МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Спеціальність: 123 «Комп'ютерна інженерія»

Освітня програма: «Комп'ютерна інженерія»

Група: 2БКС-28

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

здобувача освіти денної форми навчання

БКС.28.08.000.КРБ

ГРАБОВОГО ОЛЕКСАНДРА ВАЛЕНТИНОВИЧА

м. Одеса
2024 р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Спеціальність: **123 «Комп'ютерна інженерія»**

Освітньо-професійна програма: **«Комп'ютерна інженерія»**

Група: **2БКС-28**

ПОЯСНЮВАЛЬНА ЗАПИСКА

До кваліфікаційної роботи бакалавра на тему: **«Створення Інтернет-ресурсу для комерційного підприємства з елементами 3D-дизайну»**

Проектний матеріал складається з пояснювальної записки на 85 сторінках та графічного (презентаційного) матеріалу на 13 аркушах (слайдах)

Виконавець _____ (Грабовий О.В.)

Керівник проекту _____ (Іванова Л.В.)

Консультанти:

з розділу охорони праці та техніки безпеки _____ (Чорновол Н.І.)

з нормоконтролю _____ (Петрашова В.І.)

старший консультант _____ (Кривченко Ю.В.)

До захисту допущений

Завідувач кафедри _____ (Іванова Л.В.)

Завідувач відділення _____ (Скорнякова О.В.)

Захист «24» 06 2024 р. Протокол ЕК № 7

Оцінка ЕК 5 (відмінно) 95

Секретар ЕК _____

АНОТАЦІЯ

У роботі представлено результати розробки Інтернет-ресурсу для комерційного підприємства з елементами 3D-дизайну.

У цій роботі описаний принцип розробки сайту за допомогою бібліотек React, React Router та Redux.

У аналітичній частині приведене технічне завдання, огляд реалізації популярних комерційних веб-сайтів України та Європи, обґрунтування обраних технологій розробки. В технологічній частині описано налаштування збірника проектів Webpack, та реалізація компонентів, сторінок та всього сайту.

Ключові слова: React, React Router, Redux, React Redux, Redux Toolkit, Webpack, NPM, Git, Frontend.

ANNOTATION

The work represents the results of the development of an Internet resource for a commercial enterprise with elements of 3D design.

This work describes the principle of site development using the React, React Router, and Redux libraries.

The analytical part contains a technical task, an overview of the implementation of popular commercial websites of Ukraine and Europe, justification of the selected development technologies. The technological part describes the configuration of the Webpack project collection, and the implementation of components, pages and the entire site.

Keywords: React, React Router, Redux, React Redux, Redux Toolkit, Webpack, NPM, Git, Frontend.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Відділення *Комп'ютерних систем* Кафедра *Комп'ютерної інженерії*

Спеціальність *123 «Комп'ютерна інженерія»*

Освітньо-професійна програма *«Комп'ютерна інженерія»*

ЗАТВЕРДЖУЮ:

Заст. дир. з НВР *Беркань І.В.*

"16" 01 2024 р.

ЗАВДАННЯ

на кваліфікаційну роботу бакалавра

здобувачеві освіти *Грабового Олександра*

(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи *Створення Інтернет-ресурсу для комерційного підприємства з елементами 3D-дизайну*

затверджена наказом по коледжу від "04" 11 2023 р. № 244-А2-ОД

2. Термін здачі студентом кваліфікаційної роботи *15.06.2024*

3. Вихідні дані до роботи *Вимоги до інформаційних інтернет ресурсів*
Функціональні вимоги до програмного забезпечення
Нефункціональні вимоги до програмного забезпечення
Вимоги до дизайну та інтерфейсу користувача

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)

1. Аналітичний огляд існуючих рішень. 2. Обґрунтування вибору технологій для створення інформаційних інтернет-ресурсів. 3. Архітектура сайту. 4. Розробка елементів дизайну та інтерфейсу користувача. 5. Створення макету та прототипу сайту. 6. Тестування сайту. 7.Охорона праці. 8. Висновки. 9. Список використаних джерел інформації.

5. Перелік графічного матеріалу (слайдів мультимедійної презентації)

1. *Порівняльний аналіз існуючих рішень.*
2. *Технології інформаційних інтернет-ресурсів.*
3. *Архітектура сайту.*
4. *Макет сайту*
5. *Скріншоти /відео роботи сайту*

6. Консультанти по кваліфікаційній роботі, із зазначенням розділів, що їх стосуються

Розділ	Консультант	ПІДПИС	
		Завдання видав	Завдання прийняв
Основний розділ	*Іванова Л.В.*		
Розділ охорони праці	*Чорновол Н.І.*		
Нормоконтроль	*Петрашова В.І.*		
Старший консультант	*Кривченко Ю.В.*		

7. Дата видачі завдання _____

Керівник роботи *Іванова Л.В.* _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

Пор. №	Назва етапів кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1.	*Аналіз технічного завдання та пошук джерел інформації*	*4.05.2024*	виконав
2.	*Формування функціональних вимог до ПЗ*	*8.05.2024*	виконав
3.	*Визначення технічного завдання на розробку*	*10.05.2024*	виконав
4.	*Вибір технологій для реалізації сайту*	*12.05.2024*	виконав
5.	*Розробка архітектури сайту*	*15.05.2024*	виконав
6.	*Розробка алгоритмів роботи сайту*	*18.05.204*	виконав
7.	*Розробка елементів дизайну*	*22.05.2024*	виконав
8.	*Розробка елементів інтерфейсу*	*24.05.2024*	виконав
9.	*Створення макету сайту*	*26.05.2024*	виконав
10.	*Створення прототипу*	*28.05.2024*	виконав
11.	*Тестування ПЗ*	*31.05.2024*	виконав
12.	*Тестування виконання вимог користувача*	*2.06.2024*	виконав
13.	*Перевірка роботи на плагіат*	*6.06.2024*	виконав
14.	*Підготовка до малого захисту*	*10.06.2024*	виконав
15.	*Розробка питань з охорони праці*	*12.062024*	виконав
16.	*Підготовка роботи до захисту та тестування ПЗ*	*15.06.2024*	виконав
17.	*Оформлення слайдів мультимедійної презентації*	*18.06.2024*	виконав

Здобувач освіти _____
(підпис)

Керівник роботи _____
(підпис)

Формат	Зона	Поз.	Позначення	Найменування	Кіл.	Прим.
				Документація		
			БКС 28.08.000.00 КРБ	Кваліфікаційна робота	1	
А4			БКС 28.08.000.00 КРБ ПЗ	Пояснювальна записка	1	
			БКС 28.08.000.00 КРБ МП	Мультимедійна		
				презентація		

Ізм.	Арк.	№докум.	Підпис	Дата
Розроб.		Грабовий О.В.		13.06
Перевір.		Іванова Л.В.		13.06
Н. конт		Петрашова В.І.		13.06
Затв.		Іванова Л.В		13.06

БКС 28.08.000.00 КРБ

Створення Інтернет-ресурсу для комерційного підприємства з елементами 3D-дизайну

Літ.	Аркуш	Аркушів
Н Д Г	5	85

ВСП "ОТФК ОНТУ" гр.2БКС-28

ЗМІСТ

ВСТУП

У сучасному світі важко уявити великий магазин або мережу магазинів, які не мають власного веб-сайту. І це не дивно, адже наявність веб-версії магазину значно розширює клієнтську базу та впізнаваємість бренду, так як люди з будь-якої частини країни, або навість світу, можуть, як мінімум, отримати інформацію стосовно послуг та товарів інтернет-магазину, а як максимум, скористатися послугами, що позитивно впливає на дохід магазину. За інформацією Асоціації рітейлерів України (RAU), ринок електронної комерції у 2023 році склав $1.7 млрд., а в 2024 році очікується зріст обсягів до $2 млрд. За даними міжнародної компанії Deloittte, активність онлайн-шопінгу українців у 2023 році зросла на 63% порівняно з попереднім роком та у 1,8 раза — порівняно з 2021 роком. Також, веб-сайт магазину значно оптимізує бізнес-процеси, адже вимагає значно меншу кількість персоналу для його обслуговування, і як наслідок, меншу вартість утримування. Наявність сайту покращує управління багатьма процесами, такими як: обробка замовлення, інвентаризація тощо. Також, сайти є важливими аналітичними інструментами, допомагаючи отримувати інформацію про поведінку клієнтів, їх зацікавлення до певних послуг та товарів.

У цій роботі було описано створення веб-сайту для комерційного підприємства (інтернет-магазин), що має покращити клієнтський досвід з магазином, розширити охоплення клієнтів та впізнаваємість бренду

1. Технологічний розділ

1.1 Аналітичний огляд існуючих рішень

1.1.1 Аналіз технічного завдання

Технічне завдання дипломного проекту полягає в розробці Інтернет-ресурсу для комерційного веб-сайту. Власноруч розробляючи дизайн, функціональні можливості та клієнтську частину веб-сайту, мені необхідно зробити аналіз існуючих рішень на ринку електронної комерції в українському та європейському регіоні.

Звісно, що підсумкові рішення мають відповідати загальним вимогам до інтернет ресурсів, а саме:

- **Зручний інтерфейс користувача**: Веб-сайт повинен мати зрозумілий та легкий у використанні інтерфейс, що дозволяє користувачам швидко знаходити потрібну інформацію.
- **Адаптивний дизайн**: Веб-сайт повинен бути адаптованим під різні типи пристроїв (комп'ютери, планшети, смартфони) та розміри екранів, щоб забезпечити зручний доступ для користувачів у будь-який час та в будь-якому місці.
- **Швидке завантаження**: Веб-сайт повинен завантажуватися швидко, оскільки повільна швидкість завантаження може призвести до втрати інтересу користувачів і зменшення конверсії.
- **Безпека даних**: Забезпечення конфіденційності та безпеки особистих даних користувачів є важливою складовою веб-сайтів, особливо тих, які зберігають особисту інформацію від користувачів.
- **Якісний контент**: Інформація на веб-сайті повинна бути актуальною, коректною та інформативною. Контент має бути легко зрозумілим та доступним для цільової аудиторії.

- **Навігація і структура**: Веб-сайт повинен мати зрозумілу та логічну структуру навігації, яка дозволяє користувачам швидко переходити між різними сторінками та знаходити потрібну інформацію.
- **Можливість зворотного зв'язку** : Наявність форми зворотного зв'язку або контактних даних дозволяє користувачам звертатися до адміністраторів веб-сайту для отримання допомоги чи вирішення питань.
- **SEO-оптимізація**: Веб-сайт повинен бути оптимізований для пошукових систем з метою підвищення його видимості в пошукових результатах та привертання більшої кількості відвідувачів.

1.1.2 Аналіз існуючих рішень

У цьому підрозділі описується аналіз функціональності веб-сайтів українських та міжнародних інтернет-магазинів з метою вибору оптимального рішення для розробки проекту.

Серед українських інтернет-магазинів будуть розглянуті: Rozetka, Епіцентр. Вибір здійснювався серед найпопулярніших українських інтернет-магазинів за версією премії Ukrainian Business Award, які надають послуги, найбільш схожі до послуг сайту дипломного проекту.

Серед популярних магазинів європейського регіону будуть розглянуті: Amazon та eBay, так как як вони є найпопулярнішими за версією сайту Statista.

Rozetka:

Загальна функціональність: категорії та підкатегорії товарів, можливість сортувати за ціною та іншими фільтрами, кабінет користувача, список бажань, список замовлень, кошик, пошук по сайту, галерея зображень товару, характеристики товару, опис товару, можливість залишити відгук, порівняння товарів, запропоновані товари, можливість зворотного зв’язку, розділи з інформацію про покупку, доставку та повернення, адаптивний дизайн.

Інтерфейс та навігація: Rozetka пропонує зручний та інтуїтивно зрозумілий інтерфейс. На головній сторінці розташовано основні категорії товарів, що дозволяє користувачам швидко знайти потрібний розділ. Навігація по сайту є зручною, з можливістю фільтрації товарів за різними параметрами.

Пошук: Пошукова система досить потужна і швидка. Результати пошуку відображаються швидко та точно, дозволяючи користувачам знаходити товари за ключовими словами або категоріями.

Оцінка Google Lighthouse:

Рисунок 1.1.1 - Оцінка Google Lighthouse сайту Rozetka.

За результатами оцінки Google Lighthouse, можна сказати, що сайт має гарну оптимізацію, враховуючи великий обсяг функціоналу, гарну SEO-оптимізацію, але недостатньо реалізовану підтримку Accessibility (доступність для людей з обмеженими можливостями, підтримку керування клавіатурою тощо)

Епіцентр:

Загальна функціональність: категорії та підкатегорії товарів, можливість сортувати за ціною та іншими фільтрами, кабінет користувача, список бажань, список замовлень, кошик, пошук по сайту, галерея зображень товару, характеристики товару, опис товару, можливість залишити відгук, порівняння товарів, запропоновані товари, можливість зворотного зв’язку, розділи з інформацію про покупку, доставку та повернення, адаптивний дизайн.

Інтерфейс та навігація: Інтерфейс Епіцентру я не можу назвати однозначно зручним. Головна сторінка виглядає лаконічно, містить категорії товарів та акційні пропозиції, новини. В каталозі, у товарів зазначено забагато додаткової

інформації, використовується маленький шрифт. На сторінці товару забагато рекомендованих товарів, акційних банерів, інформації, яку можна було розмістити у згорнутому вигляді, так як вона не для всіх є актуальною . Навігація є простою, користувачі можуть легко переходити між розділами та підкатегоріями товарів.

Пошук: Пошукова система Епіцентр дозволяє швидко та ефективно знаходити потрібні товари. Результати пошуку відображаються точно та швидко, з можливістю фільтрування за різними параметрами, такими як ціна, бренд, наявність тощо. Але при цьому, пошук не відображає окремо категорії та підкатегорії.

Оцінка Google Lighthouse:

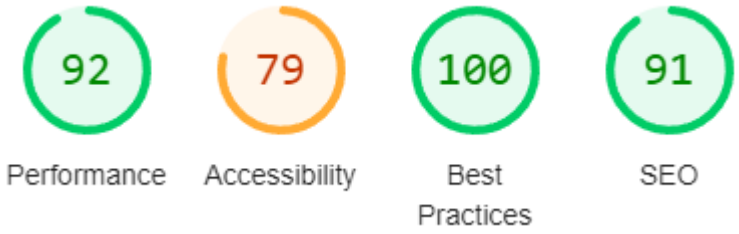


Рисунок 1.1.2 - Оцінка Google Lighthouse сайту Епіцентр.

За результатами оцінки Google Lighthouse, можна сказати, що сайт має дуже гарну оптимізацію, гарну SEO-оптимізацію, але непогано реалізовану підтримку Accessibility.

Amazon:

Загальна функціональність: категорії та підкатегорії товарів, можливість сортувати за ціною та іншими фільтрами, кабінет користувача, список бажань, список замовлень, кошик, пошук по сайту, галерея зображень товару, характеристики товару, опис товару, можливість залишити відгук, запропоновані товари, можливість зворотного зв’язку, розділи з інформацію про покупку, доставку та повернення, адаптивний дизайн, підтримка декількох мов.

Інтерфейс та навігація: Amazon відзначається своїм добре продуманим та зручним інтерфейсом. Головна сторінка сайту дуже гарно оформлена, містить широкий вибір категорій товарів, що дозволяє користувачам швидко знаходити потрібні товари. Каталог товарів не містить зайвої інформації, хоча в цілому його дизайн досить простий. Сторінка товару виглядає лаконічно та також не містить зайвої інформації, хоча в цілому, її дизайн виглядає дещо застарілим. Навігація по сайту є легкою та інтуїтивно зрозумілою.

Пошук: Пошукова система Amazon є дуже потужною та ефективною. Користувачам надається можливість швидко знаходити товари за ключовими словами, артикулом або категорією. Результати пошуку відображаються швидко та точно, з можливістю фільтрації за різними параметрами.

Оцінка Google Lighthouse:

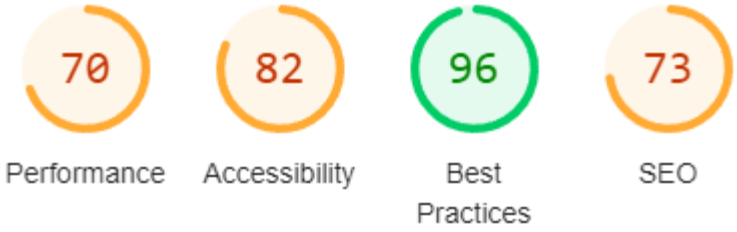


Рисунок 1.1.3 - Оцінка Google Lighthouse сайту Amazon.

За результатами оцінки Google Lighthouse, можна сказати, що сайт має задовільну оптимізацію, задовільну SEO-оптимізацію, але гарно реалізовану підтримку Accessibility.

eBay:

Загальна функціональність: категорії та підкатегорії товарів, можливість сортувати за ціною та іншими фільтрами, кабінет користувача, список бажань, список замовлень, кошик, пошук по сайту, галерея зображень товару, характеристики товару, опис товару, можливість залишити відгук, запропоновані товари, можливість зворотного зв’язку, розділи з інформацію про покупку,

доставку та повернення, адаптивний дизайн, підтримка декількох мов, можливості маркет-плейсу, можливість проведення аукціонів.

Інтерфейс та навігація: eBay відомий своїм простим та зрозумілим інтерфейсом. Головна сторінка містить розділи з різними категоріями товарів, а також акційні пропозиції. Навігація є легкою, і користувачі можуть швидко переходити між категоріями товарів та фільтрувати їх за різними параметрами.

Пошук: Пошукова система eBay є потужною та швидкою. Користувачі можуть шукати товари за ключовими словами, категоріями, артикулами тощо. Результати пошуку відображаються точно та з великою кількістю варіантів для вибору.

Оцінка Google Lighthouse:

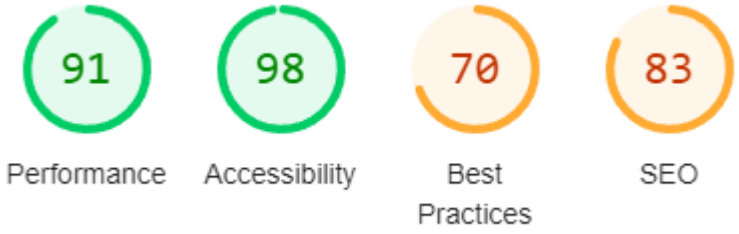


Рисунок 1.1.4 - Оцінка Google Lighthouse сайту eBay.

За результатами оцінки Google Lighthouse, можна сказати, що сайт має дуже гарну оптимізацію, гарну SEO-оптимізацію, а також повністю реалізовано підтримку Accessibility.

Висновок:

Проаналізувавши наведені вище сайти, я визначив перелік функціоналу, який є на кожному сайті, і який доцільно реалізувати у сайті дипломного проекту, це: категорії та підкатегорії товарів, можливість сортувати за ціною та іншими фільтрами, список бажань, кошик, пошук по сайту, галерея зображень товару, характеристики товару, опис товару, можливість залишити відгук, запропоновані товари, можливість зворотного зв’язку, розділи з інформацію про покупку, доставку та повернення, адаптивний дизайн.

1.1.3 Аналіз окремих елементів та сторінок

В цьому розділі описується аналіз окремих елементів та сторінок на предмет їх функціональності та дизайну.

В кожному сайті присутня головна сторінка, сторінка каталогу довільної категорії, сторінка довільного товару. На кожній сторінці присутній header (верхня навігаційна панель).

Header:

Рисунок 1.1.5 - Header сайту Rozetka.

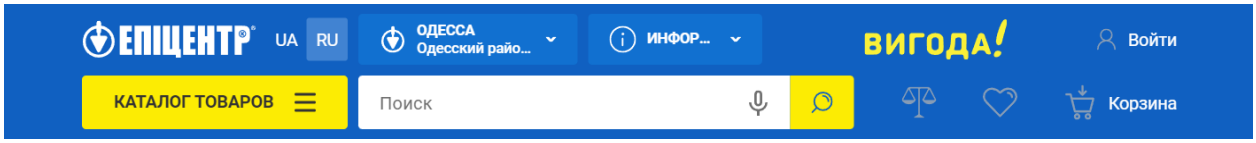


Рисунок 1.1.6 - Header сайту Епіцентр.

Проаналізувавши наведені вище приклади, можна дійти до висновку, що header повинен складатися с логотипу сайту, який одночасно є посиланням на головну сторінку, полем для пошуку по сайту, який відразу ж показує результати пошуку в скороченому вигляді, іконки-посилання на список бажань, кошик, порівняння товарів. Додатково, ці іконки можуть показувати кількість товарів у відповідній категорії. Якщо є додаткове меню (наприклад, з каталогом), то в header має бути кнопка, яка розгортає або показує це меню. Також, header має бути завжди зверху, незалежно від прокрутки сторінки.

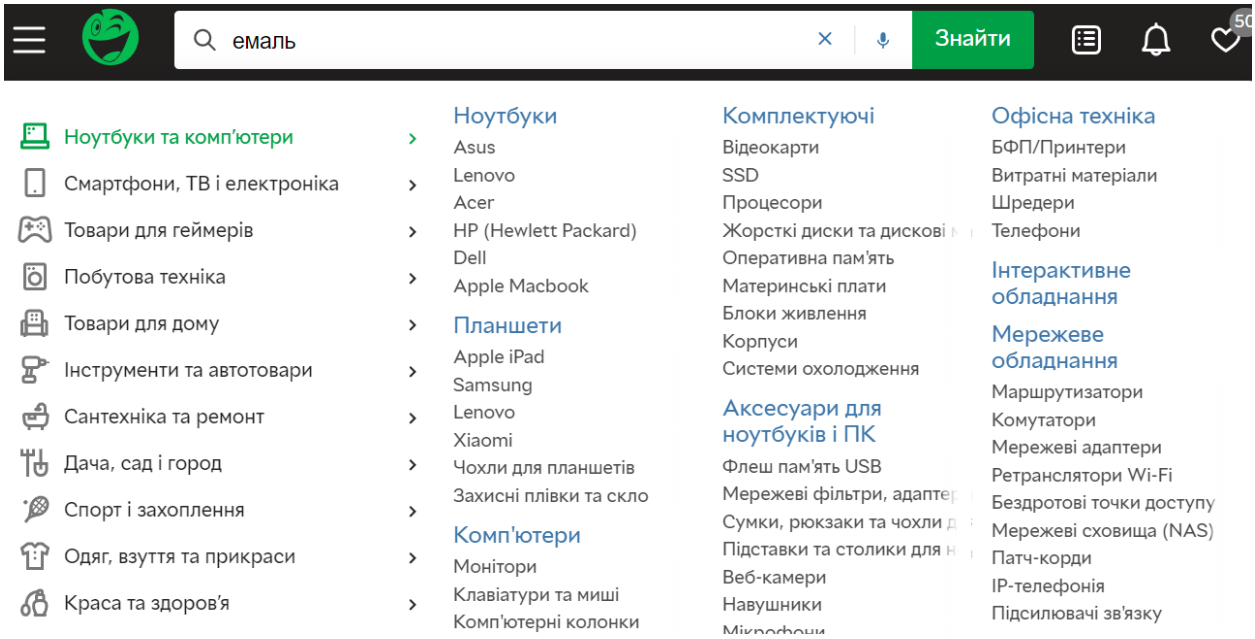


Рисунок 1.1.7 - Додаткове меню сайту Rozetka.

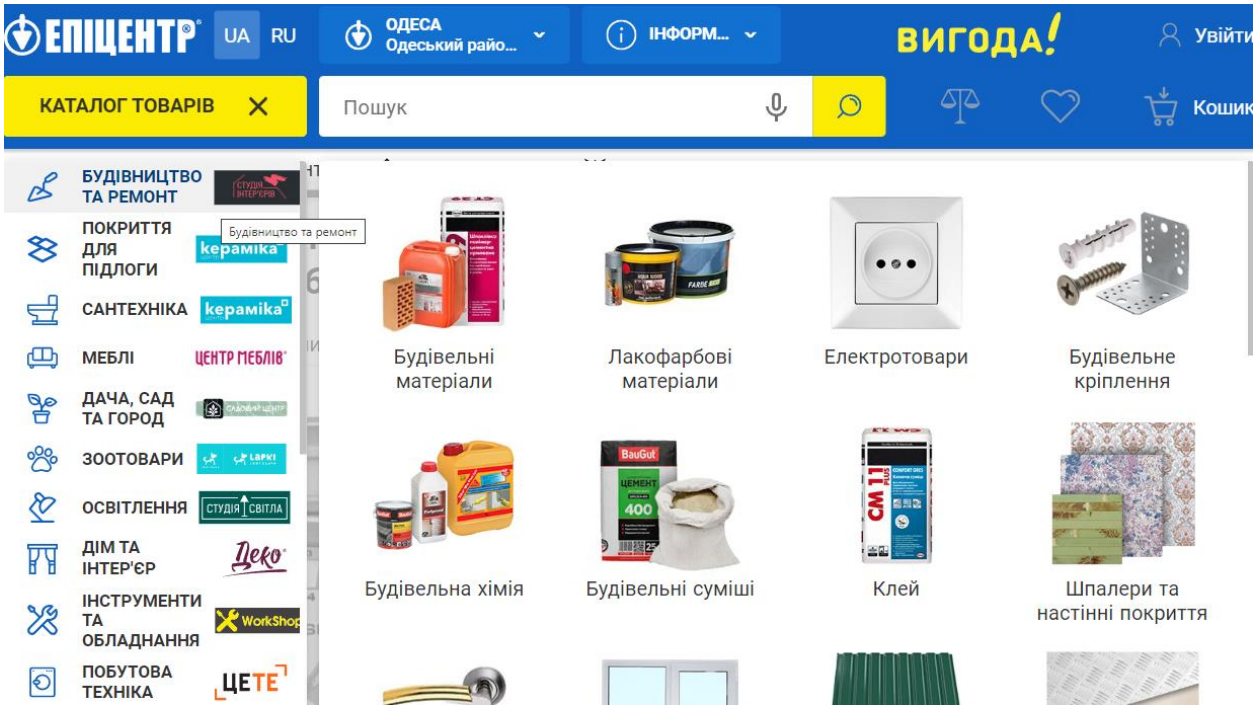


Рисунок 1.1.8 - Додаткове меню сайту Епіцентр.

В обох вище зазначених прикладах, додаткове меню показує категорії, де при натисканні або наведені на них, показуються їх підкатегорії. При цьому, якщо додаткове меню активно, то на задньому фоні з'являється темна підкладка, при натисканні на яку, меню приховується. Також, якщо додаткове меню активне, тоді прокрутка сторінки блокується.

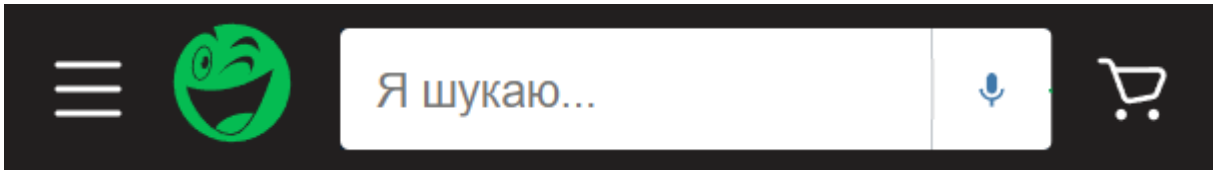


Рисунок 1.1.9 - Мобільний header сайту Rozetka.

Рисунок 1.1.10 - Мобільний header сайту Епіцентр.

Основними елементами header для мобільних пристроїв являються: кнопка для показу додаткового меню, логотип-посилання на основну сторінку, пошук по сайту та кошик. Елементи, які не вмістилися, були перенесені у додаткове меню.

Основна сторінка:

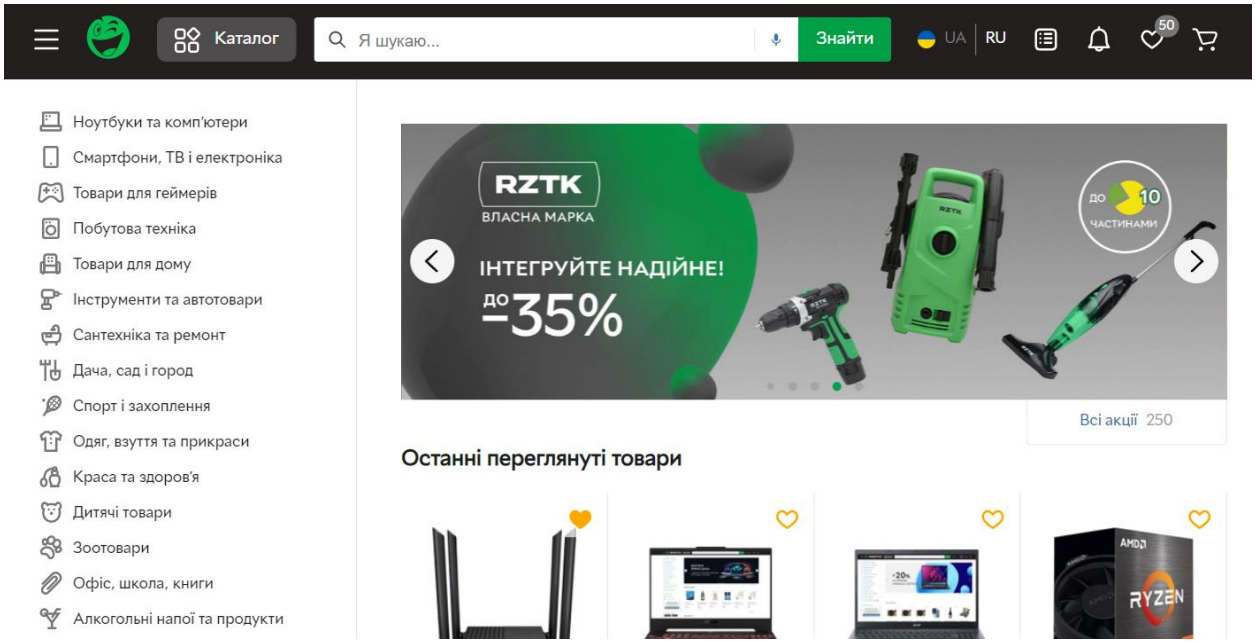


Рисунок 1.1.11 - Основна сторінка сайту Rozetka.

Рисунок 1.1.12 - Основна сторінка сайту Епіцентр.

Проаналізувавши наведені вище приклади, можна дійти до висновку, що основна сторінка буде складатися з банерів рекламних акцій та новин, розташованих зверху, часто виконаних у формі слайдеру. Поряд будуть розташовані категорії та підкатегорії, оформлені за допомогою іконок або зображень. Додатково будуть показуватися рекомендовані, або переглянуті товари.

Каталог:

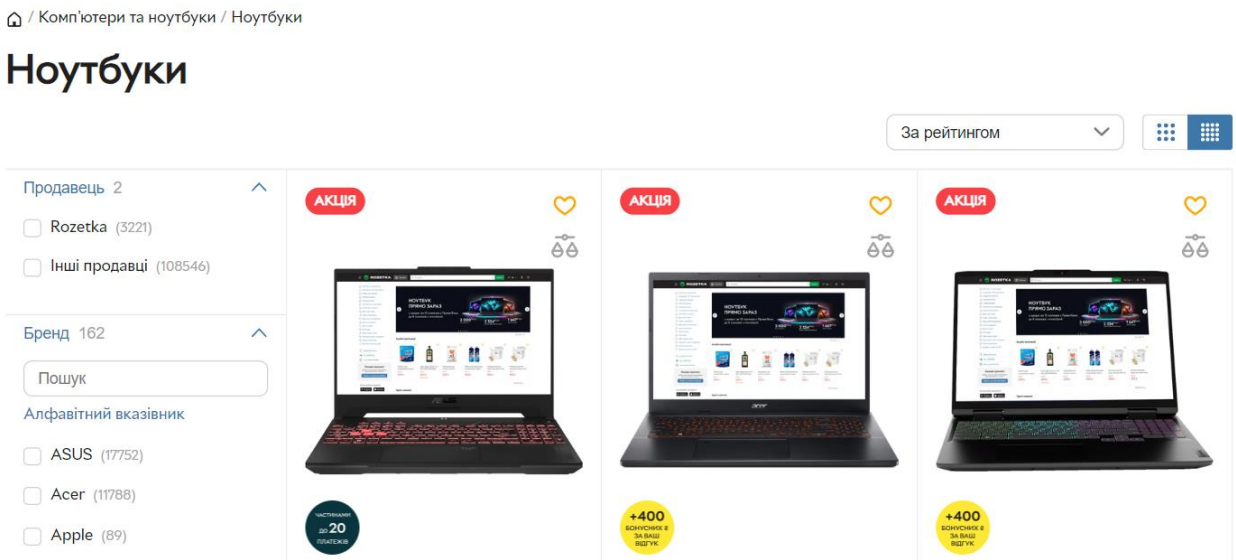


Рисунок 1.1.13 - Каталог сайту Rozetka.

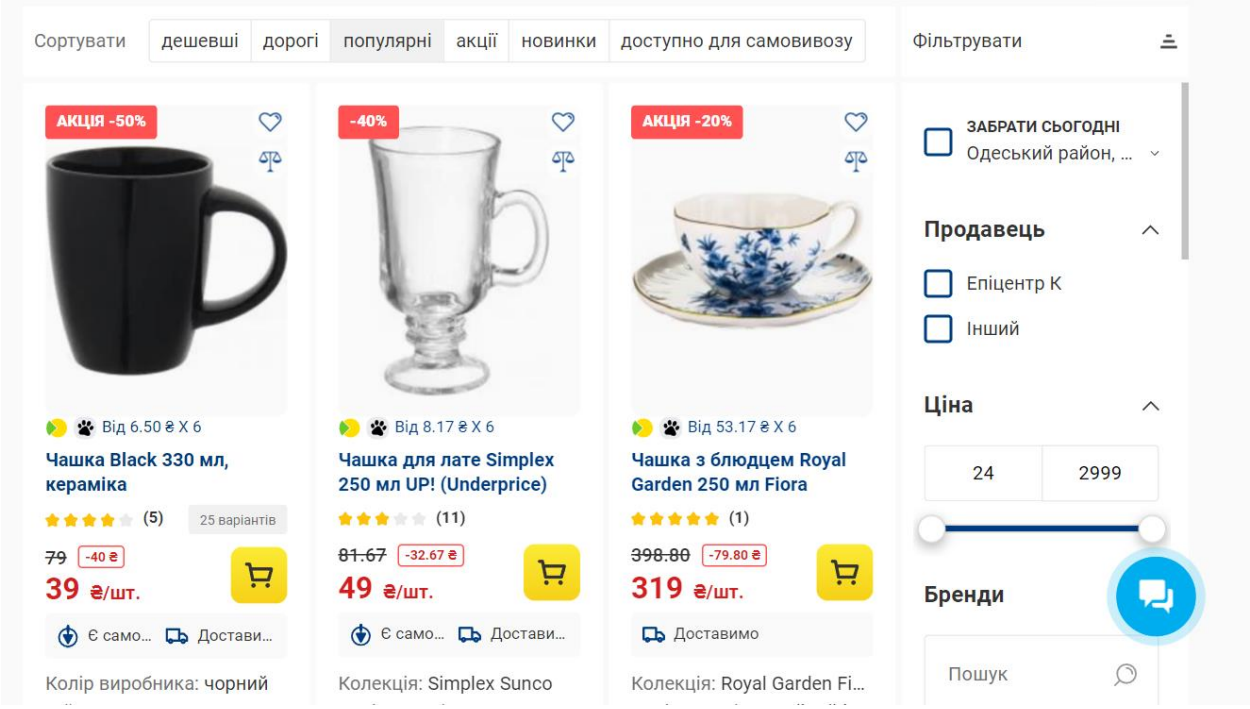


Рисунок 1.1.14 - Каталог сайту Епіцентр.

Проаналізувавши наведені вище приклади, можна дійти до висновку, що каталог буде складатися з панелі сортування товарів за ціною, назвою тощо., панелі фільтрів з лівою або правою сторони, де кожна категорія фільтрів може скриватися, а посередині мають відображатися товари. Картка товарів буде містити зображення, назву, ціну, кнопку додавання в кошик, список бажань, порівняння. Додатково можуть бути приведені короткі характеристики, стару ціну, розмір знижки та інша інформація. Додатково, на сторінці будуть реалізовані налаштування перегляду товарів (кількість товарів на сторінці, кількість товарів у рядку тощо), а також breadcrumbs - ланцюг навігації до поточної сторінки (наприклад, головна - категорія - підкатегорія)

Сторінка товару:

Рисунок 1.1.15 - Сторінка товару сайту Rozetka.

Рисунок 1.1.16 - Сторінка товару сайту Епіцентр.

Проаналізувавши наведені вище приклади, можна дійти до висновку, що сторінка товару буду містити: галерею зображень товару, з можливістю збільшення зображення для детального огляду, короткий опис основних характеристик, повні характеристики, опис товару, коментарі, статус наявності, кнопки для додання в кошик, список бажань, порівняння, ціна, стара ціна та знижка (за наявності),

форма для того, щоб залишити відгук або запитання, рекомендовані товари. Переключення між коментарями, характеристиками та іншими розділами буде реалізовано за допомогою кнопок-вкладок. Додатково, буде реалізована підтримка 3-Д моделі товару (за її наявності), інформація про можливості придбання, активні акції, які стосуються цього товару.

1.1.4 Висновок

Отже, проаналізувавши загальну функціональність наведених вище сайтів, а також функціональність та складові окремих сторінок та елементів, можна скласти перелік функціональних та нефункціональних вимог.

Функціональні:

Каталог товарів, фільтрація товарів, сортування товарів, сторінка товарів, галерея зображень товару, головна сторінка з поточними акціями та новинами, сторінки з контактною інформацією, інформацією про доставку та повернення, список бажань, кошик, можливість порівнювати товари, адаптивність.

Нефункціональні:

Технічна продуктивність, безпека користувача, доступність для людей з обмеженими можливостями, підтримка клавіатури, масштабованість, зручність використання.

1.2 Обґрунтування вибору технологій для створення інформаційних інтернет-ресурсів

1.2.1 Визначення етапів розробки відповідно функціональності сайту

Розробка подібного сайту вимагає залучення розробників різних напрямків роботи (розробник клієнтської частини, серверної частини, дизайнер, SEO-спеціаліст тощо). Нижче наведено функціонал, за який відповідає розробник архітектури сайту, клієнтської частини та дизайну:

Архітектура сайту та його загальна функціональність - етап, на якому визначається, яку саме мету має виконувати сайт. Розуміючи необхідну функціональність сайту, можна визначитись з його технічною реалізацією. Це включає і технології для розробки (збірник проекту, основна мова програмування тощо), і технічні рішення (фільтрація на стороні клієнта чи сервера, необхідні ендпоїнти (HTTP-адреси) для завантаження інформації, загальний макет для всіх сторінок та інше)

Дизайн сайту - розуміючи необхідний функціонал, можна почати розробку дизайну. Це включає загальні рішення, такі як: вибір палітри кольорів, яка відображає брендовий стиль організації та відповідає настрою сайту; шрифти, які легко читати та відображають бренд; розробку логічної структури щоб користувачі могли легко знаходити необхідну інформацію; візуальну ідентичність, яка буде відображати бренд, використовуючи логотипи, графічні елементи тощо. Також, необхідно зробити варіації дизайну для різних розмірів екрану, а також для мобільних пристроїв. Це не лише розміри елементів, це також зручність користування за допомогою сенсорних екранів.

Розробка клієнтської частини - цей етап є одним з ключових етапів у процесі створення веб-ресурсу. Цей етап включає в себе розробку та імплементацію інтерфейсу користувача, який відображається в браузері, використовуючи технології та рішення, прийняті на етапі створення архітектури сайту.

Технічна оптимізація - етап технічної оптимізації є важливою частиною процесу розробки сайту, оскільки від цього залежить швидкість завантаження, продуктивність та ефективність веб-ресурсу. Він включає в себе як автоматичні (за умови використання збірника проектів), так і мануальні варіанти оптимізації.

Базова SEO-оптимізація - хоча SEO-оптимізація є дуже комплексним процесом, для якого необхідно залучати відповідного спеціаліста, є декілька аспектів, які може і повинен виконати розробник клієнтської частини. Це включає в себе: логічну та зрозумілу структуру URL-адрес для сторінок сайту, що

відображає їхню ієрархію та зміст; використання ключових слів у URL-адресах та уникнення надмірної довжини та складності тощо. Підтримка доступності (accessibility) також позитивно впливає на SEO-оптимізацію.

Підтримка доступності (Accessibility / a11y) - цей етап включає в себе усі заходи та методи, спрямовані на забезпечення того, щоб веб-ресурс був доступним для всіх користувачів, включаючи людей з різними видами обмежень та потребами. Нижче подано огляд ключових аспектів цього етапу: дотримання міжнародних стандартів доступності, таких як WCAG (Web Content Accessibility Guidelines), щоб забезпечити, що сайт відповідає основним вимогам для доступності; врахування важливих аспектів, таких як читабельність тексту, можливість навігації за допомогою клавіатури, візуальний контраст, підтримка екранних читачів тощо; забезпечення альтернативними текстами для зображень та іншого мультимедійного контенту, щоб люди з вадами зору або іншими обмеженнями могли отримати необхідну інформацію; використання ARIA-атрибутів для додавання додаткової семантики, або забезпечення семантикою несемантичних елементів (наприклад, використання власноруч написаного селекту або кнопки замість HTML-елементів <button> та <select>); підтримку подій клавіатури для несемантичних елементів, а також додаткова підтримка для комплексних елементів, навігації по сайту тощо.

Тестування - цей етап є критично важливою частиною розробки веб сайту, оскільки він дозволяє переконатися, що веб-ресурс працює належним чином та відповідає всім вимогам та очікуванням користувачів.

1.2.2 Вибір основних технологій розробки

1.2.2.1 Основна бібліотека для написання JS та HTML-логіки

Виходячи з технічного завдання та архітектури проекту, веб-сайт буде складатися з компонентів, які мають взаємодіяти один з одним, легко додаватись до інших сторінок та компонентів, робити мережеві запити, зберігати результати таким чином, щоб вони були доступні іншим елементам та сторінкам, легко

змінюватись в залежності від збережених даних, має транспілюватися в основні технології, які підтримує браузер - JavaScript, CSS, HTML. З вище перерахованого можна виділити основні критерії для потенційної бібліотеки або фреймворку: компонентна структура, реактивність, підтримка стану (state), підтримка асинхронних подій. Зрештою я обрав бібліотеку React.

Бібліотека React - це інструмент для розробки інтерактивних та динамічних веб-інтерфейсів. Вона існує для допомоги розробникам створювати складні, ефективні та масштабовані веб-додатки з високоякісним користувацьким інтерфейсом. Ось кілька ключових переваг React: компонентна структура, віртуальний DOM (оптимізує процес оновлення елементів на сторінці), JSX-синтаксис, одиночний потік даних, розширюваність та масштабованість.

Недоліки:

- **Можливі проблеми з SEO**: Веб-сторінки, побудовані з використанням React, можуть мати проблеми з індексацією пошуковими системами через те, що вони генеруються динамічно за допомогою JavaScript. Ця проблема може бути вирішена за допомогою технології SSR (Server Side Rendering), суть якої в тому, що сервер попередньо генерує HTML-сторінку для клієнта, тому пошукові роботи мають змогу проаналізувати таку сторінку.

- **Первинне завантаження**: Хоча React дозволяє створювати швидкі та ефективні веб-додатки, перше завантаження сторінки може бути повільним через необхідність завантаження всього JavaScript коду для відображення сторінки. Цю проблему можна вирішити, використовуючи роутинг (маршрутизацію) сторінок, а також динамічне завантаження компонентів за запитом.

Узагальнюючи, бібліотека React існує для спрощення та прискорення розробки веб-інтерфейсів, забезпечуючи розробникам потужний інструментарій та ефективні методи для створення сучасних веб-додатків.

Аналоги React:

Основним аналогом React є фреймворк Vue.js.

Vue.js - є прогресивним JavaScript фреймворком, який також використовує компонентну архітектуру, але має меншу криву навчання порівняно з React. Vue.js надає простий та елегантний синтаксис, а також широкий набір інструментів для розробки веб-додатків.

Переваги: менша кількість концепцій (більш просте навчання та використання), більш гнучка архітектура (нав'язує жорстких правил щодо структури проекту, що дозволяє використовувати Vue.js в різних типах проектів), легша інтеграція з існуючими проектами, вбудовані можливості для анімації та переходів.

Недоліки: менша екосистема та ресурси, менша популярність та прийняття у галузі, менша гнучкість та вибір архітектури.

Більш короткий огляд інших аналогів:

Svelte - має кращу оптимізацію, так як компілює компоненти на стадії збирання проекту, через що клієнту не потрібно завантажувати та виконувати додатковий JS-код. Але Svelte має меншу популярність в галузі, через що велика кількість плагінів та додаткових інструментів не має офіційної підтримки, а їх налаштування вимагає багато часу.

Angular - з переваг Angular можно виділити ширший набір функцій з коробки, наприклад: маршрутизація, HTTP-запити. З недоліків: вищий поріг входження, адже Angular використовується для більш складних застосунків, великий розмір пакету, обмежена гнучкість, так як Angular накладає певні обмеження та конвенції на розробку, що може ускладнювати вирішення специфічних завдань та вимагати відступу від стандартів розробки.

1.2.2.2 Маршрутизація

Так як веб-сайт складається з багатьох сторінок і розробляється за допомогою React, то для маніпулюванням сторінками було прийнято рішення використовувати концепцію SPA (Single Page Application).

SPA (Single Page Application) - це тип веб-додатка, який працює у браузері та взаємодіє з користувачем, не потребуючи перезавантаження сторінки з сервера. У SPA весь контент завантажується один раз при першому завантаженні додатка (якщо не використовується динамічне завантаження), а подальша навігація відбувається шляхом динамічної зміни вмісту на сторінці за допомогою JavaScript. Для реалізації цієї концепції я буду використовувати React Router.

React Router - це бібліотека для маршрутизації веб-додатків, яка дозволяє створювати односторінкові додатки (SPA) в React. Використання React Router має декілька важливих переваг: маршрутизація, організація коду (розділ компонентів за маршрутами), динамічні маршрути (дозволяє передавати параметри в URL-адреси), вбудована підтримка React, підтримка HTTP-запитів.

Аналоги:

React Router не є автором концепції SPA та маршрутизації, тому існує багато альтернативних бібліотек. Але, якщо розглядати це питання в контексті React, то єдиним аналогом React Router є Reach Router.

Reach Router: Це маршрутизатор, що враховує досяжність, створений командою з React Training. Reach Router має простий та ефективний інтерфейс, а також підтримує декларативний підхід до маршрутизації.

Переваги:

- Д**осяжність**: Reach Router створений з урахуванням досяжності, що робить його більш підходящим для розробки доступних веб-додатків для користувачів з обмеженими можливостями.
- П**ідтримка статичної генерації**: Reach Router дозволяє створювати динамічні маршрути для статично згенерованих сайтів, що полегшує оптимізацію та розгортання веб-додатків.

Недоліки:

- **Обмежений функціонал**: На даний момент React Router має більший набір функцій та можливостей, порівняно з Reach Router, що може призвести до обмеженості у виборі функціональності для певних проектів.
- **Обмежена документація та підтримка**: У порівнянні з React Router, документація та підтримка Reach Router можуть бути менш розвинені, що може зробити важчим вирішення проблем та отримання допомоги в спільноті.

1.2.2.3 Інструменти для керування глобальним станом додатка

Велика кількість інформації є загальної для багатьох сторінок та компонентів сайту. В основному, це інформація яку надсилає сервер, наприклад, інформація про категорії - назва, id, список підкатегорій. Тому доцільно, з точки зору контролю мережевого трафіка, відповідності даних, оптимізації продуктивності, зберігати цю інформацію як глобальний стан додатку. Для керування глобальним станом вже давно існують рішення, які є стандартом розробки на даний момент. Наприклад, Redux. Його переваги: централізоване зберігання стану, передбачуваність та однонапрямлений потік даних, спрощення налагодження та виявлення помилок,
Також, я буду використовувати додаткові бібліотеки Redux, такі як:

React Redux - для спрощення використання Redux-логіки у React-компонентах;

Redux Toolkit - для спрощення роботи мережевих запитів, результат яких буде зберігатися у глобальному стані. Також, він автоматично кешує результати мережевих запитів;

В цілому, Redux забезпечує ефективне керування станом додатка, роблячи його більш передбачуваним, розширюваним та легко налагоджуваним.

1.2.2.4 CSS-препроцесор

Для більш зрозумілої та ефективної розробки CSS-стилів, я буду використовувати CSS-препроцесор SASS (SCSS).

SCSS (Sassy CSS) - це розширений синтаксис для CSS, який додає деякі потужні функції та можливості, що роблять стилізацію веб-сторінок більш зручною та ефективною. Перевагами SCSS є: підтримка змінних, вкладеність, міксини та функції, вбудована підтримка каскаду та можливість імпорту стилів.

Загалом, використання SCSS спрощує розробку та управління стилями веб-додатків, роблячи код більш зрозумілим, ефективним та легко змінюваним.

1.2.2.5 Система контролю версій

Для того щоб мати можливість робити резервні копії, переглядати історію змін, використовувати паралельну розробку та керувати версіями, я буду використовувати систему контроля версій Git.

Git - це розподілена система контролю версій, яка дозволяє розробникам відстежувати зміни в програмному коді, спільно працювати над проектом та керувати версіями коду. Ось деякі важливі причини використання Git: історія змін, гілки версій / варіантів розробки, спільна робота, відновлення стану проекту.

Узагальнюючи, використання Git є важливою практикою для будь-якого розробника програмного забезпечення, оскільки він полегшує керування та співпрацю над проектом, забезпечує безпеку та надійність, а також допомагає відстежувати історію змін.

1.2.2.6 Збірник проекту

Для транспілювання SCSS та JSX у HTML, CSS, JS, автоматизації збірки, а також оптимізації ресурсів та підтримки модульності, я буду використовувати збірник проектів Webpack.

Webpack - це збірник модулів для сучасних веб-додатків, який дозволяє збирати, обробляти та оптимізувати різні ресурси проекту. Ось декілька важливих причин використання Webpack: модульність, збірка та оптимізація ресурсів,

підтримка модулів та завдань, Hot Module Replacement (HMR, дає можливість вносити зміни до коду додатку та бачити їх відображення без перезавантаження сторінки), широкі можливості конфігурації.

1.3 Архітектура сайту

У даному розділі буде надано детальний огляд архітектури веб-сайту. Веб-сайт "Gymba" побудований на клієнт-серверній архітектурі, де клієнтська сторона взаємодіє з серверною стороною через HTTP-протокол. Клієнтська сторона реалізована за допомогою JS-бібліотек React, React Router, Redux, а серверна частина зроблена за допомогою JS та емулюється на стороні клієнта за допомогою API MSW (Mock Server Worker). Серверна інформація зберігається у форматі JSON.

1.3.1 Структура сайту та маршрутизація

Загалом, веб-сайт складається з 13 сторінок, 4 з яких є шаблонами для відповідної інформації з серверу.

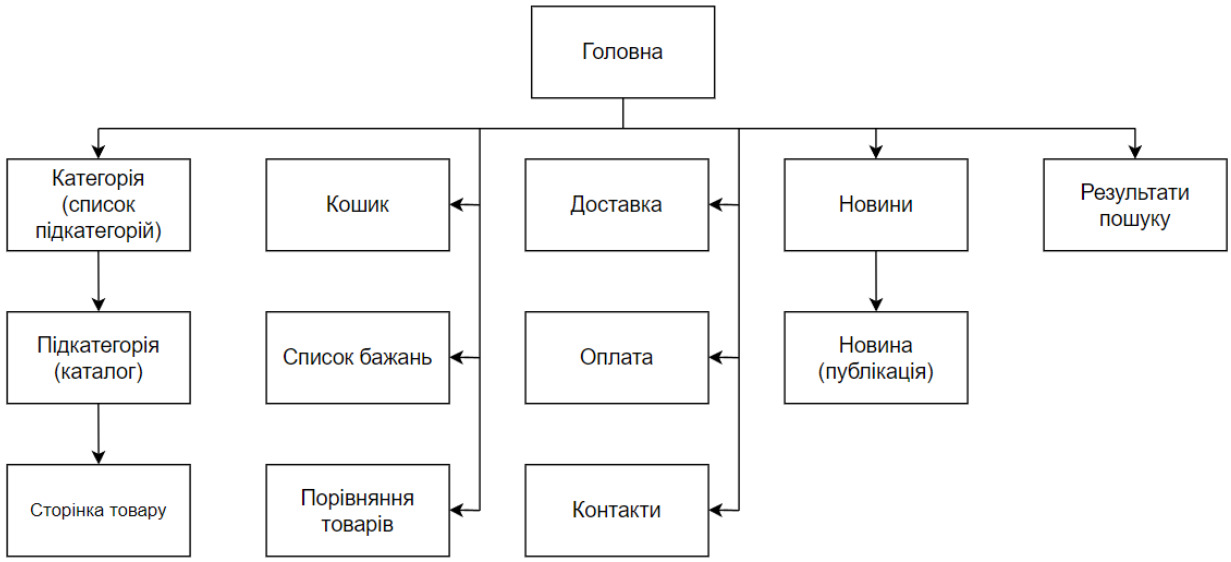


Рисунок 1.3.1 - Структура сайту Gymba.

Як було вказано раніше, для маршрутизації сайту використовується бібліотека React Router. Нижче наведено структуру маршрутів (routes), на якій також вказано адресу для кожного маршруту:

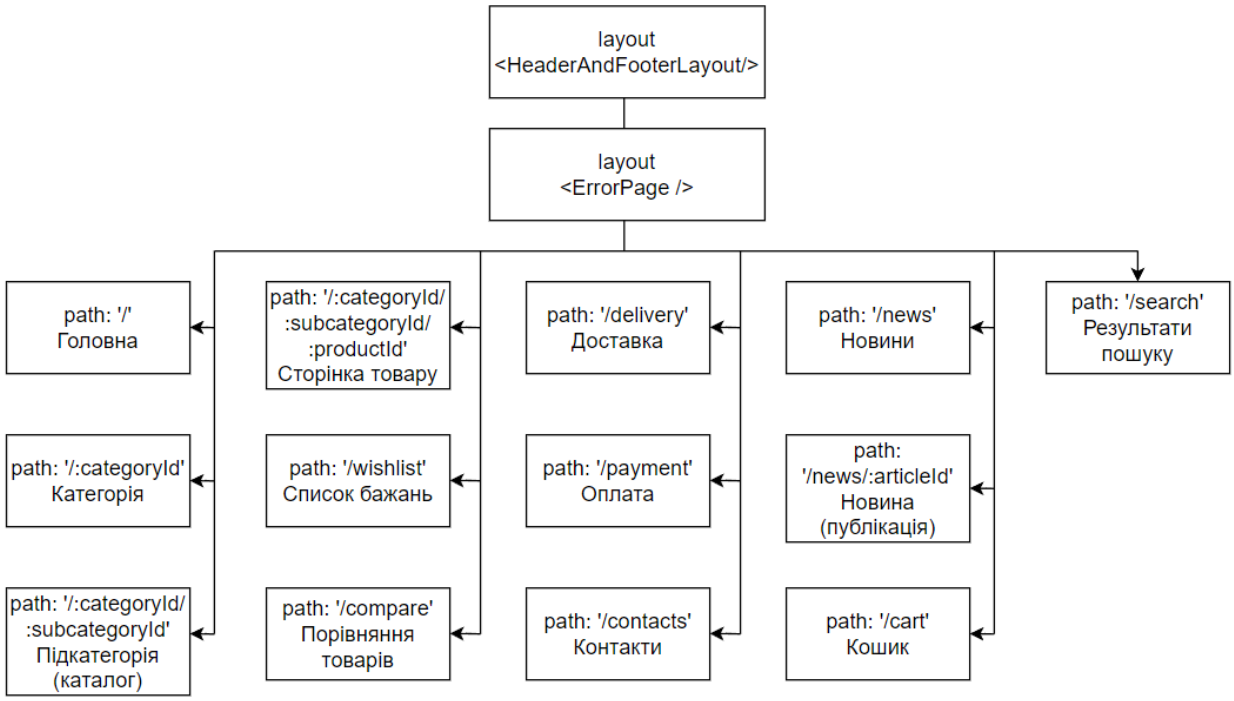


Рисунок 1.3.2 - Структура маршрутів сайту Gymba.

В цілому, схема подібна до схеми загальної структури сайту (рис. 1.3.1), але є декілька важливих моментів. Кожен маршрут відокремлений від інших, тобто вони не залежать один від одного (вони не вкладені, тобто жоден маршрут не є елементом іншого), проте, вони всі є дочірніми маршрутами двох макетів (layouts) - HeaderAndFooterLayout та ErrorPage.

Макет (layout) - це маршрут, який не має своєї адреси (path), а тому не приймає участі у маршрутизації. Він слугує обгорткою для інших маршрутів, щоб не дублювати загальний інтерфейс та функціональність. Так як header та footer є базовими елементами для кожної сторінки (маршруту), тому, щоб не дублювати код виклику цих елементів у кожній сторінці, вони були реалізовані у вигляді макету.

Другим макетом є елемент ErrorPage, який відловлює усі не перехоплені помилки і показує їх у зручному інтерфейсі. Також, це дозволяє користувачеві перейти на іншу сторінку сайту, не вдаючись до ручної зміни URL або перезавантаження сторінки.

Також, варто зауважити наявність динамічних сегментів адреси (напр. категорія - ‘:categoryId’; підкатегорія - ‘:categorydId/:subcategoryId’ тощо). Такі сегменти декларуються, починаючись з ‘:’. Коли маршрут збігається з URL-адресою, динамічний сегмент буде проаналізовано з URL-адреси та надано як параметри іншим API маршрутизатора.

1.3.2 Дані, які зберігаються на сервері

Серверна інформація зберігається у файлах формату JSON, у нормалізованому вигляді. Тобто, кожна категорія даних зберігається у простому об’єкті з полями id та entities. Id - це масив, в якому зберігаються всі id цього типу даних у строковому форматі. Entities - це простий об’єкт, в якому зберігається інформація про кожен елемент цього типу даних, де ключем є строка id, а значенням - об’єкт з необхідною інформацією. Таких підхід оптимізує процес отримання необхідної інформації. По-перше, багатьом частинам застосунку необхідне лише список id, отримання яких через цикл може викликати проблеми з оптимізацією, якщо кількість елементів обчислюється в тисячах. По-друге, часто необхідно отримати об’єкт якогось елемента за допомогою id. Без нормалізації, для цього необхідно також зробити перебір всіх елементів, де для кожного елементу буде зроблена перевірка, чи є однаковими їх id. Це не викликає багато труднощів, якщо категорія елемента, який ми шукаємо, не є чиєюсь підкатегорією. Але, якщо категорія глибоко вкладена, то кількість ітерацій значно зростає. Нормалізовані дані дозволяють отримати об’єкт за id за простою схемою [categoryName].entities[id], взагалі не використовуючи перебір.

Загалом, є два JSON-файли: categories та news. В categories зберігається інформація про категорії, підкатегорії, продукти:

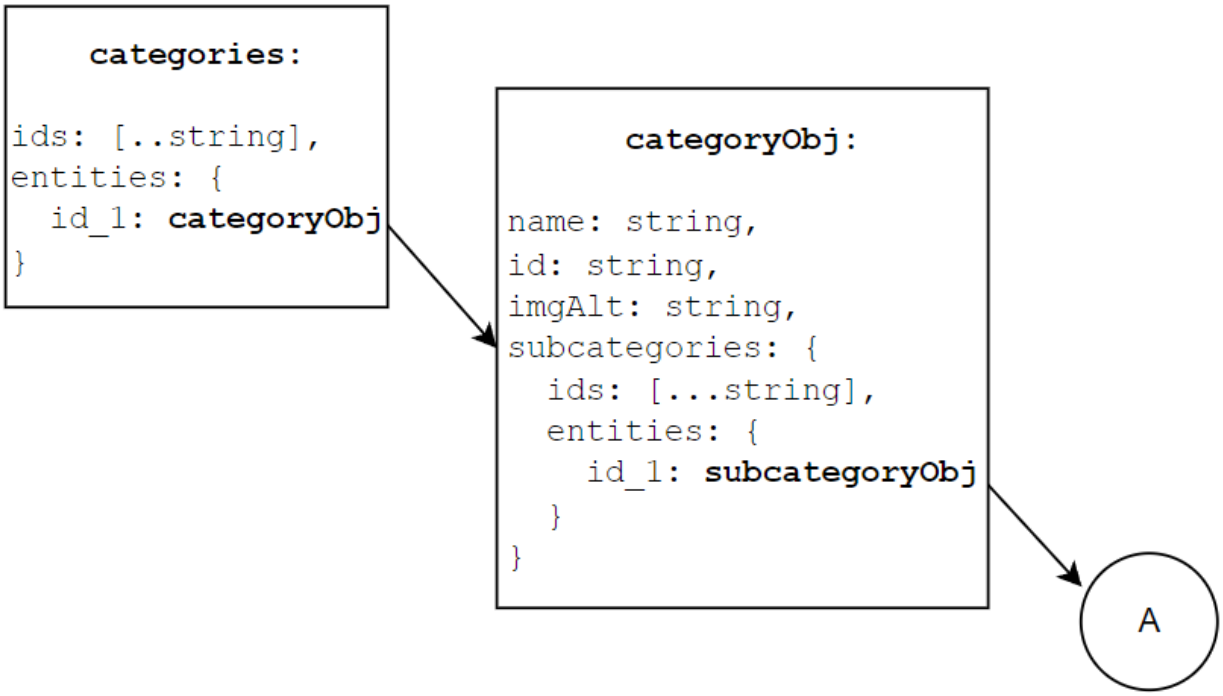


Рисунок 1.3.3 (а) - Схема даних categories.json.

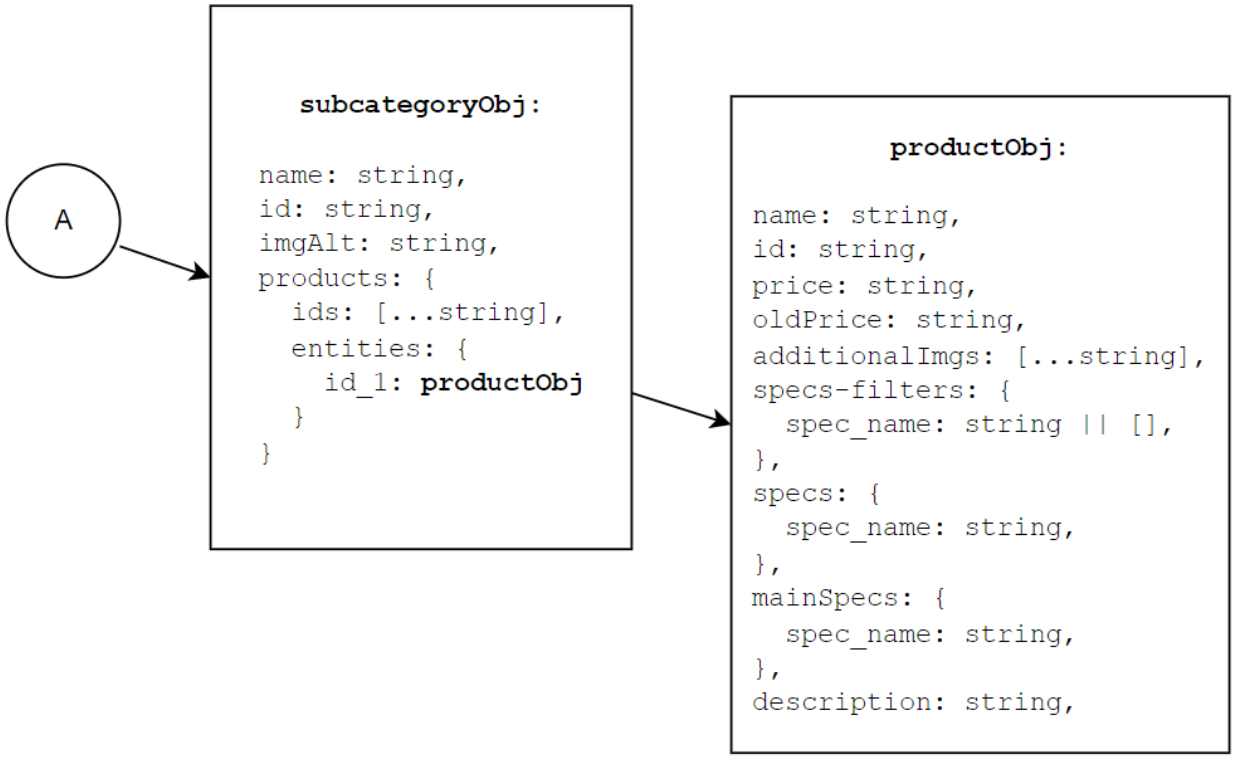


Рисунок 1.3.3 (б) - Схема даних categories.json.

В news зберігається список та контент новин та статей:

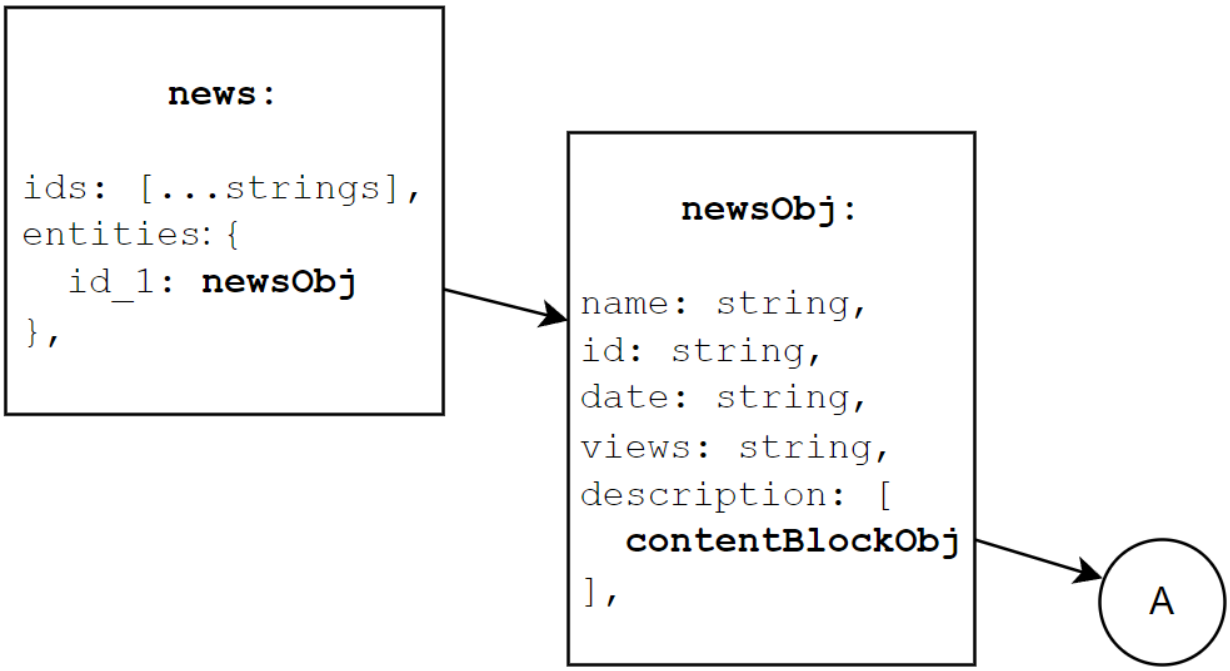


Рисунок 1.3.4 (а) - Схема даних news.json.

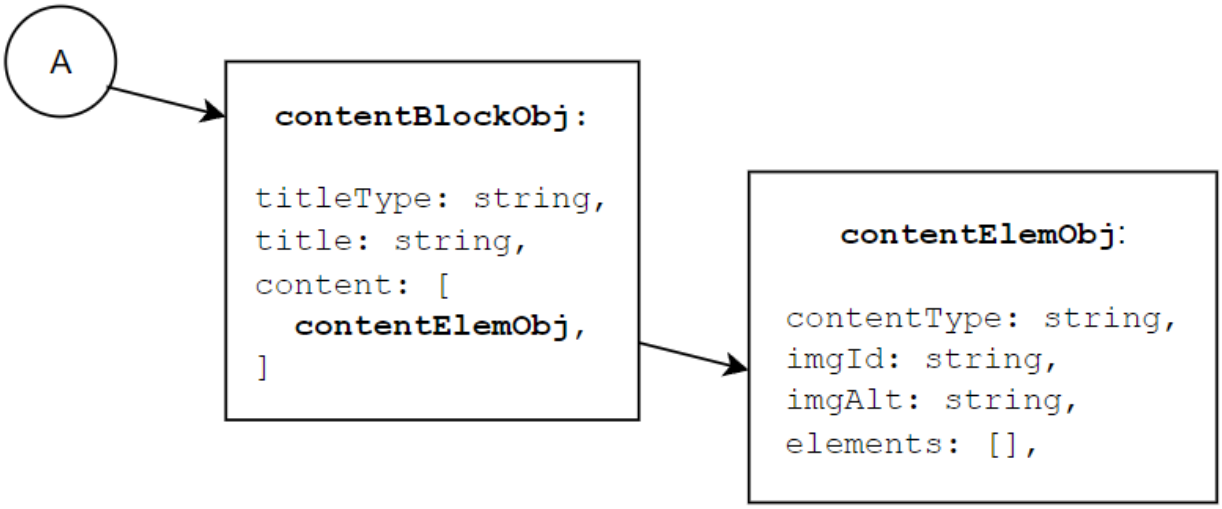


Рисунок 1.3.4 (б) - Схема даних news.json.

1.3.3 Клієнт-серверна взаємодія

Для емуляції сервера на стороні клієнта, я використовую API MSW (Mock Service Worker), який дає можливість створювати ендпоїнти (URL-адреси, з якими можна взаємодіяти, щоб отримати або відправити дані):

```
http.get('/fakeAPI/categories', async () => {
  const categories = await getCategoriesAndSubcategories();
  return HttpResponse.json(categories);
}),
http.get('/fakeAPI/getProducts/:categoryId/:subcategoryId', async ({ request, params }) => {
  const { categoryId, subcategoryId } = params;
  const { searchParams } = new URL(request.url);

  return getFilteredProductsAndMinMaxPrice(categoryId, subcategoryId, searchParams);
}),
http.get('/fakeAPI/wishlistIds', async () => {
  const wishlistIds = await getWishlistIds();

  return HttpResponse.json(wishlistIds);
}),
http.get('/fakeAPI/getWishlistProducts', async ({ request }) => {
  const pageNum = new URL(request.url).searchParams.get('page');

  return getWishlistProductsPerPageAndPageAmount(+pageNum);
}),
http.get('/fakeAPI/cartIds', async () => {
  const cartIds = await getCartIds();

  return HttpResponse.json(cartIds);
}),
http.post('/fakeAPI/cartIds', async ({ request }) => addMutation(request, addIdToCart)),
http.patch('/fakeAPI/cartIds', async ({ request }) => {
  const [categoryId, subcategoryId, productId, amount] = await request.json();

  return editProductAmountInCart(categoryId, subcategoryId, productId, amount);
}),
http.get('/fakeAPI/getCartProducts', async () => {
  const cartProducts = await getCartProducts();

  return HttpResponse.json(cartProducts);
}),
http.get('/fakeAPI/compareIds', async () => {
  const cartIds = await getCompareIds();

  return HttpResponse.json(cartIds);
}),
```

Рисунок 1.3.4 - Скриншот коду з ініціалізацією адрес в MSW.

Нижче наведено адреси ендпоїнтів з підтримуваними HTTP-методами, та вказано, що вони очікують отримати при запиті, та чим відповідають:

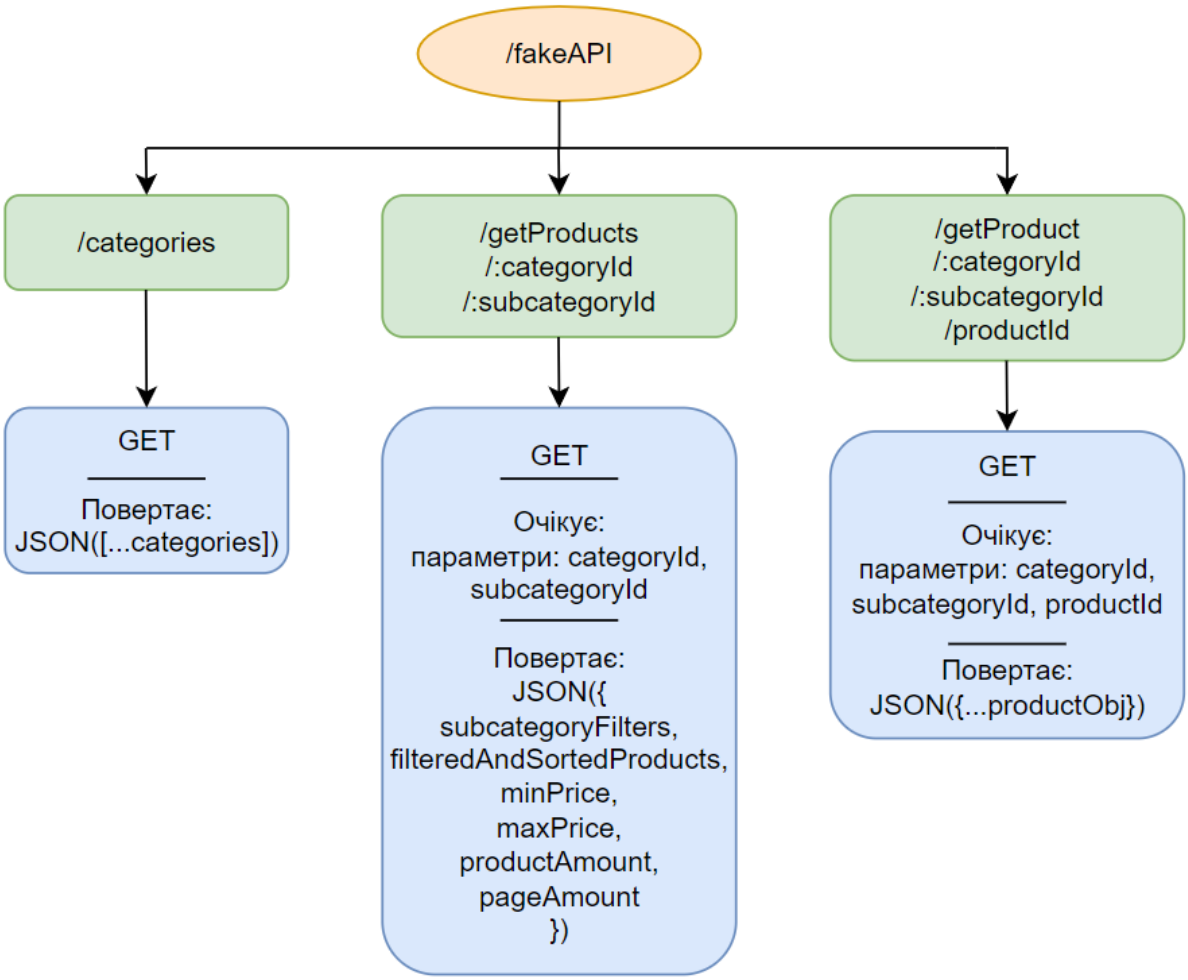


Рисунок 1.3.5 - Діаграма ендпоїнтів отримання категорії, каталогу товарів підкатегорії та повної інформації про товар.

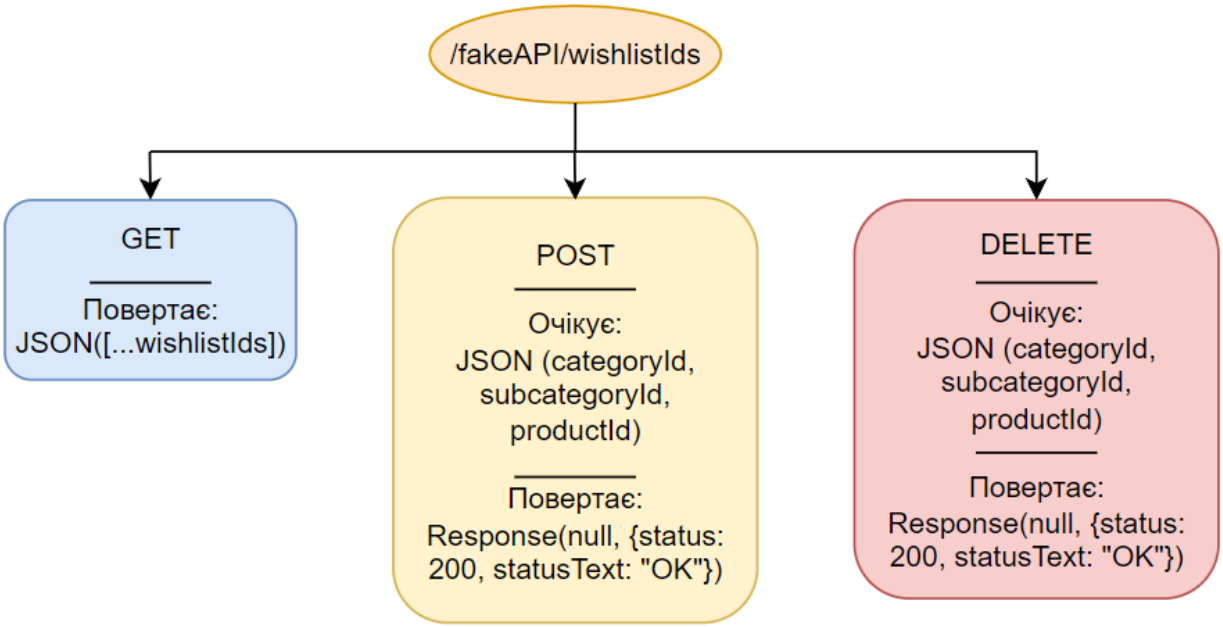


Рисунок 1.3.6 - Діаграма для ендпоїнта /wishlistIds.

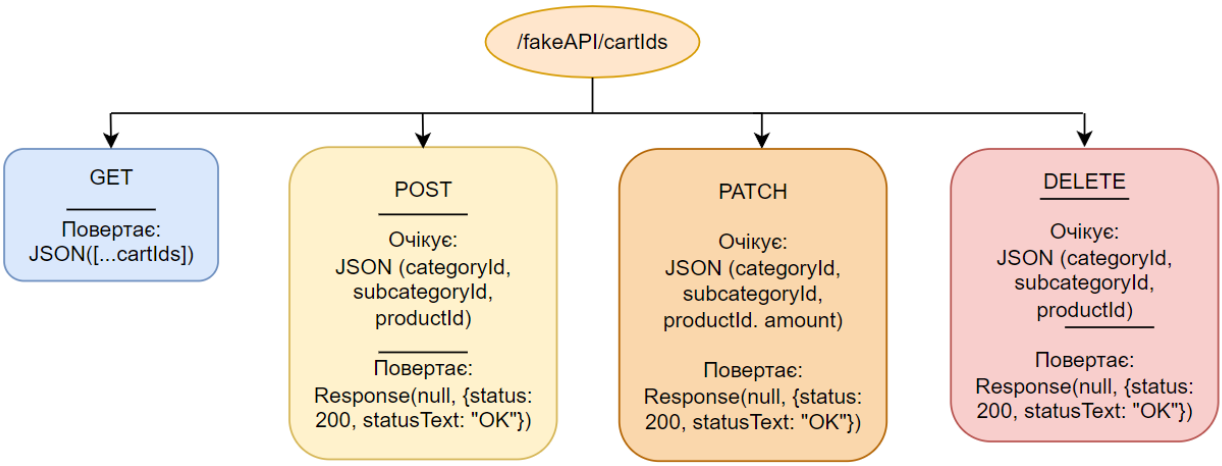


Рисунок 1.3.7 - Діаграма для ендпоїнта /cartIds.

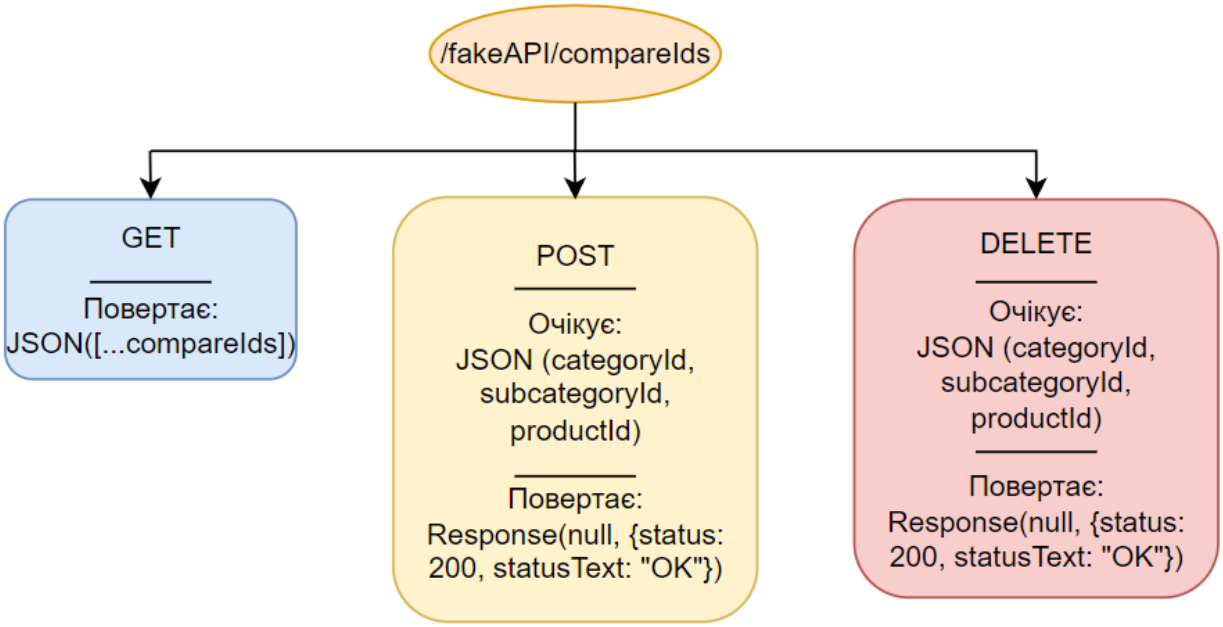


Рисунок 1.3.8 - Діаграма для ендпоїнта /compareIds.

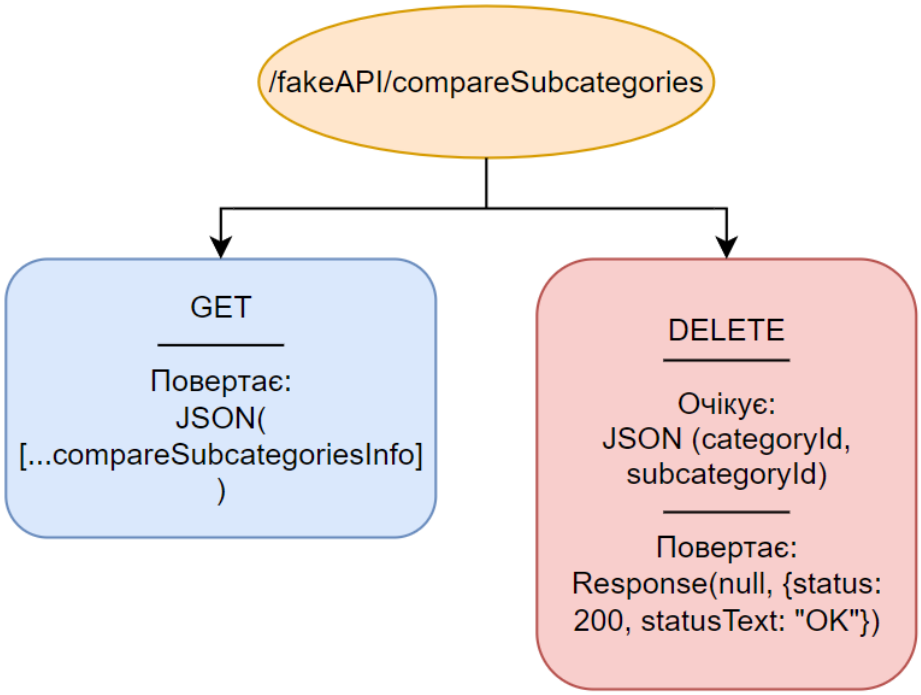


Рисунок 1.3.8 - Діаграма для ендпоїнта /compareSubcategories.

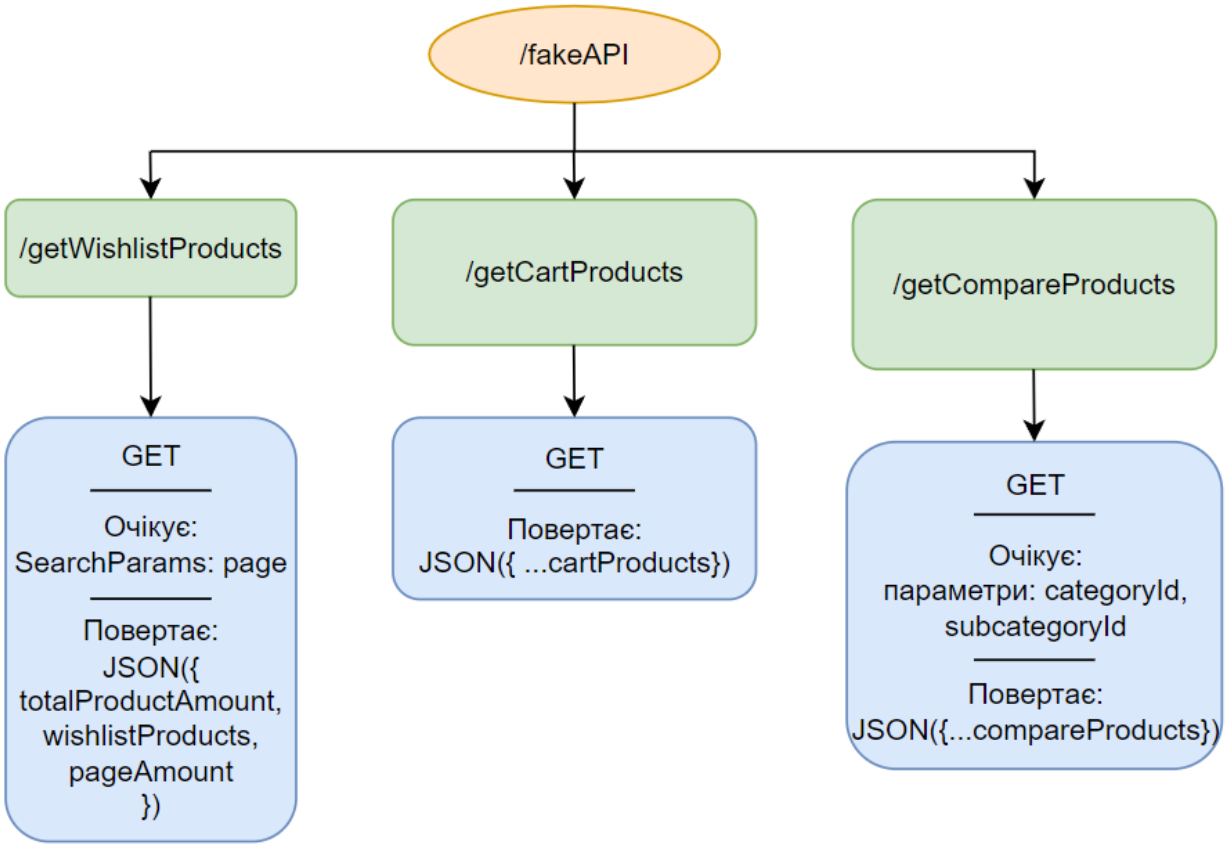


Рисунок 1.3.9 - Діаграма для ендпоїнтів отримання об’єктів товарів кошику, порівняння, списку бажань.

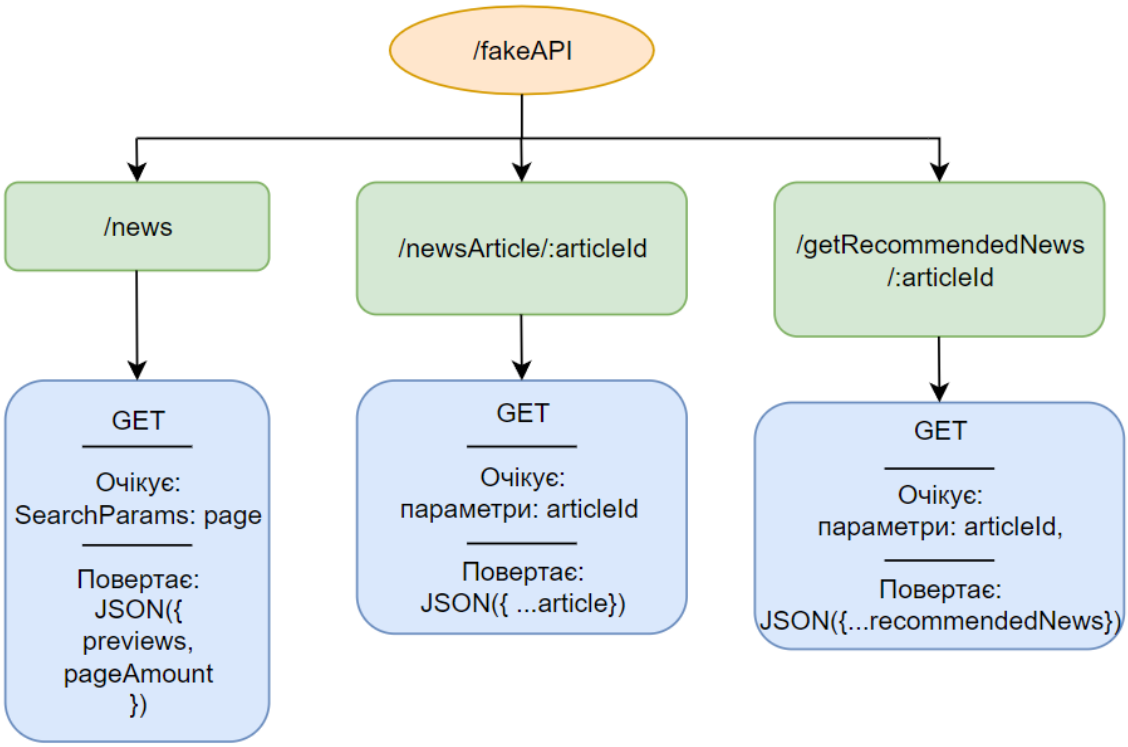


Рисунок 1.3.10 - Діаграма для ендпоїнтів отримання об’єктів новин, повної інформації про статтю, рекомендованих новин.

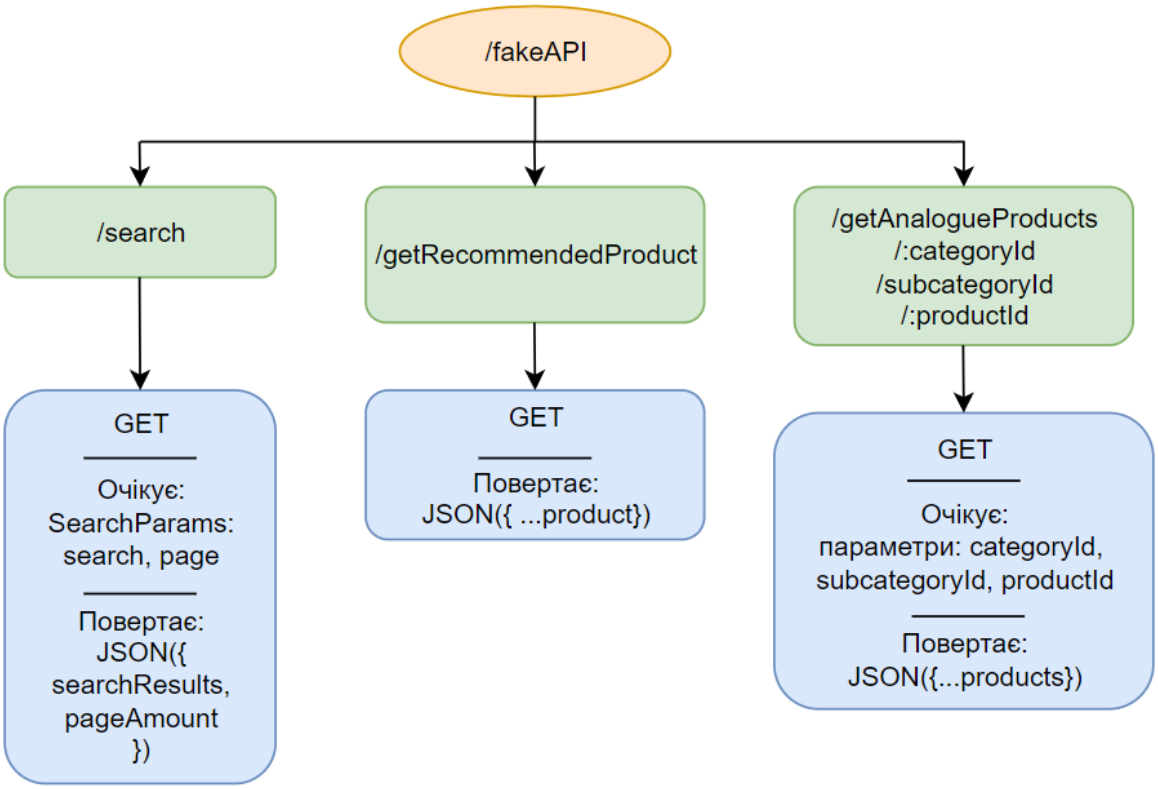


Рисунок 1.3.11 - - Діаграма для ендпоїнтів отримання об’єктів результату пошуку, рекомендованого товару, аналогічних товарів.

1.4 Налаштування збірника проекту Webpack

Налаштування збірника проекту є вкрай важливим моментом, адже він не тільки спрощує розробку та покращує робочу версію (наприклад, мініфікація коду), але і транспілює JSX та SCSS файли, які браузер не підтримує, у JS та CSS, з якими браузер вміє працювати.

Файл налаштування Webpack має експортувати об’єкт за налаштуваннями, або функцію, яка повертає такий об’єкт. Я буду використовувати варіант з функцією, адже вона буде викликатися, передаючи environment об’єкт в якості аргументу, завдяки якому, я зможу змінювати логіку налаштування, в залежності від того, це збірка під час розробки (development / dev), чи робоча версія (production / prod). Взагалі, у проекті буде існувати два типа збірки, dev та prod. Dev версія не буде використовувати багато додаткового функціоналу, такого, як наприклад, мініфікація, treeshacking (видалення не використаного коду) тощо, адже dev збірка буде робитися після кожної зміни робочого каталогу, тому вона має робитися якнайшвидше. В свою чергу, prod версія буде використовувати увесь наведений вище функціонал, адже, вона робиться дуже рідко, і часом збірки у одну-декілька хвилин можна знехтувати. Нижче описано процес налаштування збірки:

```
module.exports = (env) => {
  const isProd = env.production;
  const isShowBundleAnalyzer = env.analyze;

  const mode = isProd ? "production" : "development";
  const entry = path.resolve(__dirname, 'src/static/main.jsx');

  const output = {
    publicPath: '/gymba/',
    filename: '[name].[contenthash].bundle.js',
    path: path.resolve(__dirname, 'bundle'),
    clean: true,
  }
```

Рисунок 1.4.1 - Налаштування типу збірки та JS-коду.

Як було вказано раніше, я використовую функцію, щоб мати можливість приймати аргумент env, на основі якого я зможу вибирати тип збірки. Першим налаштування буде значення поля mode, адже Webpack має свої власні початкові налаштування

як для dev версії, так і для prod версії. А саме, він буде робити мініфікацію та treeshacking JS-коду у prod версії. Далі, я буду робити налаштування JSX коду, та його транспіляцію у JS. Спочатку, мені необхідно вказати точку входу - entry. Я використовую функції та константи Node.js, такі як __dirname та path.resolve, щоб власноруч не вводити робочу директорію проекту, а також, щоб зробити коректний абсолютний шлях, адже саме його очікує поле entry. Далі, я буду налаштовувати вихідні дані. В path я вказую абсолютний шлях до місця, де буде зберігатися зібраний проект. Для цього я також використовую функціонал Node.js. Друге, це схема найменування файлів. Якщо в проекті лише одна точка входу та не використовується код-сплітинг (розділ великого файлу на малі), то можна використовувати статичну назву. Однак, створюючи кілька файлів за допомогою кількох точок входу, розділення коду або різних плагінів, я маю використовувати наявні у Webpack шаблони (хеші), щоб надати кожному пакету унікальну назву.

Template	Description
[id]	The ID of the chunk
[name]	The name of the chunk, if set, otherwise the ID of the chunk
[chunkhash]	The hash of the chunk, including all elements of the chunk
[contenthash]	The hash of the chunk, including only elements of this content type (affected by `optimization.realContentHash`)

Рисунок 1.4.2 - Варіанти шаблонів найменувань файлів у Webpack.

У назві вихідного файлу, я буду використовувати шаблон [name], який, зберігає оригінальну назву файлу, та [contenthash] для створення унікального хешу на основі контенту файлу. Це необхідно для того, щоб при кожній зміні файлу генерувався новий хеш, через що, браузер буде завжди використовувати актуальну версію файлу, а не версію, взяту з кешу браузера. Тому, мені не прийдеться вручну очищати кеш браузера, щоб гарантовано отримати актуальну версію файлу. Опція clean очищає робочий каталог перед кожною збіркою, щоб там не залишалося старих та неактуальних файлів.

Але, Webpack не має передвстановленого функціоналу для транспіляції JSX у JS. Тому необхідно власноруч завантажити та налаштувати модуль для цього:

```
const module = {
  rules: [
    {
      test: /\.m?jsx?$/,
      exclude: /node_modules/,
      use: {
        loader: 'babel-loader',
        options: {
          presets: [
            [
              '@babel/preset-env',
              isProd && { "useBuiltIns": "usage", "corejs": "3.35.1" }
            ].filter(Boolean),
            [
              '@babel/preset-react',
              { runtime: 'automatic' },
            ],
          ],
          plugins: [!isProd && require.resolve('react-refresh/babel')].filter(Boolean),
        },
      },
    },
```

Рисунок 1.4.3 - Налаштування транспіляції JSX у JS.

Для цього (а також для інших типів файлів), я буду використовувати такий функціонал Webpack як loaders у поєднання з окремим loader - babel-loader. Loaders - це перетворення, які застосовуються до вихідного коду модуля. Вони дозволяють попередньо обробляти файли під час їх імпорту або «завантаження». Таким чином, loaders є схожими на «tasks» в інших інструментах збірки та забезпечують потужний спосіб обробки етапів початкової збірки. Завантажувачі можуть перетворювати файли з іншої мови (наприклад, TypeScript) на JavaScript або завантажувати вбудовані зображення як URL-адреси даних. Babel-loader дає можливість транспілювати JSX у JS, а також деякий інший функціонал. Більш детально про налаштування: test - це регулярний вираз, який відповідає файлам, для яких має застосовуватися цей loader. Exclude - це файли або папки, які не входять до цього переліку. В моєму випадку це /node_modules/ адже це папка зі сторонніми модулями, які необхідні для розробки і не використовуються самим сайтом. Loader - безпосередньо loader, який буде використовуватися для цих файлі. В нього можуть бути свої налаштування - options, як в моєму випадку, де я використовую

presets - готові налаштування для окремих типів файлі та видів збірки. Я використовую preset-react, щоб зробити правильну транспіляцію JSX, та preset-env, який я використовую лише при prod збірці. Він використовує налаштування файлу .browserlistrc, щоб зробити транспіляцію JS коду у більш ранній варіант, який підтримує перелік браузерів, зазначений у файлі .browserlistrc, а також додає поліфіли - нові функції JS, які старі браузери не підтримують.

Далі, необхідно налаштувати транспіляцію SCSS у CSS, а також інші функції, такі як: мініфікація, створення source map (дає можливість побачити оригінальний код, якщо виникне помилка), додавання браузерних префіксів, адже всі ці функцію не є частиною Webpack:

```
{
  test: /\.s[ac]ss$/i,
  use: [
    isProd ? {
      loader: MiniCssExtractPlugin.loader,
      options: {
        publicPath: "/gymba/",
      },
    } : "style-loader",
    {
      loader: "css-loader",
      options: {
        modules: {
          localIdentName: isProd ? '[hash:base64:8]' : '[path][name]__[local]',
        },
        sourceMap: !isProd,
      }
    },
    {
      loader: "postcss-loader",
      options: {
        sourceMap: !isProd,
        postcssOptions: {
          plugins: isProd ? ["autoprefixer"] : [],
        }
      }
    },
    "resolve-url-loader",
    {
      loader: "sass-loader",
      options: {
        sourceMap: true,
      }
    }
  ]
},
```

Рисунок 1.4.4 - Налаштування транспіляції SCSS у CSS.

На цьому прикладі можна помітити одну з важливих можливостей loader. Їх можна групувати у ланцюги знизу-вверх, або зліва-направо, де результат першого loader передається у наступний. Це дозволяє робити більш комплексні перетворення. Для мого проекту я використовую наступні loader (у порядку роботи): sass-loader - транспілює SCSS код у CSS, а також створює source-map, що зазначено у налаштування options; resolve-url-loader - дозволяє правильно знаходити файли, які використовуються у файлах SCSS (наприклад, медіа-файли, або використання міксинів з інших SCSS файлів тощо); postcss-loader - loader, який при production збірці додає браузерні префікси, використовуючи параметри налаштування із файлу .browserlistrc; css-loader - також потрібен для правильних імпортів медіа та інших файлів, які використовуються у SCSS-файлах, але поміж цього, має налаштування для підтримки CSS Modules (методологія, яка має своє API і використовується для коректного і зручного маніпулювання CSS-класами). І останні, це style-loader та MiniCssExtractPlagin.loader. При dev збірці використовується style-loader, який додає створений код безпосередньо у HTML файл (inline стилі). Це дозволяє більш швидко робити dev збірки, що відбувається постійно. MiniCssExtractPlugin.loader використовується у production збірці для створення окремих CSS-файлі, адже розділення файлів зменшує кількість мережевого трафіку, тобто дозволяє клієнту завантажувати їх при необхідності.
Далі, я налаштую збірку шрифтів та медіа-файлів:

```
{
  test: /\.(woff|woff2|eot|ttf|otf)$/i,
  type: 'asset/resource',
  generator: {
    filename: `assets/fonts/[name].[contenthash].[ext]`,
  }
},
{
  test: /\.(jpe?g|png|gif|webp)$/i,
  type: 'asset/resource',
  generator: {
    filename: `assets/images/[name].[contenthash].[ext]`,
  },
  use: isProd ? ['image-webpack-loader'] : [],
},
{
  test: /\.svg$/i,
  resourceQuery: { not: [/url/] },
  use: [
    {
      loader: '@svgr/webpack',
      options: { icon: true },
    },
  ],
},
```

Рисунок 1.4.5 - Налаштування шрифтів та медіа-файлів.

З недавніх пір, Webpack «з коробки» підтримує можливість знаходити та додавати до збірки шрифти та медіа-файли. Раніше, для цього також необхідно було завантажувати окремі loader. Для налаштування цієї функції, необхідно вказати типи вхідних файлів, вказати тип їх перетворення у полі type (в моєму випадку, Webpack буде створювати окремі файли), та вказати шлях та формат створюваних файлів у полі generator.filename. Для зображень при prod збірці, я додатково використовую image-webpack-loader який їх оптимізує. Для SVG-файлів я додатково використовую модуль SVGR, який дає можливість імпортувати ці файлі як React компоненти, що спрощує маніпулювання ними (наприклад, додавання стилів).

Далі, я налаштує плагіни:

```
const plugins = [
  new HtmlWebpackPlugin({
    filename: path.resolve(__dirname + '/bundle/index.html'),
    template: path.resolve(__dirname, 'src/static/index.html'),
  }),
  new MiniCssExtractPlugin({
    filename: 'styles/[name].[contenthash].bundle.css',
  }),
  new FaviconsWebpackPlugin({
    logo: "./src/assets/favicons/favicon.png",
    outputPath: "./assets/favicons",
    prefix: './assets/favicons/',
    inject: true,
    favicons: {
      icons: {
        yandex: false
      }
    }
  }),
]

!isProd && plugins.push(new ReactRefreshWebpackPlugin());
isShowBundleAnalyzer && plugins.push(new BundleAnalyzerPlugin());
```

Рисунок 1.4.6 - Налаштування плагінів.

HtmlWebpackPlugin додає підтримку HTML-файлів, а також автоматично підключає до них JS-файли. Його налаштування template приймає абсолютний шлях до файлу, на основі якого буде збиратися HTML-файл, а filename - шлях, де куди буде збиратися HTML-файл. FaviconsWebpackPlugin приймає шлях до фавіконки (іконка, яка відображається рядом з назвою сайту у панелі вкладок браузера), створює цю фавіконку у різних форматах для підтримки багатьма браузерами, та підключає їх у HTML-файл. ReactRefreshWebpackPlugin додає підтримку HMR (Hot Module Replacement) для React компонентів. BundleAnalyzerPlugin за запитом може виводити інформацію про розмір зібраного проекту та розмір його окремих файлів, що є корисним для аналізу та оптимізації. При production збірці використовується CssMinimizerPlugin, який використовується для мініфікації CSS-файлів.

Також, для розробки необхідно налаштувати сервер:

```
const devServer = !isProd ? {
  open: ['/gymba/'],
  watchFiles: ['src/**/*.html', 'bundle/*.html'],
  client: {
    overlay: true,
    progress: true,
  },
  compress: true,
  hot: true,
  historyApiFallback: true,
} : undefined;
```

Рисунок 1.4.7 - Налаштування Webpack DevServer.

Webpack DevServer запускає сервер з моїм проектом у локальній мережі, що дозволяє мені використовувати серверні функції, а також робити тестування сайту на телефоні. Він має наступні налаштування: open - сторінка сайту, яка автоматично відкривається при запуску серверу; watchFiles - шлях(и) до файлів, при зміні яких, сервер автоматично оновлюється; client - виводить додаткову інформацію, таку як помилки або стан компіляції; compress - використовує gzip-компресію, це не є критично необхідним, але так як реальні сервери використовують її, це може дати більш точні результати при тестуванні; historyApiFallback - включає підтримку маніпуляції API History, що є необхідним при використанні React Router. І одна з найголовніших переваг Webpack DevServer - це підтримка технології HMR (Hot Module Replacement), яка включається опцією hot. HMR - обмінюється, додає або видаляє модулі під час роботи програми без повного перезавантаження. Це може значно прискорити розробку. Наприклад, якщо при розробці ввести якусь інформацію у поле вводу, а потім змінити стилі довільного елементу, то буде зроблено повне перезавантаження сторінки і, звісно, введена інформація видалиться. З HMR, зміни будуть зроблені необхідному елементу безпосередньо у браузері, через що сторінку не буде перезавантажено, а інформація не буде видалена.

І останнє, це невелике налаштування оптимізації:

```
const optimization = {
  moduleIds: 'deterministic',
  runtimeChunk: 'single',
  splitChunks: {
    cacheGroups: {
      vendor: {
        test: /[\\/]node_modules[\\/]/,
        name: 'vendors',
        chunks: 'all',
      },
    },
  },
};
```

Рисунок 1.4.8 - Налаштування оптимізації.

Це налаштування дозволяє відокремити бібліотеки сторонніх розробників, таких як React, в окремий файл, оскільки вони менш імовірно змінюватимуться, ніж наш локальний вихідний код. Цей крок дозволить клієнтам запитувати ще менше від сервера, щоб залишатися з актуальним кодом.

1.5 NPM-скрипти збірки

Для того щоб запустити процес збірки та DevServer необхідно створити NPM-скрипти в package.json файлі, та, відповідно, запустити ці скрипти. Нижче описано всі скрипти які використовуються проектом:

```
"scripts": {
  "eslint": "eslint src/**/*.jsx src/**/*.js",
  "eslint:fix": "eslint src/**/*.jsx* --fix",
  "stylelint": "stylelint **/*.scss",
  "stylelint:fix": "stylelint **/*.scss --fix",
  "build:dev": "webpack",
  "build:dev:analyze": "webpack --env analyze",
  "build:prod": "npm run eslint:fix && npm run stylelint:fix && webpack --env production",
  "serve": "webpack serve",
  "deploy": "gh-pages -d bundle"
},
```

Рисунок 1.5.1 - NPM-скрипти проекту.

eslint - включає лінтер JS-коду, який буде аналізувати код з метою виявлення потенційних помилок, неоднозначностей, стилістичних недоліків;

eslint:fix - автоматично виправляє деякі помилки, знайдені лінтером;

stylelint - включає лінтер SCSS-коду;

stylelint:fix - автоматично виправляє деякі помилки, знайдені лінтером;

build:dev - робить development версію збірки;

build:dev:analyze - робить development версію збірки та відкриває вікно результату аналізу проекту;

build:prod - робить перевірку лінтерами, і за умови відсутності помилок, робить production збірку;

serve - робить development збірку та запускає локальний сервер за допомогою Webpack DevServer;

deploy - завантажує проект на GitHub Pages;

1.6 Розробка сторінок сайту

Процес розробки сторінки та компонентів, які вона містить, я продемонструю на прикладі сторінки «Каталог товарів».

Першим етапом є створення React-елементу сторінки, який складається менших елементів (ці елементи, в свою чергу, також можуть складатися з інших елементів) і підключення цього елементу до декларації маршрутів React Router. Для цього достатньо імпортувати необхідні елементи (їх функціонал ще не реалізовано, лише ініціалізовано та експортовано функції), створити безпосередньо елемент сторінки та експортувати його:

```jsx
import BreadCrumbs from '../../components/BreadCrumbs/BreadCrumbs.jsx';
import Products from '../../components/Products/Products.jsx';

export default function ProductsPage() {
  return (
    <>
      <BreadCrumbs />
      <Products />
    </>
  );
}
```

Рисунок 1.6.1 - Елемент сторінки ProductsPage.

Перед тим, як використовувати його у маршрутизації, я зроблю його lazy-варіант, тобто елемент, який клієнт буде завантажувати лише за необхідністю. За замовчуванням, клієнт завантажує весь не lazy-код сайту, що значно збільшує час першого завантаження сайту і погано впливає трафік. Lazy-елементи будуть завантажені клієнтом лише тоді, коли це буде дійсно потрібно, наприклад, переходячи на сторінку, де цей елемент використовується. Для створення такого елементу, мені необхідно завантажити звичайний елемент (у моєму випадку це елемент сторінки), використовуючи ES-функцію динамічного імпорту - import, та React-функцію lazy, яка дозволяє відкласти завантаження компонента. Створений lazy-елемент буде експортовано, щоб використовувати у маршрутизації:

```
import { lazy } from 'react';

const ProductsPageLazy = lazy(() => import('./Products.jsx'));

export default ProductsPageLazy;
```

Рисунок 1.6.2 - Lazy-елемент сторінки ProductsPageLazy.

Далі, у файлі маршрутизації я імпортую цей елемент та створюю для нього маршрут:

```
{
  path: ':categoryId/:subcategoryId',
  element: <ProductsPageLazy />,
},
```

Рисунок 1.6.3 - Маршрут для сторінки «Каталог товарів».

Адреса цієї сторінки складається з двох динамічних параметрів - categoryId та subcategoryId, тобто ця сторінка не є статичною та використовується як каталог товарів для будь-якої підкатегорії будь-якої категорії.

Як видно на рисунку 1.6.1, елемент сторінки складається з двох елементів: Breadcrumbs та Products.

Breadcrumbs - це елемент, який показує ланцюг навігації від головної сторінки до поточної. Він також є динамічним, адже використовується багатьма сторінками сайту. На сайті існує три види ланцюгів навігації: до товару, до новини та довільна сторінка (доставка, оплата тощо). Першим елементом кожного ланцюга

буде посилання на головну сторінку, адже вона є початком сайту. Функція створення елементу використовує хук React Router - useParams, який повертає об'єкт зі значеннями для динамічних параметрів маршруту. Це дозволить зрозуміти, який із трьох зазначених вище видів ланцюгів необхідно створити. Маючи значення динамічних параметрів, можна зробити посилання на сторінку, але не вистачає одного елементу - назви сторінки. Знаючи тип сторінки та її ID, можна дізнатися назву. Є багато варіантів для реалізації цього. Можна зберігати цю інформацію у вигляді об'єкту, як частину елементу Breadcrumbs, або робити окремий мережевий запит з інформацією саме для цього компоненту. Я обрав інший варіант - я буду робити мережевий запит, використовуючи Redux Toolkit Query API. На перший погляд, це також виглядає як окремий мережевий запит саме для цього компоненту, але це не так. Такий самий запит робить елемент Header, щоб мати інформацію про сторінки категорій та підкатегорій. Redux Toolkit Query API кешує та зберігає результат цього запиту у глобальному стані. Після чого, якщо якийсь елемент буде робити такий самий запит, він одразу отримає результати з глобального стану. Для використання Redux Toolkit Query API необхідно ініціалізувати API з основними налаштуваннями та деклараціями ендпоїнтів:

```
export const queryAPI = createApi(
  {
    keepUnusedDataFor: 0,
    reducerPath: 'globalData',
    baseQuery: fetchBaseQuery({ baseUrl: '/fakeAPI' }),
    tagTypes,
    endpoints: (builder) => ({
      getCategories: builder.query({
        query: () => '/categories',
        transformResponse(response) {
          return categoriesAdapter.addMany(categoriesAdapter.getInitialState(), response);
        },
        providesTags: ['categories'],
      }),
```

Рисунок 1.6.4 - Ініціалізація Redux Toolkit Query API та необхідного ендпоїнта.

Створене QueryAPI згенерує React хук, який дозволить робити мережеві запити, які QueryAPI буде надсилати до сервера (у моєму випадку, це локальний MSW API), і

обробляти відповідь (зберігати у глобальному стані, кешувати, трансформувати перед зберіганням та інше).

```
http.get('/fakeAPI/categories', async () => {
  const categories = await getCategoriesAndSubcategories();
  return HttpResponse.json(categories);
}),
```

Рисунок 1.6.5 - Ініціалізація ендпоїнта в MSW.

Як видно на рисунку 1.6.5, для того щоб отримати інформацію, стосовно категорій та підкатегорій, необхідно зробити GET-запит за адресою /fakeAPI/categories. На сервері буде зроблено запит до функції getCategoriesAndSubcategories, яка поверне скорочені версії об’єктів з файлу categories.json, де зберігається вся інформація про категорії та товари:

```
export async function getCategoriesAndSubcategories() {
  await fakeNetwork();

  const categoriesFullObjs = Object.values(products.categories.entities);

  const categoriesShortObjs = categoriesFullObjs.map((c) => {
    const subcategoriesIds = c.subcategories.ids;

    const subcategoriesFullObjs = Object.values(c.subcategories.entities);
    const subcategoriesShortObjs = subcategoriesFullObjs.map((subC) => ({
      name: subC.name,
      id: subC.id,
      imgAlt: subC.imgAlt,
    }));

    const subcategoryEntities = {};

    subcategoriesShortObjs.forEach((s) => {
      subcategoryEntities[s.id] = s;
    });

    return {
      ...c,
      subcategories: {
        ids: subcategoriesIds,
        entities: subcategoryEntities,
      },
    };
  });

  return categoriesShortObjs;
}
```

Рисунок 1.6.6 - Функція getCategoriesAndSubcategories.

Отже, до того моменту, як Breadcrumbs спробує зробити запит, інформація вже буде завантажуватись або зберігатися в глобальному стані, а отже, зайвих запитів зроблено не буде:

```
export default function BreadCrumbs() {
  const params = useParams();

  const [fetchedData, setFetchedData] = useState(null);

  const { data } = useGetCategoriesQuery();
  if (data && fetchedData === null) {
    setFetchedData(data);
  }
```

Рисунок 1.6.7 - Мережевий запит та зберігання відповіді у стані.

Маючи всю необхідну інформацію про посилання, можна зрозуміти, який з трьох варіантів навігаційних ланцюгів необхідно зробити і, безпосередньо, зробити та повернути сам елемент Breadcrumbs:

```
const links = [];

if (fetchedData && params.categoryId && params.subcategoryId) {
  const category = fetchedData.entities[params.categoryId];

  if (!category) return;

  links.push(createLink(category.id, category.name, `/${category.id}`));

  if (params.subcategoryId && params.productId) {
    const subcategory = category.subcategories.entities[params.subcategoryId];

    if (!subcategory) return;

    links.push(createLink(subcategory.id, subcategory.name, `/${category.id}/${subcategory.id}`));
  }
} else if (params.articleId) {
  links.push(createLink('news', 'Новини', '/news'));
}

return (
  <nav className={classNames(containerCls.container, breadcrumbsCls.nav)}>
    <ul className={breadcrumbsCls.list}>
      <li>
        <Link className={breadcrumbsCls.link} to="/" alt="Головна">
          Головна
        </Link>
      </li>
      {links}
    </ul>
  </nav>
);
```

Рисунок 1.6.8 - Створення елемента Breadcrumbs.

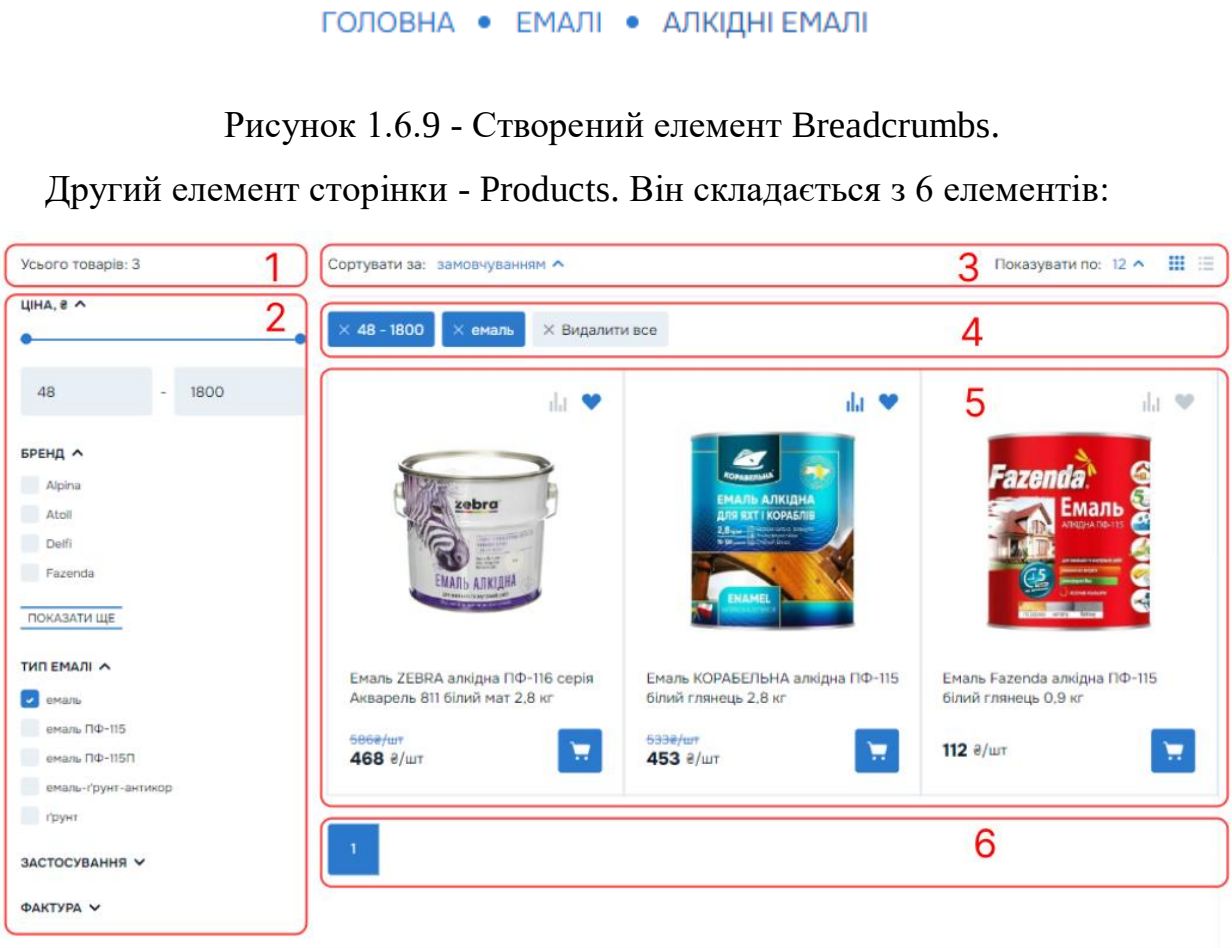


Рисунок 1.6.9 - Створений елемент Breadcrumbs.

Другий елемент сторінки - Products. Він складається з 6 елементів:

Рисунок 1.6.10 - Елементи Products.

1. Кількість товарів з урахуванням застосованих фільтрів
2. Блок фільтрів;
3. Панель сортування та відображення товарів;
4. Застосовані фільтри
5. Товари
6. Пагінація (нумерація сторінок товарів)

Елемент Products є батьківським елементом для всіх вище перерахованих, отже, він керує спільним станом для всіх цих елементів, імпортує їх та викликає, передаючи необхідні дані:

```
const [category, setCategory] = useState(null);
const [subcategoryFilters, setSubcategoryFilters] = useState(null);
const [productsAndInfo, setProductsAndInfo] = useState(null);
const [isFilterMenuOpen, setIsFilterMenuOpen] = useState(false);
const [windowWidth, setWindowWidth] = useState();
const [prevCategoryAndSubcategory, setPrevCategoryAndSubcategory] = useState(null);
```

Рисунок 1.6.11 - Ініціалізація змінних стану.

- category - інформація про категорію та підкатегорію;
- subcategoryFilters - список фільтрів та їх значення;
- productsAndInfo - об’єкт товарів та додаткова інформація (кількість сторінок товарів, загальна кількість товарів);
- isFilterMenuOpen - статус меню фільтрів (для мобільних пристроїв фільтри знаходяться в окремому меню);
- windowWidth - ширина екрану (використовується для делегування між меню фільтрів і просто елементом фільтрів);
- prevCategoryAndSubcategory - використовується для оновлення каталогу, якщо користувач з цієї ж сторінки відкриває каталог іншої підкатегорії;

Спочатку, елемент Products робить такий самий мережевий запит, як і Breadcrumbs, щоб отримати категорії:

```
const { data: fetchedCategories } = useGetCategoriesQuery();

if (fetchedCategories) {
  if (fetchedCategories.entities[categoryId] !== category) {
    setCategory(fetchedCategories.entities[categoryId]);
  }
}
```

Рисунок 1.6.12 - Мережевий запит для отримання категорії.

Він робиться один раз, незалежно від можливих переходів до каталогу інших категорій. Далі, необхідно зробити мережевий запит для отримання інформації про товари. GET запит для цього необхідно зробити за адресою /fakeAPI/getProducts/:categoryId/:subcategoryId. Так як адреса містить динамічні

параметри, необхідно зробити правильну строку адреси. Для отримання поточних ID категорії та підкатегорії використовується React Router хук useParams. Також, роблячи запит, необхідно додавати параметри пошуку (searchParams), якщо було застосовано фільтри і сортування. Для отримання поточних параметрів пошуку (якщо вони є), використовується React Router хук useSearchParams:

```
const fetchUrl = Array.from(searchParams).length ? `${categoryId}
  /${subcategoryId}?${searchParams.toString()}` : `${categoryId}/${subcategoryId}`;

const {
  data: fetchedProductsAndFilters,
  isLoading: isProductsLoading,
  isFetching: isProductsFetching,
  status: productsFetchingStatus,
} = useGetProductsQuery(fetchUrl);

if (fetchedProductsAndFilters && productsFetchingStatus === 'fulfilled') {
  if (!prevCategoryAndSubcategory
    || prevCategoryAndSubcategory.categoryId !== categoryId
    || prevCategoryAndSubcategory.subcategoryId !== subcategoryId
  ) {
    setPrevCategoryAndSubcategory({
      categoryId,
      subcategoryId,
    });
    setSubcategoryFilters(fetchedProductsAndFilters.subcategoryFilters);
  }

  if (productsAndInfo !== fetchedProductsAndFilters) {
    setProductsAndInfo(fetchedProductsAndFilters);
  }
} else if (productsFetchingStatus === 'rejected') {
  throw new Response(null, { status: 404, statusText: 'Not found' });
}
```

Рисунок 1.6.13 - Мережевий запит для отримання продуктів каталогу.

Тут також використовується хук, створений Redux Toolkit Query API за описаним раніше принципом. Крім даних запиту, хук також повертає інформацію про поточний стан запиту (змінні status, isLoading, is Fetching). Якщо категорії або підкатегорії не існує, ендпоїнт поверне помилку і значення змінної status буде ‘rejected’, що призведе до створення помилки 404 і відображення елемента Error, що дасть можливість користувачеві продовжити навігацію сайтом не роблячи

перезавантаження. Цей запит буде робитися кожен раз, коли змінюється підкатегорія каталогу, або якщо були застосовані фільтри / сортування. Маючи необхідну інформацію, можна викликати зазначені вище елементи.

Блок фільтрів (FilterBlock) отримує список фільтрів, а також стан, чи завантажуються зараз товари (щоб блокувати панель на час завантаження). Блок фільтрів складається з двох типів елементів: фільтр ціни (FilterPriceForm) і інші фільтри (FilterForm), які динамічно створюються в залежності від даних, які повернув сервер:

```
const FilterBlock = memo(({ subcategoryFilters, isFetching }) => {
  const filterElems = useMemo(() => {
    if (!subcategoryFilters) return;

    return Object.entries(subcategoryFilters).map(([key, value], i) => (
      <FilterForm
        key={key}
        name={key}
        values={value}
        initIsClosed={i > 1}
      />
    ));
  }, [subcategoryFilters]);

  return (
    <div className={classNames(
      filterCls.filterBlock,
      filterElems && isFetching && filterCls.filterBlock_inactive,
    )}
    >
      <FilterPriceForm />
      {filterElems || <ThreeDotsSpinnerBlock />}
    </div>
  );
});
```

Рисунок 1.6.14 - Елемент FilterBlock.

FilterPriceForm являє собою повзунок мінімальної і максимальної ціни і має наступний функціонал: drag`n`drop маніпулювання повзунками, підтримка сенсорних пристроїв, поля вводу як альтернатива повзункам, підтримка управління клавіатурою, параметри мають зберігатися при перезавантаженні сторінки або

переходу вперед / назад в межах каталогу, застосування фільтрів ціни для фільтрації товарів.

Спочатку необхідно налаштувати значення цін. Тут використовується багато параметрів, це: мінімальна і максимальна ціна товарів, які надіслав сервер, попередня мінімальна і максимальна ціна товарів, які надіслав сервер, загальна мінімальна і максимальна ціна, поточна мінімальна і максимальна ціна, ціна, яка введена в поля вводу, але ще не застосована як фільтр та інші. Така кількість параметрів ціни необхідно для забезпечення багатьох варіантів використання повзунка. Поки форма не використовується, тобто не було натискань на повзунки або вводу в поля вводу, поточні мінімальна і максимальна ціни завжди дорівнюють загальній мінімальній та максимальній ціні, при чому застосування інших фільтрів змінюють обидва ці значення. Якщо ж користувач змінив значення, то головне правило полягає в тому, що ніщо не повинно змінити це значення, тобто поточна мінімальна і максимальна ціна більше не змінюються автоматично, а загальна мінімальна і максимальна ціна залежать від поточною, наприклад, якщо мінімальна ціна товарів, які надіслав сервер вища, за введену користувачем, то загальна мінімальна ціна буде такою, яку ввів користувач. Також, якщо користувач робить перезавантаження сторінки, або переходить в каталог за посиланням, яке містить параметри пошуку з фільтрами ціни, то поточні ціни мають бути такими, як в параметрах пошуку. Це ж саме стосується переходу назад / вперед в межах всього каталогу:

```
function setupPrices() {
  if (prevLoaderMinPrice !== fetchedMinPrice) setPrevLoaderMinPrice(fetchedMinPrice);
  if (prevLoaderMaxPrice !== fetchedMaxPrice) setPrevLoaderMaxPrice(fetchedMaxPrice);

  if (!formWasInteracted) {
    if (searchParams.has('minPrice')) {
      const searchParamsMinPrice = Number(searchParams.get('minPrice'));
      const searchParamsMaxPrice = Number(searchParams.get('maxPrice'));

      if (fetchedMinPrice > searchParamsMinPrice && prevLoaderMinPrice !== fetchedMinPrice) {
        setTotalMinPrice(searchParamsMinPrice);
      } else if (fetchedMinPrice < searchParamsMinPrice
        && prevLoaderMinPrice !== fetchedMinPrice) {
        setTotalMinPrice(fetchedMinPrice);
      }
      if (fetchedMaxPrice < searchParamsMaxPrice && prevLoaderMaxPrice !== fetchedMaxPrice) {
        setTotalMaxPrice(searchParamsMaxPrice);
      } else if (fetchedMaxPrice > searchParamsMaxPrice
        && prevLoaderMaxPrice !== fetchedMaxPrice) {
        setTotalMaxPrice(fetchedMaxPrice);
      }

      if (searchParamsMinPrice !== currentMinPrice) {
        setCurrentMinPrice(searchParamsMinPrice);
        setMinInputValue(searchParamsMinPrice);
      }
      if (searchParamsMaxPrice !== currentMaxPrice) {
        setCurrentMaxPrice(searchParamsMaxPrice);
        setMaxInputValue(searchParamsMaxPrice);
      }
```

Рисунок 1.6.15 - Налаштування цін форми до взаємодії з користувачем.

Як було вказано раніше, однією з функцій є реалізація drag`n`drop для повзунка та підтримка сенсорних пристроїв. Необхідно відслідковувати натискання на повзунок, рух курсора або пальця, відтискання, тобто, три дії. Я буду використовувати pointerEvents події, так як вони працюють як і з сенсорними пристроями, так і з комп’ютерною мишею. При натисканні, відмічається, що з формою була взаємодія, визначається тип кнопки повзунка - мінімальна / максимальна, мінімальне і максимальне положення кнопки на лінії повзунка (наприклад, мінімальна кнопка не може виходити за межі лінії або бути над / правіше максимальної кнопки), визначаються поточні координати курсора та кнопки, які будуть необхідні для подальших розрахунків та додаються слухач події “pointermove”, тобто, рух миші / пальця по сенсорному екрану та “pointerup”, тобто, відтискання клавіши миші пальця:

```
function roundButtonOnDown(e) {
  e.preventDefault();

  setFormWasInteracted(true);

  const { buttonType } = e.target.dataset;
  const button = buttonType === 'min' ? minButtonRef.current : maxButtonRef.current;

  const roundButtonWidthInPercent = getRoundButtonPercentWidth();

  let minLeft;
  let maxLeft;

  if (buttonType === 'min') {
    minLeft = 0;
    maxLeft = maxButtonLeft - roundButtonWidthInPercent;
  } else {
    minLeft = minButtonLeft + roundButtonWidthInPercent;
    maxLeft = 100 - roundButtonWidthInPercent;
  }

  const cursorCoordXOnDown = e.clientX;
  const buttonCoordLeft = button.getBoundingClientRect().left;
  const cursorDiff = cursorCoordXOnDown - buttonCoordLeft;

  button.setPointerCapture(e.pointerId);
```

Рисунок 1.6.16 - Функція слухача подій “pointerdown”, тобто, натискання.

При русі кнопки повзунка, спрацьовує функція слухача, яка обчислює координати лінії повзунка, різницю між поточним положенням курсора та попереднім, обчислюється координати нового положення кнопки, враховуючи крайнє ліве і крайнє праве доступне положення. Після чого, маючи координати нового положення кнопки, роблять розрахунки поточної мінімальної / максимальної ціни:

```
function roundButtonOnMove(onMoveEvent) {
  const mainLineLeftCoord = mainLineRef.current.getBoundingClientRect().left;
  const cursorCoordXOnMove = onMoveEvent.clientX;
  const buttonLeftInPx = cursorCoordXOnMove - mainLineLeftCoord - cursorDiff;

  let buttonLeftInPercent = (buttonLeftInPx / getMainLinePxWidth()) * 100;

  if (buttonLeftInPercent < minLeft) {
    buttonLeftInPercent = minLeft;
  } else if (buttonLeftInPercent > maxLeft) {
    buttonLeftInPercent = maxLeft;
  }

  buttonLeftInPercent = Number(buttonLeftInPercent.toFixed(2));

  calcCurrentMinMaxPrice(buttonType, buttonLeftInPercent);
}
```

Рисунок 1.6.17 - Функція слухача подій “pointermove”, тобто, руху миші / пальця.

При відтисканні, спрацьовує функція слухача події “pointerup”, яка видаляє слухачі події “pointermove” та “pointerup”, і змінює параметри пошуку, що викликає новий GET-запит, щоб отримати актуальні відфільтровані товари.

FilterForm отримує назву фільтра, можливі значення та статус, чи розгорнуте меню цього фільтра. Цей елемент створює чекбокс-елементи FilterCheckbox, які при натисканні генерують подію “submit”, яку перехоплює елемент FilterForm та змінює поточні пошукові параметри, що призводить до GET-запиту відфільтрованих товарів. Також, чекбокс візуально змінюється в залежності від того, обраний він чи ні:

```
function onSubmitHandler(e) {
  e.preventDefault();

  const URLParams = new URLSearchParams();
  params.forEach((p) => URLParams.append(name, p));

  Array.from(searchParams).forEach(([key, value]) => {
    if (key !== name) {
      URLParams.append(key, value);
    }
  });

  setSearchParams(URLParams);
}

const checkboxes = values.slice().sort().map((value, i) => (
  <li
    key={value}
    hidden={isAdditionalAccordionNeeded && isAdditionalClosed && i > 4}
  >
    <FilterCheckbox
      value={value}
      isChecked={params.includes(value)}
      updateParams={setParams}
    />
  </li>
));
```

Рисунок 1.6.18 - Функція слухача події “submit” та функція створення чекбоксів.

FilterForm слідкує за поточними пошуковими параметри, щоб у разі перезавантаження сторінки, переходу вперед / назад, або видалення всіх фільтрів з панелі застосованих фільтрів, стан форми та чекбоксів були відповідними. Також тут реалізований механізм згортання / розгортання всього меню фільтра, а також згортання / розгортання значень, якщо їх більше або дорівнює 8.

Наступним великим елементом є панель сортування та відображення товарів. Вона складається з двох елементів-селектів Select та блоку з кнопками зовнішнього вигляду карток товарів.

Елемент Select приймає наступні аргументи: назву, варіанти відповідей, варіант відповіді, який є обраним за замовчуванням та назву пошукового параметра, який він буде додавати / змінювати. Його функціонал є наступним: при кліку по селектору, відкривається меню варіантів відповіді. При кліку по варіанту відповіді (також підтримуються події клавіатури), цей варіант стає активним, селектор закривається і вносяться зміни до параметрів пошуку, що викликає GET-запит відфільтрованих товарів:

```
function openButtonOnClick() {
  setIsOpen(!isOpen);
  setFocusedOptionIndex(-1);

  if (!isOpen) {
    setTimeout(() => {
      listRef.current.focus();
    }, 0);
  }
}

function optionListOnClick(e) {
  const option = e.target.closest('[role="option"]');
  if (!option) return;

  const newOptionId = option.id;

  setSelectedOptionId(newOptionId);
  setIsOpen(false);

  searchParams.set(searchParamName, newOptionId);
  setSearchParams(searchParams);

  openButtonRef.current.focus();
}
```

Рисунок 1.6.19 - Функція кліку по селекту та кліку по варіанту відповіді.

Як і попередні елементи, Select відслідковує пошукові параметри, щоб у разі перезавантаження сторінки, або інших варіантів зміни пошукових параметрів, селектор мав актуальний стан.

Блок кнопок зовнішнього вигляду є частиною основного елементу Products. Є два варіанту зовнішнього вигляду карток продуктів - короткі (за замовчуванням) та довгі. При натисканні на ці кнопки, змінюється значення змінної стану isProductCardsShort на один з зазначених вище варіантів, а також це значення зберігається в localStorage браузера, щоб зберегти налаштування користувача. Значення змінної стану isProductCardsShort буде передаватися елементам карток для створення відповідного зовнішнього вигляду.

```
function shortCardBtnOnClick() {
  setIsProductCardsShort(true);
  localStorage.setItem('productCardAppearance', 'short');
}

function longCardBtnOnClick() {
  setIsProductCardsShort(false);
  localStorage.setItem('productCardAppearance', 'long');
}
```

Рисунок 1.6.20 - Функції кліку по кнопкам зовнішнього вигляду.

Наступним елементом є панель застосованих фільтрів - AppliedFiltersBlock. Її мета полягає у відображенні застосованих фільтрів та можливості видалення як окремих фільтрів, так і всіх разом. Елемент відслідковує пошукові параметри, тому на кожну зміну він отримає об’єкт пошукових параметрів (searchParams) та відфільтровує деякі значення: значення сортування, так як панель застосованих фільтрів не має впливати на сортування; мінімальна і максимальна ціна, вони є окремими фільтрами, але на їх основі буде зроблена одна кнопка, яка видаляє зразу обидва фільтра; номер сторінки каталогу, адже це також не є фільтром:

```
const filteredSearchParams = Array.from(searchParams).filter(([name]) => (
  name !== 'page' && name !== 'perView' && name !== 'sortBy'
  && name !== 'minPrice' && name !== 'maxPrice'
));

const isThereAnyFilters = filteredSearchParams.length || searchParams.has('minPrice');

let priceButton;

if (searchParams.has('minPrice')) {
  const minPrice = searchParams.get('minPrice');
  const maxPrice = searchParams.get('maxPrice');

  priceButton = (
    <button
      type="button"
      className={appliedFiltersCls.button}
      aria-label="Видалити фільтр цін"
      onClick={() => {
        searchParams.delete('minPrice');
        searchParams.delete('maxPrice');
        setSearchParams(searchParams);
      }}
    >
      <Cross className={appliedFiltersCls.icon} />
      {`${minPrice} - ${maxPrice}`}
    </button>
  );
}
```

Рисунок 1.6.21 - Функція фільтрація параметрів пошуку та створення кнопки видалення фільтрів ціни.

Таким чином, при натисканні на кнопку видалення фільтрів, параметри пошуку змінюються, що викликає новий GET-запит, щоб отримати відфільтровані товари. А при зміні пошукових параметрів іншими елементами (FilterForm, FilterPriceForm тощо), елемент оновлює свій стан і, відповідно, кнопки.

Наступним, та одним з найважливіших елементів є картка товару. Головний елемент Products, отримавши з серверу об’єкти відфільтрованих та відсортованих товарів для певної сторінки каталогу, створює елементи карток товарів - ProductCard:

```
const productCards = useMemo(() => (
  filteredAndSortedProducts?.map((p) => (
    <ProductCard
      key={p.id}
      name={p.name}
      categoryId={categoryId}
      subcategoryId={subcategoryId}
      productId={p.id}
      price={p.price}
      oldPrice={p.oldPrice}
      isShortCard={windowWidth <= 576 ? true : isProductCardsShort}
    />
  ))
), [filteredAndSortedProducts, categoryId,
  subcategoryId, windowWidth, isProductCardsShort]);
```

Рисунок 1.6.22 - Функція створення елементів ProductCard.

Елемент ProductCard приймає у вигляді аргументів інформація про товар (назва, ціна тощо), а також налаштування зовнішнього вигляду (короткий / довгий), про які розповідалось раніше. Під час створення (рендеру), елемент одразу починає завантаження зображення товару, і поки воно відбувається, показує анімацію завантаження. Також, елемент робить три GET-запити, щоб отримати інформацію, чи є цей елемент у списку бажань, кошику та списку для порівняння, використовуючи Redux Toolkit Query API. Замість того, щоб зберігати повну інформацію про товар, щоб знати, чи є він у списку бажань, кошику або списку порівняння, зберігається лише інформація про id категорії, підкатегорії та товару. На перший погляд, може здатися, що, якщо кожен товар робить 3 GET-запити, то в сумі їх відбудеться: к-сть товарів помножити на три. Але, сервер повертає інформацію не про конкретний товар, а масив з усіма товарами, які містяться у списку бажань / кошику / списку порівняння, тобто всі товари роблять однакові запити, а потім перевіряють, чи є їхній товар у переліку, який повернув сервер. А використовуючи Redux Toolkit Query, дублюючих запитів не буде, тому у підсумку, загалом відбувається три запити:

```
const { data: wishlistFetcherData } = useGetWishlistIdsQuery();

if (wishlistFetcherData) {
  const isInWishlistIdsList = !!wishlistFetcherData.find(([cId, subcId, pId]) => (
    cId === categoryId && subcId === subcategoryId && pId === productId
  ));

  if (productInWishlist !== isInWishlistIdsList) {
    setProductInWishlist(isInWishlistIdsList);
  }
}

const { data: cartFetcherData } = useGetCartIdsQuery();

if (cartFetcherData) {
  const isInCartIdsList = !!cartFetcherData.find((cId) => (cId.categoryId === categoryId
    && cId.subcategoryId === subcategoryId && cId.productId === productId));

  if (productInCart !== isInCartIdsList) {
    setProductInCart(isInCartIdsList);
  }
}

const { data: compareFetcherData } = useGetCompareIdsQuery();

if (compareFetcherData) {
  const isInCompareIdsList = !!compareFetcherData.find(([cId, subcId, pId]) => (
    cId === categoryId && subcId === subcategoryId && pId === productId
  ));

  if (productInCompare !== isInCompareIdsList) {
    setProductInCompare(isInCompareIdsList);
  }
}
```

Рисунок 1.6.23 - GET-запити елементу ProductCard.

Взагалі, функціонал елемента ProductCard полягає у демонстрації товару та можливості додати / видалити його зі списку бажань / кошику / списку порівняння. Для цього, по кліку на відповідні кнопки, викликається необхідний хук, згенерований Redux Toolkit Query API, з аргументом, у вигляді JSON-строки, яка містить id категорії, підкатегорії та товару. Кожен ендпоїнт Redux Toolkit Query API отримуючи цю інформацію, робить POST-запит до сервера, інвалідуючи відповідний тег. Інвалідація тега призводить до автоматичних повторних GET-запитів, які повертають інформацію, яка перебуває під цим тегом. Наприклад, GET-запит id товарів, які знаходяться у списку бажань, відбувається під тегом ‘wishlistIds’, тому, POST-запит, який видаляє id товару зі списку бажань, інвалідує

тег ‘wishlistIds’, що призводить до автоматичного повторного GET-запиту, щоб отримати актуальну інформацію. Схема виглядає приблизно так:

POST-запит → обробка сервером → відповідь сервера → інвалідація тега → автоматичний GET-запит → обробка сервером → відповідь сервера

Так як надсилання запиту, його обробка, та надсилання відповіді займає час, який при поганому зв’язку може складати десятки секунд, тоді інтерфейс, який показує стан товару, буде мати велику затримку та відчуватися зависаючим. Тому, у проекті використовуються оптимістичні (optimistic) оновлення стану. Їх суть полягає у тому, що коли робиться певний POST-запит, його тіло (body) перехоплюється функцією, яка робить оптимістичне оновлення, і на основі цього тіла, одразу і без затримки оновлюється глобальний стан. Після чого відбувається інвалідація тега і автоматичні GET-запити, які без затримки повертають оновлену інформацію з глобального стану. Тим часом, після отримання відповіді з серверу на POST-запит, відбувається реальний GET-запит на сервер, і у разі успішної відповіді глобальний стан оновлюється вже справжньою актуальною інформацією з сервера. Якщо ж виникає помилка, то глобальний стан скидається до внесення змін і відповідно елементи, які використовують цей стан, також відкатуються до стану до внесення вниз. Тому, користувач може побачити, що операція була неуспішна.

```
function sendMutationRequest(state, addMutationFunc, deleteMutationFunc) {
  const body = JSON.stringify([categoryId, subcategoryId, productId]);

  if (!state) {
    addMutationFunc(body);
  } else {
    deleteMutationFunc(body);
  }
}

const [addToWishlistRequest] = useAddWishlistIdMutation();
const [deleteFromWishlistRequest] = useDeleteWishlistIdMutation();

function wishlistButtonOnClick() {
  sendMutationRequest(productInWishlist, addToWishlistRequest, deleteFromWishlistRequest);

  if (wishlistMutationFunc) {
    wishlistMutationFunc(categoryId, subcategoryId, productId);
  }
}
```

Рисунок 1.6.24 - Функція кліку по кнопці, яка додає / видаляє товар зі списку бажань.

```
deleteWishlistId: builder.mutation(
  createPatchDeleteMutation(
    (body) => ({
      url: '/wishlistIds',
      method: 'DELETE',
      body,
    }),
    [{ type: 'wishlistIds', id: 'LIST' }, 'wishlistProducts'],
    (args) => (
      queryAPI.util.updateQueryData('getWishlistIds', undefined, (draft) => {
        const [categoryId, subcategoryId, productId] = JSON.parse(args);
        const index = draft.findIndex(([draftCaId, draftSId, draftPId]) => (
          draftCaId === categoryId
          && draftSId === subcategoryId
          && draftPId === productId
        ));
        draft.splice(index, 1);
      })
    ),
  ),
),
```

Рисунок 1.6.25 - Ендпоїнт Redux Toolkit Query API, який відповідає за видалення товару зі списку бажань і робить POST-запити до сервера та оптимістичні оновлення.

І останнім основним елементом є пагінація (нумерування сторінок) - PaginationBlock. У якості аргументів він приймає загальну кількість сторінок та елемент, до якого треба прокручувати сторінку браузера при зміні сторінки каталогу (в моєму випадку, при зміні сторінки каталогу, браузер прокручує сторінку до заголовку категорії). Його принцип роботи схожий з попередніми компонентами: елемент створює певну кількість кнопок зміни сторінки в залежності від загальної кількості сторінок, і при натисканні, змінює параметри пошуку, що призводить до GET-запиту товарів для певного номера сторінки каталогу. Він також відслідковує значення сторінки у параметрах пошуку, щоб при перезавантаженні або інших змінах номеру сторінки, його стан був актуальним:

```
const buttons = [];

for (let i = firstMainButtonId; i <= lastMainButtonId; i += 1) {
  buttons.push((
    <Fragment key={i}>
      <button
        type="submit"
        id={i}
        onClick={() => setCurrentPageNum(i)}
        className={classNames(
          paginationCls.paginationButton,
          i === currentPageNum && paginationCls.paginationButton_active,
        )}
        style={{
          borderRadius: i === firstMainButtonId ? '4px 0 0 4px'
            : i === lastMainButtonId ? '0px 4px 4px 0' : '',
        }}
        aria-label={`Перейти на сторінку товарів ${i}`}
      >
        {i}
      </button>
      {i !== lastMainButtonId && (
        <span className={paginationCls.greyLine} />
      )}
    </Fragment>
  ));
}
```

Рисунок 1.6.26 - Функція створення кнопок для зміни номера сторінки.

Також, якщо загальна кількість сторінок більше за 9, тоді створюються додаткові кнопки:

Рисунок 1.6.27 - Блок пагінації з додатковими кнопками.

Отже, всі дочірні елементи компонента Products маніпулюють параметрами пошуку. Products, в свою чергу, відслідковує зміни цих параметрів, і на кожну зміну він робить GET-запити, інформацію з яких він і його дочірні елементи використовують для відображення актуальної кількості товару, фільтрів, карток товарів, кількості сторінок.

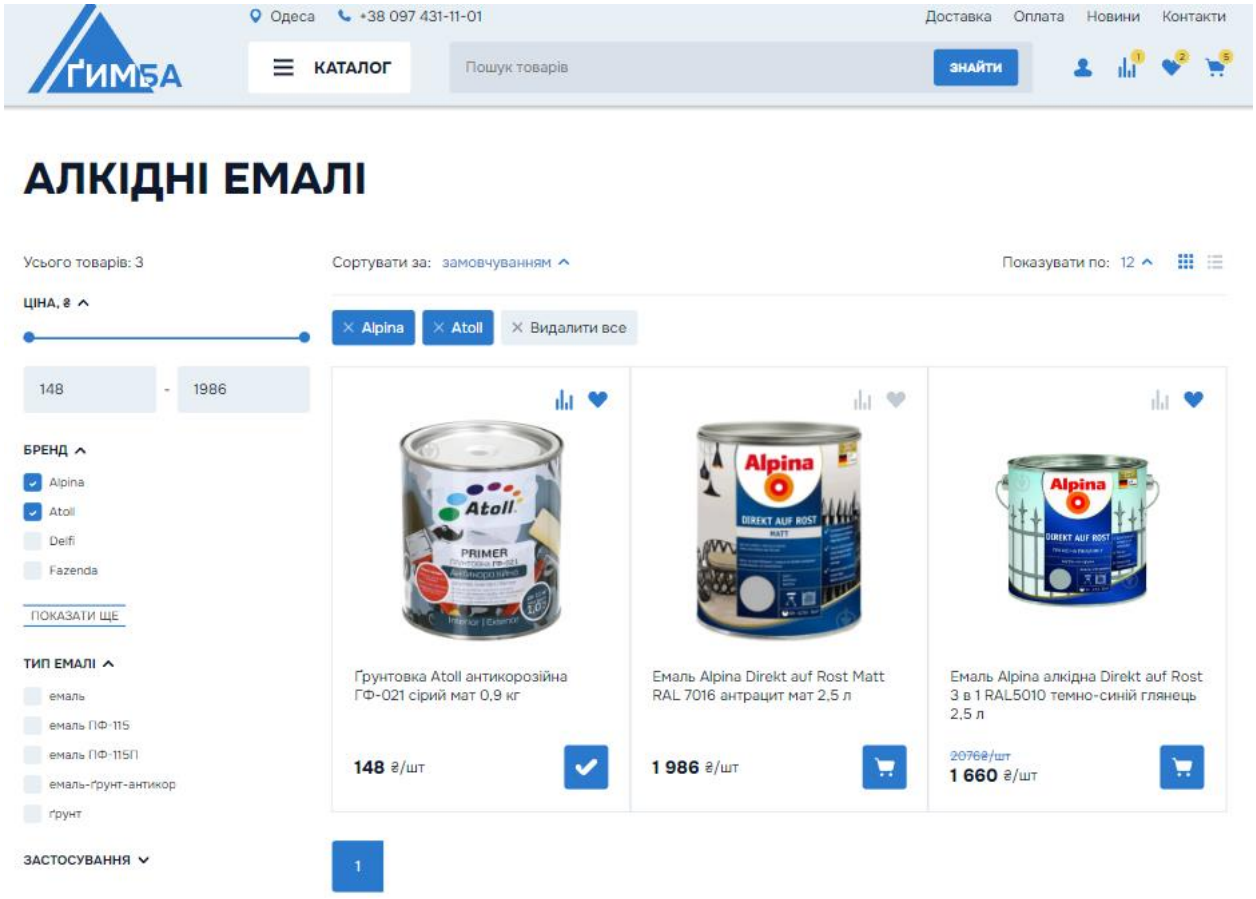


Рисунок 1.6.28 - Підсумковий вигляд сторінки.

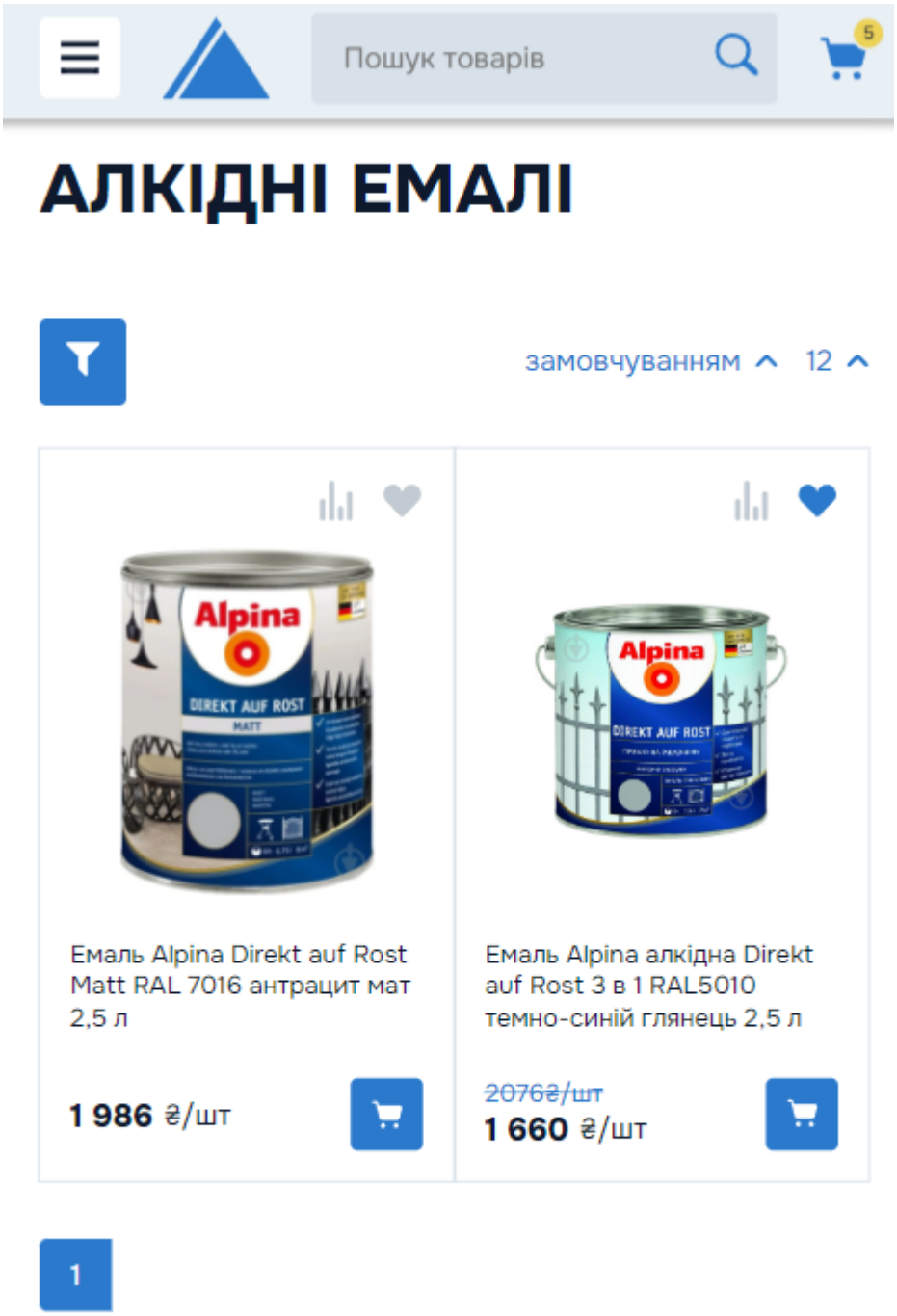


Рисунок 1.6.29 - Підсумковий вигляд мобільної версії сторінки.

2. Охорона праці

Вступ

Охорона праці є ключовим елементом для забезпечення безпечних та здорових умов праці на будь-якому підприємстві, незалежно від його сфери діяльності. В сучасному виробництві вона виконує декілька важливих функцій. По-перше, охорона праці сприяє збереженню здоров'я та життя працівників через створення безпечних умов праці, що мінімізує ризики для їхнього здоров'я. По-друге, вона допомагає підвищити продуктивність праці, оскільки здорові працівники працюють ефективніше. По-третє, впровадження заходів з охорони праці дозволяє знизити витрати на компенсації, лікування, а також на втрати через тимчасову непрацездатність працівників.

Розробка заходів з охорони праці є необхідною для попередження нещасних випадків, систематичного аналізу і усунення потенційних небезпек, що дозволяє уникнути травм і аварій. Також вона необхідна для профілактики професійних захворювань, оскільки включає контроль над шкідливими факторами, які можуть спричинити такі захворювання. Забезпечення комфортних умов праці, включаючи відповідний мікроклімат, освітлення, шумоізоляцію та інші параметри робочого місця, також є важливою частиною заходів з охорони праці.

Умови праці на робочому місці веб-розробника мають відповідати вимогам безпеки праці, у відповідності з нормативно-правовим актом з охорони праці (НПАОП 0.00-7.15- 18) «Вимоги щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями», а також НПАОП 0.00-1.28-10 "Правила охорони праці під час експлуатації електронно-обчислювальних машин", затверджені наказом Державного комітету України з промислової безпеки, охорони праці та гірничого нагляду від 26.03.2010 № 65.

Розробка веб-сайту для комерційного підприємства виконується веб-розробником, робочим місцем якого є персональний комп’ютер в умовах офісу або домашніх

умовах. Нижче буде розглянуто умови праці та забезпечення безпеки праці при виконанні основних видів робіт на робочому місці веб-розробника.

2.1 Аналіз шкідливих та небезпечних факторів виробничого середовища

Шкідливими факторами під час праці веб-розробника є: підвищена або знижена температура повітря робочої зони, підвищений рівень шуму на робочому місці, підвищена або знижена вологість повітря, підвищена або знижена рухливість повітря, підвищена або знижена іонізація повітря, підвищений рівень електромагнітних випромінювань, відсутність або недостача природного освітлення, недостатня освітленість робочої зони, підвищена яскравість світла, прямий та відбитий відблиск, підвищена пульсація світлового потоку, нервово-психічні перевантаження (перенапруга аналізаторів, монотонність праці, емоційні перевантаження).

2.2 Вимоги до приміщення

Площа приміщення для роботи з комп'ютерами повинна становити не менше 6 кв.м на одного працівника. Це визначено в ДСанПіН 3.3.2.007-98 "Державні санітарні норми та правила роботи з відеодисплейними терміналами електронно-обчислювальних машин". Рівень шуму на робочих місцях операторів ПК не повинен перевищувати 50 дБ, а вібрація має бути зведена до мінімуму, відповідно до ДСанПіН 3.3.2.007-98. Робочий стіл повинен мати регульовану висоту, а монітор слід розташовувати на відстані 50-70 см від очей користувача. Відповідно до ДБН В.2.5-23:2010 "Електроустановки будівель. Основні положення", всі електроустановки в приміщеннях для роботи з комп'ютерами повинні відповідати вимогам електробезпеки. Електропроводка має бути захищеною, а обладнання заземленим.

2.3 Освітлення приміщення

Згідно з ДБН В.2.5-28:2006 "Природне і штучне освітлення", в приміщеннях для роботи з ПК має бути забезпечене комбіноване освітлення, що складається з природного та штучного освітлення. Коефіцієнт природного освітлення (КПО) повинен становити не менше 1,5%. Це означає, що рівень природного освітлення повинен бути достатнім для нормальної роботи без надмірного навантаження на зір.

Штучне освітлення має бути організоване таким чином, щоб забезпечити рівень освітленості робочих поверхонь від 300 до 500 люкс. Загальне освітлення приміщення забезпечується за допомогою світильників, рівномірно розташованих по всій площі приміщення, а місцеве освітлення – за допомогою настільних ламп або інших джерел світла, що забезпечують додаткове освітлення робочої зони.

Відповідно до ДСанПіН 3.3.2.007-98 "Державні санітарні норми та правила роботи з відеодисплейними терміналами електронно-обчислювальних машин", рівень освітленості на робочих місцях операторів ПК повинен відповідати наступним значенням:

Зона освітлення	**Рекомендована освітленість (люкс)**
Основне робоче місце	300-500
Зона для документів	500
Прохідні зони і коридори	150-200

Таблиця 2.1 - Рекомендовані рівні освітленості

Освітлення не повинно створювати відблисків на екранах моніторів, тому джерела світла слід розташовувати таким чином, щоб прямі промені не потрапляли на екран. Для цього використовуються світильники з розсіяним або відбивним світлом, а також екрани або шторки на вікнах для регулювання інтенсивності природного світла.

Джерела штучного світла повинні мати спектр, максимально наближений до природного денного світла, з колірною температурою 4000-5000 К.

Також важливо враховувати розташування робочих місць відносно джерел світла. Робочі місця повинні бути розташовані перпендикулярно до вікон, щоб уникнути відблисків, і на відстані не менше 1,5 м від стін з вікнами для забезпечення рівномірного освітлення.

2.4 Вентиляція та мікроклімат

Вентиляція та мікроклімат приміщень для операторів ПК мають відповідати вимогам ДСанПіН 3.3.2.007-98, забезпечуючи температурний режим від 18 до 24°С і відносну вологість повітря від 40 до 60%. Повітрообмін має гарантувати видалення шкідливих речовин та забезпечення свіжого повітря.

Пора року	Категорія робіт	Температура повітря, С, не більше	Відносна вологість повітря %	Швидкість руху повітря, м/с
Холодна	Легка-1а	22-24	40-60	0,1
	Легка-1б	21-23	40-60	0,1
Тепла	Легка-1а	23-25	40-60	0,1
	Легка-1б	22-24	40-60	0,1

Таблиця 2.2 - Норми мікроклімату для приміщень з ВДТ ЕОМ та ПЕМ

2.5 Пожежна безпека

Пожежна безпека — це стан захищеності особистості, суспільства та держави від пожеж, що забезпечується комплексом організаційних і технічних заходів, направлених на запобігання пожежам, а також захист людей, матеріальних

цінностей та довкілля у випадку їх виникнення. Основни вимогами до пожежної безпеки є: евакуаційні шляхи та виходи - відповідно до ДБН В.1.1-7-2016, евакуаційні шляхи та виходи повинні бути розташовані таким чином, щоб забезпечити швидке та безпечне залишення приміщення у разі пожежі. Евакуаційні виходи мають бути позначені світловими знаками та забезпечені вільним доступом. Системи пожежної сигналізації та оповіщення - згідно з ДБН В.2.5-56:2014 "Системи протипожежного захисту", у приміщеннях для роботи з ПК необхідно встановлювати системи пожежної сигналізації, які автоматично виявляють ознаки пожежі (дим, високу температуру) та подають сигнал тривоги. Системи оповіщення мають інформувати працівників про необхідність евакуації. Засоби пожежогасіння - приміщення повинні бути обладнані первинними засобами пожежогасіння, такими як вогнегасники, пожежні крани, пожежні ковдри. Згідно з ДБН В.2.5-56:2014, кількість і тип вогнегасників залежать від площі приміщення та категорії пожежної небезпеки. Наприклад, на кожні 100 кв.м площі приміщення має бути встановлено один вогнегасник вуглекислотний або порошковий.

2.6 Висновки

Важливість дотримання правил охорони праці полягає у зниженні ризику виробничих травм і професійних захворювань, підвищенні продуктивності праці та створенні комфортних і безпечних умов праці. Дотримання вимог нормативно-правових актів, регулярне проведення інструктажів та навчань, забезпечення належного мікроклімату, освітлення, пожежної безпеки та ергономічних умов на робочому місці є ключовими аспектами ефективної системи охорони праці, що сприяє збереженню здоров'я працівників і забезпеченню стабільного розвитку підприємства.

ВИСНОВКИ

В даному дипломному проекті було створено веб-сайт інтернет-магазину Gymba, використовуючи сучасні технології розробки, такі як: React, React Router, Redux, React Redux, Redux Toolkit. Сайт має наступний функціонал:

- Зручна навігація;
- Пошук товарів, категорій, новин по сайту;
- Розподіл товарів по категоріям та підкатегоріям;
- Фільтрація та сортування товарів;
- Демонстрація зображень, опису, характеристик товарів;
- Додавання та видалення товарів зі списку бажань;
- Додавання та видалення товарів та категорій зі сторінки порівняння. Можливість порівняння характеристик товарів;
- Додавання та видалення товарів із кошику, редагування кількості товарів;

З точки зору підтримки та розвитку було розроблено:

- Адаптивний дизайн, який підтримується великою кількістю пристроїв.
- Lazy-завантаження компонентів, що прискорює швидкість завантаження сторінки;
- Гарна загальна швидкість роботи сайту;
- SEO-оптимізація;
- Підтримка доступності (accessibility / a11y);
- Зручність розширення;
- Автоматична генерація сторінок для категорій, підкатегорій, товарів;
- Великий функціонал створення компонентів для статей та новин;

Сайт відповідає сучасним вимогам та має гарні показники у Google Lighthouse.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. DOM Living Standart [Електронний ресурс] Режим доступу до ресурсу: https://dom.spec.whatwg.org/ (дата звернення 20.05.2024)

2. HTML Living Standart [Електронний ресурс] Режим доступу до ресурсу: https://html.spec.whatwg.org/ (дата звернення 20.05.2024)

3. Compatibility Living Standart [Електронний ресурс] Режим доступу до ресурсу: https://compat.spec.whatwg.org/ (дата звернення 20.05.2024)

4. React Reference [Електронний ресурс] Режим доступу до ресурсу: https://react.dev/reference/react (дата звернення 20.05.2024)

5. React Router Documentation [Електронний ресурс] Режим доступу до ресурсу:
https://reactrouter.com/en/main (дата звернення 20.05.2024)

6. Redux Documentation [Електронний ресурс] Режим доступу до ресурсу:
https://react-redux.js.org/introduction/getting-started (дата звернення 20.05.2024)

7. Redux Toolkit Documentation [Електронний ресурс] Режим доступу до ресурсу:
https://redux-toolkit.js.org/introduction/getting-started (дата звернення 20.05.2024)

8. React Redux Documentation [Електронний ресурс] Режим доступу до ресурсу:
https://react-redux.js.org/introduction/getting-started (дата звернення 20.05.2024)

9. Sass: Documentation [Електронний ресурс] Режим доступу до ресурсу:
https://sass-lang.com/documentation/ (дата звернення 20.05.2024)

10. Git - Reference [Електронний ресурс] Режим доступу до ресурсу:
https://git-scm.com/docs (дата звернення 20.05.2024)

11. Webpack Documentation [Електронний ресурс] Режим доступу до ресурсу: https://webpack.js.org/concepts/ (дата звернення 20.05.2024)
12. NPM Documentation [Електронний ресурс] Режим доступу до ресурсу: https://docs.npmjs.com/ (дата звернення 20.05.2024)
13. MDN Web Docks [Електронний ресурс] Режим доступу до ресурсу: https://developer.mozilla.org/en-US/ (дата звернення 20.05.2024)
14. Scott Chacon, Ben Straub. ProGit. apress 2021
15. ДСанПіН 3.3.6.042-99 «Державні санітарні норми мікроклімату виробничих приміщень».
16 .ДСанПіН 2.3.6.037-99 «Державні санітарні норми виробничого шуму, ультразвуку та інфразвуку»

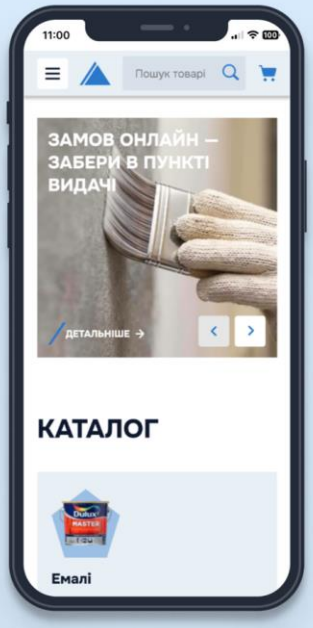
ДОДАТОК А

Слайди мультимедійної презентації

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Створення Інтернет-ресурсу для комерційного підприємства з елементами 3D-дизайну

Виконав: Грабовий О.В., 2БКС-28
Керівник: Іванова Л.В.



СТРУКТУРНА СХЕМА САЙТУ

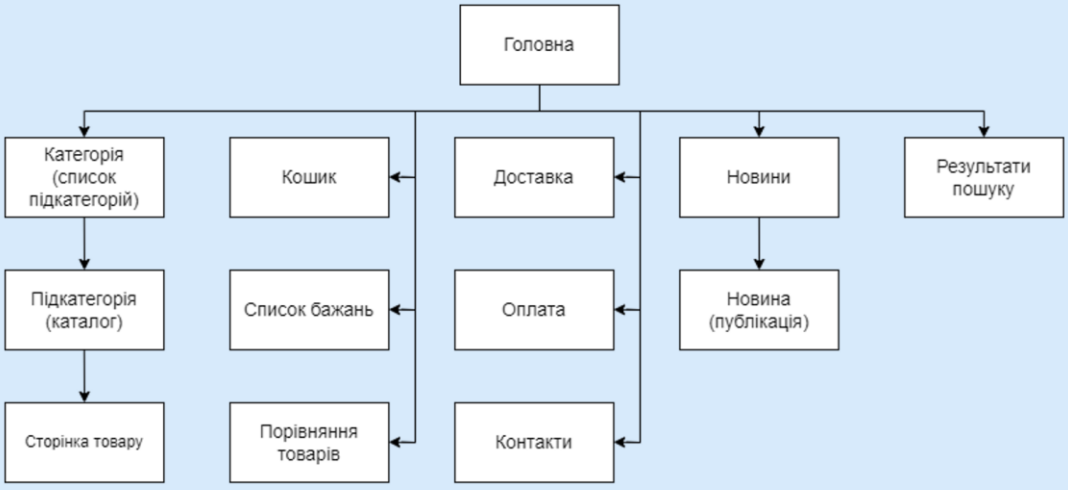
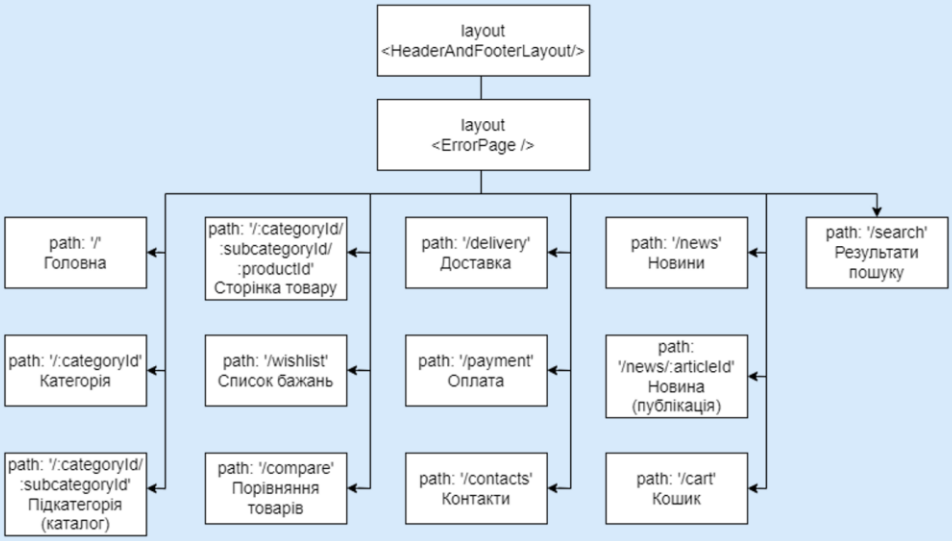
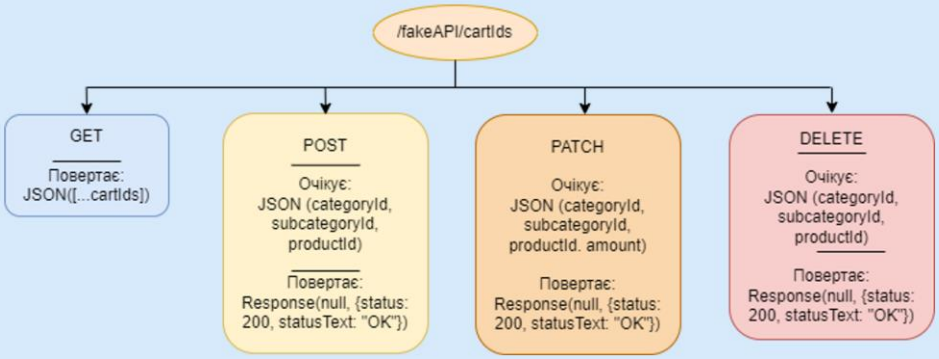


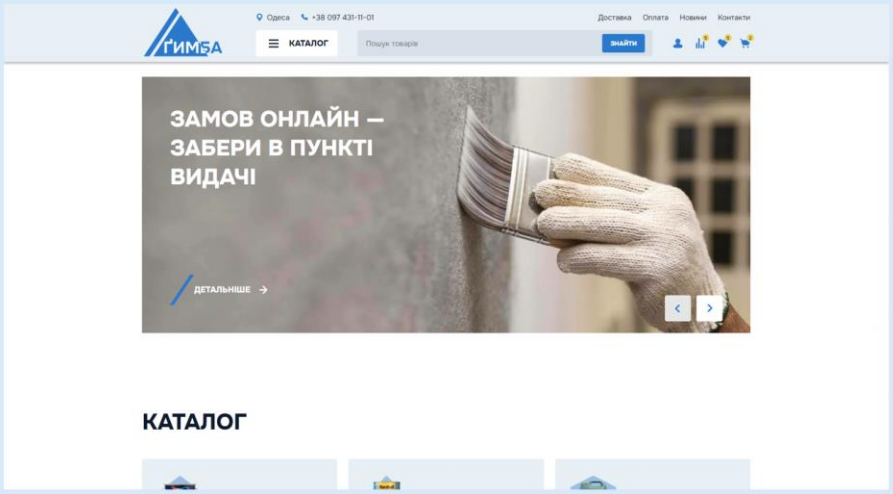
СХЕМА МАРШРУТИЗАЦІЇ САЙТУ

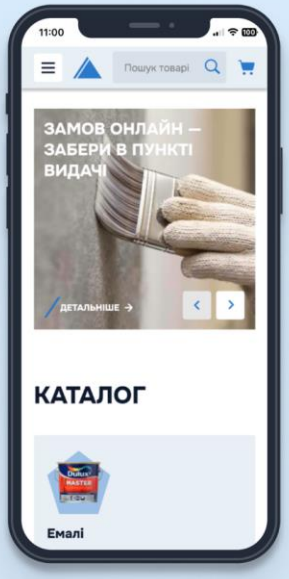


ПРИКЛАД ЕНДПОЇНТУ САЙТУ

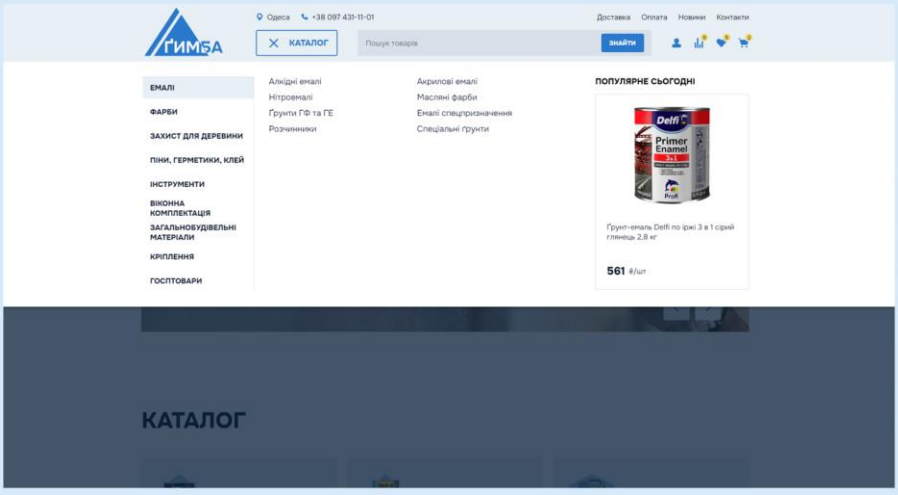


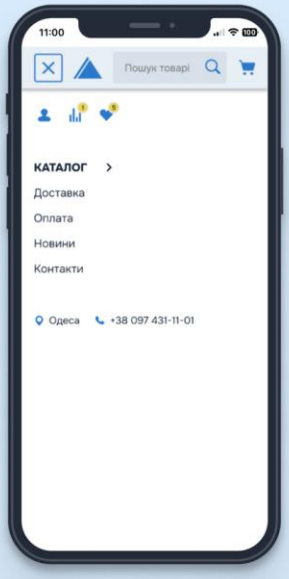
ГОЛОВНА СТОРІНКА



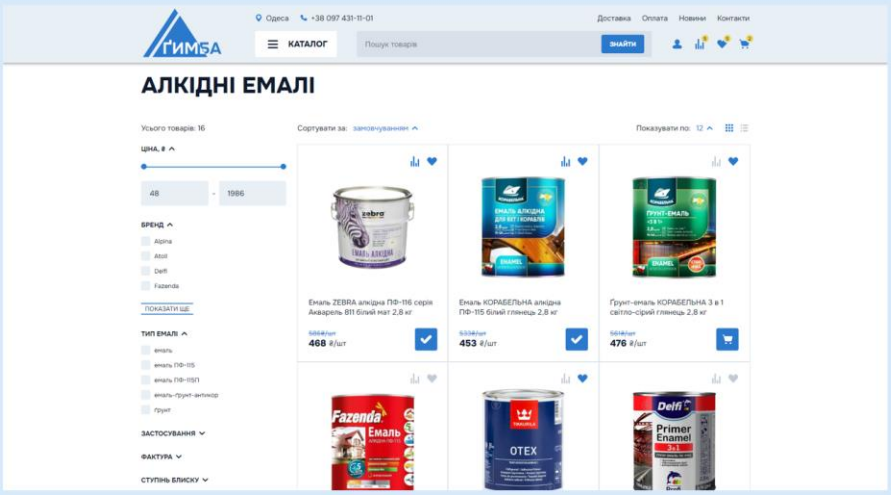


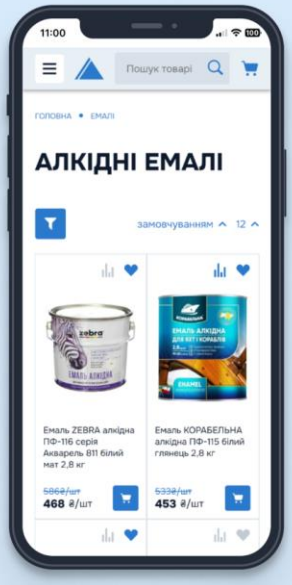
ГОЛОВНЕ МЕНЮ



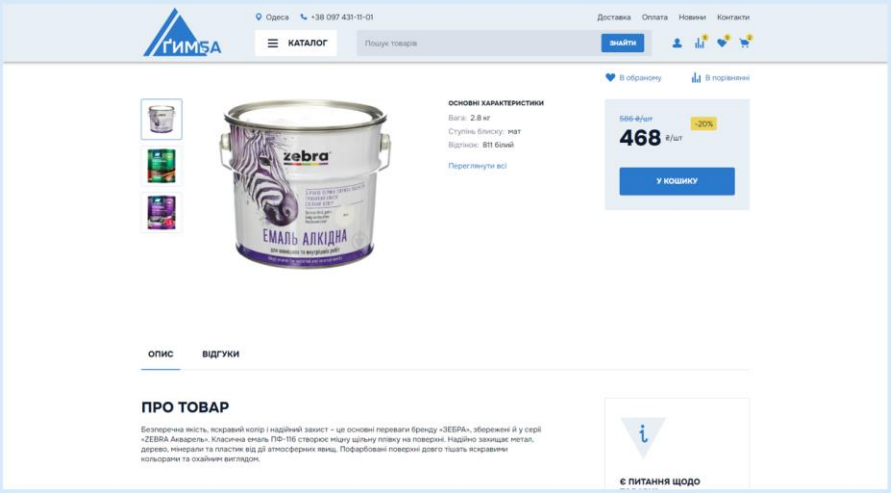


СТОРІНКА КАТАЛОГУ

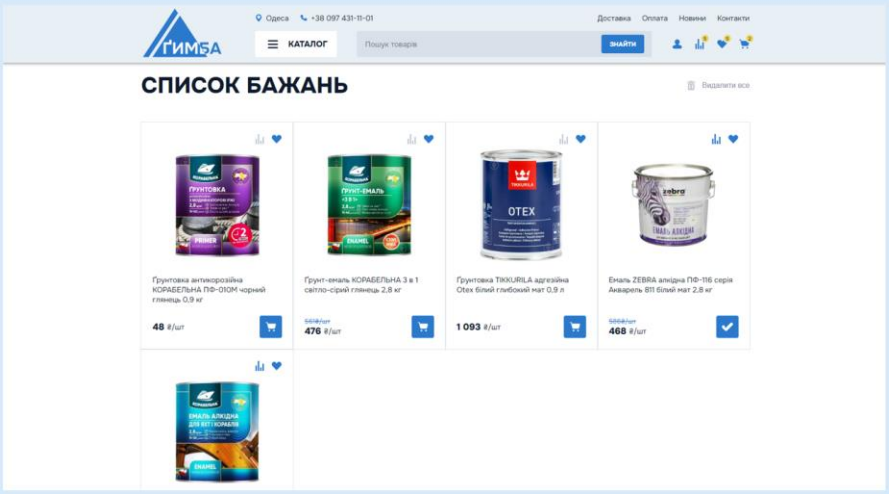


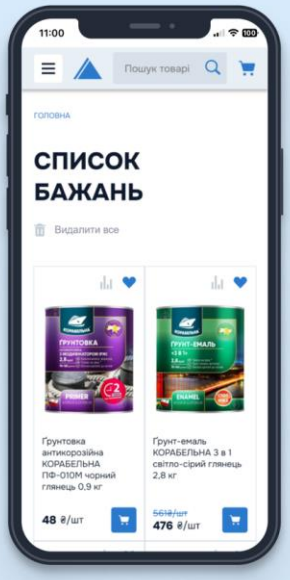


СТОРІНКА ТОВАРУ

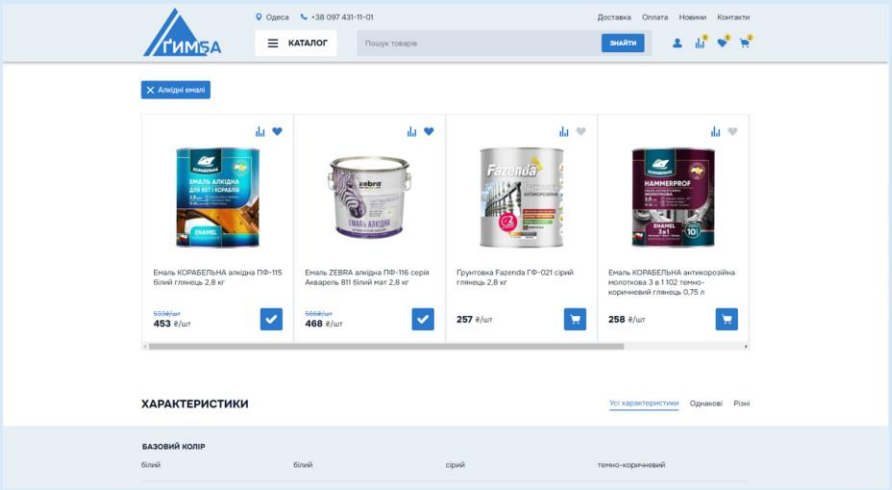


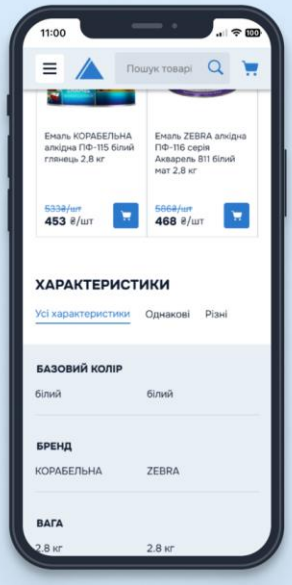
СТОРІНКА СПИСКУ БАЖАНЬ



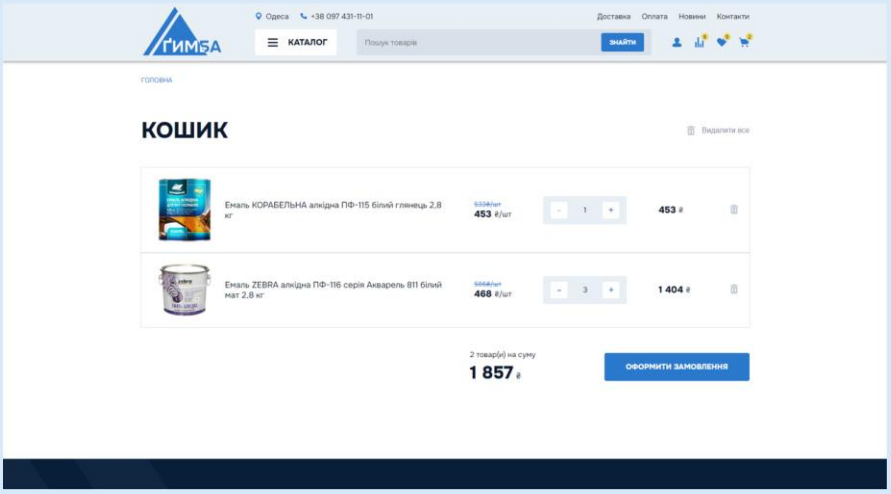


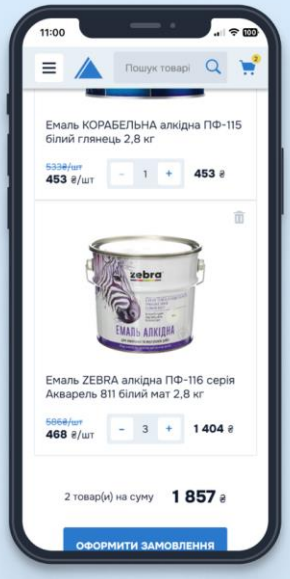
СТОРІНКА ПОРІВНЯННЯ ТОВАРІВ



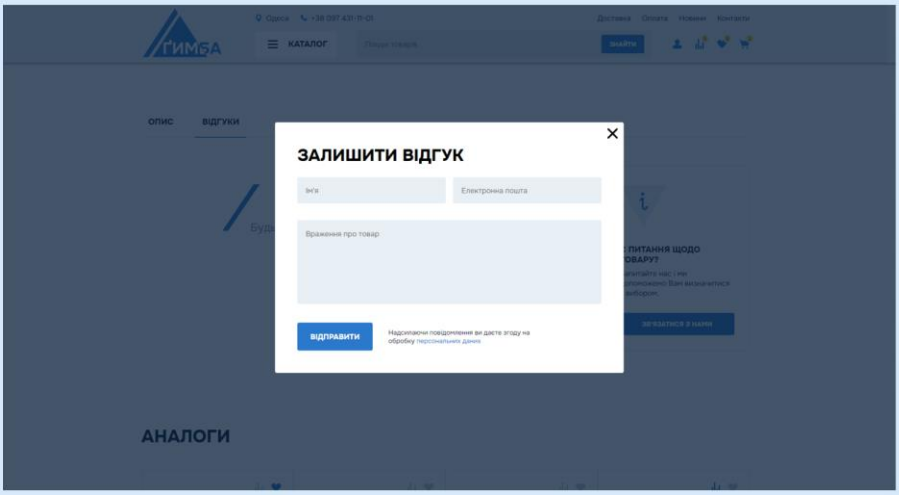


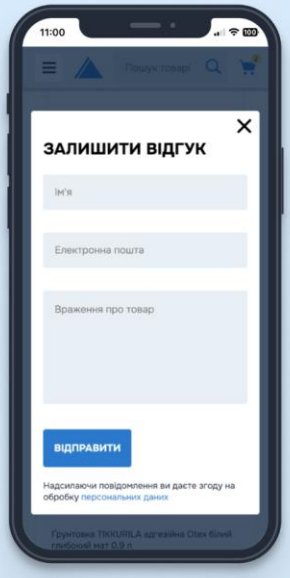
СТОРІНКА КОШИКА





ПРИКЛАД МОДАЛЬНОГО ВІКНА





ДЯКУЮ ЗА УВАГУ!

Відокремлений структурний підрозділ

Одеський технічний фаховий коледж ОНТУ

ВІДГУК

Керівника про кваліфікаційну роботу бакалавра

Грабового Олександра Валентиновича

(прізвище, ім'я та по батькові)

Освітньо-професійна програма *«Комп'ютерна інженерія»*

Спеціальність *123 «Комп'ютерна інженерія»*

Тема кваліфікаційної роботи

«Створення Інтернет-ресурсу для комерційного підприємства з елементами 3D-дизайну»

ХАРАКТЕРИСТИКА ДИПЛОМНОГО ПРОЕКТУ (РОБОТИ)

а) Обсяг і якість виконання роботи (розрахунково-пояснювальної записки)

Пояснювальна записка виконана якісно, у достатньому обсязі, відповідно до індивідуального завдання та теми дипломного проекту, розділи пояснювальної записки відповідають етапам рішення завдання, поставленого у дипломному проекті

Презентація виконана якісно, у достатньому обсязі. Презентація наочно демонструє результати роботи.

б) Самостійність роботи над кваліфікаційною роботою

Студент самостійно обрала напрям та тематику кваліфікаційної роботи. Провів аналіз існуючих рішень і зробив необхідні висновки для реалізації проекту. Виявив навички самостійно опрацьовувати новий матеріал та виконувати пошук необхідної літератури та інших джерел інформації

в) Теоретична підготовка бакалавра

відповідає вимогам, що надаються до бакалавра зі спеціальності «Комп'ютерна інженерія»

г) Вміння розв`язувати виробничі і конструкторські питання на базі останніх досліджень науки і техніки, передових методів виробництва

У дипломному проекті розглянута та реалізована сучасна тема створення інтернет-ресурсів для організації комерційної діяльності описані принципи розробки сайту у сучасному середовищі програмування та процес підбору інструментів та технологій для розробки графічного дизайну та ергономіки . Обґрунтовано вибір програмних засобів для реалізації програмного забезпечення, проведено його тестування

Загальна оцінка 5(відмінно)

Прізвище, ім`я, по батькові Іванова Лілія Вікторівна

Місто роботи і посада керівника проекту ВСП «Одеський технічний фаховий коледж ОНТУ» к.т.н., зав. кафедрою Комп'ютерної інженерії

Підпис

«13» 06 2024р.

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

РЕЦЕНЗІЯ

на кваліфікаційну роботу здобувача (здобувачки) освіти

відділення комп'ютерних систем

Грабового Олександра Валентиновича

(прізвище, ім`я та по батькові)

Спеціальність 123 Комп'ютерна інженерія

Освітня програма Комп'ютерна інженерія

Керівник кваліфікаційної роботи Іванова Лілія Вікторівна

(прізвище, ім`я та по батькові)

Тема кваліфікаційної роботи *«Створення Інтернет-ресурсу для комерційного підприємства з елементами 3D-дизайну»*

Обсяг розрахунково-пояснювальної записки 78 сторінок

Обсяг графічної (презентаційної) частини 13 аркушів (слайдів)

ХАРАКТЕРИСТИКА КВАЛІФІКАЦІЙНОЇ РОБОТИ

а) заключення про ступінь відповідності виконаного кваліфікаційної роботи завданню

кваліфікаційна робота у повному обсязі відповідає темі та завданню

б) характеристика виконання кожного розділу кваліфікаційної роботи

Кваліфікаційна робота складається з розділів: Передмова. 1. Аналітичний огляд існуючих рішень. 2. Обґрунтування вибору технологій для створення інформаційних інтернет-ресурсів. 3. Архітектура сайту. 4. Розробка елементів дизайну та інтерфейсу користувача. 5. Створення макету та прототипу сайту. 6. Тестування сайту. 7.Охорона праці. 8. Висновки. 9. Список використаних джерел інформації.інформацію щодо результатів виконаної роботи.

в) оцінка якості виконання пояснювальної записки та графічної частини кваліфікаційної роботи

Пояснювальна записка виконана якісно, у достатньому обсязі, відповідно до індивідуального завдання та теми дипломного проекту, розділи пояснювальної записки відповідають етапам рішення завдання, поставленого у кваліфікаційній роботі. Презентація виконана якісно, у достатньому обсязі. Презентація наочно демонструє результати роботи.

г) перелік позитивних якостей кваліфікаційної роботи

1. Актуальна тематика.
2. Сучасні технології реалізації програмного продукту
3. Якісне подання результатів роботи.

д) основні недоліки кваліфікаційної роботи

1. Варто було б додати блок-схеми алгоритмів для опису реалізованих скриптів.
2. Пояснювальна записка містить помилки оформлення.

Оцінка розрахункової частини Відмінно

Оцінка графічної частини Добре

Загальна оцінка Відмінно

Прізвище, ім'я, по батькові рецензента Василіу Євген Вікторович

Місце роботи і посада рецензента Державний університет інтелектуальних технологій і зв'язку, д.т.н., проф. кафедри КБ та ТЗІ

Підпис:

«10» 06 2024 р.

UNICHECK
by Turnitin

Ім'я користувача:
Катерина Григоріївна Краснокутська

ID перевірки:
1016344232

Дата перевірки:
10.06.2024 20:35:53 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
10.06.2024 20:38:19 EEST

ID користувача:
100011688

Назва документа: **2БКС-28 Грабовий_Олександр_Валентинович**

Кількість сторінок: **88** Кількість слів: **11615** Кількість символів: **90059** Розмір файлу: **11.12 MB** ID файлу: **1016145767**

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

6.22%
Схожість

Найбільша схожість: **2.83%** з Інтернет-джерелом (https://card-file.ontu.edu.ua/server/api/core/bitstreams/9908b7a9-6b3.

6.22% Джерела з Інтернету | **1000** ..... Сторінка 90

Не знайдено джерел з Бібліотеки

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи | **7**

Підозріле форматування | **29 сторінок**

ДОЗВІЛ
НА РОЗМІЩЕННЯ
ВИПУСКНОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ
В ЕЛЕКТРОННОМУ РЕПОЗИТАРІЇ ВСП «ОТФК ОНТУ»

Ми, що нижче підписалися,

Грабовий Олександр Валентинович
здобувач освіти гр. 2БКС-28, та

Іванова Лілія Вікторівна,
керівник випускної кваліфікаційної роботи,

не заперечуємо щодо розміщення електронного варіанту пояснювальної записки до випускної кваліфікаційної роботи бакалавра на тему:

«Створення Інтернет-ресурсу для комерційного підприємства з елементами 3D-дизайну» (автор роботи – Грабовий О.В., керівник роботи – Іванова Л.В.)

виконаного у ВСП «Одеський технічний фаховий коледж Одеського національного технологічного університету» в 2024 році, у повному обсязі в електронному репозитарії ВСП «ОТФК ОНТУ» для вільного доступу через мережу Інтернет.

Несемо відповідальність за ідентичність електронного та друкованого варіантів випускної кваліфікаційної роботи і даємо згоду на обробку персональних даних.

Виконавець _____ / Грабовий О.В. /

Керівник _____ / Іванова Л.В. /

«13» червня 2024 р.