Приватний заклад вищої освіти
Одеський технологічний університет «ШАГ»
Кафедра інформаційних технологій та фундаментальної підготовки

Випускна кваліфікаційна робота бакалавра

<<СИСТЕМА УПРАВЛІННЯ КОНТЕНТОМ ЕЛЕКТРОННОЇ КОМЕРЦІЇ (за прикладом сервісу Amazon)>>

на здобуття бакалаврського рівня вищої освіти
зі спеціальності «122 Комп'ютерні науки»
Виконавці проекту

№	П.І.Б.	Ролі	Група
1	Семенюк Олег Олегович	Backend	КН-П-201
2	Пшеничний Олександр Сергійович	Backend	КН-П-201
3	Вацько Ірина Олександрівна	Frontend	КН-П-201
4	Рибальченко Денис Леонідович	Frontend	КН-П-201
5	Гетманцева Вікторія Юріївна	Designer	КН-Д-201
6	Гарбуз Дар'я Вячеславівна	Designer	КН-Д-202
7	Петров Всеволід Едуардович	DevOps	КН-А-201

Автор звіту
_____ Рибальченко Денис Леонідович

АНОТАЦІЯ

У цій роботі проводиться аналіз та розробка системи управління контентом електронної комерції за прикладом одного з найвідоміших інтернет магазинів у світі - Amazon. Основними функціями функціями проекту є: реєстрація, автентифікація, авторизація, керування профілем, керування наявними користувачами, створення та редагування категорій, створення та редагування товарів, створення та редагування відгуків до товарів, перегляд та фільтрація товарів, перегляд товару, перегляд свого та чужих відгуків, перегляд статистики товару, додавання товарів у кошик, додавання товарів у список бажаного, оформлення замовлення, перегляд зроблених замовлень та списку бажаного.

При реалізації проекту особлива увага була пригорнута до того, як користувач взаємодіє з сайтом, які елементи він використовує найчастіше, та що він найбільше хоче побачити на кожній сторінці. Була розроблена детальна система фільтрації товарів та їх пошуку.

Актуальною є також система валідації форм, з миттєвим відгуком користувачу при введенні некоректної інформації і можливістю відображення помилок які повертає сервер.

Взаємодія з сервером є однією з найголовніших частин цієї роботи, тому був зроблений акцент на тому щоб користувач знав про статус мережевих запитів та випадково не зробив зайві дії.

Цей документ описує реалізацію клієнтської частини додатку, а також технології які були для цього використані.

ЗМІСТ

ВСТУП

У останнє десятиліття електронна комерція невпинно розвивається. Все більше людей стають зацікавленими у тому, щоб купляти онлайн, і вслід за цим все більше бізнесів замислюються над можливістю продавати товари та надавати послуги через мережу Інтернет. Ця популяризація онлайн магазинів пов’язана в першу чергу з постійно зростаючою кількістю людей що мають доступ до інтернету, а також розвитком інтернет технологій та можливостей бездротового підключення.

Не останню роль зіграла пандемія COVID-19 у 2019 році, що спричинила масові локдауни у багатьох країнах світу, в наслідок чого потреба у інтернет маркетингу значно зросла.

У зв’язку з цим наша команда вирішила розробляти саме платформу електронної комерції, орієнтуючись на світових лідерів у цій галузі, аналізуючи їх сильні сторони та помилки. Клієнти нашої платформи мають змогу зручно відсортувати товари за допомогою великої кількості фільтрів, подивитись детальну інформацію про них, корисний досвід інших користувачів з цим товаром, а також поділитися своєю думкою.

Далі у цьому документі наявне технічне завдання, яке описує основні вимоги з реалізації інтернет магазину, загальної характеристики системи, яка коротко характеризує кожну з частин готового проекту, реалізації системи покупця, яка описує конкретно мою частку роботи над тією частиною сайту яку бачить покупець, реалізацію адміністративної панелі, у якій адміністратор системи може керувати контентом магазину, висновки та перелік джерел, що були використані при розробці та написанні цього звіту.

ТЕХНІЧНЕ ЗАВДАННЯ ДО ПРОЕКТУ

ІНФОРМАЦІЙНА СИСТЕМА УПРАВЛІННЯ ЖИТТЄВИМ ЦИКЛОМ ЕЛЕКТРОННОГО МАРКЕТИНГУ
(за зразком Amazon)

Загальний опис.
Програмне забезпечення (ПЗ) призначене для створення розподіленої системи управління контентом, який призначений для електронної комерції. Функції системи мають забезпечити повний життєвий цикл управління контентом: створення, актуалізація, модифікація та знищення. Головна особливість системи полягає у тому, що вона передбачає механізми узгодження контента з віддаленими джерелами даних , а також управління замовленнями, їх оформленням, статусом, а також зберіганням та переданням їх користувачу або зовнішнім системам (платіжна система, служба доставки, тощо).

Призначення:
ПЗ орієнтується на створення контейнеру управління інформаційними одиницями комерційного призначення. Значна увага приділяється маркетинговим інструментам, що дозволяють проводити дослідження ефективності роботи системи, зокрема, зворотному зв'язку між сторонами комерційної взаємодії. ПЗ створюється за прикладом мережевого магазину, у той же час має складатись за принципом універсальності, мати змогу змінити цільове призначення.

Вимоги до функціоналу. Обов'язкова частина
Головна сторінка
Стартовий інтерфейс ПЗ (головна сторінка або активність) має за мету популяризацію системи. На ньому має бути забезпечено відображення популярних показників як-то загальна кількість користувачів та комерційних одиниць (або їх груп), вибірка з найбільш популярного та новітнього контенту, а також нових дописів (відгуків), зовнішня реклама. Головна сторінка обмежує функції системи для незареєстрованих користувачів лише переглядом, інші

функції надаються у разі авторизації користувача. Має бути забезпечений пошук контенту, групування та фільтрування результатів за різними ознаками, наявними у знайденому контенті. Для об'ємних результатів має бути забезпечена пагінація.

Реєстрація користувачів:
Користувач ПЗ має мати змогу зареєструватись за допомогою авторизаційних даних (наприклад електронної пошти та паролю). При реєстрації обов'язково зазначається засіб оперативного контактування (ЗОК) - номер телефону, електронна пошта тощо. На тестовому етапі, за умови ускладнення замовлення послуг телефонії чи СМС, можливе використання лише електронної пошти. Засіб оперативного контактування (ЗОК) при реєстрації підлягає верифікації через введення коду з надісланого листа на введену пошту користувачем. Рекомендується використовувати ЗОК у якості авторизаційного логіну користувача.
За умови що користувач забув пароль, ЗОК має бути використаний для його відновлення. Схема відновлення паролю жорстко не регламентується.
Також передбачається наявність у системі користувачів з додатковими правами: модераторів та адміністраторів. Реєстрація кореневого адміністратора системи має бути реалізоване при інсталяції системи, наступним користувачам змінюють права наявні адміністратори.

Особистий кабінет користувача:
Основне робоче місце зареєстрованого користувача ПЗ. Повинен забезпечити усі необхідні засоби для

- Управління власними замовленнями: створення, видалення, перегляд усіх замовлень з можливістю більш детального огляду
- Встановлення, зміна та вилучення персональних даних, у т.ч. платіжного засобу, зміна чи відновлення авторизаційних даних (паролю)
- Перегляд поточної активності: історії пошуку, перелік відзначеного контенту (кошик), виконані замовлення.

- Формування зворотного зв'язку: надання оцінок, складання відгуків та рекомендацій

Також засобами особистого кабінету має бути передбачено можливість зміни персональних даних.

Зворотній зв'язок

Користувач має змогу встановити власну оцінку (лайк або бал) для довільного контенту, щодо якого зафіксована його активність (замовлення). За умови виконаного замовлення (купівлі) користувачеві надається можливість створювати відгуки та рекомендації щодо контенту. Засоби пошуку, фільтрування та групування мають забезпечити врахування рейтингу контенту.

Вимоги до функціоналу. Опціональна частина

Модерація

Для забезпечення дотримання загальноприйнятих норм спілкування, а також з метою недопущення публікування контенту, який містить заборонені компоненти (ненормативні висловлювання, заклики до протиправних дії, приниження чи дискримінація, тощо), усі створені користувачами дані (відгуки) мають в обов'язковому порядку пройти погодження (модерацію). Якщо контент порушує норми, кореневий адміністратор має можливість видаляти подібний контент.

Архітектура

ПЗ має бути спроектоване за клієнт-сервеною розподіленою архітектурою з чітко відокремленими структурними вузлами системи:

Mockup - візуальна модель інтерфейсу користувача із зазначенням зв'язків і переходів між різними частинами інтерфейсу.

Backend - серверне програмне забезпечення, що забезпечує збереження та оброблення поточних даних, взаємодію з базою даних. Надає програмний інтерфейс взаємодії (API) з іншими вузлами.

Web App - додаток, доступний з веб-простору (сайт), що забезпечує візуальний інтерфейс користувачів усіх категорій

Усі модулі ПЗ (окрім візуальної моделі Mockup) повинні обов'язково включати до свого складу засоби

- розгортання на новому пристрої (інсталяції) із встановленням початкових таблиць у БД та кореневого користувача-адміністратора

Технології

Зберігання вихідних кодів ПЗ повинно здійснюватись за допомогою системи контроля версій (вибір конкретної системи узгоджується групою проекту). Структура файлів та каталогів повинна відбивати архітектуру ПЗ, на зразок:

- root
    - Backend
    - Frontend
    - Mockup (вихідні файли або посилання на зовнішній проект)
    - DevOps

Для кожного з модулів рекомендується (з можливістю часткової зміни) наступні технології створення:

База даних - MySQL
Backend - ASP .NET Core Web-API.
Frontend - JS фреймворк Next.js на основі бібліотеки React
Mockup - визначається ментором з дизайну
DevOps - GitLab, Docker, MySQL, WireGuard VPN

Команда проєкту

Для виконання поставлених завдань передбачається робота команди розробників окремо за кожним з модулів проекту.
Мінімальна команда проекту - чотири учасники, у такій команді кожен з учасників відповідає за одну з частин: Backend, Frontend, Mockup, DevOps.
Дозволяється включення до команди більшої кількості учасників, у такому разі нові учасники будуть розподілені між існуючих частин (Mockup, Backend, або Frontend).

Звітність

За результатами виконання завдань із створення ПЗ кожен з учасників команди проекту складає власний (індивідуальний) звіт, що складається з практичної частини та автореферату. Практична частина є прямим результатом роботи і подається у вигляді посилання на репозиторій (чи інший URL) або у вигляді архіву на довільному носієві даних. Вихідний код повинен оформлятися у відповідності до загальноприйнятих стандартів, на зразок стандартів GNU http://www.gnu.org/prep/standards/, у тому числі з додержанням вимог до коментування та документування коду.

Автореферат містить описову частину, у якій розкриваються основні концепти проекту, деталізуються завдання до виконавця, встановлюється актуальність роботи, її об'єкт та предмет, обґрунтовується вибір технологій та відзначається шлях виконання та ступінь досягнення поставлених задач, посилання на джерела інформації.

РОЗДІЛ 1

ЗАГАЛЬНА ХАРАКТЕРИСТИКА СИСТЕМИ

В архітектурі проекту наявні такі частини:

- Design
- Frontend
- Backend
- DevOps

Design - дуже важлива частина, яка визначає візуальну частину проекту, а також вивчає цільову категорію, створює структуру, продумує шлях користувача на сайті та його досвід з додатком. Для детального вивчення потреб користувача та цільової категорії проводиться дослідження, після чого розробляється Wireframe, який є каркасом майбутнього сайту. Після визначення структури та шляху користувача на сайті створюється дизайн система, визначаються кольори, шрифти та елементи, за допомогою яких розробляється макет, який відображає модель завершеного продукту.

Frontend - є односторінковим додатком (SPA), що дозволяє досягти максимальної інтерактивності і зробити сайт зручним для користувача, але також має у своєму складі окремий сервер, який рендерить сторінки коли це необхідно, для досягнення найкращого користувацького досвіду та продуктивності роботи сайту у різних сценаріях. Реалізовано за допомогою бібліотеки React та фреймворка Next.js.
Основні цілі, яких необхідно досягти для успішного виконання цієї частини:

- **Узгоджений дизайн** - більшість елементів мають бути стандартизовані, мати однаковий зовнішній вигляд, колірну схему, стани, тощо.
- **Взаємодія з сервером** - відповіді з серверу повинні кешуватись для зменшення часу очікування при виконанні однакових запитів, але також повинні оновлюватись після певних дій для відображення актуальних даних.

- **Адаптивність** - сайт має адаптуватись під різні розміри екранів, зберігаючи оригінальний дизайн, але перебудовуючи його під конкретний пристрій для зручності використання. Особлива увага приділяється мобільним пристроям, тому що значна частина користувачів буде користуватися саме телефонами.
- **Користувацький досвід (UX)** - якість користувацького досвіду повинна бути висока, чого можна досягти за допомогою таких елементів як: кешування запитів, відображення стану завантаження даних, виділення елементів з якими взаємодіє користувач, зручний одноманітний інтерфейс, наявність версій сайту під різні пристрої.

Backend - невід'ємна частина системи, яка виконує усю основну роботу з базою даних, та надає інтерфейс у стилі REST API. За допомогою цього інтерфейсу клієнтський додаток через HTTP-запити отримує та відправляє дані на сервер, де вони валідуються та обробляються. У разі виникнення помилок сервер відправляє об'єкт з детальним описом помилок, у разі успішного виконання відправляється об'єкт з необхідними даними та додатковими метаданими.

DevOps - автоматичне налаштування та розгортання проекту після зміни вихідного коду будь-якої частини проекту. Для налаштування безперервної інтеграції (CI/CD) використовується розгорнутий екземпляр безкоштовного open-source проекту GitLab, який дозволяє як зберігати вихідний код за допомогою системи контролю версій Git, так і налаштовувати його автоматичне розгортання при змінах. Також, для моніторингу статусу розгорнутого додатка використовуються такі сервіси як Prometheus та Grafana.

РОЗДІЛ 2

РЕАЛІЗАЦІЯ КЛІЄНТСЬКОЇ ЧАСТИНИ

Використані технології

При створенні технічного завдання до системи були висунуті такі вимоги:

- Висока інтерактивність
- Висока кількість сторінок і схожих елементів
- Швидкість завантаження у користувача
- Швидкість роботи
- Можливість швидкого змінення зовнішнього вигляду та/або роботи схожих елементів
- Робота з мережевими запитами
- Безпека
- Оптимізація для пошукових систем (CEO)

Зважаючи на це, нами було прийнято рішення використовувати наступні технології:

- TypeScript
- React
- Next.js
- shadcn/ui
- React Query
- React Hook Forms
- Zod
- Zustand
- Tailwind CSS

Далі детальніше за кожну технологію та причину її використання у цьому проекті

TypeScript - це надмножина мови програмування JavaScript, розроблена компанією Microsoft для вирішення проблеми відсутності типізації у JavaScript. TypeScript надає можливість створювати типи, які присвоюються змінним та перевіряють на етапі трансляції у JavaScript, який буде виконуватись на кінцевому пристрої. Не

дивлячись на те що TypeScript потребує етапу трансляції у JavaScript для виконання, TypeScript дозволяє писати безпечний код та не допускати різноманітні поширені помилки які призводять до багів. Ми вирішили зробити проект саме на TypeScript, тому що він дозволяє нам швидше писати код та робити менше помилок, а також більшість сучасних бібліотек мають відмінну інтеграцію з TypeScript.

React - бібліотека для відображення користувацьких інтерфейсів. Ця бібліотека дозволяє створювати високо інтерактивні веб додатки за допомогою JavaScript, а також використовує компонентний підхід, який дозволяє розбивати елементи інтерфейсу на окремі компоненти які розділяють один зовнішній вигляд та поведінку, що дає можливість задовольнити вимоги до нашого проекту.
Під час проектування були також розлянуті альтернативи, такі як Angular та Vue, але врешті решт був обраний саме React через те що він має дуже велику екосистему різноманітних бібліотек та технологій, які можуть допомогти нам у реалізації поставлених вимог.
Але все ж React має деякі недоліки, які притаманні кожному SPA (Single Page Application) фреймворку:

- Користувач має дочекатись завантаження усього JavaScript коду для того щоб побачити сторінку
- Після завантаження сторінки користувач має дочекатися завантаження даних з серверу (профіль користувача, список товарів, тощо)
- Боти пошукових систем можуть не дочекатись рендерингу усього сайту, що значно вплине на індексацію та може навіть зовсім виключити сайт з пошукової видачі.

Для вирішення цих проблем була обрана наступна технологія.

Next.js - серверний JavaScript фреймворк, який використовує бібліотеку для створення користувацьких інтерфейсів React для відображення інтерфейсу а також надає підтримку просунутих можливостей.
Ці додаткові можливості дозволяють нам вирішити проблеми React та реалізувати значну частину вимог проекту, тому нами була обрана саме ця технологія.

Деякі можливості Next.js які використовувались у цьому проекті:

- **Server-Side Rendering** - рендеринг сторінки на сервері за допомогою React та відправлення користувачу готової HTML розмітки, з додаванням інтерактивності (гідратації) після первинного завантаження. Цей підхід дозволяє користувачу швидше побачити сторінку не чекаючи завантаження усього необхідного JavaScript коду, а також значно спрощує роботу пошукових ботів, які одразу отримують готову сторінку для індексації, що дозволяє отримати вищий рейтинг у пошуковій системі. Також на етапі серверного рендеринга до сторінки можно додати дані які необхідно відобразити користувачу, що дозволяє йому не чекати завантаження даних після завантаження сторінки.
- **React Server Components** - рендеринг React компоненту на сервері та відправка його на клієнт у спеціальному форматі, де він додається до працюючого React додатку та відображається користувачу. Цей підхід дозволяє гранулярно налаштувати серверний рендеринг, та найчастіше використовується для відправки користувачу компонента з предзавантаженими даними. У час коли серверний компонент завантажується, Next.js дозволяє відобразити на його місці єлемент який відображає процес завантаження. Також серверні компоненти за замовчуванням кешуются.
- **Caching** - завдяки зміненню (патчингу) браузерної функції *fetch* Next.js автоматично виконує дедуплікацію запитів та дозволяє гнучко налаштувати кешування кожного окремого запиту. Також кешуються сторінки та серверні компоненти, що дозволяє зменшити кількість запитів та час очікування користувача.
- **Middleware** - функция, яка виконується для кожного запиту (можна налаштувати на виконання тільки для окремих запитів). У цій функції можна перевірити, наприклад, чи має користувач доступ до сторінки на яку він намагається потрапити, та перевести його на іншу сторінку якщо ні. Плюс цього підходу у тому що це відбувається на сервері, тому ми можемо бути впевнені що користувач не отримує інформацію яку він не повинен бачити.

- **Rewrites** - перенаправлення обраних запитів на іншу адресу. У нашому випадку використовувалось для перенаправлення запитів на бекенд, який знаходиться у локальній мережі та не має публічної адреси.

shadcn/ui - бібліотека готових компонентів React. Від інших бібліотек відрізняється тим, що не має окремого пакету який експортує компоненти. Усі компоненти копіюються з офіційного сайту бібліотеки або додаються за допомогою спеціального CLI безпосередньо до файлів проекту. Це надає змогу змінювати або доповнювати будь-яку частину компоненту під вимоги проекту. Для стилізації використовується Tailwind CSS, також за замовчуванням налаштовує дизайн систему Tailwind CSS.
Цю бібліотеку ми взяли через те що ми можемо модифікувати її як завгодно, підлаштовуючись під вимоги дизайну

React Query - бібліотека для взаємодії за мережевими запитами. Надає зручний інтерфейс для виконання мережевих запитів, відстежування їх стану, трансформування відповіді та просунутої функціональності, такої як кешування, кількість повторних спроб виконати запит який завершився помилкою та повторне виконання запиту при деяких діях користувача (наприклад повторне відкриття вікна сайту після того як воно було згорнуте).
Оскільки у нашому проекті є велика кількість мережевих запитів, стан яких нам потрібно відстежувати а також застосовувати кешування, ми вирішили додати цю бібліотеку до проекту.

React Hook Forms - бібліотека для керування станом форм. Чим більше у форми полів, тим складніше їх обробляти, валідувати та оновлювати. React не має вбудованого функціоналу керування формами, і підхід з використанням таких примітивів як useState дуже швидко стає занадто великим та неефективним. React Hook Forms - одна з бібліотек яка вирішує цю проблему. Ми обрали її з-поміж інших бібліотек тому що у неї зручний інтерфейс, дуже широкий та гнучкий функціонал, а також є інтеграція з бібліотекою валідації Zod. Також ця бібліотека працює швидше інших, оскільки мінімізує кількість рендерів при вводі значень.

Zod - бібліотека для валідації даних. Має дуже багато примітивів валідації, які можно об’єднувати в схеми та валідувати складі об’єкти (наприклад форми). Ця бібліотека була обрана за широкі можливості валідації складних об’єктів, а також хорошу інтеграцію з іншими бібліотеками, такими як React Hook Forms. У нашому проекті використовується для валідації усіх форм.

Zustand - бібліотека для керування глобальним станом. Оскільки в React немає вбудованого рішення щодо керування глобальним станом, використовуются різні зовнішні бібліотеки. Звичайним вибором більшості розробників є бібліотека Redux, але вона вимагає написання великої кількості одноманітного коду, а також не підтримує просунуту функціональність, таку як асинхронні операції.

Tailwind CSS - фреймворк CSS, який сконцентрований на утилітарних классах, які мають одне або декілька властивостей CSS і виконують одну конкретну задачу (наприклад зміна розміру або кольору тексту). При такому підході можна у переважній більшості випадків не писати окремі CSS класи, а комбінувати класи Tailwind для отримання бажаного результату. Tailwind сканує файли проекту та генерує тільки ті класи які використовуються, що зменшує кількість CSS яку має завантажити користувач. Також Tailwind має гнучку конфігурацію, куди можна внести свою дизайн-систему та потім не вносити зміни по всьому коду у випадку зміни дизайну. Оскільки ми спочатку планували робити макет, і потім додавати до нього дизайн, ця функціональність була для нас однією з головних причин використання Tailwind у проекті. Також бібліотека компонентів shadcn/ui, яку ми також використовуємо, використовує Tailwind для стилізації компонентів, що було для нас додатковим плюсом до рішення про використання цієї технології.

Це були основні бібліотеки та технології, які використовуються на цьому проекті. Звісно, ми використовували ще деякі допоміжні бібліотеки, але оскільки їх роль в проекті не дуже велика, у цьому звіті вони не описуються.

Архітектура

Сам сайт у нас ділиться на 2 частини:

- Магазин - основний сайт який бачить користувач, і з яким він взаємодіє
- Адмін панель - окрема частина сайту, де адміністратор може керувати наявним у магазині контентом (категорії, товари, користувачі)

Для відображення користувацького інтерфейсу використовується бібліотека React. Ця бібліотека оперує компонентами - функціями, які у собі інкапсулюють поведінку, дані та розмітку.
Ми створили велику кількість компонентів, від карток та полів вводу, до цілих сторінок. Усі ці компоненти можна перевикористовувати та змінювати їх вигляд або поведінку, передаючи певні дані.

Маршрутизація між сторінками відбувається за допомогою Next.js App Router, де структура папок та файлів відбиває структуру сторінок на сайті. Але нами були зроблені деякі зміни у цей процес. Оскільки ми вирішили відображати адмін панель з субдомену admin, був написаний middleware, який викликається на кожен запит на сторінку, та визначає на якому субдомені знаходиться користувач, після чого говорить Next.js у якій папці шукати необхідну сторінку. Якщо користувач знаходиться на кореневому домені, за замовчуванням він отримує сторінку магазину. Таким чином, магазин доступний або з кореневого домену, або з субдомену shop.

Також цей middleware перевіряє, чи має користувач право передивлятись певну сторінку. Не авторизований користувач не може зайти на сторінку аккаунту, а покупець не може відкрити жодну зі сторінок адмін панелі крім логіну. Альтернативою middleware є перевірка привілей на клієнті, але у такому випадку користувач спершу отримує дані сторінки, що може призвести до потенційного витоку інформації, тож ми не використовували цей підхід.

Комунікація з сервером відбувається шляхом відправлення запиту з префіксом /api на сервер Next.js, який перенаправляє цей

запит на аналогічний шлях на Backend. Це дозволяє взаємодіяти з сервером який не має виходу у публічний інтернет, а також додавати власні обробники Next.js для інтеграції зі сторонніми сервісами.

Здебільшого, дані з сервера завантажуються вже після завантаження сторінки, але деякі сторінки потребують предзавантажених даних. Наприклад, на сторінці товару дані про товар отримуються з серверу на етапі рендерингу сторінки, а дані про відгуки вже після (рис. 1). На рис. 2 представлена для порівняння типова для SPA схема роботи, де можна побачити більшу кількість запитів, що виконуватимуться довше. На рис. 3 відображена загальна схема роботи запитів нашого додатку.

Рисунок 1 - запити під час рендерингу сторінки товару

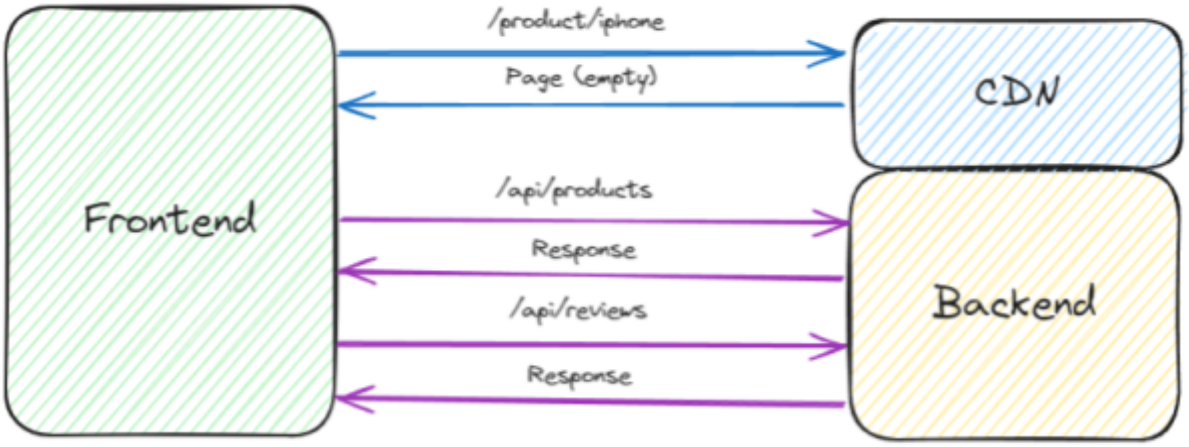


Рисунок 2 - традиційна схема мережевих запитів SPA

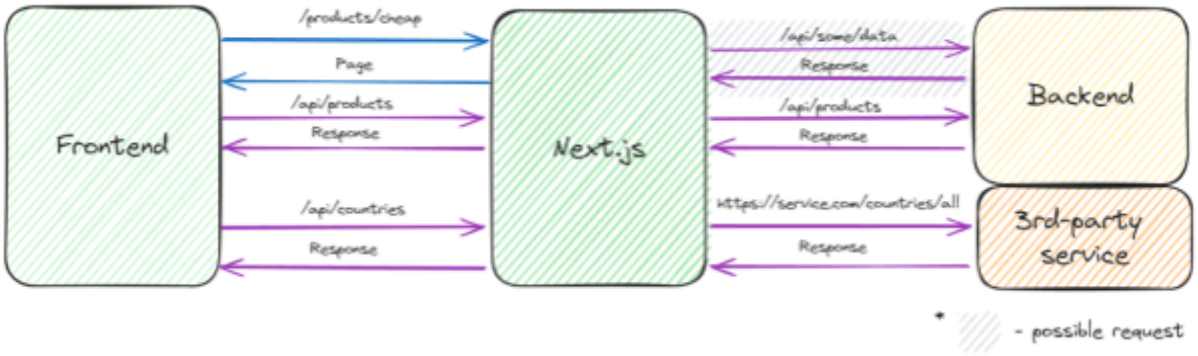


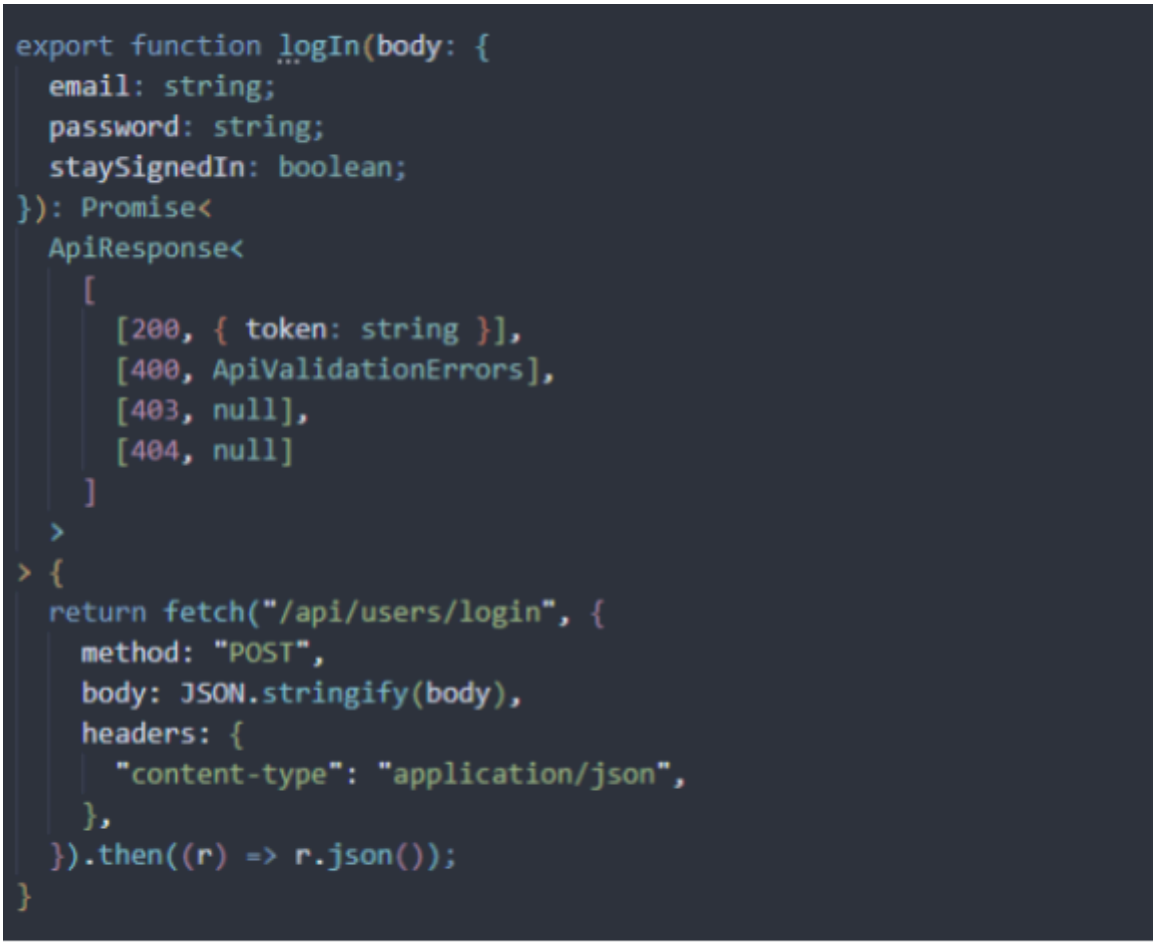
Рисунок 3 - загальна схема запитів у цьому проекті

Для отримання та відправлення даних використовується бібліотека React Query, за допомогою якої відстежується стан запиту для відображення стану завантаження, а також реалізується кешування. При необхідності, після певних дій кеш скидається і завантажуються нові дані. Наприклад, після написання відгуку у товарі оновлюються каруселі з товарами на головній сторінці, щоб користувач побачив зміни якщо там був цей товар.

Для типізації серверних запитів був розроблений окремий інтерфейс ApiResponse (рис. 4), який на основі переданих даних співвідносить результат запиту та дані що прийшли у ньому. Приклад типізації запиту можна побачити на рис. 5.

```
export type ApiResponse<T extends [number, unknown][]> = {
  [index in keyof T]: {
    status: T[index][0];
    message: string;
    data: T[index][1];
  };
}[number];
```

Рисунок 4 - інтерфейс ApiResponse



```
export function logIn(body: {
  email: string;
  password: string;
  staySignedIn: boolean;
}): Promise<
  ApiResponse<
    [
      [200, { token: string }],
      [400, ApiValidationErrors],
      [403, null],
      [404, null]
    ]
  >
> {
  return fetch("/api/users/login", {
    method: "POST",
    body: JSON.stringify(body),
    headers: {
      "content-type": "application/json",
    },
  }).then((r) => r.json());
}
```

Рисунок 5 - типізація функції входу користувача

Реалізація сторінок у нас йшла у такому порядку:

- Верстка Wireframe
- З’єднання з сервером та верстка готового дизайну

Далі детальніше про кожну частину

Реалізація сторінок магазину

Оскільки саме цю частину бачить переважна більшість користувачів, особливий акцент був зроблений на зручності користування та адаптивності. Кожна сторінка була розроблена адаптивною під різні типи пристроїв (стаціонарні комп’ютери, ноутбуки, планшети, телефони) відповідно до дизайну.

Здебільшого, для цього використовувалися медіа запити (CSS Media Queries) для відображення відповідного контенту на різних пристроях, але не дивлячись на всі плюси цього підходу інколи цього

буває недостатньо. Є ситуації, коли треба щоб контент відображався певним чином саме в залежності від розміру контейнера в якому він знаходиться, що неможливо реалізувати за допомогою медіа запитів.

Саме така проблема у мене виникла з карткою товару, яка відображається по різному в залежності від доступного місця. Для вирішення цієї проблеми були використані відносно недавні контейнерні запити (CSS Container Queries), які дозволяють визначити певний елемент як контейнер, та додавати до його дочірніх елементів стилі в залежності від розміру контейнеру. Такий же підхід був також використаний у картці товару.

Окрема увага була приділена оптимізації картинок. Усі картинки які додаються у якості графіки (банери) було переведено у формат WebP, що дозволило значно зменшити їх розмір та пришвидшити завантаження сайту. Також, для всіх картинок було використане ліниве завантаження, що дозволяє відобразити сторінку не чекаючи завантаження всіх картинок.

Реалізацію сторінок я почав з головної сторінки, де користувач може побачити банери, а також каруселі з рекомендованими товарами та категоріями. Картинки в банерах завантажуються у форматі WebP, елементи каруселі завантажуються після відкриття користувачем сторінки та відображають статус завантаження у вигляді скелетів (Skeleton) карток (рис. 6). Кількість карток у каруселі змінюється зі зміною розміру екрану.

Рисунок 6 - завантаження даних у каруселі

На головній сторінці також можно побачити хедер та футер, які є однаковими на всіх сторінках магазину, тому що вказані на загальному для всього магазину layout.

У футері наявні посилання на соціальні мережі та сторінки, де можна прочитати правила користування сервісом.

У хедері користувач може перейти до свого профілю, відкрити модальне вікно корзини або відкрити бокове меню, звідки можна ініціювати вхід чи реєстрацію, побачити дані авторизованого користувача (при наявності), перейти до налаштувань акаунту та вийти з акаунту. Також у хедері наявне поле пошуку, де користувач може ввести здійснити пошук по назві товару, після чого він переходить на сторінку товарів.

Реєстрація та автентифікація відбуваються за допомогою серії модальних вікон. Ці вікна були створені іншим учасником проекту, в той час коли я займався їх інтеграцією з сервером. При успішній реєстрації чи автентифікації користувач отримує токен, який зберігається у глобальне сховище та згодом додається до запитів які цього потребують. Після зміни токену автоматично відбувається запит на отримання профілю користувача. Також токен зберігається до localStorage, та навіть після перезавантаження сторінки якщо користувач авторизован він залишається на поточній сторінці і знову отримує дані профілю.

На сторінці товарів я здебільшого займався дороблennям її за прикладом фінальної версії дизайну, а також зв’язком з сервером (отримання товарів, отримання доступних фільтрів, отримання товарів по фільтру, відображення отриманих даних). Оскільки фільтри та товари також завантажуються з серверу, для них були розроблені окремі скелети завантаження (рис. 7). Відповіді сервера щодо фільтрів та товарів кешуються.

Також э майже ідентична сторінка товарів по категорії, де на додачу до фільтрів товари також фільтруються по обраній категорії. Натиснувши на картку товару, користувач переходить на сторінку товару.

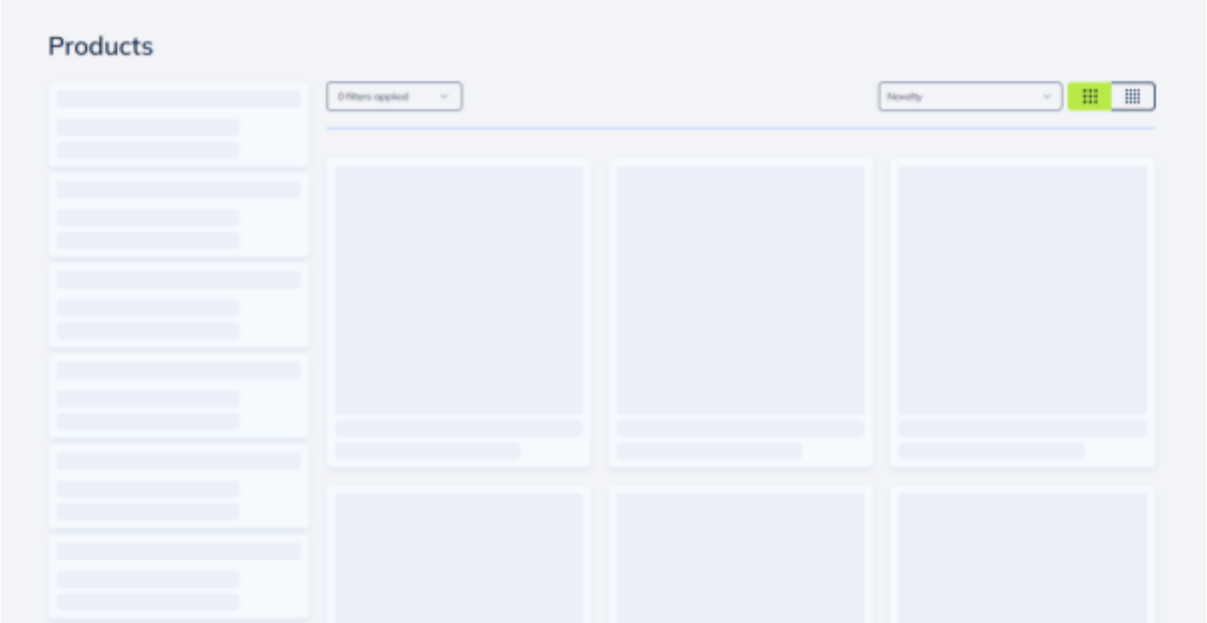


Рисунок 7 - завантаження товарів та фільтрів на сторінці товару

Як вже згадувалося раніше, дані товару потрапляють на сторінку ще до того як сторінка потрапляє до користувача, а інші дані (відгуки та рекомендовані товари) вже після. Рішення організувати рендеринг таким чином було обрано через те, що дані про товар це критична інформація яку користувач повинен бачити одразу після відкриття сторінки, в той час як відгуки та рекомендації знаходяться внизу сторінки і користувач може до них навіть не дійти. Також, якщо Next.js сервер розгорнутий в одній локальній мережі з Backend сервером, він може отримати дані про товар значно швидше, ніж користувач, якому довелося б зробити додатковий запит. Такої моделі рендерингу вдалось досягнути перетворивши сторінку товара на Server Component, який може виконувати асинхронні операції (такі як виконання мережевого запиту).

На сторінці товару користувач може подивитись дані товару, інформацію про оплату та умови повернення, фотографії товару у каруселі (при натисканні на фотографію вона відкривається у окремому вікні у збільшеному вигляді), додати товар до кошика або списку бажаного.

Для відображення стану завантаження відгуків, відгука користувача та статистики відгуків по товару реалізовані окремі скелети (рис. 8).

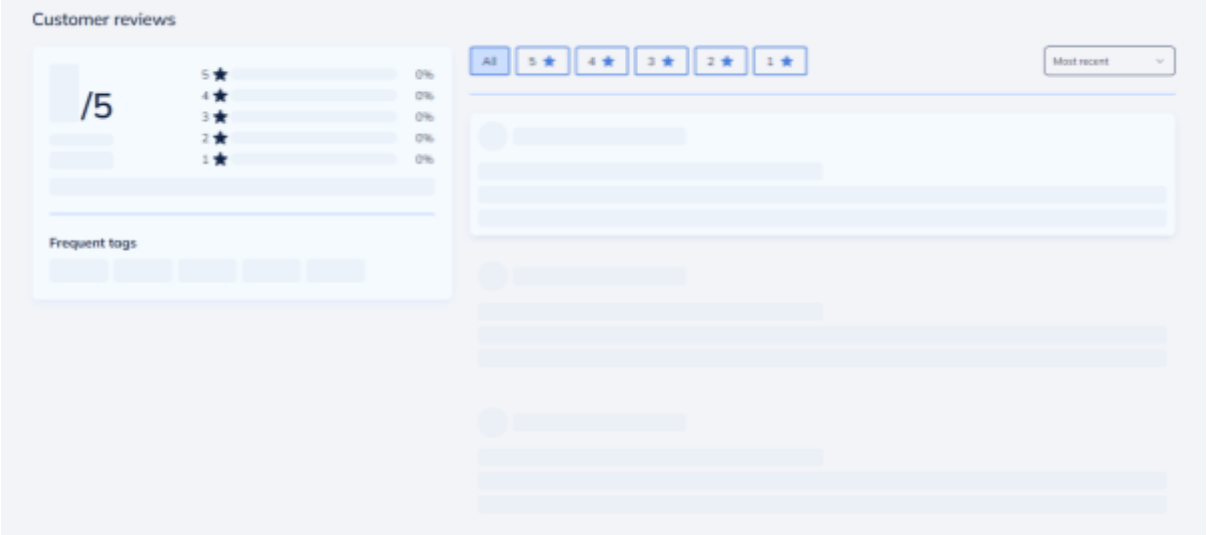


Рисунок 8 - завантаження відгука користувача, інших відгуків та статистики

У разі якщо користувач написав відгук цього товару, він може його переглянути, та при бажанні відредагувати чи видалити. Якщо відгук користувача не знайдено, у нього з’являється кнопка, яка при натисканні відкриває модальне вікно створення відгуку, через це ж вікно також відбувається і редагування. Окремо від рейтингу та тексту до відгука можно додати картинки а також вибрати теги, які найкраще описують досвід користувача з товаром. При створенні та редагуванні автоматично оновлюється інформація про статистику відгуків.

Після відгука користувача йдуть відгуки інших користувачів. При натисканні ці відгуки можна переглянути у розгорнутому форматі, а також подивитись прикріплені до них фотографії у збільшеному вигляді. На відгук можна поставити лайк, що миттєво відобразиться на картці. Це вдалося реалізувати за допомогою оптимістичних оновлень - спочатку у користувача відображаються зміни як вже виконані, а потім якщо серверний запит поверне помилку зміни буде скасовано.

РОЗДІЛ 3

РЕАЛІЗАЦІЯ АДМІНІСТРАТИВНОЇ ПАНЕЛІ

Адміністративна панель реалізована з використанням тих самих технологій що й частина покупця, і є частиною загального проекту, яка має окремий layout та додатковий захист сторінок.

На відміну від частини покупця, адміністративна панель не передбачається для використання на мобільних пристроях, тому ця частина була адаптована лише під стаціонарні комп'ютери та ноутбуки.

Наразі у нас реалізовано 4 сторінки адмін панелі:

- Логін
- Категорії
- Користувачі
- Товари

Сторінка логіну представляє собою просту форму входу з поштою та паролем. Вхід відбувається через той же серверний оброблювач що і вхід покупця, але якщо користувач не має прав адміністратора то він отримує помилку. Усі інші сторінки адмін панелі захищені на стороні сервера Next.js та не відкриються у користувача який не є адміністратором.

Успішно пройшовши логін, адміністратор потрапляє на сторінку категорій. Тут з випадаючого списку можна вибрати одну з кореневих категорій, та побачити їх підкатегорії у вигляді дерева з чекбоксами. Для реалізації цього списку було написано декілька функцій для перетворення масиву категорій в дерево та роботу з цим деревом. Серед функцій є: змінення, додавання, видалення, фільтрування, пошук. Згодом ці функції були використані ще в декількох місцях, де був потрібен перегляд категорій у вигляді дерева. На рис. 9 зображений приклад функції для оновлення статусу чекбоксу у дереві.

У випадаючому списку з кореневими категоріями є кнопка яка відкриває вікно створення кореневої категорії, також після вибору

кореневої категорії стає доступна кнопка створення дочірньої категорії. Для створення та редагування кореневої та дочірньої категорій використовується однаковий компонент форми, який відображає різні поля в залежності від переданих параметрів. У цій формі окрім базової інформації можна також створити property keys - спеціальні ключі, які адміністратор повинен описати при створенні товару у цій категорії.

```
export function updateTree<TValue, TId>(
  root: TreeNodeType<TValue>,
  updatedNode: TreeNodeType<TValue>,
  updatedNodeChecked: CheckedState,
  options: NodeOptions<TValue, TId>
): TreeNodeType<TValue> {
  if (options.getId(root.value) === options.getId(updatedNode.value)) {
    return {
      value: updatedNode.value,
      isRoot: updatedNode.isRoot,
      checked: updatedNodeChecked,
      nodes: updateNodesCheckedState(updatedNode.nodes, updatedNodeChecked),
    };
  }

  const copiedNodes: TreeNodeType<TValue>[] = [];
  for (let node of root.nodes) {
    copiedNodes.push(
      updateTree(node, updatedNode, updatedNodeChecked, options)
    );
  }

  const checkedValues = copiedNodes.map((n) => n.checked);
  const checked = getCheckedState(checkedValues, root.checked, root.isRoot);

  return {
    value: root.value,
    isRoot: root.isRoot,
    checked: checked,
    nodes: copiedNodes,
  };
}
```

Рисунок 9 - оновлення елементів дерева категорій

При натисненні на категорію у списку у картці праворуч відкривається більш детальна інформація про цю категорію. Тут же категорію можна відредагувати або видалити. Також є можливість видалення декількох категорій одразу, виділивши їх чекбоксами та натиснувши кнопку видалення. Категорії не видаляються назавжди, а просто деактивуються, після чого більше не відображаються у покупців.

Сторінка користувачів найпростіша - тут у таблиці можна переглянути всіх користувачів які зареєстровані на сайті, деактивувати чи змінити їх роль (покупець/адміністратор). Також можна здійснити пошук та налаштувати відображення даних у таблиці.

Сторінка товарів схожа на сторінку категорій, але тут можна вибрати не лише кореневу категорію, а будь-яку, після чого відобразиться таблиця товарів з вибраної категорії. Ці товари також можна обирати, детально передивлятись, видаляти. Тут же знаходиться кнопка яка відкриває форму створення нового товару або редагування існуючого. Це найскладніша форма на сайті, тут можна налаштувати всі характеристики і опис товару, а також додати фотографії.

ВИСНОВКИ

У процесі розробки проекту було проаналізовано і випробувано різні підходи до будування веб додатку, які стосуються архітектури, структуризації компонентів, взаємодією з віддаленим сервером через API, а також таких тем як безпека та оптимізація різних показників, таких як швидкість завантаження.

Спроектовано й розроблено систему яка може змінюватись при змінах в дизайні, що дало змогу не переробляти кольори та шрифти кожен раз коли у дизайні відбуваються зміни.

Проаналізована взаємодія користувача з додатком та реалізовані необхідні зміни для покращення користувацького досвіду.

Стандартизована взаємодія з сервером, який використовує інтерфейс REST API, реалізоване належне сповіщення користувача про наявні запити на завантаження даних, їх вигляд та місце майбутнього розташування.

Звісно, залишається ще багато простору для покращення. Наприклад, обробка невдалих запитів та помилку відсутності у користувача інтернету, оповіщення користувача про ці помилки.

Також в планах залишається реалізація ще двох сторінок адміністративної панелі - замовлень та відгуків. Адміністратору важливо мати змогу модерувати весь контент який наявний на сайті, особливо той який створюють користувачі.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

- TypeScript Docs
- React Docs
- Next.js Docs
- shadcn/ui Docs
- React Query Docs
- React Hook Forms Docs
- Zod Docs
- Zustand Docs
- Tailwind CSS Docs
- Git Docs
- StackOverflow