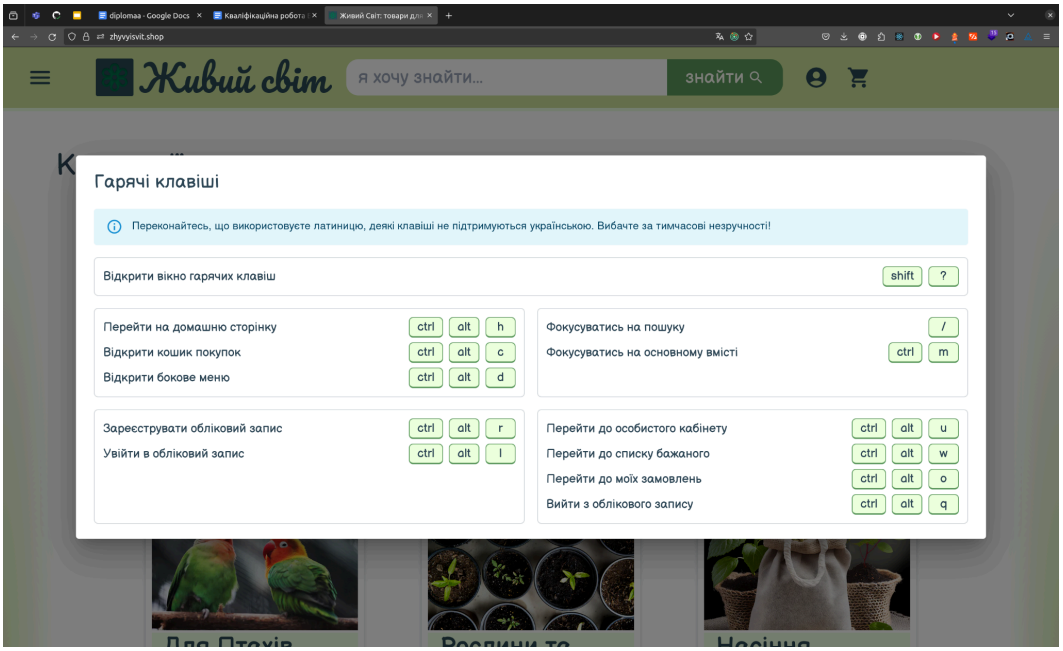


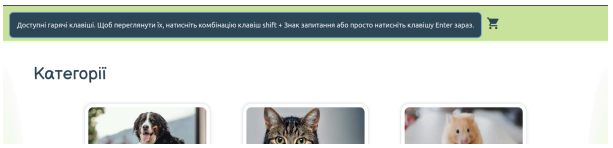
Гарячі клавіші

Для розробки гарячих клавіш для навігації, фокусування та взаємодії з додатком була використана бібліотека **react-hotkeys-hook**. Це бібліотека для React, яка дозволяє визначати комбінації клавіш і асоціювати їх з функціями зворотного виклику, які виконуються, коли користувач натискає вказані клавіші. Для цього використовується хук **useHotkeys** (keys, callback), як зображено на скріншоті нижче.

Також, було розроблене модальне вікно з переліком усіх гарячих клавіш і елемент, який стоїть першим у черзі фокусування кожної сторінки, що повідомляє користувачів, що користуються клавіатурою, про їх наявність.

При розробці було протестовано, що використані комбінації клавіш не є типовими для гарячі клавіш браузера або читача екрана





```
useHotkeys('shift+?, shift+.', () => toggle(HOTKEYS_MODAL), [dispatch])

  //navigation
  useHotkeys('ctrl+alt+h, ctrl+alt+р', () => navigateTo('/'))
  useHotkeys('ctrl+alt+u, ctrl+alt+г', () => navigateTo('/user/personal_data'))
  useHotkeys('ctrl+alt+w, ctrl+alt+ц', () => navigateTo('/user/wish_list'))
  useHotkeys('ctrl+alt+o, ctrl+alt+щ', () => navigateTo('/user/orders_list'))

  useHotkeys('ctrl+alt+c, ctrl+alt+с', () => toggle(CART_MODAL), [dispatch])
  useHotkeys('ctrl+alt+l, alt+shift+д', () => toggle(SIGN_IN_MODAL), [dispatch])
  useHotkeys('ctrl+alt+r, alt+shift+к', () => toggle(SIGN_UP_MODAL), [dispatch])
  useHotkeys('ctrl+alt+d, ctrl+alt+в', () => toggle(MAIN_OFFCANVAS), [dispatch])
  // da
  //focus management
  useHotkeys('/', () => focusOn('search_bar_input'))
  useHotkeys('ctrl+m, ctrl+ь', () => focusOn('main_content'))

  //user
  useHotkeys('ctrl+alt+q, ctrl+alt+й', () => signOut({ callbackUrl: '/' }), [
    dispatch,
  ])
```

Сумісність з читачами екрану (Orca, JAWS, NVDA, VoiceOver і т.д.)

Забезпечена можливість переміщатися по веб-сайту без використання миші.

Також, для всіх елементів, що можуть отримати фокус клавіатурою та не містять текст, який би достатньо описував призначення було використано aria-label, це дозволяє SR (screen reader) прочитати його значення замість умісту елемента.

```
<ul className={`${s.controls}`}>
                <PaginationItem
                  pageId={1}
                  ariaLabel='Перейти на найпершу сторінку'
                  disabled={isActive(FIRST_PAGE)}
                >
                  <KeyboardDoubleArrowLeftRounded />
                </PaginationItem>
                <PaginationItem
                  pageId={Math.max(1, Number(activePageId) - 1)}
                  ariaLabel='Перейти на попередню сторінку'
                  disabled={isActive(FIRST_PAGE)}
                >
                  <KeyboardArrowLeftRounded />
                </PaginationItem>
              </ul>
```

Для всіх елементів, aria-label відповідає його стану (наприклад, елемент пагінації скаже поточна сторінка, якщо поточні сторінка відповідає посиланню, замість “Перейти на …”)
А також, використовується aria-disabled для повідомлення, що з елементом неможливо взаємодіяти. (Наприклад для навігації до попередньої сторінки, коли поточна дорівнює першій)

```jsx
const PaginationItem = ({
  pageId,
  onClick,
  children,
  disabled = false,
  ariaLabel = null,
}) => {
  return (
    <li className={`${s.item}`}>
      <Link
        className={`${isActive(pageId) ? s.active : ''} ${
          disabled ? s.disabled : ''
        }`}
        aria-label={
          ariaLabel
            ? ariaLabel
            : isActive(pageId)
              ? 'Поточна сторінка'
              : `Перейти на сторінку номер ${pageId}`
        }
        aria-disabled={disabled}
```

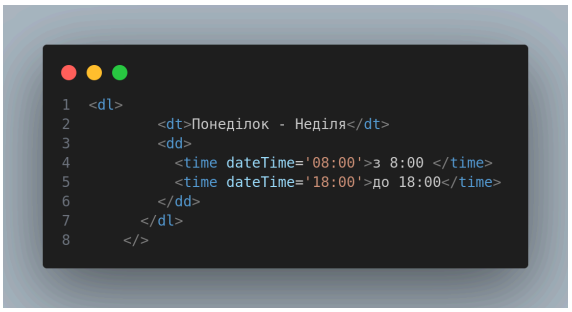
Усі зображення мають атрибут alt, що дозволяє SR передати зміст тим, хто не може побачити зображення.

```jsx
<Image
          src={'/assets/logo.svg'}
          alt='Логотип магазину'
          width={30}
          height={30}
        />
```

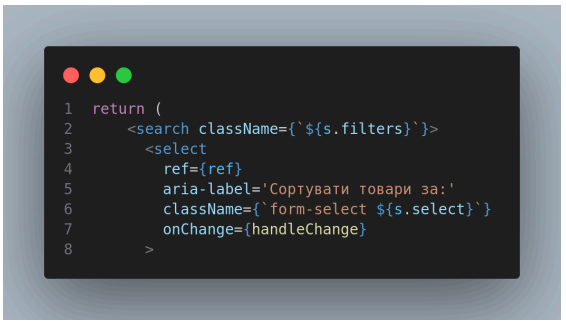
Використання семантичного html

Використання семантичного HTML важливе для читання екрану, оскільки воно допомагає краще розуміти семантичне значення контенту на веб-сторінці

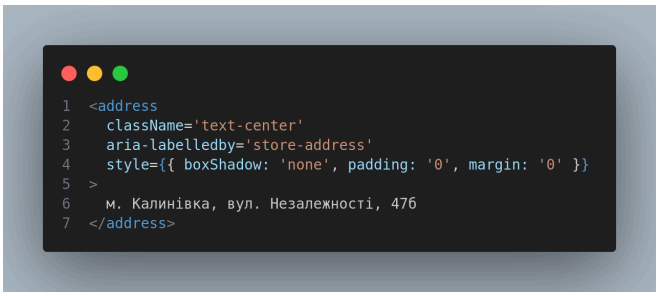
Приклади використання semantic html:



```
<dl>
          <dt>Понеділок - Неділя</dt>
          <dd>
            <time dateTime='08:00'>з 8:00 </time>
            <time dateTime='18:00'>до 18:00</time>
          </dd>
        </dl>
      </>
```



```
return (
    <search className={`${s.filters}`}>
      <select
        ref={ref}
        aria-label='Сортувати товари за:'
        className={`form-select ${s.select}`}
        onChange={handleChange}
      >
```



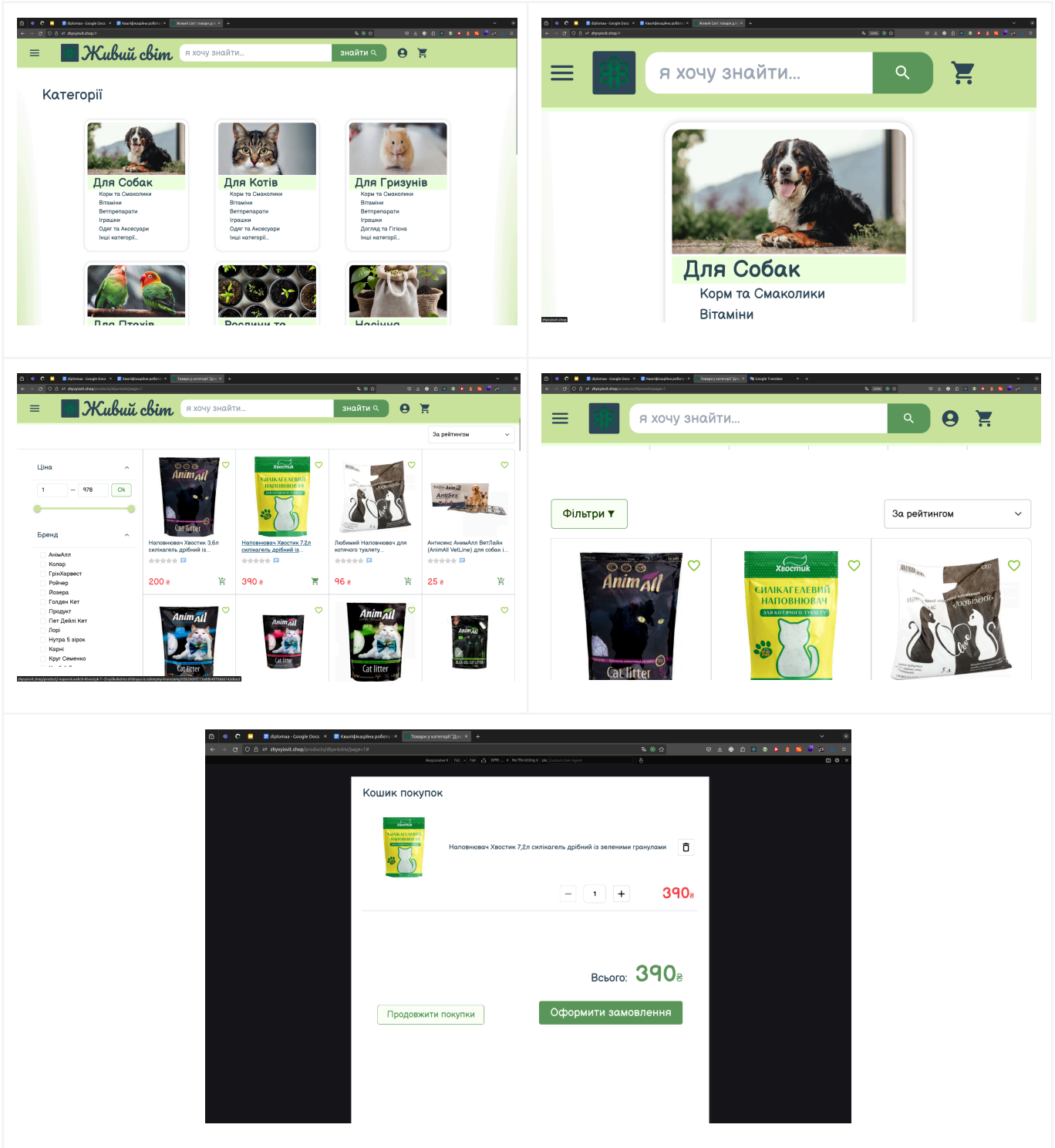
```
<address
  className='text-center'
  aria-labelledby='store-address'
  style={{ boxShadow: 'none', padding: '0', margin: '0' }}
>
  м. Калинівка, вул. Незалежності, 476
</address>
```

```
<header className={`${s.header_container}`}>
    <nav className={`${s.header} ${balsamiqSans.className}`}>
      <OffcanvasToggler />
      <Logo />
      <SearchBar />
```

Адаптивність дизайну сторінок

Розмір компонентів на сторінці має змінюватися, не порушуючи відображення сторінки, щоб люди з вадами зору могли збільшувати вміст і легше читати.
Було реалізовано адаптивний дизайн для кожної компоненти та сторінки. Застосунок пристосований як для великих моніторів, так і для мобільних девайсів.

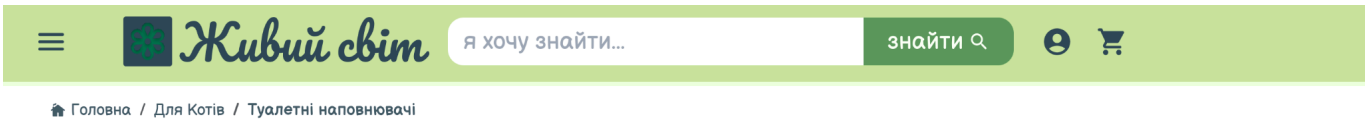
Були також розроблені спеціальні компоненти для підтримки рівних можливостей серед великих та малих екранів (як наприклад: бокова панель з фільтрами на малих екранах зникає зі сторінки, її можна відкрити та закрити за допомогою окремої кнопки). Також, модальні вікна на маленьки екранах розширюються стають повноекранними



Незмінне місце певних компонентів

Компоненти, які існують на кількох веб-сторінках, як-от навігація, верхні та нижні колонтитули та бічні панелі, мають постійно відображатися в тих самих місцях на всьому сайті, щоб люди завжди знали, як їх знайти, незалежно від того, на якій сторінці вони перебувають [25].

Наприклад, як сторінка індивідуального компонента, так і сторінка каталогу, має компонент навігації **breadcrumbs**, що зберігає своє місце на обох сторінках.



Стандарти WAI-ARIA для створення доступних компонентів [26]

Було забезпечено дотримання стандартів **WAI-ARIA для** користувацького інтерфейсу (акордеон, сітка, кнопки, модальні вікна та ін. компонентів) [26]

Наприклад, для списків **WAI-ARIA** зазначає, що клавіші:

Стрілка вниз переміщує фокус до наступного параметра. За бажанням у списку з одним вибором виділення також може переміщатися з фокусом.
Стрілка вгору переміщує фокус до попереднього параметра. За бажанням у списку з одним вибором виділення також може переміщатися з фокусом.

Це було реалізовано у опціях фільтру, де кожна опція - це елемент списку, навігація по них здійснюється за допомогою власного react хука **useArrowKeysNavigation.**

```
function useArrowKeyNavigation() {
  return useCallback((e, targets = {}) => {
    // Set default targets
    const defaultTargets = {
      ArrowDown: document.body.firstChild,
      ArrowUp: document.body.lastChild,
      Home: document.body.firstChild,
      End: document.body.lastChild,
      ...targets,
    }

    const nextElement = defaultTargets[e.key]
    if (nextElement) {
      e.preventDefault()
      nextElement.focus()
    }
  }, [])
}
```

Передання стану додатку за допомогою різних способів

Передання стану додатку за допомогою різних способів(кольору, тексту, компонентів).

Для відображення стану додатку використовується компонент Alert, який має відповідний колір та зображення (icon) залежно від атрибуту severity. Також, надається відповідний текст, який описує проблему, попереджає чи інформує про щось.

```jsx
import { Alert } from '@mui/material'
import s from './alert.module.scss'

const CustomAlert = ({ text, severity = 'info', animated = true }) => {
  return (
    <Alert
      severity={severity}
      className={`${s.alert} ${animated ? s.animated : ''}`}
      key={text}
    >
      {text}
    </Alert>
  )
}

export default CustomAlert

```

Додатково:

- Всі веб-сторінки мають **належну структуру** рівня **заголовків**
- Бібліотека **MUI** надає перехоплення табуляції на модальних вікнах, бокових меню, компонентів, на popover та ін.

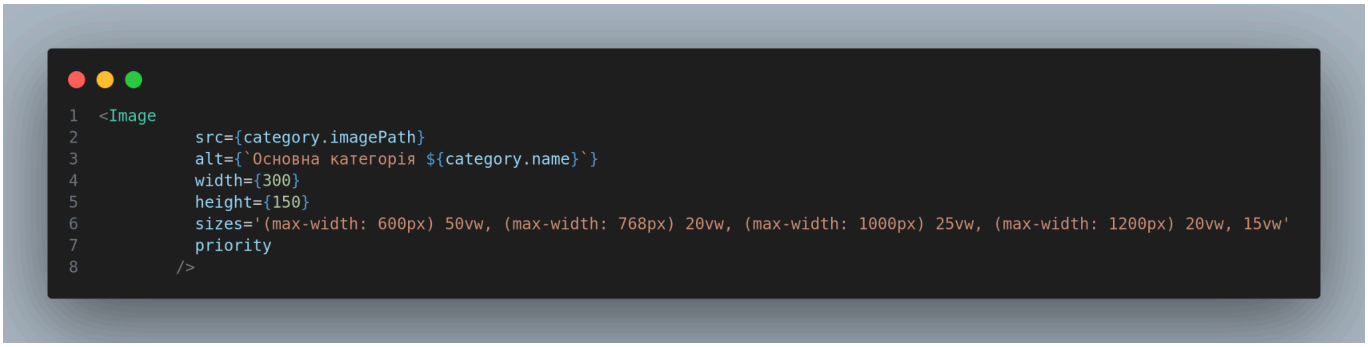
4.3.2 Оптимізація продуктивності

Next.js пропонує кілька вбудованих компонентів, що покращують оптимізацію:

- **next/image**

  Next.js надає компонент **Image**, який служить заміною стандартного HTML-елемента img, що автоматично оптимізує завантаження зображення, використовуючи такі методи:

  - Ліниве завантаження. При цьому є можливість встановлення високого пріоритету для важливих для відображення інтерфейсу зображень.
  - Перетворення форматів зображень на сучасні, ті, які важать менше, наприклад WebP.
  - Атрибут Sizes, що дозволяє використовувати зображення, розміри яких адаптуються залежно від вікна перегляду клієнта
  - Атрибути width and height, що запобігають Content Layout Shift



```
<Image
          src={category.imagePath}
          alt={`Основна категорія ${category.name}`}
          width={300}
          height={150}
          sizes='(max-width: 600px) 50vw, (max-width: 768px) 20vw, (max-width: 1000px) 25vw, (max-width: 1200px) 20vw, 15vw'
          priority
        />
```

- **next/dynamic**

  Next.js надає інструмент для динамічного імпорту компонентів, що автоматично оптимізує завантаження компонентів ліниво завантажуючи їх. Це підвищує code splitting та допомагає не завантажувати ті компоненти, що не відображаються. Ефективно використовувати для модульних вікон, бокових меню, загалом компонентів, що відображаються тільки після певної взаємодії користувача. Також, потрібно ліниво завантажувати компоненти, що не одразу видимі на сторінці, наприклад компонент пагінації, або для тих, що

відображаються не у всіх випадках, як компонент для відображення помилки про не знайдені товарів за пошуковим запитом

```
const lazyload = (importStatement) =>
  dynamic(importStatement, { loading: () => <LoadingOverlay loading={true} /> })

const MainOffcanvas = lazyload(
  () => import('comps/modals/main_offcanvas/main_offcanvas.js'),
)

const ChangePasswordModal = lazyload(
  () => import('comps/modals/change_password/change_password_modal'),
)
const SignInModal = lazyload(
  () => import('comps/modals/auth/sign_in_modal/sign_in_modal'),
)
const SignUpModal = lazyload(
  () => import('comps/modals/auth/sign_up_modal/sign_up_modal'),
)
const DeleteAccountModal = lazyload(
  () => import('comps/modals/delete_account/delete_account_modal.js'),
)
const CartModal = lazyload(() => import('comps/modals/cart/cart_modal'))
const WriteReviewModal = lazyload(
  () => import('comps/modals/reviews/write_review_modal'),
)
const HotkeysModal = lazyload(
  () => import('comps/modals/hotkeys/hotkeys.modal'),
)

```

```
const NoProductYet = dynamic(() => import('comps/warnings/no_products.js'), {
  loading: () => <LoadingSpinner />,
  ssr: false,
})

const FiltersAccordion = dynamic(
  () =>
    import(
      'features/products/listing/comps/filter/filters_accordion/filters_accordion'
    ),
  { loading: () => <LoadingSpinner /> },
)

const FiltersOffcanvas = dynamic(
  () =>
    import(
      'features/products/listing/comps/filter/filiters_offcanvas/filters_offcanvas.js'
    ),
  { loading: () => <LoadingSpinner /> },
)

const ProductsPagination = dynamic(
  () =>
    import('features/products/listing/comps/gallery/pagination/pagination.js'),
  { ssr: false },
)

const Selected = dynamic(
  () => import('features/products/listing/comps/filter/selected.js'),
  { ssr: false },
)
```

- **next/router**: - оптимізований маршрутизатор, який автоматично розділяє код та ліниво завантажує сторінки.

- **next/link**: Next.js надає компонент Link, який служить заміною стандартного HTML-елемента <a>. Він автоматично оптимізує створення посилань між сторінками та навігацію, використовуючи такі методи:
    - Попередньо **завантажує сторінки в фоновому режимі**, що забезпечує швидкий і більш плавний перехід між сторінками
    - Дозволяє використовувати **динамічні маршрути**, використовуючи шаблонні літерали для створення шляху посилання
- **next/font**: інструмент для оптимізації веб-шрифтів. Він автоматично завантажує файли шрифтів під час збірки та розміщує їх разом з іншими статичними активами.

```
import { Balsamiq_Sans } from 'next/font/google'
import { Pacifico } from 'next/font/google'
const balsamiqSans = Balsamiq_Sans({
  weight: '400',
  subsets: ['latin'],
  display: 'swap',
  adjustFontFallback: false,
  fallback: ['Arial', 'sans-serif'],
})
const pacifico = Pacifico({
  weight: '400',
  subsets: ['latin'],
  display: 'swap',
  adjustFontFallback: false,
  fallback: ['Arial', 'sans-serif'],
})
export { balsamiqSans, pacifico }
```

Очищення CSS

Очищення CSS – це практика, що передбачає видалення невикористаного CSS із таблиць стилів проекту, що зменшує розмір файлів, які потрібно завантажувати для користувача.

Для цього використовується бібліотека **@fullhuman/postcss-purgecss.**
Вона аналізує вміст додатку (атрибут content конф. файлу **postcss.config.js**.), щоб виявити селектори CSS, які використовуються, та видаляє ті, які не були знайдені.

```
module.exports = {
  plugins: [
    'postcss-flexbugs-fixes',
    [
      'postcss-preset-env',
      {
        autoprefixer: {
          flexbox: 'no-2009',
        },
        stage: 3,
        features: {
          'custom-properties': false,
        },
      },
    ],
    [
      '@fullhuman/postcss-purgecss',
      {
        content: [
          './pages/**/*.{js,jsx,ts,tsx}',
          './comps/**/*.{js,jsx,ts,tsx}',
          './features/**/*.{js,jsx,ts,tsx}',
          './styles/**/*.{scss,css}',
          './node_modules/@mui/**/*.{js,jsx,ts,tsx}',
        ],
        safelist: {
          standard: ['html', 'body'],
        },
        defaultExtractor: (content) => content.match(/[\w-/:]+(?<!:)/g) || [],
      },
    ],
  ],
}

```

Ліниве завантаження компонентів товарів на сторінці каталогу

Для лінивого завантаження товарів була використана бібліотека **react-virtualized.** Бібліотека допомагає рендерити лише частини каталогу товарів, достатнього для заповнення вікна перегляду користувача [36].

Як це допомагає продуктивності:

- Дозволяє сторінці швидше завантажувати стилі і рендерити сторінку, через відсутність необхідності рендерити всі 50 товарів одразу.
- Зменшує обсяг затребуваної пам’яті, уникаючи надмірного розподілу вузлів DOM (проблема відома як **excessive DOM size**).

Однак, з використанням react-virtualized приходить проблема навігації за допомогою клавіатури по сітці товарів. Адже компоненти завантажуються лише тоді, коли потрапляють в поле зору, а отже не фіксуються браузером як елементи, які можливо фокусувати, бо вони просто не присутні у DOM сторінки. Це вирішується за допомогою використання так званих скелетонів (легких до завантаження компонентів), що будуть рендеритись незважаючи на те, чи вони видимі. Вони матимуть можливість фокусу. Коли людина здійснить навігацію до нього, скелетон буде замінений на картку товару.

```
if (!isVisible) {
      return (
        <div role='row' key={key}>
          <div role='gridcell'>
            <TabIndexButton
              aria-label={`Товар завантажується. Зачекайте будь ласка`}
              style={{ ...style }}
            >
              <LoadingSpinner />
            </TabIndexButton>
          </div>
        </div>
      )
    }
```

React.js UseCallback

useCallback — це хук в React.js, що створює мемоізовану версію функції, яка змінюється, лише якщо одна із залежностей змінюється. Це корисно під час передачі функцій зворотного виклику (callbacks) оптимізованим дочірнім компонентам, які покладаються на рівність посилань, щоб запобігти непотрібним обчисленням.

```
//actions
  const add = useCallback(
    async function () {
      dispatch(crtSlice.add(this))
    },
    [dispatch],
  )

  const remove = useCallback(
    async function () {
      dispatch(crtSlice.removeOne(this))
    },
    [dispatch],
  )

  const removeAll = useCallback(
    async function () {
      dispatch(crtSlice.removeAll(this))
    },
    [dispatch],
  )

  return [{ items, totalCost }, add, remove, removeAll]
```

useCallback було застосовано, аби передати кожному компоненту товару функції для можливості додати до кошику або списку бажаних товарів, таким чином коли картки товарів ререндеряться, коли потрапляють у viewport користувача, не повинні обчислювати функції ще раз.

4.4 Бекенд та побудова API

Необхідно було створити RESTful прикладний програмний інтерфейс та проміжне програмне забезпечення за допомогою Express.js та Node.js.

Спершу було змодельована архітектура серверної частини. Вона полягає в тому, що кожна кінцева точка прикладного програмного інтерфейса використовує відповідний контролер, який в свою чергу використовує один або більше сервісів.

Сервіси ж використовують визначені моделі сутностей за допомогою ODM-бібліотеки mongoose для мутацій даних в базі даних. Потім дані повертаються до контролерів, які ті відсилають клієнту.

Таким чином, контролери є менеджерами запитів, коли сервіси виконують роль робітників. Це полегшує підтримку коду, оскільки кожна частина системи відповідає за конкретну функцію.

Додатково до архітектури входять проміжні функції, що перевіряють, чи користувач авторизований (якщо endpoint вимагає цього), виконують кешування, й логування та обробку помилок.

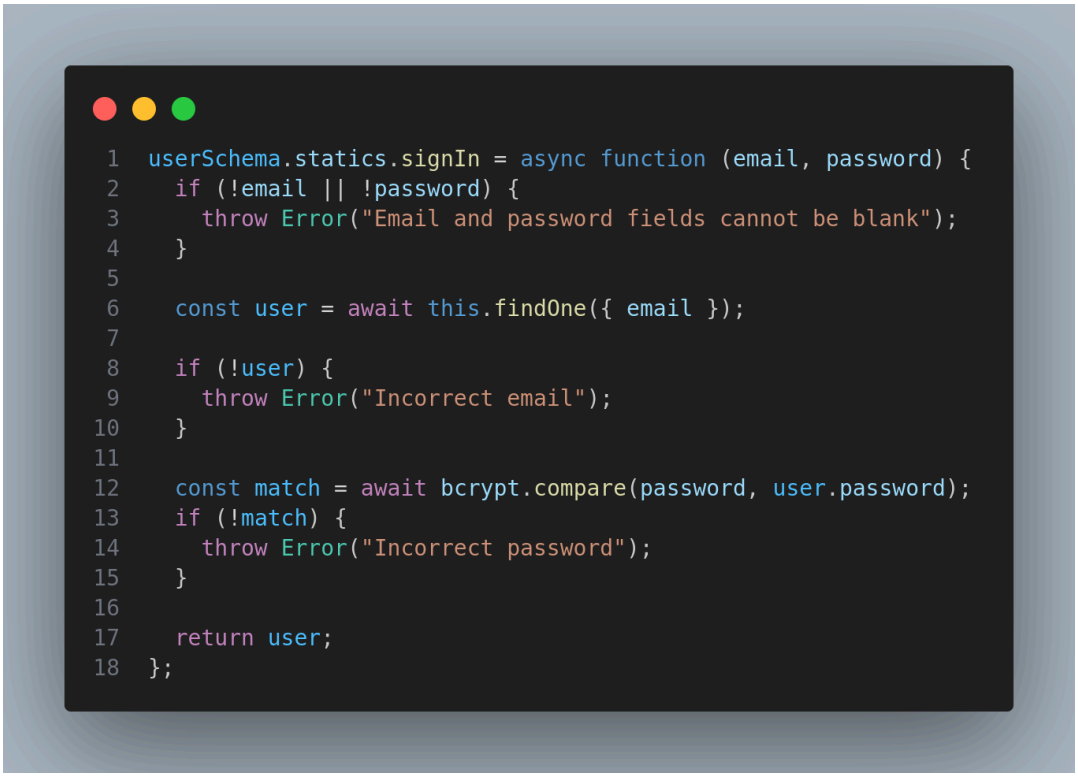
Приклад визначеного набору кінцевих точок можемо бачити знизу. Це частина API для функціоналу користувача. Вона надає точки для: реєстрації, авторизації, синхронізації кошику та списку бажаних товарів.

```
import express from "express";
import { requireAuth } from "../middleware/auth.js";

import { signIn, signUp } from "#src/controllers/user/auth.user_controller.js";
import { addLikedProduct } from "#src/controllers/user/like.user_controller.js";
import {
  addCartItem,
  syncCart,
  getCart,
  deleteCartItem,
} from "#src/controllers/user/cart.user_controller.js";

const router = express.Router();

router.post("/signIn", signIn);
router.post("/signUp", signUp);

router.use(requireAuth);

router.get("/cart/:userId/fetch", getCart);
router.patch("/cart/:userId/sync", syncCart);

router.post("/cart/:userId/add/:productId", addCartItem);
router.delete("/cart/:userId/delete/:productId", deleteCartItem);

router.patch("/like/:userId", addLikedProduct);

export { router as userRoutes };

```

Розглянемо детальніше як побудований один з endpoint-ів інтерфейсу на прикладі. Візьмемо до уваги endpoint router.post(“/signIn”, SignIn), можемо зауважити, що він використовує контролер signUp

```
export const signIn = asyncErrorHandler(async (req, res, next) => {
  const { email, password } = req.body;

  const { user, token } = await authService.signIn(email, password);

  res.status(200).json({
    id: user._id,
    firstName: user.firstName,
    secondName: user.secondName,
    email: email,
    token: token,
    likedProducts: user.likedProducts,
    cart: user.cart,
    image: user.image,
  });
});
```

Можемо бачити, що контролер взаємодіє з моделлю користувача та викликає його статичний метод SignIn. Цей метод визначений на самій моделі.



```
userSchema.statics.signIn = async function (email, password) {
  if (!email || !password) {
    throw Error("Email and password fields cannot be blank");
  }

  const user = await this.findOne({ email });

  if (!user) {
    throw Error("Incorrect email");
  }

  const match = await bcrypt.compare(password, user.password);
  if (!match) {
    throw Error("Incorrect password");
  }

  return user;
};
```

Після того, як ми запевнились, що користувач з наданими даними існує, ми повертаємо його json у відповіді.

Отже, ми показали, як працює endpoint, де контроллер служить керуючою частиною, сервіс (у даному випадку статичний метод на моделі) служить за виконавця роботи, а визначення його шляху відбувається у файлі user.routes .

Варто зазначити, що розглянутий нами endpoint використовує проміжне програмне забезпечення. asyncErrorHandler який огортає його тіло слугує передає всі помилки, які трапляються в глобальний проміжну функцію для обробки помилок

```
export const asyncErrorHandler = (controller) => {
  return (req, res, next) => {
    controller(req, res, next).catch((err) => next(err));
  };
};

```

Воно визначає яку відповідь у разі помилки та який статус код, надсилати назад користувачеві, базуючись на тому, чи помилка викликана некоректними діями користувача, чи набором зазначених в переліку помилок, обирається повідомлення, яке найкраще опише проблему. Якщо викликану помилку не зазначено, надсилається повідомлення загального змісту.

```
import _Error from "#src/utils/error.js";
import { mainLogger as ml } from "#src/utils/loggers.js"; // Adjust the path as needed

const DUPLICATE_FIELD_ERROR = 11000;
export const errorHandlingMiddleware = (error, req, res, next) => {
  ml.error(error.stack);

  const resError = {
    statusCode: error.statusCode || 500,
    status: error.status || "error",
  };

  if (process.env.NODE_ENV === "production") {
    let e = { ...error, name: error.name };

    if (!e.isOperational) {
      switch (e.name) {
        case "CastError":
          e = new _Error(`Invalid value for ${error.path}: ${error.path}`, 400);
          break;
        case "ValidationError":
          const errors = Object.values(error.errors).map((el) => el.message);
          e = new _Error(`Invalid input data. ${errors.join(". ")}`, 400);
          break;
        case "JsonWebTokenError":
          e = new _Error("Invalid token. Please log in again", 401);
          break;
      }

      switch (e.code) {
        case DUPLICATE_FIELD_ERROR:
          e = new _Error(
            `Duplicate field value: ${Object.keys(error.keyValue)}: ${
              Object.values(error.keyValue)[0]
            }`,
            400
          );
          break;
      }
    }

    const GENERIC_ERROR_MESSAGE =
      "Something went wrong! Please try again later.";
    resError.message = e.isOperational ? e.message : GENERIC_ERROR_MESSAGE;

    resError.statusCode = e.statusCode || resError.statusCode;
  } else {
    resError.stack = error.stack;
    resError.error = error;
    resError.message = error.message;
  }

  res.status(resError.statusCode).json(resError);
};

```

Проміжне програмне забезпечення для обробки помилок

Якщо ж контролер не був обгорнутий в asyncErrorHandler, глобальні обробники помилок зупинять додаток.

```
process.on("uncaughtException", (err) => {
  ml.error(`UncaughtException occured. ${err.message}`);
  process.exit(ERROR_EXIT_CODE);
});

process.on("unhandledRejection", (err) => {
  ml.error(err.message);
  cleanup(server, ERROR_EXIT_CODE);
});
```

Також визначено проміжна функція для захисту ресурсів від неавторизованих користувачів та використовується **express-winston** та **winston-mongodb бібліотеки** для логування помилок та невдалих запитів у базу даних.

```
export const infoLogger = expressWinston.logger({
  winstonInstance: ml,
  expressFormat: true,
  statusLevels: true,
})

const errorLoggerTransports = [new transports.Console()]

if (process.env.NODE_ENV !== 'production') {
  errorLoggerTransports.push(new transports.File({ filename: 'errors.log' }))
}

export const errorLogger = expressWinston.errorLogger({
  transports: errorLoggerTransports,
  format: winston.format.combine(
    winston.format.json(),
    winston.format.errors({ stack: true }),
    winston.format.timestamp(),
    winston.format.prettyPrint(),
    winston.format.printf(
      (info) => `[${info.timestamp}] ${info.level}: ${info.meta.message}`,
    ),
  ),
  expressFormat: true,
  statusLevels: true,
})
```

4.4.1 Система фільтрації та сортування

Система дозволяє фільтрувати за ціною, брендом, країної виробництва та іншими унікальними характеристиками товарів. Кожна категорія містить свій унікальний набір фільтрів. Наприклад, категорія “Для Собак, Ветпрепарати” має такі фільтри: "Бренд", "Лікарська форма","Призначення", "Тип", "Хвороба", "Країна-виробник товару". Більш загальні категорії містять тільки "Бренд" та "Країна-виробник товару".

Реалізована система підтримує актуальність фільтрів. Тобто, при застосуванні певного фільтру, буде згенерована нова мапа фільтрів, яка буде актуальна до поточного каталогу товарів.

Це реалізується за допомогою перетину мап характеристик наборів товарів, що були створені за допомогою застосування кожного з фільтрів. Таким чином, обираються тільки ті фільтри та їх опції, що будуть актуальні в поточному контексті.

```
if (slugKey === 'tsina') {
  filteredCharacteristics = await characteristicsQuery
    .where('price')
    .gte(slugOptions[0])
    .lte(slugOptions[1])
    .exec()
} else {
  const regexFilterValues = options.map(
    (value) => new RegExp(`^${value}$`, 'i'),
  )

  filteredCharacteristics = await characteristicsQuery
    .where(`characteristics.${key}`, {
      $in: regexFilterValues,
    })
    .exec()
}

const filterMap = getFiltersMap(filteredCharacteristics, activeCategory)

if (slugKey !== 'tsina') {
  const allFilterValues = await Product.distinct(
    `characteristics.${key}`,
    query.getQuery(),
  )

  filterMap.set(key, allFilterValues)
}

allFilterMaps.push(filterMap)
}
```

```
let filtersMap = []
  const ONLY_PAGE_FILTER = 1
  if (filters.size > ONLY_PAGE_FILTER) {
    let allFilterMaps = await getAllFilterMaps(query, filters)
    filtersMap = intersectMaps(...allFilterMaps)
  } else {
    let allProducts = await Product.find(query.getQuery())
      .select('characteristics')
      .sort({ createdAt: -1 })
      .exec()

    filtersMap = getFiltersMap(allProducts, activeCategory)
  }
```

Також, товари можна сортувати за ціною та рейтингом. Реалізується за допомогою стандартного інструмента mongoose **sort:**

```
query = query.sort({ price: 1 })
```

4.4.2 Система для пошуку

Реалізована за допомогою створення текстового індексу на масиві ключових слів кожного з товарів [42]. Індекс створюється наступним чином: productSchema.index({ keywords: 'text' }). Для кожного продукту було визначено його найважливіші ключові слова, приорітизуючи назву над описом та збережено у його документ.

```
function getKeywords(text) {
  // Split the text into words
  let words = text?.split(' ') || []


  // Count the frequency of each word
  let wordFrequency = {}
  words.forEach((word) => {
    if (word) {
      wordFrequency[word] = (wordFrequency[word] || 0) + 1
    }
  })


  // Sort the words by frequency
  let sortedWords = Object.entries(wordFrequency).sort((a, b) => b[1] - a[1])


  return sortedWords.map((word) => word[0])
}


const products = await Product.find({}).sort({ createdAt: -1 }).exec()
const MAX_KEYWORDS = 20


const productsKws = products.map((product) => {
  const productString = stringify(product)


  const allKeywords = getKeywords(productString)


  const processedName = process(product.name)
  const nameKeywords = getKeywords(processedName)


  const remainingKeywords = MAX_KEYWORDS - nameKeywords.length
  const additionalKeywords =
    remainingKeywords > 0 ? allKeywords.slice(0, remainingKeywords) : []


  return {
    id: product._id,
    productData: {
      keywords: Array.from(new Set([...nameKeywords, ...additionalKeywords])),
    },
  }
})


// updateProducots
const result = await updateProducts(productsKws)


return [result]
```

Знайдені товари можна фільтрувати та сортувати, так само, як і товари у певній категорії. Однак, фільтрація доступна тільки в межах загальних характеристик, таких як "Бренд" та "Країна-виробник товару".

4.4.3 Оптимізація продуктивності

Кешування на стороні сервера

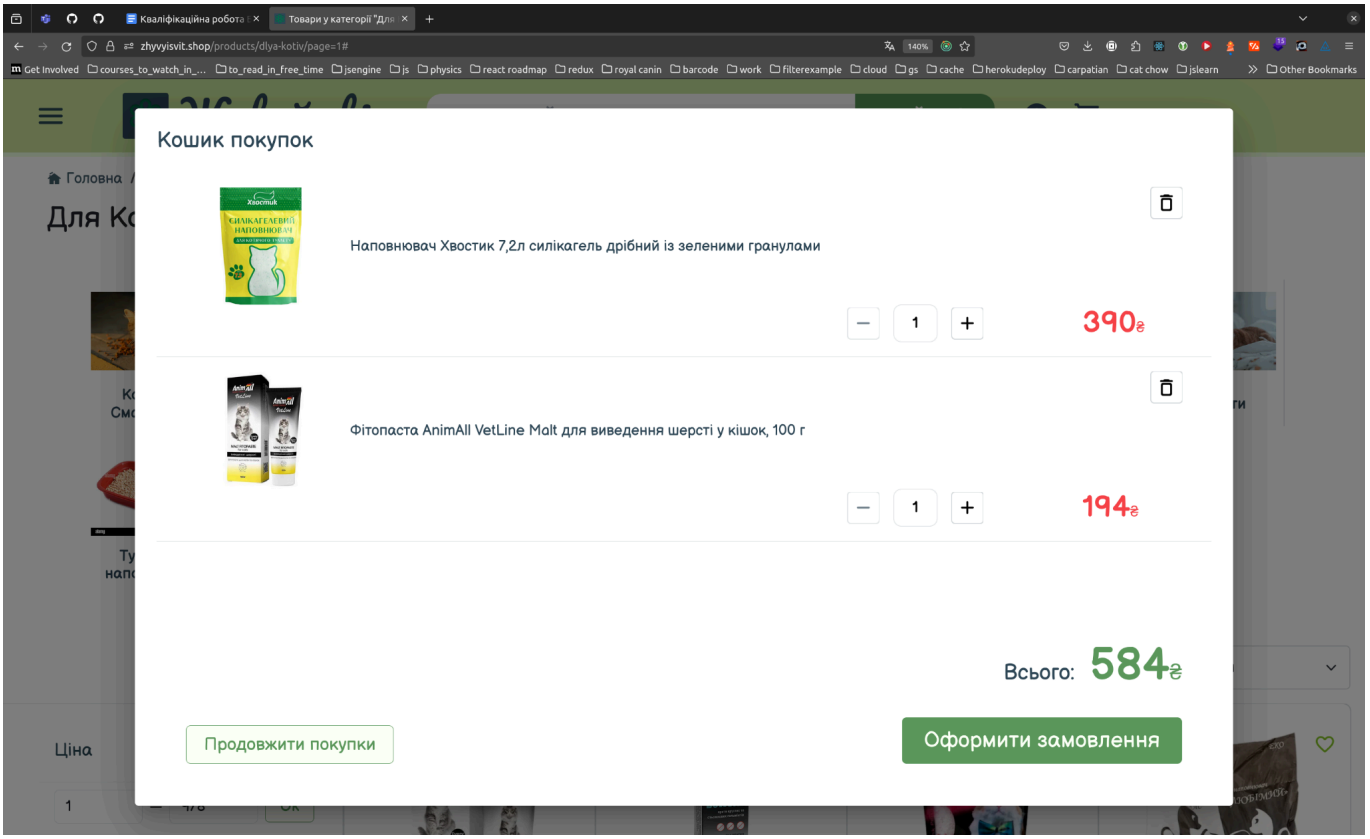
Для кешування на стороні сервера була використана бібліотека **memory-cache.** Це підвищує продуктивність, зменшує навантаження на сервер і забезпечує менший час відповіді. Це просте та легке рішення, яке підходить для малих та середніх програм [37].

Однак у нього є обмеження, такі як обмежений розмір кешу та видалення при холодних стартах **serverless** серверу [37].

```
import cache from 'memory-cache'

let cacheMiddleware = (duration) => {
  return (req, res, next) => {
    let key = '__express__' + req.originalUrl || req.url
    let cachedBody = cache.get(key)
    if (cachedBody) {
      res.send(cachedBody)
      return
    } else {
      res.sendResponse = res.send
      res.send = (body) => {
        cache.put(key, body, duration * 1000)
        res.sendResponse(body)
      }
      next()
    }
  }
}

export default cacheMiddleware

```

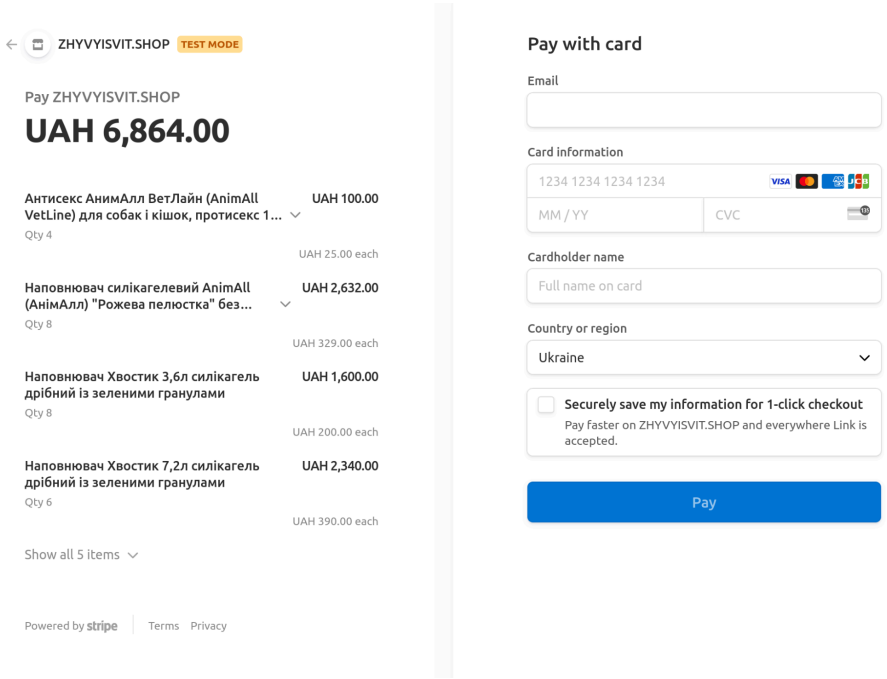
4.5 Система покупок

Покупки можливі виключно через модальне вікно кошика. В ньому представлений список товарів, обраних для покупки, а також надана детальна інформація про кожен товар.



Варто зазначити, що система оптимізована для кращої продуктивності. Коли користувач додав товар до кошику, стан додатку змінюється для відображення цього користувачеві. Якщо ж користувач не авторизований, товари зберігаються в localStorage. Це відбувається, коли він відкриває вікно кошику або коли компонент галереї товарів зникає з DOM. Для авторизованих користувачів процес є подібним, але додатково відбувається синхронізація даних користувача на стороні сервера. Крім того, при вході в акаунт, всі товари, що були в localStorage, додаються до тих, що користувач вже мав в своєму обліковому записі.

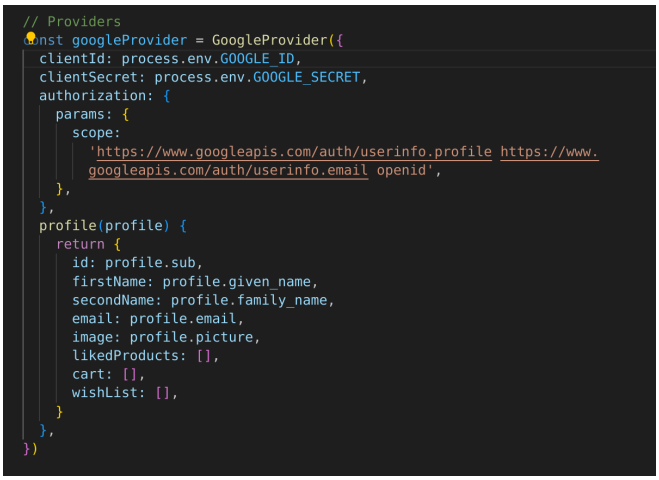
Платежі проводяться у тестовому режимі за допомогою платіжного обробника **Stripe.** Платіжний обробник проводить платежі від клієнтів, перевіряючи деталі платіжної картки та забезпечуючи безпечну передачу коштів між рахунками.



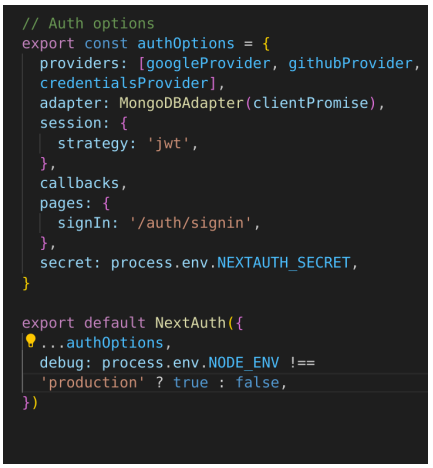
4.6 Система авторизації

Виконана за допомогою **nextauth.js** підтримує авторизацію з використання Google та Github, а також звичайних логіну та паролю.

У конфігурації nextauth.js використовуються провайдери авторизації відповідних сервісів та одногу для Credentials. А також, використовується адаптер MongoDB для зберігання сесій та облікових записів користувачів.



```
// Providers
const googleProvider = GoogleProvider({
  clientId: process.env.GOOGLE_ID,
  clientSecret: process.env.GOOGLE_SECRET,
  authorization: {
    params: {
      scope:
        'https://www.googleapis.com/auth/userinfo.profile https://www.
        googleapis.com/auth/userinfo.email openid',
    },
  },
  profile(profile) {
    return {
      id: profile.sub,
      firstName: profile.given_name,
      secondName: profile.family_name,
      email: profile.email,
      image: profile.picture,
      likedProducts: [],
      cart: [],
      wishList: [],
    }
  },
})
```



```
// Auth options
export const authOptions = {
  providers: [googleProvider, githubProvider,
  credentialsProvider],
  adapter: MongoDBAdapter(clientPromise),
  session: {
    strategy: 'jwt',
  },
  callbacks,
  pages: {
    signIn: '/auth/signin',
  },
  secret: process.env.NEXTAUTH_SECRET,
}

export default NextAuth({
  ...authOptions,
  debug: process.env.NODE_ENV !==
  'production' ? true : false,
})
```

Однак, дані про сесію користувача зберігаються в JSON Web Token (JWT).

Коли користувач входить у систему, сервер генерує JWT, який містить дані про сесію, і відправляє його клієнту. Після цього клієнт відправляє цей токен при кожному запиті до сервера, що дозволяє серверу перевірити, чи є користувач авторизованим.

Це має кілька переваг:

- JWT може бути налаштований на автоматичне закінчення терміну дії, що забезпечує додаткову безпеку.
- Не вимагає від сервера зберігати сесії, що зменшує навантаження на базу даних.
- JWT може бути використаний для авторизації на декількох серверах

4.7 Рекомендаційна система

Для створення рекомендацій використовуєть окремий Web Worker, який отримує ключові слова та id відповідних товарів з сервера, та виконує процес порівняння схожості ключових слів вибраного користувачем товару з іншими.

Використання Web Worker приносить покращення, оскільки воно дозволяє виконувати важкі обчислення в окремому потоці, не блокуючи основний потік браузера.

Для порівняння вибирається набір товарів, що належать до всіх підкатегорій батьківської категорії вибраного товару. Тобто, якщо товар знаходиться у категорії “Для Котів, Ветпрепарати”, буде обрано всі підкатегорії “Для Котів”

Для алгоритму порівняння був використаний Term Frequency Inverse Document Frequency **(TF-IDF)**.

TF-IDF - це статистичний метод, який широко використовується в обробці природної мови та інформаційному пошуку. Він вимірює, наскільки важливий термін у документі відносно колекції документів [43].

TF-IDF складається з двох частин:

Term Frequency (TF): Це кількість разів, коли термін з’являється в документі, порівняно з загальною кількістю слів у документі. Формула для обчислення TF є наступною:

$$TF = \frac{\text{кількість разів, коли термін з’являється в документі}}{\text{загальна кількість термінів в документі}}$$

Inverse Document Frequency (IDF): IDF терміна відображає пропорцію документів у корпусі, які містять термін. Слова, які є унікальними для невеликої частини документів (наприклад, технічний жаргон), отримують вищі значення важливості, ніж слова, які є загальними для всіх документів (наприклад, “або”, “для”, “та”). Формула для обчислення IDF є наступною:

$$IDF = \log\left(\frac{\text{кількість документів у корпусі}}{\text{кількість документів у корпусі, які містять термін}}\right)$$

Вага TF-IDF терміна обчислюється шляхом множення TF та IDF:

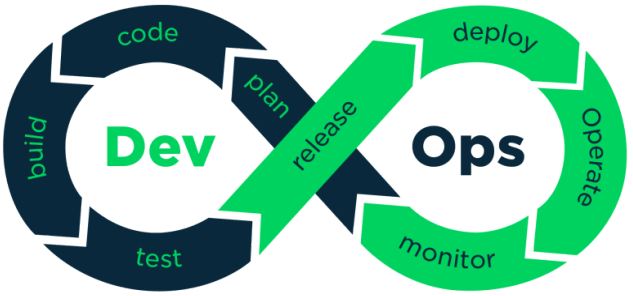
$$\text{TF-IDF} = TF * IDF$$

Таким чином, важливість терміна висока, коли він часто з’являється в даному документі та рідко з’являється в інших. Загалом, поширеність у документі, виміряна TF, збалансована рідкістю між документами, виміряною IDF. Результатом є показник TF-IDF, який відображає важливість терміна у документі порівняно з іншими у корпусі [43].

У контексті обробки природної мови та інформаційного пошуку, “корпус” - термін, що позначає велику та структуровану колекції текстів.

4.8 Побудова CI/CD пайплайнів

Для створення CI/CD пайплайну було використано GitHub Actions. Ця технологія дозволяє налаштувати автоматизовані процеси розробки, що будуть виконуватись у певних сценаріях (наприклад на pull request або push до певної гілки/ок) або за певним графіком.

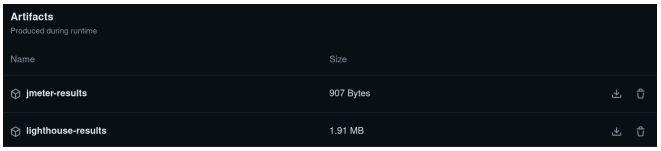


Було створено три головних робочих процеса для автоматизації інтеграції та розгортання додатку:

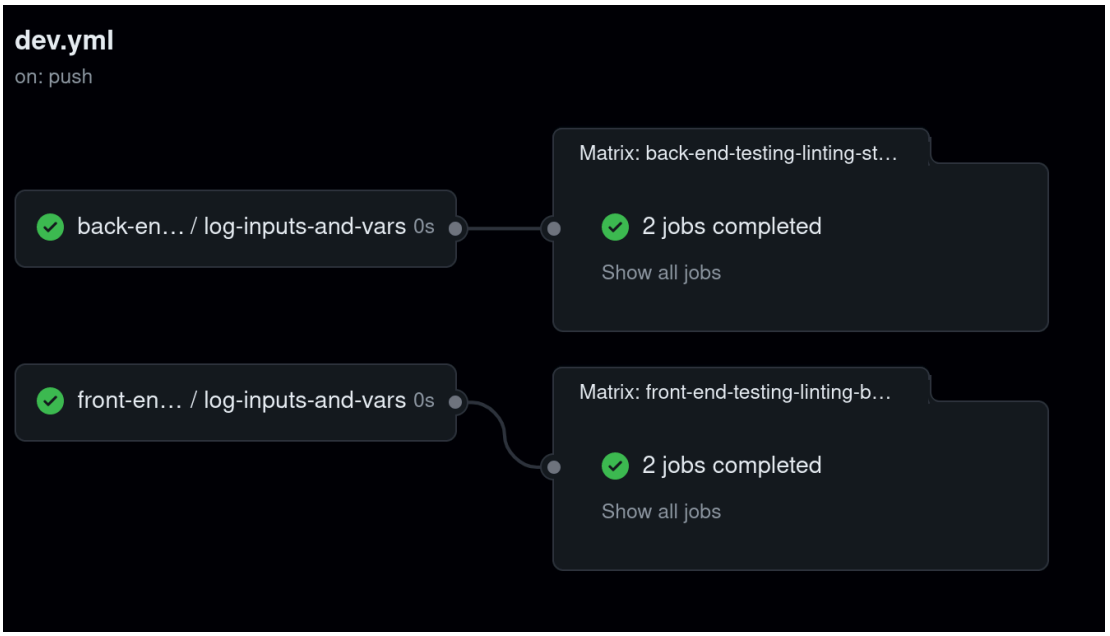
1. Deploy, що виконує наступні роботи:

- Лінтування та тестування Бек-енду та Фронтенду
- Розгортання Бек-енду та Фронтенд на хмарну платформу
- Моніторинг
  - продуктивності за допомогою Apache JMeter
  - відповідності стандартам Web Vitals Core (вимірюється продуктивність, доступність, SEO, дотримання найкращих практик у розробці)

На схемі нижче зображена схема виконання робочого процеса. Бачимо, що процеси розгортання бек-енду вимагає успішного завершення процесу його лінтування і тестування, аналогічно для фронтенду. Також, варто зазначити, що розгортання фронтенду залежить від успішного розгортання бек-енду. Після ж розгортання відбувається паралельні роботи для моніторингу застосунку, інструменти зберігаючи результати аудитів у робочому процесі.



2. Test Back-end and Front-end in development environment , що виконує лінтування та тестування Бек-енду та Фронтенду на push до **dev** гілки. Його схема зображена нижче.

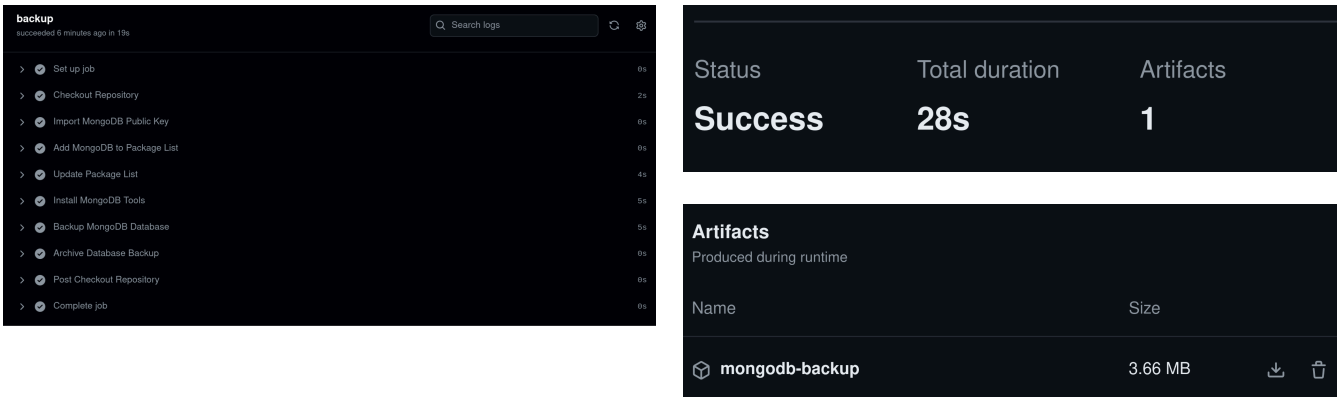


3. Database Backup, що виконує збереження стану бази даних (бекап) кожної півночі, зберігаючи архів з даними, як артефакт (github назва для результуючого файлу або набір файлів робочого процесу). Це може бути корисним для забезпечення безпеки даних користувачів та можливості відновлення в разі втрати.

Нижче зображена схема виконання робочого процеса. Процес складається з наступних основних кроків:

- Встановлення інструментів MongoDB, що включає в себе **mongodump**
- Використання команди **mongodump**, що з’єднується з Atlas MongoDb за допомогою стрічки з’єднання та копіює усі дані.
- Збереження результату як артефакту за допомогою **actions/upload-artifact@v2**

Бачимо, що процес завершено вдало з одним збереженим артефактом.



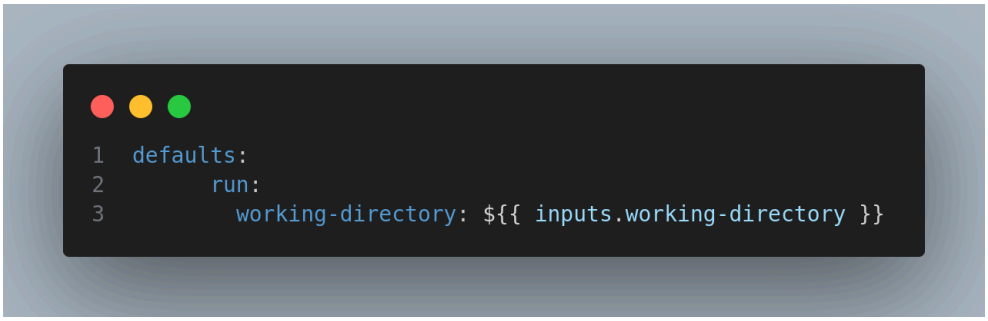
Використання повторно використовуваних робочих процесів **(reusable workflows):** GitHub Actions дозволяє вам створювати повторно використовувані робочі процеси, які можна використовувати в декількох проектах. Це дозволяє уникнути дублювання коду та забезпечує консистентність між різними проектами.

Саме Back-end Testing, Linting, and Starting та Front-end Testing, Linting, and Building є повторно використовуваними. І вони використовуються як у **deploy,** так і **dev** процесах. Ми маємо змогу сконфігурувати їх використовувати те, чи інше середовище з репозиторію github, що дозволяє підтягувати особливі до середовища значення секретних та простих змінних для застосунку.

Також, використовуються матриці для створення більш читабельних багаторазових робочих процесів, адже вони вимагають передачі параметрів секретних змінних середовища, що значно погіршує чистоту конфігурації. Крім того, це допомагає оголосити середовище, директорію виконання та інші конфігурації один раз для всіх робіт створених матрицею.

```
1  strategy:
2        fail-fast: true
3        matrix:
4          action: ['lint-test', 'build']
```

Оскільки, було використано **багатодиректорну структуру репозиторію** разом для фронтенду та бек-енду**,** налаштування активної директорії для виконання тих чи інших робіт виконувалось за допомогою опції **working-directory [35].** Вона передається з керуючого процесу до повторно використовуваних, а потім виставляється на рівні роботи, що генерує матрицю, слугуючи одним джерелом правди для всіх створених матрицею робіт.



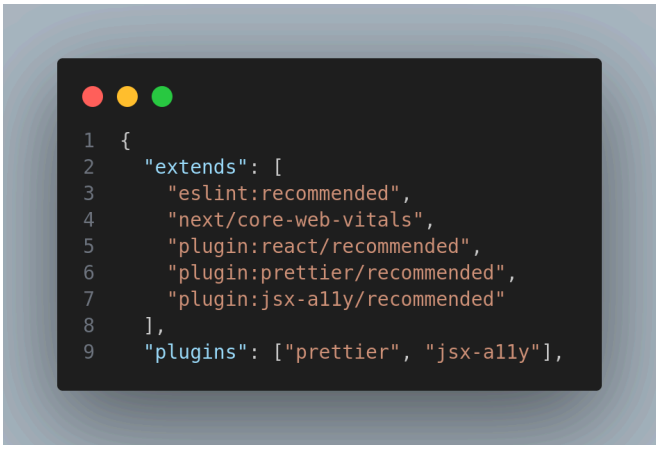
```
defaults:
      run:
        working-directory: ${{ inputs.working-directory }}
```

4.9 Лінтування

Для лінтування було використано ESLint.

Також у його конфігурацію було визначено:

- Набір рекомендованих правил ESLint, а також специфічні для Next.js, React і Prettier для фронту, та Node для беку
- Плагіни Prettier і jsx-a11y для підтримки форматування коду і доступності JSX відповідно.



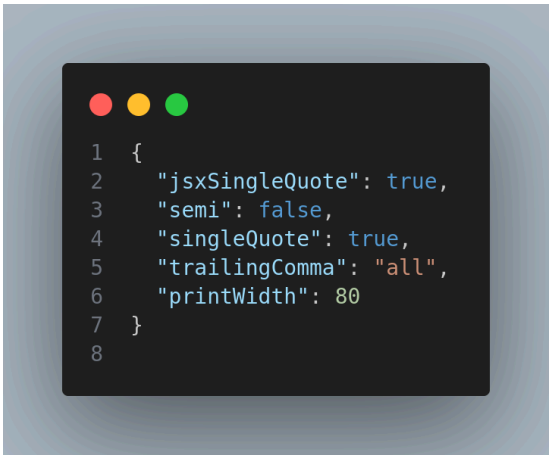
```
{
  "extends": [
    "eslint:recommended",
    "next/core-web-vitals",
    "plugin:react/recommended",
    "plugin:prettier/recommended",
    "plugin:jsx-a11y/recommended"
  ],
  "plugins": ["prettier", "jsx-a11y"],
```

```
languageOptions: {
  globals: {
    ...globals.node, // Node.js global variables
    ...globals.es2017, // ES6 global variables
    ...globals.jest, // Jest global variables
  },
  parserOptions: {
    ecmaVersion: 2020,
    sourceType: 'module',
  },
},
plugins: {
  prettier: prettier,
},
```

4.10 Форматування

Для автоматичного форматування було використано **Prettier.** Його конфігурація включає наступні налаштування:

- **jsxSingleQuote**: у JSX використовуються одинарні лапки для рядків
- **semi**: крапки з комою в кінці рядків коду не використовуються
- **singleQuote**: - використовуються одинарні лапки замість подвійних в js
- **trailingComma**: - видаляє завершальні коми в кінці об’єктів, масивів і параметрів функцій
- **printWidth**: - довжина рядка обмежується 80 символами.

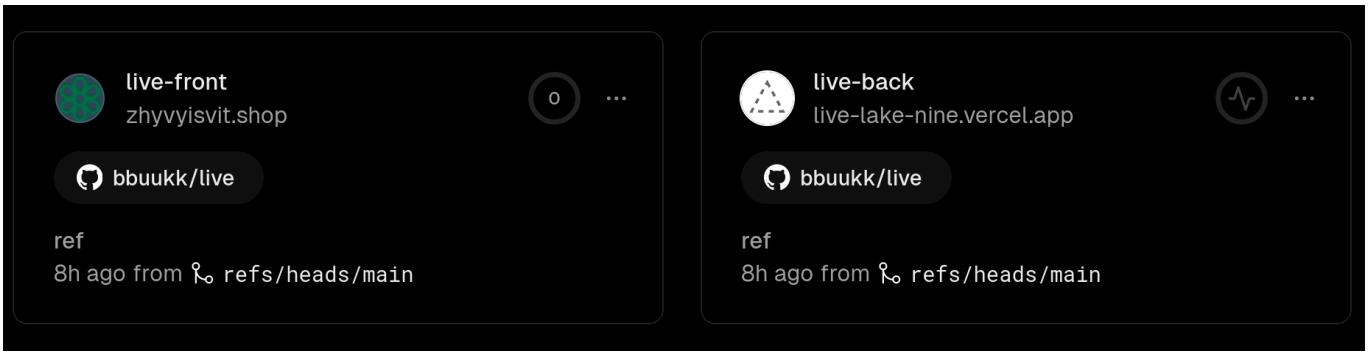


```
{
  "jsxSingleQuote": true,
  "semi": false,
  "singleQuote": true,
  "trailingComma": "all",
  "printWidth": 80
}
```

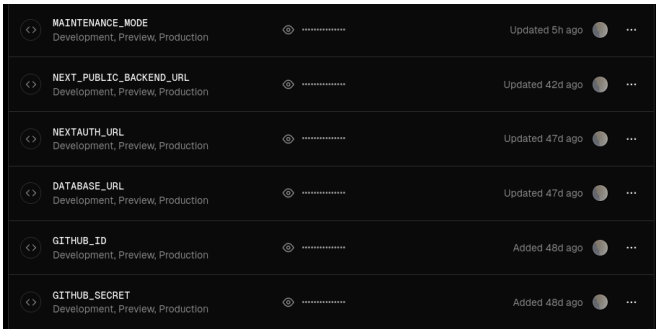
Така конфігурація набагато покращує читабельність коду

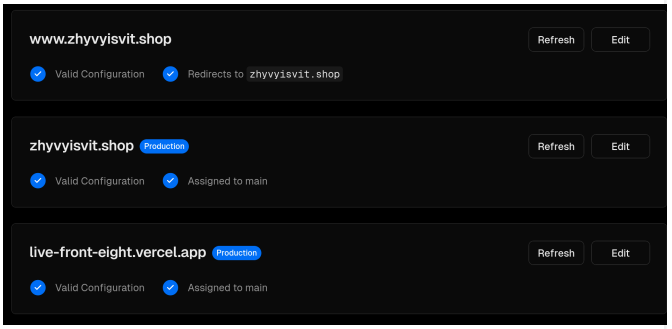
4.11 Налаштування хмарної платформи

Оскільки, як вже зазначалось, було використано **багатодиректорну структуру** на github, потрібно було створити два проекта, що будуть розгортатись з різних підпапок репозиторію.



Було встановлено змінні середовища та відключено автоматичний CI/CD. Доменне ім’я zhyvyisvit.shop було придбано через **Namecheap**. Після придбання, домен було успішно перенесено на Vercel та налаштовано для використання з фронтенд проектом.

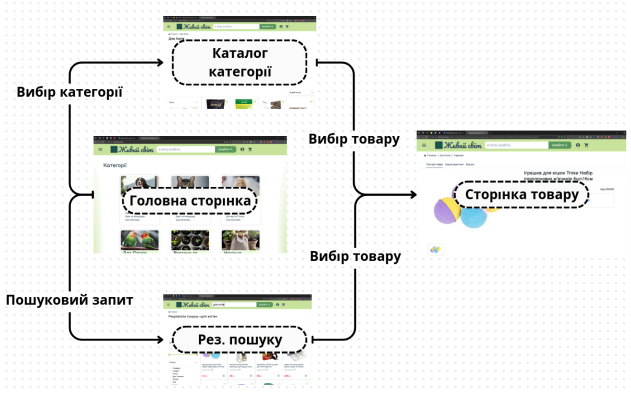




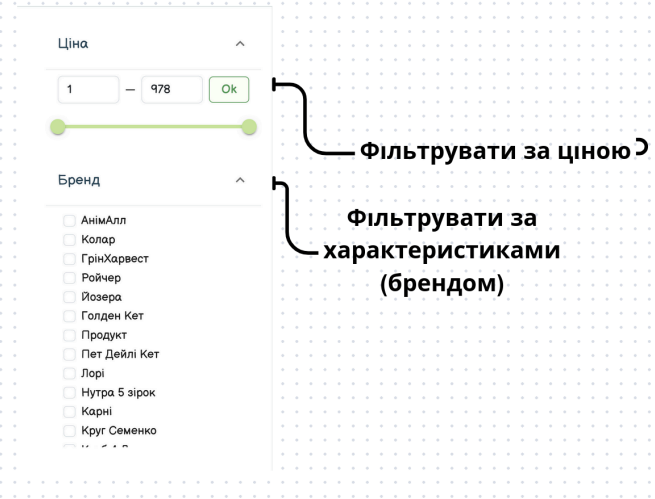
4.12 Базові сценарії функціонування проекту

Базовим сценарієм використання системи є перегляд товарів. Головна сторінка представляє набір головних категорій та їх п’яти основних підкатегорій, за якими впорядкований асортимент. Користувач обирає категорію або користається пошуком для того, щоб перейти на сторінку каталогу товарів. Звідти людина може обрати товар, що вона вподобала та перейти до індивідуальної сторінки товару для того, щоб дізнатись більше інформації.

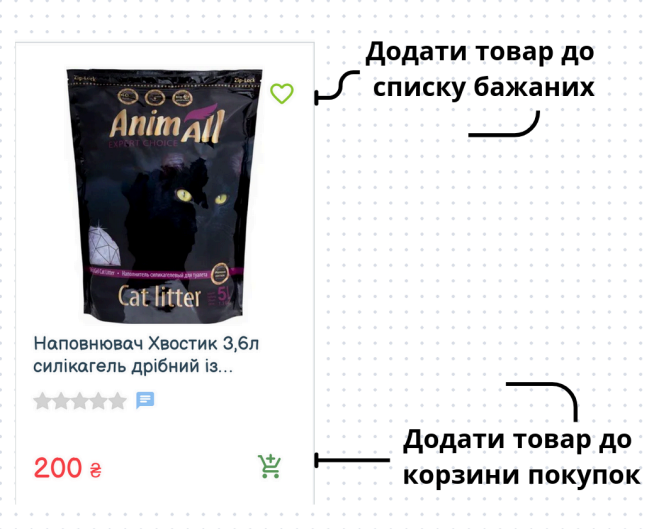
На будь-якій сторінці користувач може скористатись панеллю навігації зверху звідки він може дістатись до всіх найважливіших сторінок застосунку. Звідси, користувач може відкрити **вікно авторизації**, щоб увійти або зареєструватись або ж відкрити **кошик покупок. Крім того,** активувати **бокове меню**, що надає також посилання на сторінки загальної інформації про магазин та застосування (“Про магазин”, “Умови використання сайту”, “Правила приватності”).



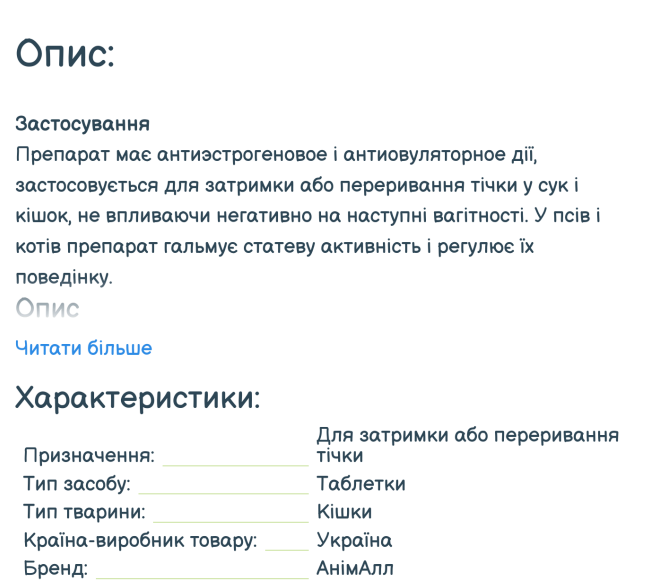
Якщо користувач має якійсь специфічні вимоги до шуканих товарів, на сторінці каталогу користувач може відфільтрувати або відсортувати асортимент.

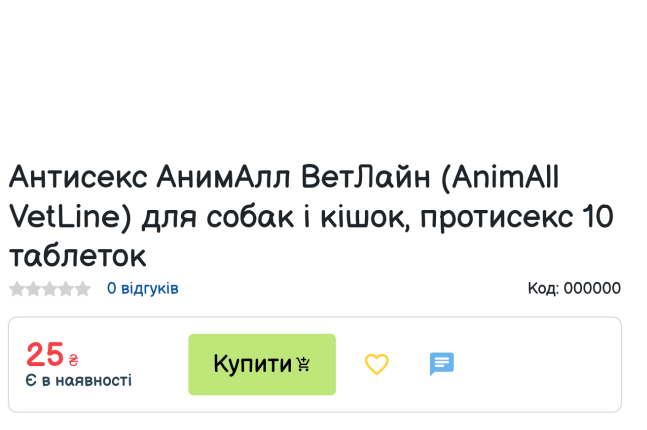


На цій же сторінці, якщо користувач може додати товар до списку бажань або до кошику покупок.

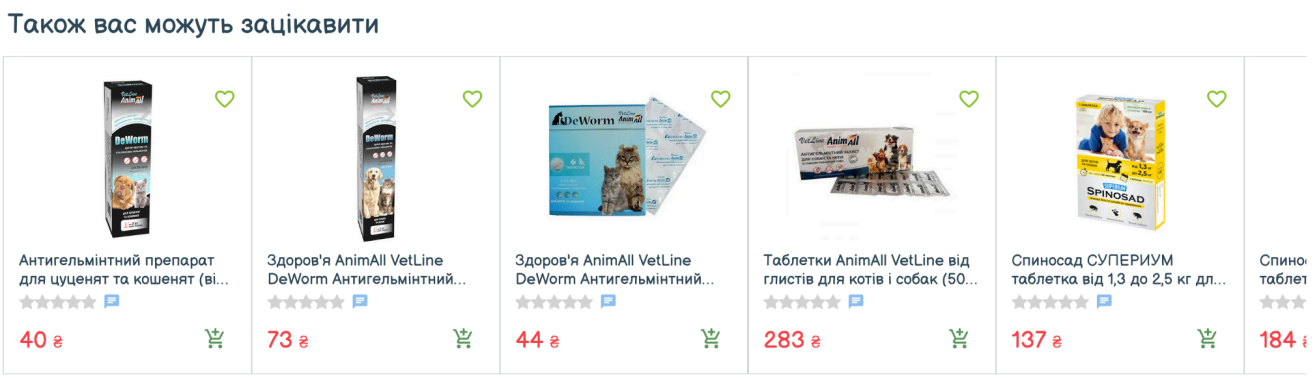


На індивідуальній же сторінці товару, користувач може детальніше вивчити характеристика товару та прочитати його опис, дізнатись ціну, додати до кошику або бажань.





Також, на цій сторінці розміщена карусель рекомендованих товарів, що може зацікавити користувача.



ВИСНОВОК

Цей розділ описує детальний алгоритм реалізації багаторівневого веб-застосунку, включаючи фронтенд, бекенд, інтеграцію CI/CD пайплайнів та систем авторизації, пошуку, рекомендацій та купівлі товарів. Також, було продемонстровані базові сценарії використання функціоналу.

Мною було успішно розроблено та стилізовано інтерфейс користувача веб-застосунку з використанням фреймворку React.js, бібліотеки компонентів MUI та препроцесора CSS Sass. Забезпечено доступність веб-застосунку для людей з обмеженими фізичними можливостями, що відповідає W3C-стандартам та рекомендаціям щодо доступності. Додатково, оптимізовано продуктивність веб-застосунку.

Також, мною був змодельований та розроблений бекенд з використанням фреймворку Express.js та Node.js, що надає API для маніпулювання та отримання даних з бази даних. Забезпечено підключення до бази даних MongoDB за допомогою Mongoose ODM. Створено моделі даних для представлення сутностей веб-застосунку. Реалізовано логіку API для обробки запитів та повернення відповідних відповідей. Розроблені проміжні функції для логування, обробки помилок та авторизації.

Потім, було інтегровано NextAuth.js для забезпечення безпечного доступу користувачів до веб-застосунку. Налаштовано провайдерів Google та Github авторизації та адаптер бази даних для NextAuth.js.

Було автоматизовано процес розгортання веб-застосунку за допомогою CI/CD пайплайнів. Забезпечено безперебійну інтеграцію та розгортання веб-застосунку за допомогою GitHub Actions. Також, за допомогою цього інструмента відбувається резервне копіювання бази даних.

Було розроблено систему для купівлі товару та забезпечено інтеграцію додатку з обробник платежів Stripe у тестовому режимі.

Було впроваджено рекомендаційну систему товарів на основі TF-IDF алгоритму. Він дозволяє порівнювати документ товару з усіма іншими у корпусі, що надає можливість відобразити найбільш релевантні рекомендації користувачу.

У кінці розділу були продемонстрований базові сценарії використання проекту кінцевим користувачем. Зображені схеми переходу користувача від однієї сторінки до іншої та описаний функціонал, що доступний на тій чи іншій сторінці.

Загалом, алгоритм реалізації демонструє, як розробити веб-додаток, що відповідатиме заявленим вимогам та буде підготовлений для майбутнього масштабування.

РОЗДІЛ 5: ТЕСТУВАННЯ, ВЕРИФІКАЦІЯ ТА ВДОСКОНАЛЕННЯ ВЕБ-ДОДАТКУ

5.1 Вступ до тестування та верифікації

Тестування та верифікація є критично важливими етапами життєвого циклу розробки веб-додатків. Вони гарантують, що додаток відповідає своїм вимогам і працює оптимально за різних умов. У цьому розділі описано системний підхід до тестування багаторівневого веб-додатку, аналізу результатів та внесення пропозицій щодо вдосконалення.

5.2 Інтеграційне тестування

Тести інтеграції призначені для перевірки того, як різні частини програми взаємодіють одна з одною.

Розглянемо приклад тесту endpoint-а POST /categories (зоб. 26). Тест перевіряє, чи можна успішно створити нову категорію та чи відповідає відповідь сервера очікуваному результату.

Це передбачає взаємодію між контролером, проміжним програмним забезпеченням (для автентифікації), сервісом та базою даних. Для тестування ми використовуємо бібліотеку **mongodb-memory-server**, яка створює екземпляр MongoDB у RAM. Це оптимальний варіант для тестів, адже базу можна легко стартувати та зупинити, вона забезпечує швидкісне читання та запис даних, ефективно імітує реальну базу даних.

```
import dotenv from "dotenv";
dotenv.config();

import supertest from "supertest";
import app from "@src/app.js";

import { setupDB, teardownDB } from "#src/__tests__/in_memory_db/db_utils.js";
import { adminToken } from "#src/__tests__/utils/admin_token.js";
import { randomCategory } from "#src/__tests__/utils/data_generator.js";

beforeAll(async function () {
  await setupDB();
});
afterAll(async function () {
  await teardownDB();
});

describe("POST /categories", () => {
  it("should create new category", async () => {
    const testCategory = {
      ...randomCategory(),
      _id: "65ad3ec1864784208de09952",
      path: "test_path",
    };

    const { statusCode, body, type } = await supertest(app)
      .post("/categories")
      .set("Authorization", `Bearer ${adminToken}`)
      .send(testCategory);

    expect(statusCode).toBe(200);
    expect(type).toBe("application/json");
    expect(body).toEqual(testCategory);
  });
});

```

5.3 Тестування продуктивності

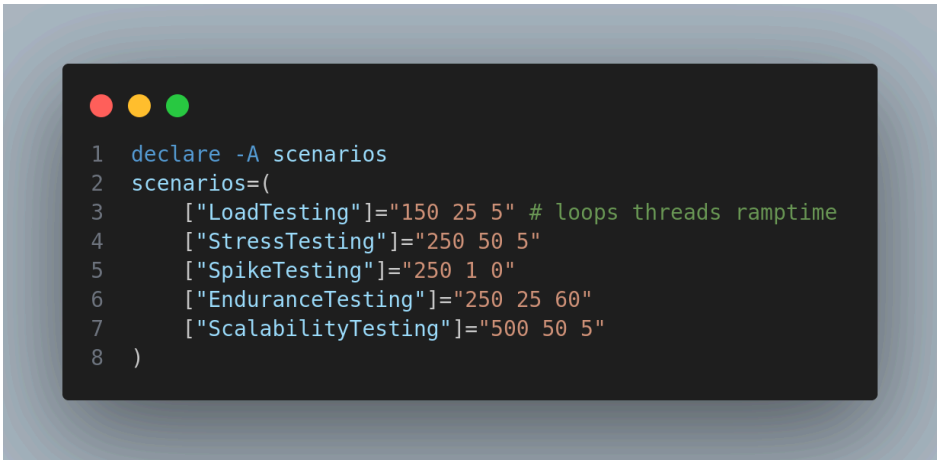
Перевірка додатку під навантаженням, щоб переконатися, що він відповідає критеріям продуктивності. Використано такий інструмент інструмент, як **Apache JMeter**, що імітує запити реальних користувачів.

Були створені також різні сценарії тестування:

- **Тестування навантаження** - для перевірки системи під очікуваним навантаженням.
- **Стрес-тестування** - перевіряє поведінку системи під інтенсивними навантаженнями та допомагає визначити точку зламу системи.

- **Тестування стрибків трафіку** - перевіряє реакцію системи на раптові великі стрибки навантаження.
- **Тестування на витривалість** - перевіряє поведінку системи під постійним навантаженням протягом тривалого періоду. Це може допомогти виявити такі проблеми, як витік пам’яті.
- **Тестування масштабованості** - перевіряє здатність системи масштабуватися для обробки збільшеного навантаження.

Кожен з них застосовується до трьох основних сторінок додатку та використовує власні параметри. Параметри включають: кількість ітерації запитів (кількість циклів, які виконуватиме кожен потік), кількість кількість одночасних потоків (імітація користувачів), час для запуску всіх потоків (**ramp Time**). Нижче вказані значення параметрів для кожного сценарію.



```
declare -A scenarios
scenarios=(
    ["LoadTesting"]="150 25 5" # loops threads ramptime
    ["StressTesting"]="250 50 5"
    ["SpikeTesting"]="250 1 0"
    ["EnduranceTesting"]="250 25 60"
    ["ScalabilityTesting"]="500 50 5"
)
```

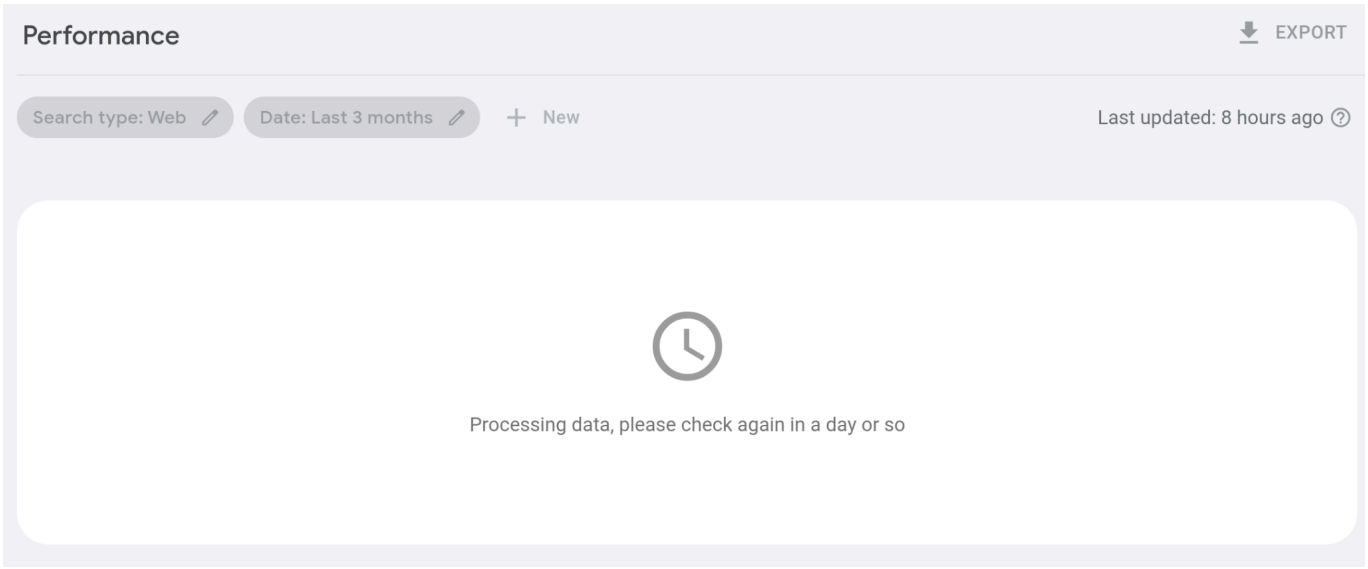
Крім того, в конфігурації **JMeter** було використано **GaussianRandomTimer**, що дозволяє моделювати випадкову затримку між запитами. Це моделює реальне навантаження на сервер, коли користувачі навігуються по сторінках з різними проміжками у часі.

Виконання тестування JMeter був вбудований в Github Actions CI/CD пайплайн. Аудит виконується за допомогою звичайного скрипту. У ньому зазначено колекція шляхів для тестування та доменне ім’я. Виконання відбувається в подвійному циклі, для кожного сценарію та шляху. Результат зберігається як артефакт у пайплайні

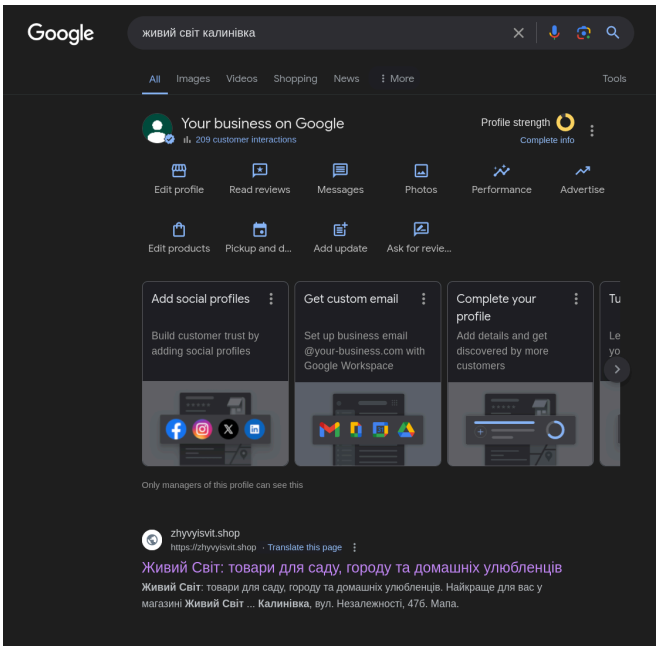
```
# Domain
DOMAIN="zhyvyisvit.shop"
# URL paths to test
URL_PATHS="/,/products/dlya-kotiv/page=1,product/
-fitopasta-animall-vetline-malt-dlya-vyvedennya-shersti-u-kishok---100-h/
65b2606f213addb487b8cab22/about"

IFS=',' read -ra ADDR <<<"$URL_PATHS"

# Run JMeter for each scenario and each URL path
for scenario in "${!scenarios[@]}"; do
    params=${scenarios[$scenario]}
    IFS=' ' read -ra PARAMS <<<"$params"

    echo PARAMS: loops: ${PARAMS[0]},threads: ${PARAMS[1]}, ramptime: ${PARAMS[2]}
    for URL_PATH in "${ADDR[@]}"; do
        echo "Running $scenario for '$URL_PATH' path"
        o
        SLUG_PATH=$(echo $URL_PATH | tr '/' '_')
        SCENARIO_PATH=$(echo $scenario | tr ' ' '_')

        mkdir -p "./results/$SCENARIO_PATH/$SLUG_PATH"

        jmeter -n -t ./.github/jmeter/test_plan.jmx -Jdomain=$DOMAIN -Jpath=$URL_PATH
        -l "./results/$SCENARIO_PATH/$SLUG_PATH/$SLUG_PATH-results.jtl" -e -o "./
        results/$SCENARIO_PATH/$SLUG_PATH/res" -Jloops=${PARAMS[0]} -Jthreads=${PARAMS
        [1]} -Jramptime=${PARAMS[2]}
    done
done
```

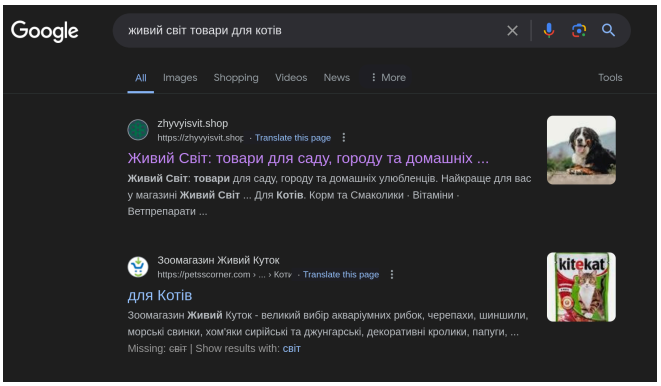
5.4 Тестування SEO

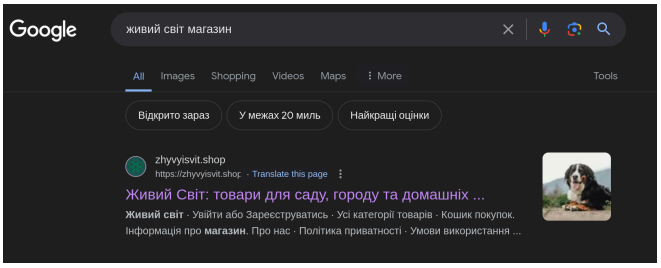
Використавши **Google Search Console** для автоматичної оцінки SEO, бачимо, що Google ще не має достатньо даних для відображення. Зазвичай Google займає деякий час, щоб просканувати та проіндексувати вміст нового сайту.

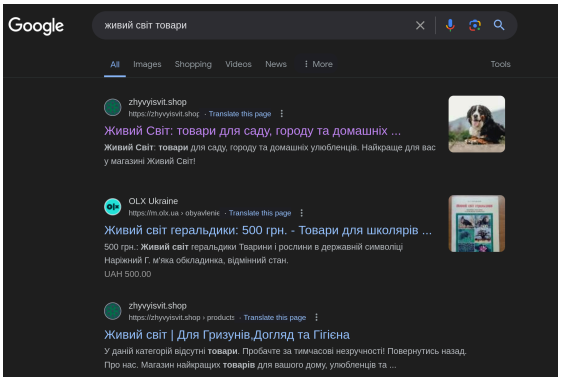


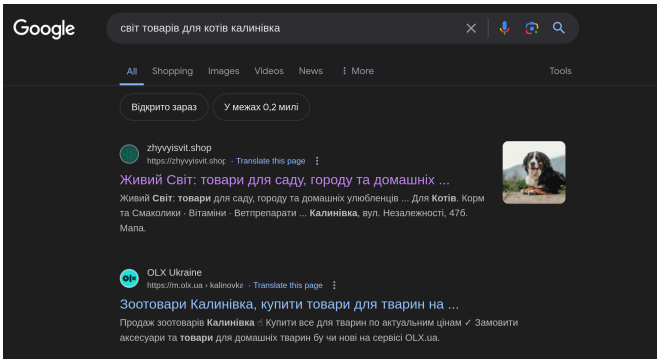
Однак, спробувавши знайти додаток самотужки в google, можемо упевнитись, що застосунок справді знаходитьсься у топі пошуку за основними ключовими словами.









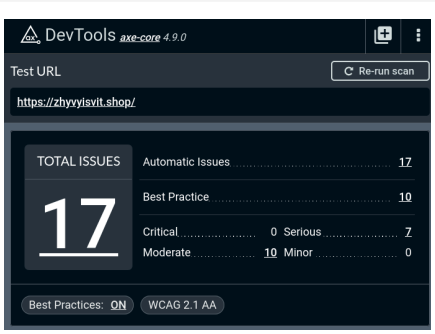


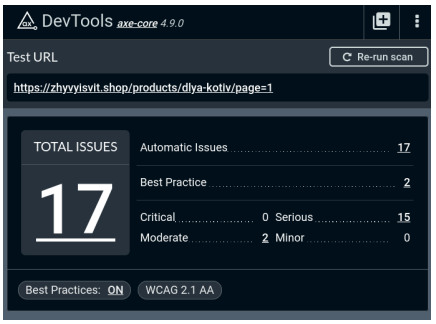
5.5 Тестування доступності

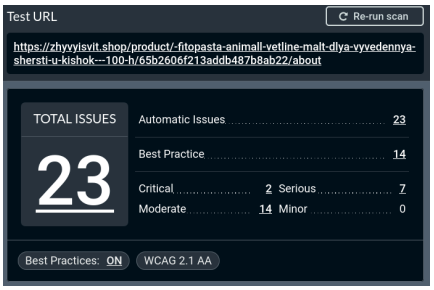
Навігація по веб-сайту за допомогою клавіатури була ретельно перевірена вручну за допомогою програми для читання екрана, Orca Screen Reader (Linux).

Також, було використано **Axe DevTools**. Це інструмент автоматичного тестування веб-доступності, розроблений, щоб допомогти розробникам створювати більш доступні веб-сайти та веб-додатки. Це розширення Chrome, яке працює на основі всесвітньо надійного механізму тестування axe-core [45]. Бачимо, що додаток

дотримується WCAG 2.1 AA на трьох основних сторінках (Головній, каталогу та індивідуальній для товару відповідно).

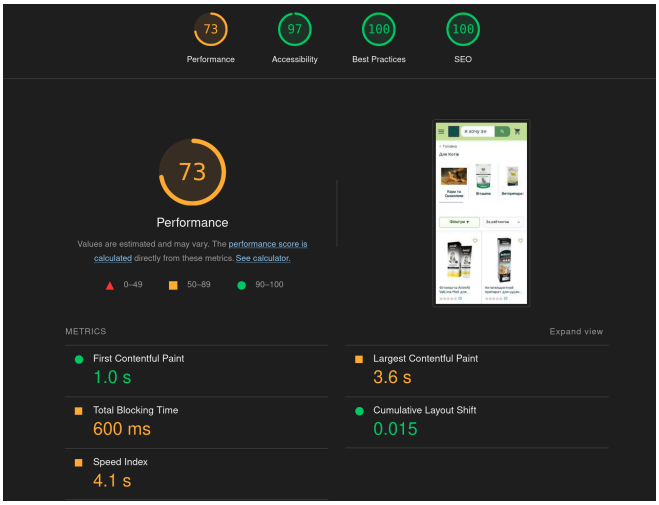


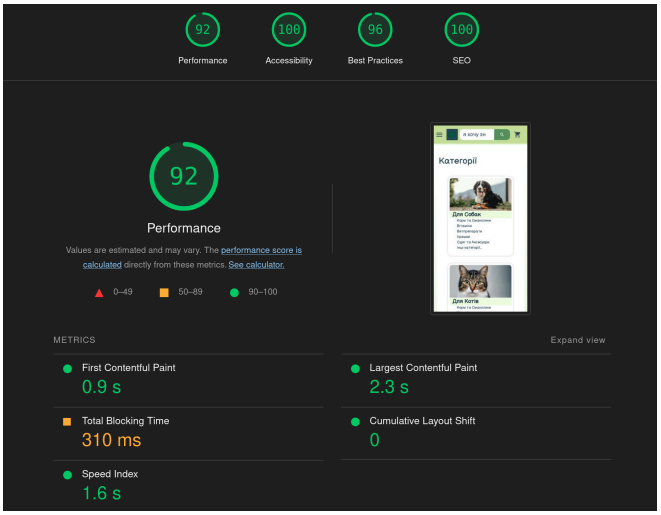


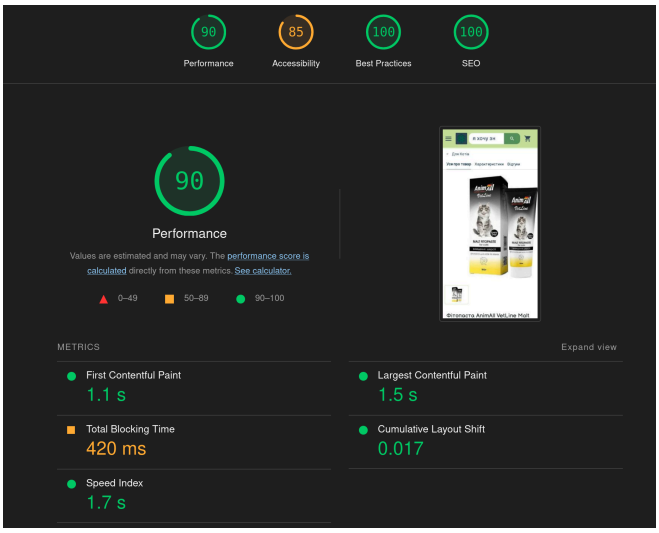


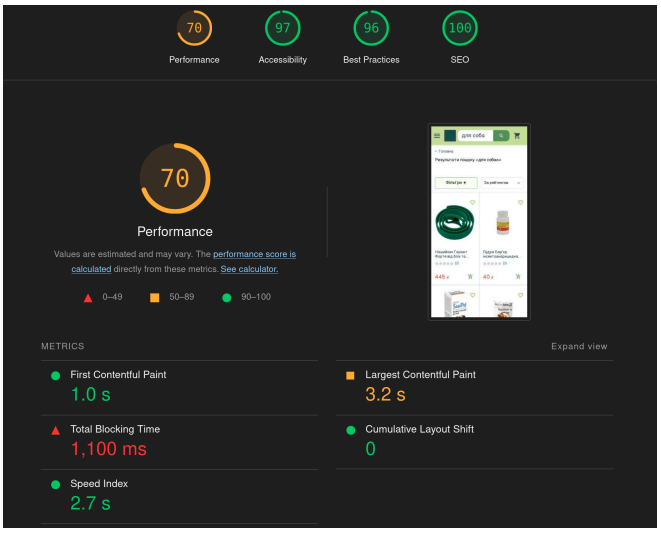
5.6 Core Web Vitals тестування

Google Lighthouse проводить комплексне тестування додатку на Core Web Vitals.









Отже, за результати можна надати увагу наступним показникам:

- Сторінки мають різні показники продуктивності, що варіюються від помірних до високих.
- **FCP** відносно швидкий у всіх тестах
- **LCP** показує, що є потреба для покращення розмірів асетів
- **TTI** загалом низький, що свідчить про низький час відгуку
- **TBT** варіюється, але залишається низьким у більшості випадків, що свідчить про мінімальні затримки через виконання скриптів
- **CLS** чудова на всіх сторінках, що свідчить про стабільні візуальні макети

Отже, є деяка варіативність в показниках продуктивності, і метрики, такі як **LCP**, можуть бути оптимізовані краще. Однак, загалом, додаток має потенціал для надання гарного користувацького досвіду.

5.7 Пропозиції щодо подальшого вдосконалення

На основі аналізу результатів тестування висувається кілька пропозицій щодо вдосконалення веб-додатку:

Покращення користувацького досвіду

Редизайн користувацьких потоків та інтерфейсів на основі відгуків користувачів для підвищення зручності використання.

Налаштування продуктивності

Оптимізація запитів до бази даних, впровадження стратегій кешування для покращення часу завантаження та загальний рефакторинг для покращення Page Load Time.

Безперервний моніторинг

Налаштування розширених інструментів моніторингу для виявлення та проактивного реагування на проблеми.

Посилення безпеки

Регулярні аудити коду, покращене автоматизоване тестування безпеки, використання інструментів для ідентифікації та виправлення вразливостей в реальному часі.

Використання TypeScript

Використання TypeScript у веб-додатку може значно покращити якість коду та продуктивність розробки.

Розробка системи рекомендацій, заснованої на користувачах

Розробка персоналізованої системи рекомендацій, яка аналізує поведінку користувачів, їхні вподобання та історію переглядів, може значно покращити користувацький досвід та успішність бізнесу.

Покращення асетів

Використання платіжного обробника Google Pay або Monobank, що повністю підтримуються в Україні та піднімають рівень довіри клієнтів.

Покращення асетів

Використання високоякісних, привабливих зображень може зробити сайт більш привабливим та інтуїтивно зрозумілим.

Використання Microdata та Schema.org для SEO

Використання Microdata та Schema.org може покращити SEO, оскільки це допомагає пошуковим системам краще розуміти контент сайту.

Використання системи голосового управління

Інтеграція системи голосового управління може зробити сайт ще більш доступним для людей з обмеженими можливостями. Можна буде використати бібліотеку **react-speech-recognition.**

Надання можливості модифікувати гарячі клавіші

Надання користувачам можливості налаштовувати гарячі клавіші відповідно до їхніх потреб може допомогти уникнути конфліктів з гарячими клавішами скрін рідера та покращити доступність.

ВИСНОВОК

В даному розділі мною було описано системний підхід до тестування та аналізу веб-додатку. Також, я висунув ряд пропозицій щодо його вдосконалення. Впровадження цих пропозицій дозволить покращити функціональність, продуктивність, доступність та безпеку веб-додатку, а також зробити його більш зручним та привабливим для користувачів

Було проведене тестування:

- **Інтеграційне тестування**, що допомагає перевірити взаємодію компонентів веб-додатку. Наприклад, тестується створення категорії та відповідь сервера.
- **Тестування продуктивності** під навантаженням**.** Для цього був використаний Apache JMeter з різними сценаріями навантаження.
- Навігація по сайту була перевірена вручну за допомогою програми для читання екрана. Також, був використаний автоматичний інструмент **Axe DevTools**. Тестування показало, що додаток відповілдає стандартам WCAG 2.1 AA на трьох ключових сторінках.
- Провів комплексну оцінку додатку на метрики від Google (Core Web Vitals) за допомогою Google Lighthouse. Тестування показало, що FCP є швидким, LCP потребує покращення розмірів асетів, TTI та TBT обидва низькі, а CLS є чудовим.

Пропозиції щодо вдосконалення:

- Налаштувати продуктивність (оптимізувати запити, впровадити кращі стратегії кешування).
- Забезпечити безперервний моніторинг безпеки (проводити аудити, покращити тестування безпеки).
- Використовувати TypeScript.
- Розробити user-based систему рекомендацій.
- Покращити асети

- Використовувати Microdata та Schema.org для SEO.
- Впровадити систему голосового управління.
- Надати можливість модифікувати гарячі клавіші.
- Змінити обробник платежів (Google Pay або Monobank).

ВИСНОВОК

Мною було ретельно вивчено тонкощі проектування, реалізації та розгортання веб-додатку на хмарній платформі. Завдяки всебічному аналізу, розробці та тестуванню було створено масштабований, доступний та продуктивний додаток, який використовує стек MERN, Next.js та хмарну платформу Vercel.

Основні висновки

Ефективність технологічного стеку

Обраний стек технологій, що включає MongoDB, Express.js, React, Node.js та Next.js, виявився високоефективним для створення сучасних веб-додатків, які вимагають масштабованості, супроводу та бездоганного користувацького досвіду.

Переваги хмарної платформи

Vercel, як хмарна платформа, продемонструвала свою цінність, надаючи зручне для розробників середовище з готовою підтримкою Next.js, полегшуючи безперервну інтеграцію та розгортання, а також пропонуючи можливості глобального розповсюдження.

Орієнтований на користувача дизайн

Дизайн та функціональність додатку були розроблені з урахуванням потреб та очікувань користувачів, що гарантує відповідність кінцевого продукту їхнім вимогам.

Бачення майбутнього

Майбутнє багаторівневих веб-додатків на хмарних платформах є яскравим, з величезним потенціалом для зростання та інновацій. З розвитком технологій з'являтимуться нові можливості для підвищення ефективності, покращення користувацького досвіду та інтеграції з новими технологіями, такими як штучний інтелект та Інтернет речей (IoT).

Отже, це дослідження не лише розглядає сучасний ландшафт розробки веб-додатків, але й закладає основу для майбутніх досягнень. Надані в ньому рекомендації

спрямовані на подальший розвиток і успіх доступних та багаторівневих веб-додатків в епоху хмарних обчислень.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. System Design: Multi-Tier Architecture. Ayokoding. URL: https://medium.com/ayokoding/system-design-multi-tier-architecture-3213ea3e206a (date of access: 23.05.2024).
2. IaaS, PaaS, SaaS. IBM. URL: https://www.ibm.com/topics/iaas-paas-saas (date of access: 23.05.2024).
3. Web Performance Optimization. LoadNinja. URL: https://loadninja.com/articles/web-performance-optimization/ (date of access: 23.05.2024).
4. What is a CDN? AWS. URL: https://aws.amazon.com/what-is/cdn/ (date of access: 23.05.2024).
5. React Native. React Native. URL: https://reactnative.dev/ (date of access: 23.05.2024).
6. MongoDB Atlas Tutorial. MongoDB. URL: https://www.mongodb.com/resources/products/platform/mongodb-atlas-tutorial (date of access: 23.05.2024).
7. Most Used Languages. Statista. URL: https://www.statista.com/statistics/793628/worldwide-developer-survey-most-used-languages/ (date of access: 23.05.2024).
8. MongoDB Atlas: Technical Overview & Benefits. Medium. URL: https://medium.com/@nparsons08/mongodb-atlas-technical-overview-benefits-9e4cff27a75e (date of access: 23.05.2024).
9. The Pros and Cons of Node.js in Web Development. GeeksforGeeks. URL: https://www.geeksforgeeks.org/the-pros-and-cons-of-node-js-in-web-development/ (date of access: 23.05.2024).
10. Introduction to Express/Node.js. MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction (date of access: 23.05.2024).

11. Most Used Frameworks Web. Statista. URL: https://www.statista.com/statistics/1124699/worldwide-developer-survey-most-used-frameworks-web/ (date of access: 23.05.2024).
12. Virtual DOM in React. LogRocket Blog. URL: https://blog.logrocket.com/virtual-dom-react/ (date of access: 23.05.2024).
13. Introducing JSX. React (Legacy). URL: https://uk.legacy.reactjs.org/docs/introducing-jsx.html (date of access: 23.05.2024).
14. Sass Documentation. Sass. URL: https://sass-lang.com/documentation/ (date of access: 23.05.2024).
15. Getting Started with NextAuth.js. NextAuth.js. URL: https://next-auth.js.org/getting-started/introduction (date of access: 23.05.2024).
16. Client-Server Overview. MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Client-Server_overview (date of access: 23.05.2024).
17. Core Web Vitals. Google Developers. URL: https://developers.google.com/search/docs/appearance/core-web-vitals (date of access: 23.05.2024).
18. Online Shopping Behavior. Statista. URL: https://www.statista.com/topics/2477/online-shopping-behavior/#topicOverview (date of access: 23.05.2024).
19. Eurostat News. European Commission. URL: https://ec.europa.eu/eurostat/web/products-eurostat-news/-/ddn-20211209-2 (date of access: 23.05.2024).
20. How to Use Analyzer in LoadRunner 12.0. Guru99. URL: https://www.guru99.com/how-to-use-analyzer-in-loadrunner-12-0.html (date of access: 23.05.2024).
21. Learn Accessibility. Netlify. URL: https://learn-a11y.netlify.app/ (date of access: 23.05.2024).

22. Introduction to Web Accessibility. W3C. URL: https://www.w3.org/WAI/fundamentals/accessibility-intro/#what (date of access: 23.05.2024).
23. Types of Cloud Computing. Google Cloud. URL: https://cloud.google.com/discover/types-of-cloud-computing (date of access: 23.05.2024).
24. Data Briefs - Number 20. CDC. URL: https://www.cdc.gov/nchs/data/databriefs/db20.pdf (date of access: 23.05.2024).
25. What is accessibility? - Learn web development | MDN. *MDN Web Docs*. URL: https://developer.mozilla.org/en-US/docs/Learn/Accessibility/What_is_accessibility (date of access: 23.05.2024).
26. (WAI) W. W. A. I. Patterns. *Web Accessibility Initiative (WAI)*. URL: https://www.w3.org/WAI/ARIA/apg/patterns/ (date of access: 23.05.2024).
27. NoCoffee – Vision Simulator for Chrome. *Access Garage*. URL: https://accessgarage.wordpress.com/2013/02/09/458/ (date of access: 23.05.2024).
28. What is Software Composition Analysis (SCA)? | Synopsys. *Synopsys | EDA Tools, Semiconductor IP and Application Security Solutions*. URL: https://www.synopsys.com/glossary/what-is-software-composition-analysis.html (date of access: 23.05.2024).
29. Bootstrap Vs Tailwind Vs MUI Frameworks: Key Differences | Talentica.com. *Talentica.com*. URL: https://www.talentica.com/blogs/bootstrap-vs-tailwind-vs-mui-frameworks-key-differences/ (date of access: 23.05.2024).
30. Next.js by Vercel - The React Framework. *Next.js by Vercel - The React Framework*. URL: https://nextjs.org/ (date of access: 23.05.2024).
31. Topic: Online shopping behavior in the United States. *Statista*. URL: https://www.statista.com/topics/2477/online-shopping-behavior/ (date of access: 23.05.2024).

32. Disability. *World Health Organization (WHO)*. URL: http://www.who.int/en/news-room/fact-sheets/detail/disability-and-health (date of access: 23.05.2024).

33. Evaluating Vercel for Web Development: Pros and Cons. *Whoooop*. URL: https://whoooop.co.uk/post/evaluating-vercel-for-web-development-pros-and-cons/ (date of access: 23.05.2024).

34. What Is Managed Cloud? | IBM. *IBM in Deutschland, Österreich und der Schweiz*. URL: https://www.ibm.com/topics/managed-cloud (date of access: 23.05.2024).

35. Festus O. Configuring Github Actions in a multi-directory repository structure. *Medium*. URL: https://medium.com/@owumifestus/configuring-github-actions-in-a-multi-directory-repository-structure-c4d2b04e6312 (date of access: 23.05.2024).

36. react-virtualized. *npm*. URL: https://www.npmjs.com/package/react-virtualized (date of access: 23.05.2024).

37. Node.js. Simple server side cache for Express.js with Node.js. *Medium*. URL: https://medium.com/the-node-js-collection/simple-server-side-cache-for-express-js-with-node-js-45ff296ca0f0 (date of access: 23.05.2024).

38. WebAIM: Contrast Checker. *WebAIM: Web Accessibility In Mind*. URL: https://webaim.org/resources/contrastchecker/?fcolor=fff&bcolor=000 (date of access: 23.05.2024).

39. Understanding Success Criterion 1.4.3: Contrast (Minimum) | WAI | W3C. *W3C*. URL: http://www.w3.org/WAI/WCAG21/Understanding/contrast-minimum.html (date of access: 23.05.2024).

40. (WAI) W. W. A. I. Listbox Pattern. *Web Accessibility Initiative (WAI)*. URL: https://www.w3.org/WAI/ARIA/apg/patterns/listbox/ (date of access: 23.05.2024).

41. Puppeteer | Puppeteer. *Puppeteer | Puppeteer*. URL: https://pptr.dev/ (date of access: 23.05.2024).

42. Indexes - MongoDB Manual v7.0. *MongoDB: The Developer Data Platform | MongoDB*. URL: https://www.mongodb.com/docs/manual/indexes/ (date of access: 23.05.2024).

43. Understanding TF-IDF (Term Frequency-Inverse Document Frequency) - GeeksforGeeks. *GeeksforGeeks*. URL: https://www.geeksforgeeks.org/understanding-tf-idf-term-frequency-inverse-document-frequency/ (date of access: 23.05.2024).

44. Axe devtools - web accessibility testing. *Chrome Web Store*. URL: https://chromewebstore.google.com/detail/axe-devtools-web-accessib/lhdoppojpmngadmnindnejefpokejbdd?hl=en-US (date of access: 23.05.2024).

АНОТАЦІЯ

Тема: Побудова багаторівневого веб-застосування на хмарній платформі

Студент: Бучок Богдан Сергійович

Факультет інформатики

Науковий керівник: Черкасов Дмитро Іванович

Захищена «___» _____ 200_ р.

Короткий зміст роботи

Це дослідження присвячене створенню доступного, продуктивного та багаторівневого веб-додатку з використанням стеку MERN, Next.js та інших сучасних веб-технологій на хмарній платформі Vercel

Актуальність цього дослідження полягає в дослідженні ефективної розробки доступних веб-застосунків за допомогою сучасного інструментарію та хмарної інфраструктури, яка забезпечує масштабованість, продуктивність та економічну ефективність. Метою цього дослідження є надання вичерпного посібника для створення надійної веб-програми, яка відповідає галузевим стандартам і найкращим практикам.

Робота, яка складається з п’яти основних розділів.У першому надається визначенням основним поняттям. У другому розділі виконується аналіз вимог до веб-застосування. У третьому розділі обираються технології, які будуть використані для розробки. У четвертому розділі описується поглиблений алгоритм для реалізації. У п’ятому розділі проводиться тестування розробленого веб-застосування та описуються рекомендації щодо подальшого вдосконалення. У висновку подається підсумок

проведеного дослідження.

Основні результати аналізу демонструють ефективність вибраних технологій, дотримання W3C-стандартам та рекомендаціям щодо доступності у розробці, слідуванням практикам у створенні надійного, масштабованого та безпечного веб-додатку. Дослідження завершується рекомендаціями щодо майбутнього розвитку, наголошуючи на необхідності постійної адаптації та вдосконалення у динамічній сфері хмарних веб-додатків.

Ключові слова: хмарна платформа, багаторівневе веб-застосування, веб-розробка, хмарні технології, моделювання, архітектура веб-застосування, програмування, хмарні сервіси, API, JavaScript, HTML, CSS, Стек MERN, Next.js, Vercel, NextAuth.js, сучасні веб-технології, Accessibility, WCAG, TF-IDF, оптимізація, веб-продуктивність