**АНОТАЦІЯ**

Розробка вебсайту магазину аксесуарів «Atna» // Дипломний проект // Аліна ПОПИК// Відокремлений структурний підрозділ «Тернопільський фаховий коледж» Тернопільського національного технічного університету імені Івана Пулюя, спеціальність: комп’ютерні науки, група: КН-421// Тернопіль, 2023

Тема дипломного проекту: Розробка вебсайту магазину аксесуарів «Atna».

Пояснювальна записка складається з п'яти розділів.

У загальній частині описуються аналітичний огляд існуючих рішень та аналіз технічного завдання.

У другому розділі представлено процес створення вебсайту, опис та обґрунтування вибору структури сайту. Тут подано опис процесу створення клієнтської частини магазину, описані його компоненти та їх функціонал. Також описано процес створення API магазину, тобто його серверної частини. Подано основні підходи до тестування сайту.

В спеціальній частині описані процес розгортання програмного продукту на вільному хостингу, інструкція з обслуговування та наповнення сайту, інструкція з популяризації та підтримки сайту.

Розрахунок вартості розробки та економічної ефективності приведено в економічній частині.

Основні питання охорони праці та техніки безпеки розглянуто в п’ятому розділі.

Обсяг пояснювальної записки 129 сторінок формату А4.

До складу дипломного проекту входить графічна частина, яка складається з структурної схеми інтерфейсу вебсайту, блок-схеми алгоритму оформлення замовлення, лістингу функції корзини, таблиці техніко-економічних показників, що виконані на окремих аркушах формату А1.

**ЗМІСТ**

[ВСТУП 10](#_Toc137902773)

[ЗАГАЛЬНИЙ РОЗДІЛ 12](#_Toc137902774)

[1.1 Аналітичний огляд існуючих рішень 12](#_Toc137902775)

[1.2 Технічне завдання 14](#_Toc137902776)

[1.2.1 Найменування та область застосування 14](#_Toc137902777)

[1.2.2 Призначення розробки 15](#_Toc137902778)

[1.2.3 Вимоги до функціоналу вебсайту 15](#_Toc137902779)

[1.2.4 Вимоги до програмної документації 16](#_Toc137902780)

[1.2.5 Техніко-економічні показники 17](#_Toc137902781)

[1.2.6 Стадії та етапи розробки 18](#_Toc137902782)

[1.2.7 Порядок тестування та прийому 19](#_Toc137902783)

[2 РОЗРОБКА ТЕХНІЧНОГО ТА РОБОЧОГО ПРОЄКТУ 21](#_Toc137902784)

[2.1 Розробка структури сайту і веб-сторінок 21](#_Toc137902785)

[2.2 Створення та верстка сторінок сайту 23](#_Toc137902786)

[2.3 Розробка структури бази даних сайту 26](#_Toc137902787)

[2.4 Програмування сайту 28](#_Toc137902788)

[2.4.1 Написання клієнтської частини 28](#_Toc137902789)

[2.4.2 Написання серверної частини 42](#_Toc137902790)

[2.5 Тестування вебсайту 47](#_Toc137902791)

[3 СПЕЦІАЛЬНИЙ РОЗДІЛ 50](#_Toc137902792)

[3.1 Інструкція з розміщення сайту в Інтернеті 50](#_Toc137902793)

[3.2 Інструкція з обслуговування та наповнення сайту 53](#_Toc137902794)

[3.3 Інструкція з популяризації та підтримки сайту 55](#_Toc137902795)

[4 ЕКОНОМІЧНИЙ РОЗДІЛ 58](#_Toc137902796)

[4.1 Визначення стадій технологічного процесу та загальної тривалості проведення НДР 58](#_Toc137902797)

[4.2 Визначення витрат на оплату праці та відрахувань на соціальні заходи 59](#_Toc137902798)

[4.3 Розрахунок матеріальних витрат 61](#_Toc137902799)

[4.4 Розрахунок витрат на електроенергію 62](#_Toc137902800)

[4.5 Визначення транспортних затрат 62](#_Toc137902801)

[4.6 Розрахунок суми амортизаційних відрахувань 63](#_Toc137902802)

[4.7 Обчислення накладних витрат 63](#_Toc137902803)

[4.8 Складання кошторису витрат та визначення собівартості НДР 64](#_Toc137902804)

[4.9 Розрахунок ціни НДР 64](#_Toc137902805)

[4.10 Визначення економічної ефективності і терміну окупності капітальних вкладень 65](#_Toc137902806)

[5 ОХОРОНА ПРАЦІ, ТЕХНІКА БЕЗПЕКИ ТА ЕКОЛОГІЧНІ ВИМОГИ 67](#_Toc137902807)

[5.1 Гігієнічні вимоги до організації і обладнання робочих місць з ВДТ 67](#_Toc137902808)

[5.2 Факти впливу ВДТ на людину 68](#_Toc137902809)

[5.3 Способи та засоби пожежогасіння 74](#_Toc137902810)

[ВИСНОВКИ 79](#_Toc137902811)

[ПЕРЕЛІК ПОСИЛАНЬ 80](#_Toc137902812)

[ДОДАТКИ 82](#_Toc137902813)

[Додаток А – Лістинг файлу src\index.js 82](#_Toc137902814)

[Додаток Б – Лістинг коду компоненту <App/> 83](#_Toc137902815)

[Додаток В – Лістинг коду компоненту <Navigation /> 85](#_Toc137902816)

[Додаток Г – Лістинг коду компоненту <Login /> 89](#_Toc137902817)

[Додаток Д – Лістинг коду компоненту <Signup /> 91](#_Toc137902818)

[Додаток E – Лістинг коду компоненту <appAPI /> 93](#_Toc137902819)

[Додаток Є – Лістинг коду компоненту <Home /> 96](#_Toc137902820)

[Додаток Ж – Лістинг коду компоненту <ProductPage /> 98](#_Toc137902821)

[Додаток З – Лістинг коду компоненту <CartPage /> 101](#_Toc137902822)

[Додаток И – Лістинг коду компоненту <CheckoutForm /> 104](#_Toc137902823)

[Додаток І – Лістинг коду компоненту <NewProduct /> 107](#_Toc137902824)

[Додаток Ї – Лістинг коду компоненту <AdminDashboard /> 110](#_Toc137902825)

[Додаток Й – Лістинг коду компоненту <EditProductPage /> 112](#_Toc137902826)

[Додаток К – Лістинг файлу server.js 116](#_Toc137902827)

[Додаток Л – Лістинг файлу routes\userRoutes.js 117](#_Toc137902828)

[Додаток М – Лістинг файлу routes\productRoutes.js 118](#_Toc137902829)

[Додаток Н – Лістинг файлу routes\orderRoutes.js 122](#_Toc137902830)

[Додаток О – Лістинг файлів схем і моделей бази даних 124](#_Toc137902831)

[Додаток П – Лістинг файлу src\App.test.js 127](#_Toc137902832)

[Додаток Р – Лістинг файлу src\AppInt.test.js 128](#_Toc137902833)

[Додаток С – Приклад інтеграційного тесту для маршруту на Express 129](#_Toc137902834)

ВСТУП

Web-розробка відіграє критичну роль у створенні і успішному функціонуванні сучасних магазинів електронної комерції.

Веб-розробка дозволяє створювати привабливий та функціональний веб-сайт, який служить основним каналом зв'язку з клієнтами. Відкритий цілодобовий доступ до магазину дозволяє клієнтам зробити покупку в зручний для них час

Якість користувацького досвіду є вирішальним фактором у залученні та збереженні клієнтів. Через web-розробку можна створити інтуїтивно зрозумілий і легко навігаційний інтерфейс, забезпечити швидке завантаження сторінок і оптимізувати сайт для різних пристроїв.

Створення веб-додатків дозволяє реалізувати широкий спектр функцій, які покращують комфорт і зручність покупок. Наприклад, можливість реєстрації користувача, керування кошиком, забезпечення безпеки платежів, розрахунок вартості доставки, інтеграцію з соціальними медіа, організацію акцій та знижок, збір і аналіз даних про клієнтів та багато іншого.

Електронні магазини дозволяють підключати різноманітні платіжні системи (наприклад, PayPal, Stripe, Apple Pay тощо), що полегшують процес здійснення платежів та забезпечують зручність для клієнтів. Це дозволяє забезпечити безпеку платежів і розширити способи оплати, що впливає на зростання продажів.

Оптимізований сайт забезпечує його видимість у пошукових системах, підвищує ймовірність того, що потенційні клієнти відвідають електронний магазин під час пошуку відповідних товарів.

Сучасні магазини електронної комерції дозволяють налаштувати інструменти аналітики, які дозволяють відстежувати та аналізувати поведінку клієнтів на веб-сайті. Це надає важливу інформацію про популярні товари, конверсію, покинуті кошики та інші метрики, які допомагають вдосконалювати стратегії маркетингу та привертати більше продажів.

Отже web-розробка є важливим елементом для створення магазинів електронної комерції, оскільки вона забезпечує функціональність, зручність, безпеку та ефективність таких магазинів. Це допомагає залучати та задовольняти клієнтів, покращувати продажі і забезпечувати успішну онлайн присутність вашого бізнесу.

В даному дипломному проекту ведеться розробка веб-сайту магазину аксесуарів «Atna». Очікується, що результатом цього дипломного проекту буде повнофункціональний вебсайт магазину аксесуарів "Atna", який забезпечить зручний та безпечний процес онлайн-покупок користувачів. Розробка цього проекту дозволить набути практичного досвіду програмування магазинів електронної комерції.

Цей проект є актуальним та значущим, оскільки веб-сайти магазинів аксесуарів стають все більш популярними серед споживачів. Розробка ефективного та зручного онлайн-магазину забезпечить конкурентні переваги для бізнесу та задоволення потреб клієнтів.

ЗАГАЛЬНИЙ РОЗДІЛ

1.1 Аналітичний огляд існуючих рішень

В сучасному світі, де електронна комерція швидко розвивається і стає все більш популярною, створення магазинів електронної комерції стає незамінним інструментом для бізнесу. При створенні електронно магазину можна скористатися двома підходами:

* скористатися платформою для створення інтернет-магазину;
* реалізувати власний проект інтернет-магазину.

Платформи для створення магазинів надають можливість підприємцям легко створити й управляти власними онлайн-магазинами без необхідності глибоких технічних знань або значних витрат на розробку з нуля.

Розглянемо кілька ключових переваг і особливостей створення магазинів електронної комерції на основі платформ для створення магазинів електронної комерції. Такі платформи, як Shopify (див. рис. 1.1), WooCommerce, Magento, BigCommerce і Squarespace, пропонують різні набори функціональних можливостей, які відповідають потребам різних бізнесів.

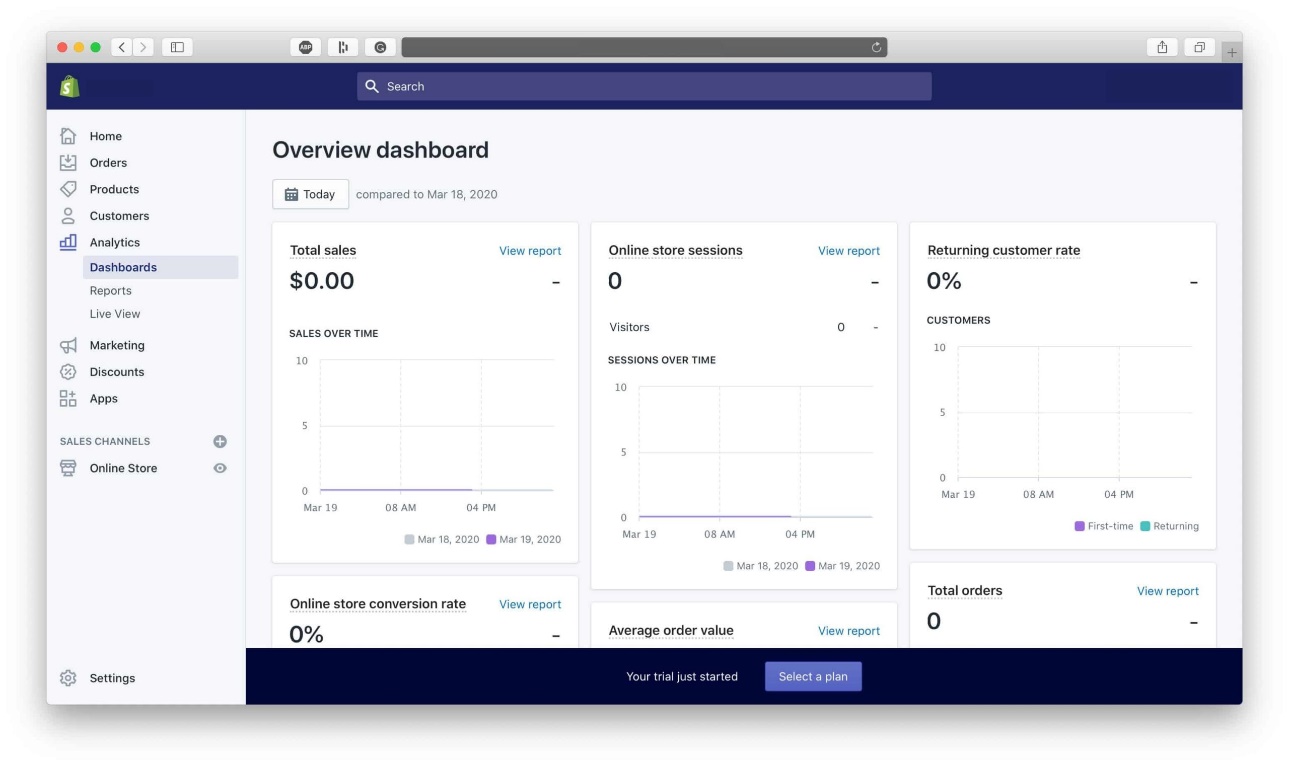


Рисунок 1.1 – Платформа Shopify

Інтерфейс таких платформ інтуїтивно-доступний, що дозволяє користувачам швидко налаштовувати магазин, додавати товари, керувати інвентарем, обробляти замовлення та встановлювати способи доставки і оплати. Важливою перевагою таких платформ є можливість вибору з великої кількості готових тем та макетів, які допомагають створити привабливий та професійний вигляд магазину згідно з потребами власного бренду.

Однією з ключових переваг використання платформи є забезпечення безпеки та захисту від шахрайства. Вони надають захищені платіжні шлюзи, SSL-сертифікати для шифрування передачі даних, а також інструменти для виявлення та запобігання фроду. Це робить платформи для створення магазинів електронної комерції надійними і безпечними для клієнтів, сприяючи збільшенню довіри до бренду.

Інша важлива перевага платформ для створення магазинів електронної комерції полягає в їхній масштабованості. Вони можуть легко розширюватись і адаптуватись до зростаючих потреб бізнесу. За допомогою додаткових плагінів, розширень і інтеграцій з іншими сервісами, можна розширити функціональність магазину і використовувати різноманітні інструменти для маркетингу, аналітики та управління клієнтами.

Окрім того, платформи для створення магазинів електронної комерції забезпечують підтримку і технічну допомогу. Вони надають документацію, онлайн-підтримку, форуми спільноти та інші ресурси, які допомагають вирішувати проблеми і відповідати на питання користувачів. Це особливо корисно для підприємців без значного досвіду веб-розробки, які можуть швидко отримати допомогу і поради щодо роботи магазину.

Розробка власного проекту інтернет-магазину має кілька переваг. Такий підхід дозволяє створити унікальний брендовий образ та дизайн, який відповідає компанії та продуктам. Можна створити індивідуальну корпоративну ідентичність, використовуючи логотипи, шрифти та інші елементи дизайну, що сприятимуть впізнаваності бренду серед конкурентів.

При реалізацій власного інтернет-магазину проекту доступний повний контроль над його функціоналом, розширеннями та налаштуваннями. Тобто можна вибирати рішення для оплати, доставки, аналітики та інших аспектів свого магазину, що дозволяє відповідати конкретним потребам бізнесу.

Зовнішні платформи можуть стягувати комісію з кожного продажу або вимагати місячних платежів за використання. Розробка власного магазину дозволяє уникнути додаткових витрат та збільшити прибутковість за рахунок уникнення залежності від зовнішніх платформ. Адже існує повний контроль над магазином та відсутня залежність від обмежень зовнішніх сервісів.

Захист конфіденційної інформації клієнтів та транзакцій є також надзвичайно важливим чинником. У власному магазині можна встановити власні протоколи безпеки, використовувати шифрування даних та забезпечити високий рівень конфіденційності.

Звичайно, розробка власного проекту інтернет-магазину вимагає більшого зусилля та ресурсів порівняно з використанням зовнішніх платформ. Адже потрібно мати технічні знання або співпрацювати з командою розробників, а також вкласти час в розробку, налаштування та підтримку магазину. Однак, ці інвестиції можуть бути вигідними на довгостроковій основі, забезпечуючи вам більший контроль, гнучкість та можливості для розвитку вашого бізнесу в інтернеті.

1.2 Технічне завдання

1.2.1 Найменування та область застосування

Тема дипломного проекту: Розробка вебсайту магазину аксесуарів «Atna».

Область застосування веб-додатку – надання користувачам можливості купувати цікаві та унікальні аксесуари.

1.2.2 Призначення розробки

Метою створення вебсайту магазину аксесуарів «Atna» є створення зручної та привабливої онлайн-платформи, де користувачі зможуть знайти та придбати різноманітні аксесуари. Основними цілями проєкту є:

* надати користувачам простий та інтуїтивно зрозумілий інтерфейс, де вони зможуть легко знайти потрібні аксесуари;
* організувати можливість фільтрування для забезпечення зручного вибору продуктів;
* надати детальний опис, фотографії та характеристики кожного аксесуару, щоб користувачі мали повну інформацію перед покупкою;
* забезпечити безпеку та надійність збереження облікових даних користувачів, а також безпеку транзакцій. Для цього повинні бути реалізовані такі заходи безпеки, як шифрування даних та безпечний протокол передачі. Крім того, важливо забезпечити надійну інтеграцію з платіжними системами для безпроблемного здійснення оплати;
* забезпечити зручне адміністрування та керування товарами, замовленнями та іншою інформацією.

Загалом, створення вебсайту магазину аксесуарів «Atna» має на меті створити привабливу та функціональну платформу, яка буде задовольняти потреби користувачів та сприятиме успішному функціонуванню бізнесу.

1.2.3 Вимоги до функціоналу вебсайту

При розробці інтернет-магазину аксесуарів «Atna» основні вимоги до функціоналу інтернет-магазину включають:

* наявність каталогу товарів, тобто магазин повинен мати зручний і структурований каталог товарів, де товари будуть розподілені за категоріями;
* кожен товар повинен мати свою сторінку з докладним описом, зображеннями, ціною та іншими характеристиками;
* повинен бути реалізований функціонал кошика та оформлення замовлення: Користувачі повинні мати можливість додавати товари до кошика, керувати кількістю товарів та оформляти замовлення;
* магазин повинен підтримувати систему оплати, яка б забезпечувала зручність для різних клієнтів;
* магазин повинен мати систему реєстрації та авторизації користувачів. Зареєстровані користувачі зможуть переглядати свої замовлення;
* наявні пошукової система, яка б дозволяла користувачам швидко знаходити потрібні товари за ключовими словами. Вона повинна бути швидкою, точною і зручною для навігації.
* магазин повинен мати адаптивний інтерфейс для мобільних платформ. Забезпечення мобільної оптимізації є важливим, оскільки все більше користувачів здійснюють покупки через мобільні пристрої. Магазин повинен мати адаптивний дизайн, який забезпечує зручне відображення та навігацію на різних пристроях;
* важливо мати механізм підтримки клієнтів, такий як онлайн-чат, електронна пошта або телефонна лінія, для відповіді на запитання та вирішення проблем.

1.2.4 Вимоги до програмної документації

Після завершення процесу розробки магазину потрібно підготувати:

* інструкцію з розміщення веб-додатку в Інтернеті, тобто описати процес розміщення веб-додатку на веб-сервері, надати вказівки щодо налаштування доменного імені для належного використання сайту, навести інструкцію щодо налаштування бази даних та з'єднання з нею для збереження інформації про товари та користувачів; описати параметри необхідних залежностей проекту.
* інструкцію з обслуговування та наповнення сайту, тобто опис процесу оновлення та підтримки програмного забезпечення сайту для забезпечення його безперебійної роботи;
* інструкцію оновлення контенту, включаючи додавання нових товарів, зміну описів, оновлення зображень тощо;
* рекомендації з резервного копіювання бази даних та файлів сайту для запобігання втрати даних.
* інструкцію з популяризації та підтримки сайту, тобто опис стратегії популяризації сайту, включаючи SEO-оптимізацію, рекламні кампанії та використання соціальних медіа.
* інструкції щодо взаємодії з клієнтами, включаючи відповіді на запитання та обробку замовлень;
* загальні відомості про можливості вебдодатку, опис його функціоналу та основних можливостей;
* список усіх ендпоінтів (API), які використовуються вебдодатком, разом з описом їх функціональності та параметрів виклику.

1.2.5 Техніко-економічні показники

Розрахунок економічної ефективності та витрат на розробку інтернет-магазину аксесуарів «Atna» повинен бути реалізований у відповідності до методичних вказівок для виконання економічного розділу дипломного проекту.

Ключовими параметрами для такого розрахунку повинні бути:

* час розробки веб-додатку – не більше 120 годин;
* кінцева загальна вартість розробки та впровадження веб-додатку не повинна перевищувати 30000 грн;
* планований термін окупності має бути менше 3-х років.

1.2.6 Стадії та етапи розробки

Розробка інтернет-магазину магазину аксесуарів «Atna» повинна включати такі стадії та етапи:

* аналіз та планування, тобто визначення вимог та функціональності інтернет-магазину, планування архітектури та дизайну, визначення технологій, які будуть використані;
* прототипування та дизайн, тобто створення прототипу вебсайту для визначення його структури та навігації, розробка дизайну, включаючи колірну палітру, типографіку та графічні елементи;
* розробка серверної частини (backend), тобто реалізація середовища розробки та конфігурація сервера, розробка бази даних та моделей даних для зберігання інформації про товари, користувачів, замовлення тощо, створення API для обробки запитів від клієнтської частини та взаємодії з базою даних, реалізація бізнес-логіки, обробка замовлень та операцій з товарами.
* розробка клієнтської частини (frontend), тобто реалізація інтерфейсу користувача з використанням ReactJS, реалізація адаптивного дизайну для різних пристроїв, реалізація динамічних елементів, таких як фільтри, додавання до кошика тощо, інтеграція з REST API для отримання та відправлення даних.
* тестування та оптимізація, тобто виконання тестів для перевірки функціональності, сумісності та безпеки вебсайту, виявлення та виправлення помилок та недоліків, оптимізація продуктивності та швидкодії вебсайту, перевірка на кінцевих користувачах для аналізу досвіду використання;
* реліз та розгортання, тобто підготовка інтернет-магазину до релізу та випуску в продакшн, розгортання на хостинг-платформі, перевірка правильності роботи інтернет-магазину після розгортання;
* підтримка та післярелізний розвиток, тобто забезпечення підтримки та вирішення проблем, які можуть виникнути після релізу, впровадження оновлень та розширення функціональності інтернет-магазину на основі зворотного зв'язку та потреб користувачів.

1.2.7 Порядок тестування та прийому

Порядок тестування та прийому інтернет-магазину аксесуарів «Atna» повинен включати такі етапи:

1. Функціональне тестування, тобто перевірка правильності роботи всіх функцій та функціональних можливостей інтернет-магазину, таких як пошук товарів, додавання до кошика, оформлення замовлення тощо.

2. Тестування сумісності, тобто перевірка роботи інтернет-магазину у різних веб-браузерах (Chrome, Firefox, Safari і т.д.) та пристроях різних розмірів (настільні комп'ютери, ноутбуки, планшети, мобільні телефони), аналіз того, чи веб-сторінки відображаються коректно;

3. Тестування продуктивності та швидкодії, тобто вимірювання часу завантаження сторінок, швидкості відповіді сервера та оптимізація ресурсів (зображень, стилів, скриптів) для забезпечення оптимальної швидкодії роботи веб-сайту.

4. Перевірка наявності заходів безпеки, таких як захист від SQL-ін'єкцій, XSS-атак, CSRF-атак та інших загроз безпеки.

5. Залучення реальних користувачів для виконання тестування зворотного зв'язку, збирання їхніх вражень та пропозицій щодо поліпшення роботи інтернет-магазину.

6. Перевірка інтернет-магазину замовником або стейкхолдерами для остаточного підтвердження його готовності та відповідності вимогам. Для прийому роботи Виконавець повинен представити:

* діючий веб-додаток, який повністю відповідає даному технічному завданню;
* вихідний програмний код, записаний на оптичний носій інформації.

Прийом інтернет-магазину повинен відбуватися комісією з двох осіб, які є представниками сторін Замовника та Виконавця у такій послідовності:

* доповідь Виконавця про виконану роботу;
* демонстрація Виконавцем роботи веб-додатку;
* контрольні випробовування роботи веб-додатку;
* відповіді на запитання і зауваження.

2 РОЗРОБКА ТЕХНІЧНОГО ТА РОБОЧОГО ПРОЄКТУ

2.1 Розробка структури сайту і веб-сторінок

Веб-сайту магазину аксесуарів «Atna» являтиме собою інтернет-магазин. З точки зору кінцевого відвідувача інтернет магазину, його повинен зустрічати відповідний інтерфейс, який повинен містити:

* назву інтернет-магазину;
* зображення (банер), яке давало б представлення про перелік товарів;
* перелік категорій товарів;
* можливість перегляду усіх товарів магазину;
* короткий опис товару.

Якщо користувача зацікавив певний товар магазину, він повинен мати можливість переглянути деталізовану інформацію, яка повинна містити:

* назву товару;
* категорію товару;
* детальний опис;
* ціну.

Якщо користувач вирішить здійснити покупку, то повинна бути передбачена можливість:

* авторизації існуючого користувача;
* реєстрації нового користувача.

Для реєстрації нового користувача магазину повинен бути передбачений функціонал виведення форми із такими полями:

* ім’я користувача;
* електронна адреса користувача;
* пароль користувача.

При заповненні поля електронної пошти, повинна працювати валідація на відповідність введеної інформації електронним адресам.

Для авторизованого користувача повинен бути реалізований функціонал кошика. Тобто авторизований користувач при перегляді детальної інформації про товар повинен мати можливість:

* вибрати потрібну кількість товарів;
* додати вибір до кошика.

В інтерфейсі кошика для користувача повинна бути передбачена можливість:

* зміни кількості товару;
* виведення загальної суми;
* видалення товару.

Якщо користувач вирішить оплатити покупку, то для цього повинна бути передбачена наявність форми для вводу:

* номера карти;
* місця та року карти;
* CVC-коду карти.

Номер карти повинен валідуватися на відповідність існуючим платіжним системам.

Для підтвердження оплати повинна передбачатися відповідна кнопка.

З точки адміністратора повинна бути передбачена можливість:

* створювати товари;
* редагувати властивості товарів;
* видаляти товари.

При створенні товару повинна бути передбачена можливість:

* вводу назви продукту;
* вводу опису продукту;
* вводу ціни продукту;
* вибору категорії продукту;
* завантаження зображень продукту;
* підтвердження створення нового продукту магазину.

Також повинна бути передбачена можливість редагування усіх властивостей створених продуктів магазину.

Адміністратор магазину також повинен мати можливість видалити кожен окремий продукт.

Для того, щоб розділи функціонал рядового користувача магазину від функціоналу адміністратора, повинна бути передбачена можливість реєстрації та авторизації користувачів. Кожен користувач магазину повинен мати можливість вийти із свого облікового запису магазину.

В результаті проаналізованих вимог до функціоналу магазину була створена структурна схема інтерфейсу інтернет-магазину «Atna» (див. рис. 2.1), яка подана на окремому плакаті 2023.ДП.122.421.18.00.00 СС в графічній частині дипломного проекту.

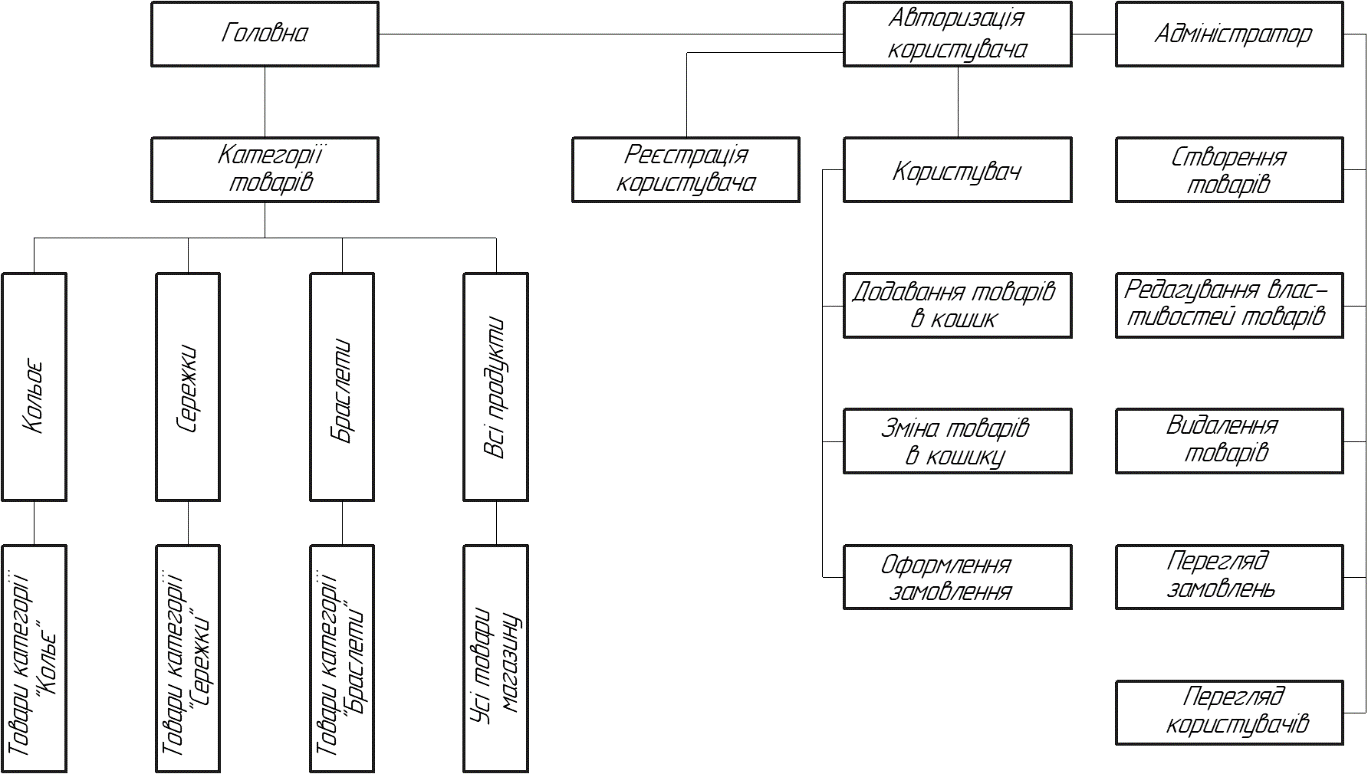


Рисунок 2.1 - Структурна схема інтерфейсу інтернет-магазину «Atna»

2.2 Створення та верстка сторінок сайту

Клієнтську частину інтернет-магазину «Atna» буде реалізовано на основі бібліотеки ReactJS.

Для цього потрібно:

1. Налаштувати середовища розробки, тобто встановіть середовище розробки для роботи з ReactJS [1], таке як Node.js [2] та npm (Node Package Manager) [3], для можливості встановлювати та керувати залежностями проекту;

2. Створити новий проект React з використанням Create React App [4] або аналогічного інструменту, що допомагає налаштувати початкову структуру та налаштування проекту.

3. Розбити сторінку магазину на різні компоненти, такі як заголовок, навігація, список товарів, окремий товар, кошик, форма оформлення замовлення тощо. Визначити їх ієрархію та взаємодію.

5. Розробити компонент, який відповідає за відображення списку товарів з докладними даними, зображеннями, ціною та кнопкою додавання до кошика. Також будемо використовувати пагінацію для полегшення навігації користувачів.

5. Використовувати стейт-менеджмент Redux [5] для збереження даних про вибрані товари, кількість товарів у кошику та інші потрібні дані. Також будемо використовувати події і обробники подій, щоб змінювати стан і взаємодіяти з компонентами.

6. Розробити компоненти форм для оформлення замовлення, введення адреси доставки та інших необхідних даних. Використовувати валідацію форм для перевірки правильності введених даних перед відправкою.

7. Реалізувати функціонал додавання товарів до кошика, підрахунку загальної вартості, видалення товарів з кошика та оформлення замовлення. Забезпечити можливість редагування кількості товарів у кошику.

8. Інтегрувати клієнтську частину з API. Серверна частина інтернет-магазину буде реалізована окремим проектом, тому для клієнтської частини буде використовуватися зовнішні API для отримання даних про товари, ціни, тощо. Будемо інтегрувати API у клієнтську частину за допомогою HTTP-запитів, тобто будемо використовуючи бібліотеки для роботи з API.

Для того, щоб створити новий проект React з використанням Create React App, слід виконати наступні кроки:

1. Установити Node.js, або переконатися, що на комп'ютері встановлено Node.js. Завантажити його можна з офіційного веб-сайту Node.js.

2. Встановити Create React App. Для цього створити папку проекту інтернет-магазину «Atna», відкрити командний рядок або термінал у цій папці і виконати наступну команду, щоб встановити Create React App:

npx create-react-app atna-mern

Ця команда створить нову папку atna-mern з початковою структурою проекту React (див. рис. 2.2).

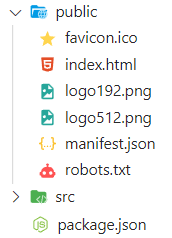


Рисунок 2.2 – Початкова структура проекту клієнтської частини інтернет-магазину «Atna»

Далі переходимо до папки проекту:

cd atna-mern

Запускаємо проект, виконавши наступну команду:

npm start

Ця команда запустить робочий сервер Create React App і відкриє проект у веб-браузері. Тут можна спостерігати зміни в реальному часі при редагуванні файлів.

В папці public буде створено індексний файл проекту index.html, який буде мати один порожній <div> елемент з id=”root” (див. рис. 2.3). В цей елемент будемо поміщати інший контент сторінки засобами JavaScript.

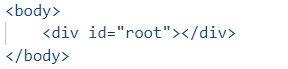


Рисунок 2.3 – Кореневий елемент клієнтської частини

Далі потрібно розпочати розробку React-проекту з використанням Create React App, який забезпечує швидке відновлення, гаряче перезавантаження модулів та інші корисні функції, які полегшують розробку.

При створенні нового проекту за допомогою Create React App повинен бути доступ до Інтернету під час виконання команди, оскільки вона завантажує пакети з мережі.

2.3 Розробка структури бази даних сайту

В якості бази даних інтернет-магазину «Atna» була обрана база даних MongoDB. MongoDB є документною базою даних, яка зберігає дані у форматі BSON (Binary JSON). У MongoDB дані організовуються у колекції, які можна порівняти з таблицями у реляційних базах даних. Кожен документ в MongoDB представляє собою набір пар ключ-значення, аналогічно до рядка в таблиці реляційної бази даних [6].

Ключові поняття та структура записів в базі даних MongoDB:

* колекції (Collections) є аналогом таблиць у реляційних базах даних. Вони містять набір документів, які можуть мати різну структуру, але пов'язані загальною темою чи контекстом;
* документи (Documents) є основною одиницею зберігання даних в MongoDB. Кожен документ представляє собою JSON-подібний об'єкт з набором пар ключ-значення. Ключі - це поля документа, а значення - це відповідні дані, такі як рядки, числа, масиви, вкладені об'єкти тощо;
* поля (Fields) представляють собою окремі ключі в документі. Вони мають імена та значення, які відображають дані.
* кожен документ в колекції може мати унікальний ідентифікатор (Unique Identifiers), який називається \_id. Він служить для однозначної ідентифікації документа в межах колекції;
* можна вкладати один документ всередині іншого документа (Nested Documents). Це називається вкладеною структурою даних і дозволяє представляти більш складену ієрархію даних;
* також можна мати масиви (Arrays) в полях документів. Масиви дозволяють зберігати набір значень у впорядкованій послідовності.

Структура бази даних інтернет-магазину «Atna» буде складатися з таких колекцій (Collections):

* User – для збереження документів користувачів;
* Product – для збереження документів продуктів;
* Order – для збереження документів замовлень.

Колекція User міститиме документи з такими полями:

* name (type: String) – для збереження імені користувача;
* email (type: String) – для збереження електронної адреси користувача;
* password (type: String) – для збереження паролю користувача;
* isAdmin (type: Boolean) – для збереження ролі користувача.
* cart (type: Object) – для збереження параметрів кошика користувача;
* orders (type: Object) – для збереження замовлень користувача.

Колекція Product міститиме документи з такими полями:

* name (type: String) – для збереження назви продукту;
* description (type: String) – для збереження опису продукту;
* price (type: String) – для збереження ціни продукту;
* category (type: String) – для збереження категорії продукту;
* pictures (type: Array) – для збереження параметрів зображень продукту.

Колекція Order міститиме документи з такими полями:

* products (type: Object) – для збереження параметрів продуктів замовлення;
* owner (type: Object) – для збереження власника замовлення;
* status (type: String) – для збереження статусу замовлення;
* total (type: Number) – для збереження загальної суми замовлення;
* count (type: Number) – для збереження кількості замовлень;
* date (type: String) – для збереження дати замовлення;
* address (type: String) – для збереження адреси доставки;
* country (type: String) – для збереження країни.

Алгоритми взаємодії з базою даних будуть описані в розділі 2.4.2.

2.4 Програмування сайту

2.4.1 Написання клієнтської частини

Після створення нового проекту з допомогою Create React App в папці src буде створено файл index.js (src\index.js). В даному файлі потрібно підключити кореневий компонент <App />. Даний компонент буде обгорнутий у менеджер станів компонент, тобто у сам об’єкт станів та у функціонал для його зміни (див. рис. 2.4).

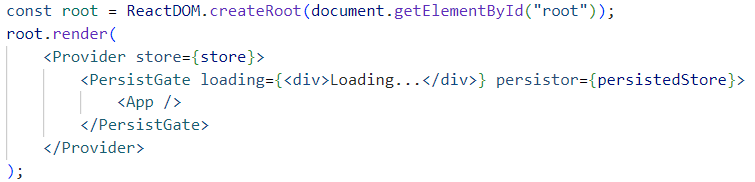


Рисунок 2.4 – Розміщення компонент <App /> у файлі src\index.js

Лістинг коду файлу src\index.js поданий в Додатку А дипломного проекту.

Розробка веб-додатку на ReactJS передбачає використання компонентного підходу [7]. Цей підхід передбачає розбиття веб-інтерфейсу на невеликі, повторно використовувані компоненти, які керують своїм станом і візуалізацією. Кожен компонент може бути розглянутий як окрема "будівельна одиниця", яку можна складати разом, формуючи складніші інтерфейси. Тобто весь інтерфейс розбивається на невеликі компоненти, які виконують конкретні функції. Наприклад, кнопка, форма, заголовок, список тощо.

Компоненти можуть бути вкладені один в одного для створення складніших інтерфейсів.

Компоненти можуть бути повторно використовувані в різних частинах додатку. Це сприяє покращенню продуктивності, зменшенню дублювання коду і підтримці єдиної бази компонентів. Кожен компонент має свій власний стан (state) і незалежність від інших компонентів. Це сприяє легкому тестуванню, відлагодженню і реорганізації компонентів без впливу на інші частини додатку.

Дані передаються вниз по ієрархії компонентів з використанням пропсів (props). Батьківські компоненти можуть передавати дані та функції своїм дочірнім компонентам, що дозволяє оновлювати стан та здійснювати взаємодію між компонентами.

Компоненти оновлюють свій стан через механізм зворотнього зв'язку. Коли стан компонента змінюється, React автоматично оновлює відповідні частини інтерфейсу, що дозволяє створювати реактивні та динамічні додатки.

Усі компоненти пишуться з використанням синтаксису JSX. JSX є розширенням синтаксису JavaScript, яке дозволяє описувати відображення компонентів у вигляді HTML-подібного коду безпосередньо в JavaScript. Це дозволяє легко комбінувати логіку JavaScript з декларативним описом інтерфейсу.

Компонентний підхід в ReactJS сприяє модульності, читабельності і реактивності веб-додатків. Він дозволяє ефективно керувати станом додатку, забезпечувати повторне використання компонентів і швидко реагувати на зміни даних. Компоненти можуть бути розроблені незалежно один від одного, сприяючи розподіленому робочому процесу і співпраці у команді розробників.

У компоненті <App /> (файл src\App.js) підключається компонент <Navigation /> та визначаються доступні маршрути для переходу по інших частинах інтернет-магазину «Atna» (див. рис. 2.5) в залежності від ролі авторизованого користувача.



Рисунок 2.5 – Структура компоненту <App />

Лістинг коду компоненту <App/> (файл src\App.js) поданий в Додатку Б дипломного проекту.

Реалізуємо в структурі каталогів клієнтської частини інтернет-магазину «Atna» (див. рис. 2.6) такий підхід:

* у папку «pages» будемо поміщати сторінки (головної сторінки, авторизації, реєстрації, категорій товарів, адміністративної панелі, створення нового продукту, редагування параметрів продукту, відображення продукту, кошика, замовлень);
* у папку «components» будемо розміщати функціональні компоненти, які будуть використовуватися компонентами з папки «pages» (виведення схожих продуктів, виведення повідомлень про додавання в кошик, навігація, пагінація, перегляд деталей продукту);
* у папку «features» розмістимо функціонал для роботи з менеджером станів компонент;
* у папку «services» розмітимо функціонал для взаємодії з API серверної частини проекту.

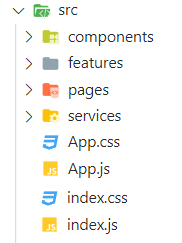


Рисунок 2.6 – Структура каталогу «src»

Функціонал компонента <Navigation /> відповідає за виведення верхньої частини сторінки, тобто навігаційного меню. Логіка цього компоненту побудована таким чином, що дане меню буде мати різний вигляд для:

* неавторизованого користувача (див. рис. 2.7);
* авторизованого покупця (див. рис. 2.8);
* адміністратора магазину (див. рис. 2.9).

Лістинг коду компоненту <Navigation /> поданий в Додатку В.



Рисунок 2.7 – Вигляд компоненту <Navigation /> для неавторизованого користувача

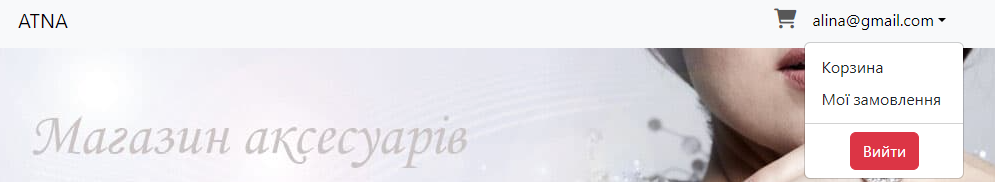


Рисунок 2.8 – Вигляд компоненту <Navigation /> неавторизованого покупця

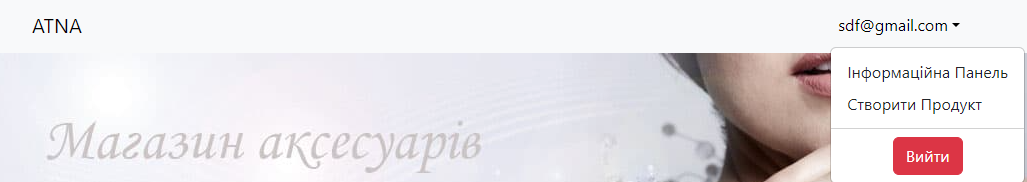


Рисунок 2.9 – Вигляд компоненту <Navigation /> для адміністратора

Для неавторизованого користувача компонент <Navigation /> виводить лінк по маршруту «/login», який в маршрутизаторі (див. рис. 2.5) співставляється з компонентом <Login />. Компонент <Login /> (файл src\pages\Login.js) відповідає за виведення вікна авторизації (див. рис. 2.10 ).

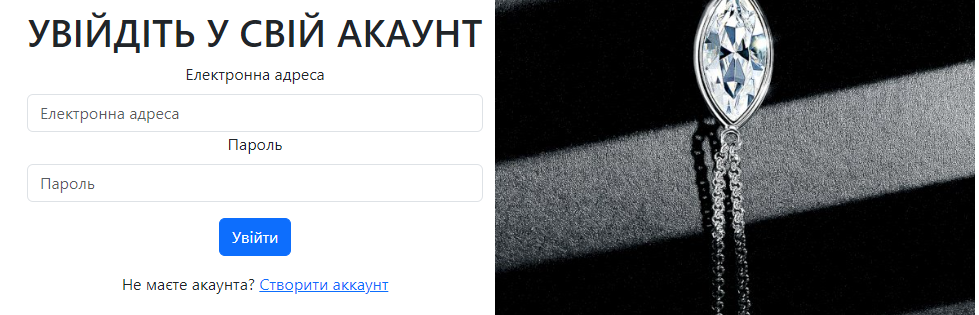


Рисунок 2.10 – Вигляд компоненту <Login />

Лістинг коду компоненту <Login /> поданий в Додатку Г дипломного проекту.

Якщо користувач ще жодного разу не реєструвався в магазині, йому потрібно натиснути посилання «Створити аккаунт» (див. рис. 2.10). Дане посилання в маршутизаторі (див. рис. 2.5) співставляється з компонентом <Signup /> (файл src\pages\Signup.js). Результат відпрацювання даного компоненту поданий на рисунку 2.11.

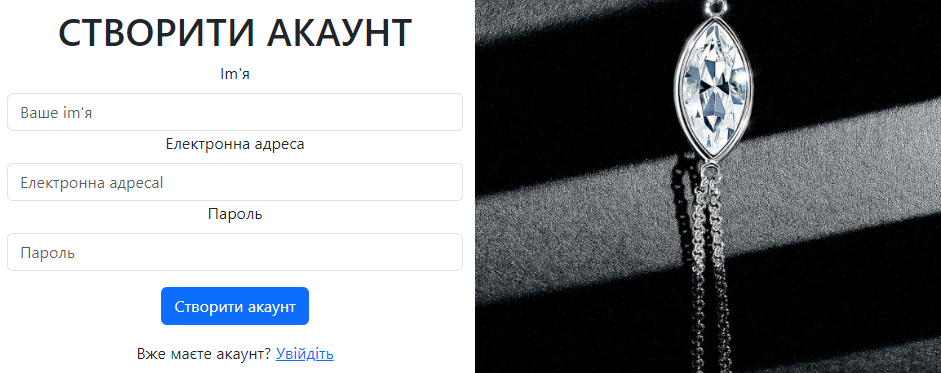


Рисунок 2.11 – Компонент <Signup />

Лістинг коду компоненту <Signup /> поданий в Додатку Д дипломного проекту.

Заповнення форми та натискання кнопки «Створити аккаутн» призведе до спрацювання обробника handleSignup, який визначений у компоненті <appAPI /> (файл src\services\appApi.js). Компонент <appAPI /> являє собою сервіс для відправки HTTP-запитів на серверну сторону веб-додатку. Лістинг коду компоненту < appAPI /> поданий в Додатку Е дипломного проекту.

Якщо не клацнули жодне з посилань маршрутизатора (див. рис. 2.5), буде відображено індексний маршрут, який співставляється з компонентом <Home /> (файл src\pages\Home.js). У даному компоненті реалізований вивід банера, категорій товарів та нових продуктів (див. рис. 2.12) інтернет-магазину «Atna». Лістинг коду компоненту <Home /> поданий в Додатку Є дипломного проекту.

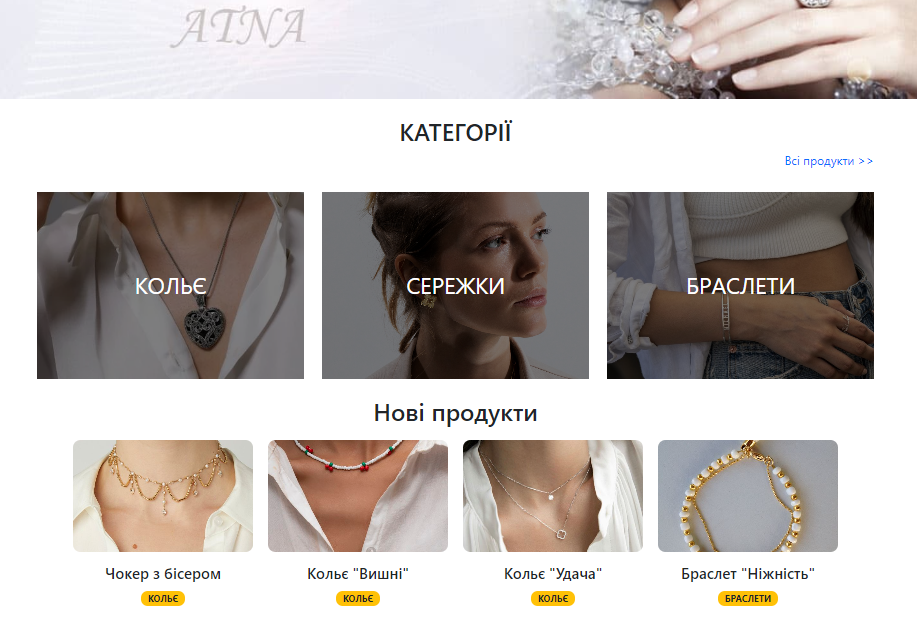


Рисунок 2.12 – Компонент <Home />

Категорії товарів визначаються в масиві categories у файлі src\categories.js.

При виборі певної категорії відбувається відпрацювання маршруту /category/${category.name.toLocaleLowerCase(), який співставляється в маршрутизаторі (див. рис. 2.5) з компонентом <CategoryPage />, код якого розміщений у файлі src\pages\CategoryPage.js. Дане відпрацювання приводить до виведення товарів вибраної категорії (див. рис. 2.13).

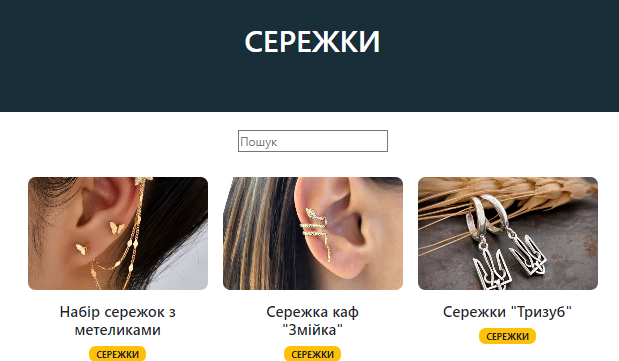


Рисунок 2.13 – Компонент < CategoryPage />

У компоненті <CategoryPage /> визначено поле пошуку, зміна значення якого призводить до відпрацювання змінної searchTerm, значення якої змінюється і передається в контекст, що призводить до спрацювання фільтра productsSearch(), вибірки певних товарів через відповідний сервіс, реалізований у компоненті <appAPI /> (файл src\services\appApi.js). Даний механізм призводить до пошуку та виведення товарів, в назві яких наявний пошуковий термін (див. рис. 2.14).

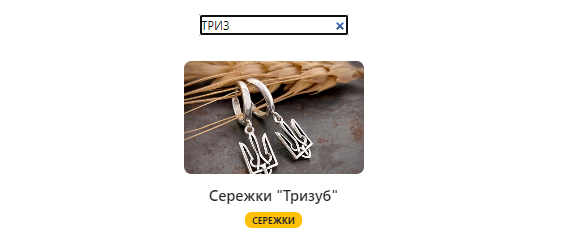


Рисунок 2.14 – Фільтрація продуктів

При кліку на певній позиції товару відпрацьовує функціонал компоненту <ProductPage />, який виводить детальні характеристики продукту та, в залежності від ролі зареєстрованого користувача, або кнопку «Додати до корзини» (див. рис. 2.15), або «Редагувати продукт» (див. рис. 2.16).

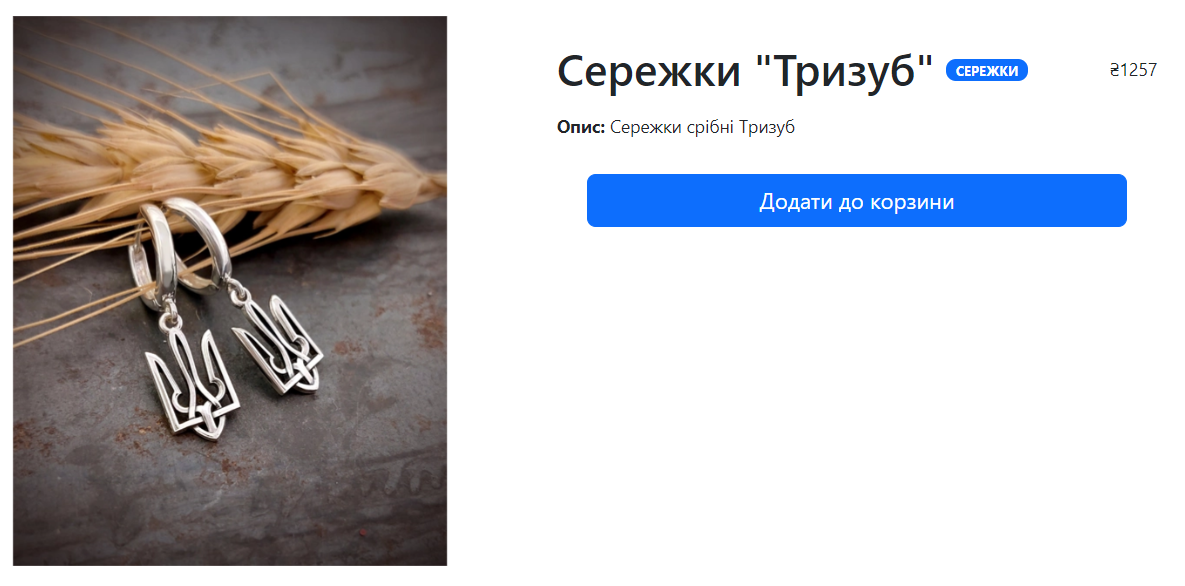


Рисунок 2.15 – Вигляд компоненти <ProductPage /> для авторизованого покупця

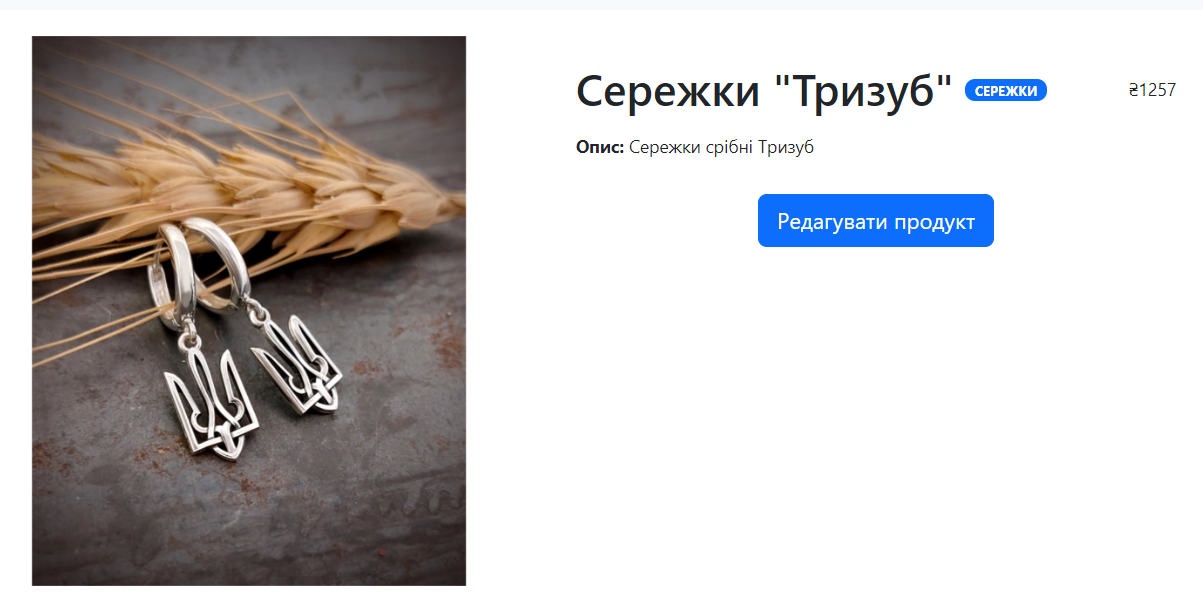


Рисунок 2.16 – Вигляд компоненти <ProductPage /> для адміністратора

Для неавторизованого користувача на будуть виведені жодні кнопки, просто буде відображатися детальна інформація про продукт. Лістинг коду компоненту <ProductPage /> поданий в Додатку Ж дипломного проекту.

Натискання кнопки «Додати до корзини» призводить до зміни стану змінної addToCart, відсилання даних кошика через відповідний сервіс на серверну сторону, виводу відповідного повідомлення (відпрацювання компоненту <ToastMessege /> з файлу src\components\ToastMessage.js, зміни значення змінної user.cart.count у компоненті <Navigation />, та ререндирингу інтерфейсу (див. рис. 2.17).

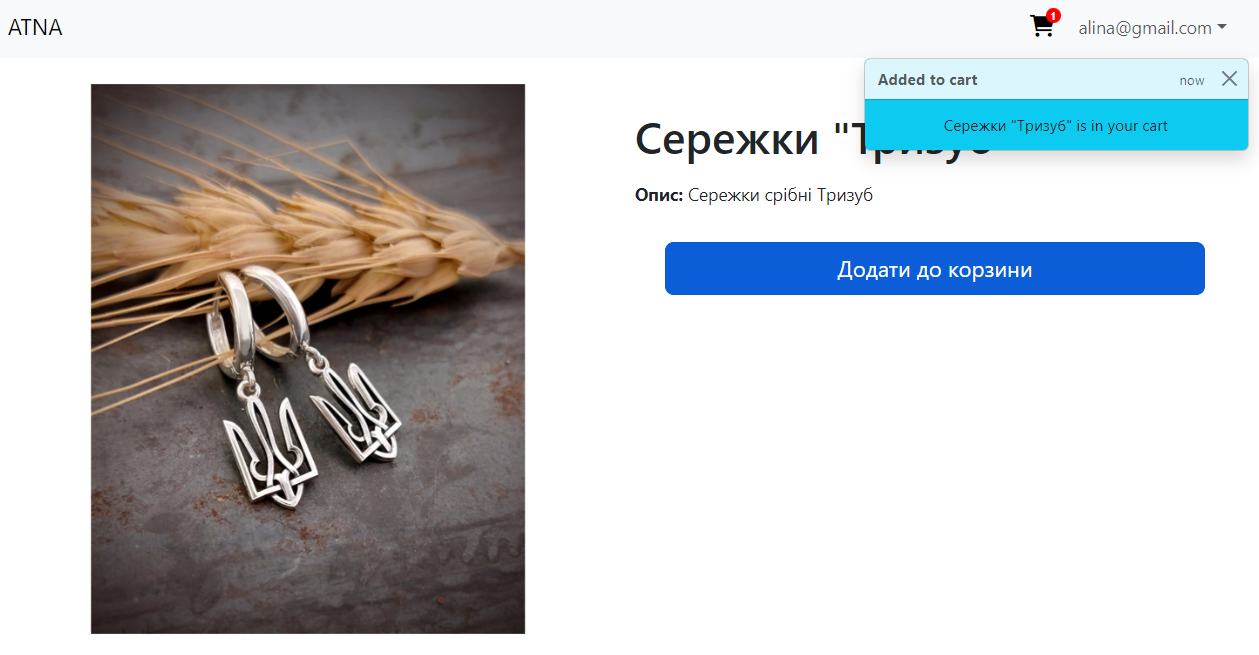


Рисунок 2.17 – Додавання товару в кошик

Натискання піктограми кошика приводить до відпрацювання компоненту <CartPage /> (файл src\pages\CartPage.js). З якого відкривається можливість (див. рис. 2.18):

* змінити кількість товарів у кошику;
* видалити продукти з кошика;
* оплатити замовлення.

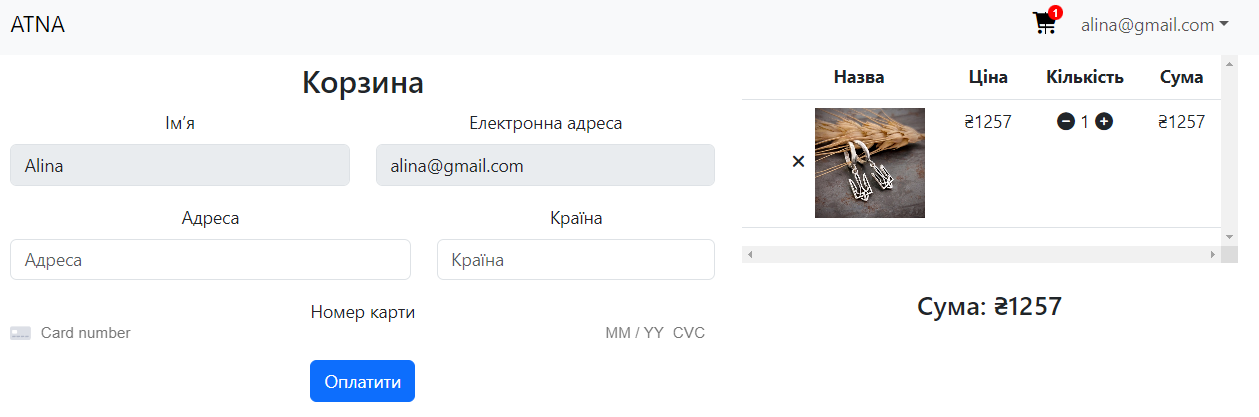


Рисунок 2.18 – Інтерфейс кошика

Лістинг коду компоненту <CartPage /> поданий в Додатку З дипломного проекту.

За виведення форми оплати товару (див. рис. 2.18) відповідає компонент <CheckoutForm/> (файл src\components\CheckoutForm.js). Лістинг коду компоненту <CheckoutForm /> поданий в Додатку И дипломного проекту. Оплата платежів реалізована на основі системи Stripe.

За виведення замовлень клієнта відповідає компонент <ClientsAdminPage /> (файл src\components\ClientsAdminPage.js). Доступитися до замовлень користувача можна з навігаційного меню «Мої замовлення» (див. рис. 2.8).

Якщо авторизований користувач є адміністратором магазину, то він має можливість:

* створювати нові товари;
* редагувати властивості товарів;
* видаляти товари;
* переглядати замовлення;
* переглядати зареєстрованих користувачів.

Для створення нового товару, потрібно зайти в обліковий запис адміністратора. Інструкція того, як змінити роль зареєстрованого користувача на адміністратора подана в розділі 3.2.

Для створення нового продукту магазину потрібно перейти у відповідне навігаційне меню «Створити продукт» (див. рис. 2.9). Функціонал створення нових продуктів реалізований у компоненті <NewProduct /> (файл src\pages\NewProduct.js). У даному компоненті передбачена відповідна форма (див. рис. 2.19), у якій потрібно:

* ввести назву продукту;
* ввести опис продукту;
* ввести ціну продукту;
* вибрати категорію продукту;
* завантажити зображення продукту.

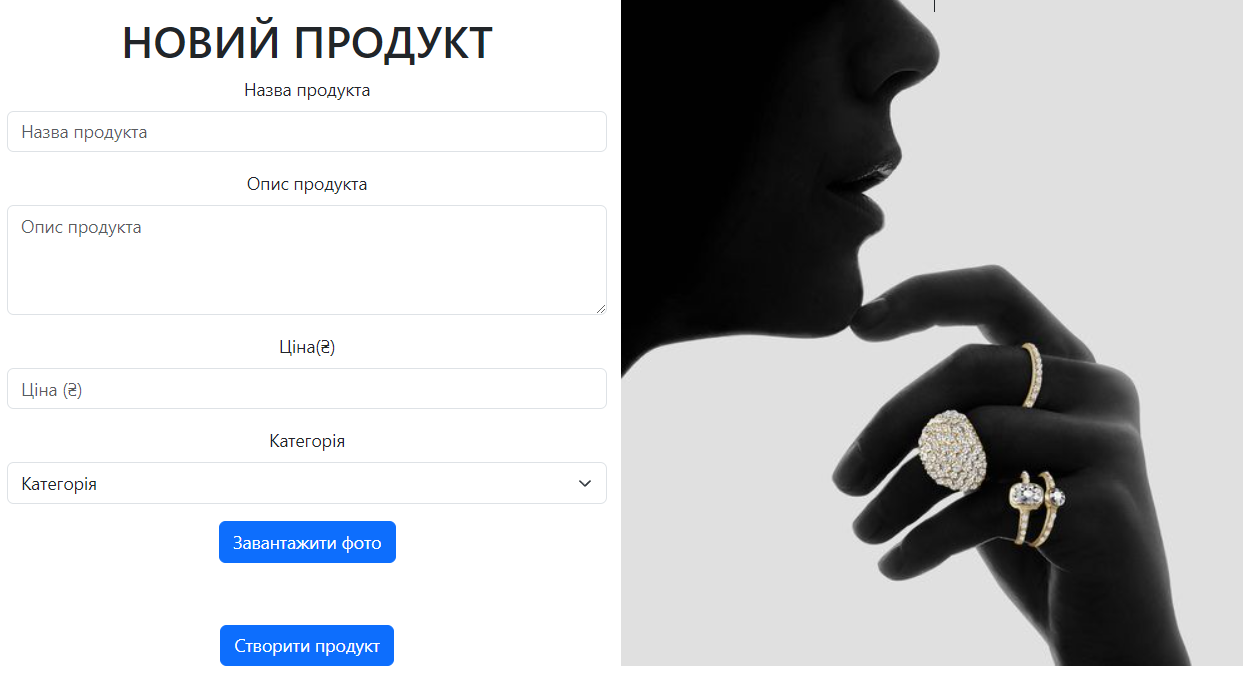


Рисунок 2.19 – Створення нового продукту

Усі зображення продуктів будуть зберігатися у хмарному сервісі cloudinary. Cloudinary - це хмарний сервіс для керування медіаресурсами, який надає широкий спектр можливостей для зберігання, оптимізації, маніпуляції та доставки зображень для веб-додатків. Cloudinary надає зручний API для завантаження зображень, відео та інших медіаресурсів у хмарне сховище [8].

Завантажуватися зображення будемо за допомого налаштування та підключення відповідного пресету (uploadPreset), інтерфейс якого дозволяє завантажувати зображення у форму (див. рис. 2.20)

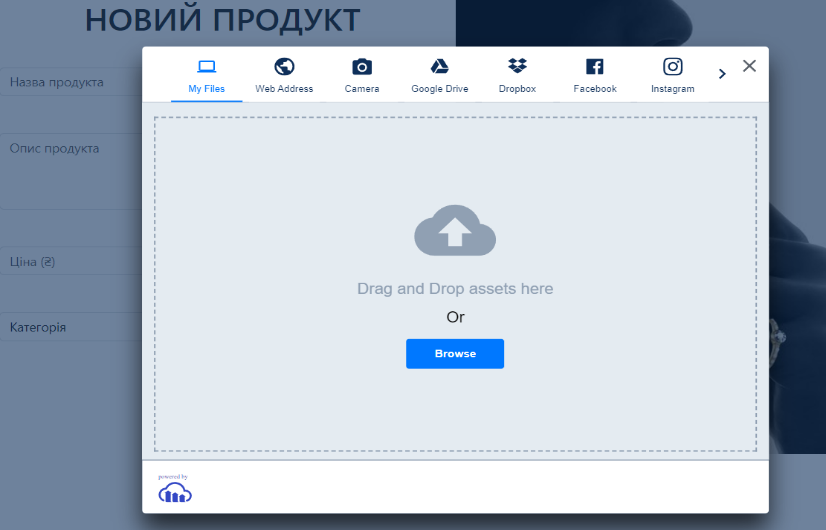


Рисунок 2.20 – Завантаження зображень продуктів

Завантажені зображення будуть розміщені хмарному сервісі cloudinary (див. рис. 2.21).

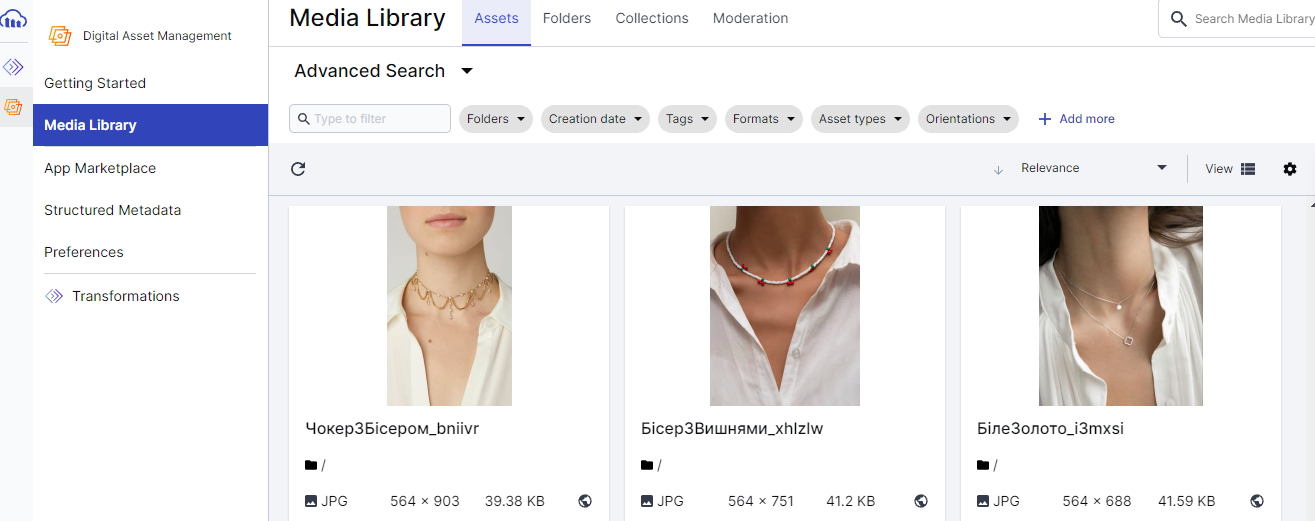


Рисунок 2.21 – Зображення продуктів в cloudinary

Лістинг коду компоненту <NewProduct /> поданий в Додатку І дипломного проекту.

Для доступу до можливостей редагування властивостей товарів та видалення товарів магазину служить відповідне меню «Інформаційна панель», доступне в профілі адміністратора (див. рис. 2.9), з якої відкривається можливість (див. рис. 2.22):

* переглядати, редагувати властивості та видаляти товари магазину;
* переглядати замовлення;
* переглядати користувачів.

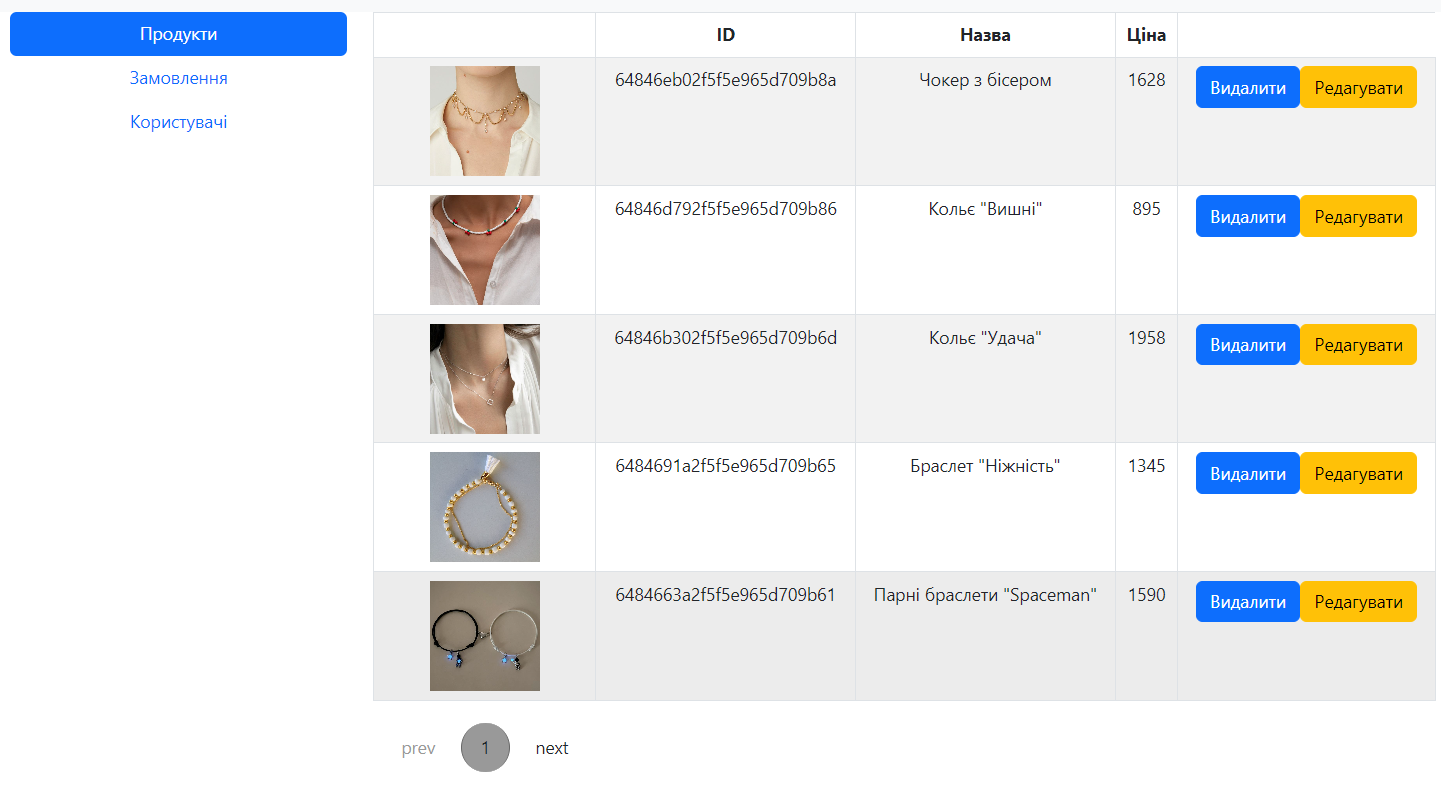


Рисунок 2.22 – Інформаційна панель адміністратора

Функціонал інформаційної панелі реалізований у компоненті <AdminDashboard /> (файл src\pages\AdminDashboard.js). В структуктуру даного компоненту входять такі компоненти:

* <DashboardProducts /> (файл src\components\DashboardProducts.js), у якому реалізований функціонал виведення продуктів магазину;
* <OrdersAdminPage /> (файл src\components\OrdersAdminPage.js) , у якому реалізований функціонал виведення замовлень магазину;
* <ClientsAdminPage /> (файл src\components\ClientsAdminPage.js) , у якому реалізований функціонал виведення клієнтів магазину.

Лістинг коду компоненту <AdminDashboard /> поданий в Додатку Ї дипломного проекту.

У компоненті <DashboardProducts /> підключається функціонал пагінації, визначений у компоненті <Pagination /> (файл src\components\Pagination.js).

При натисканні кнопки «Редагувати» відпрацьовує маршрут “/product/${\_id}/edit”, який співставляється в маршрутизаторі з компонентом <EditProductPage/> (файл src\pages\EditProductPage.js). В результаті відпрацювання коду даного компоненту виводиться вікно редагування властивостей товару (див. рис. 2.23).

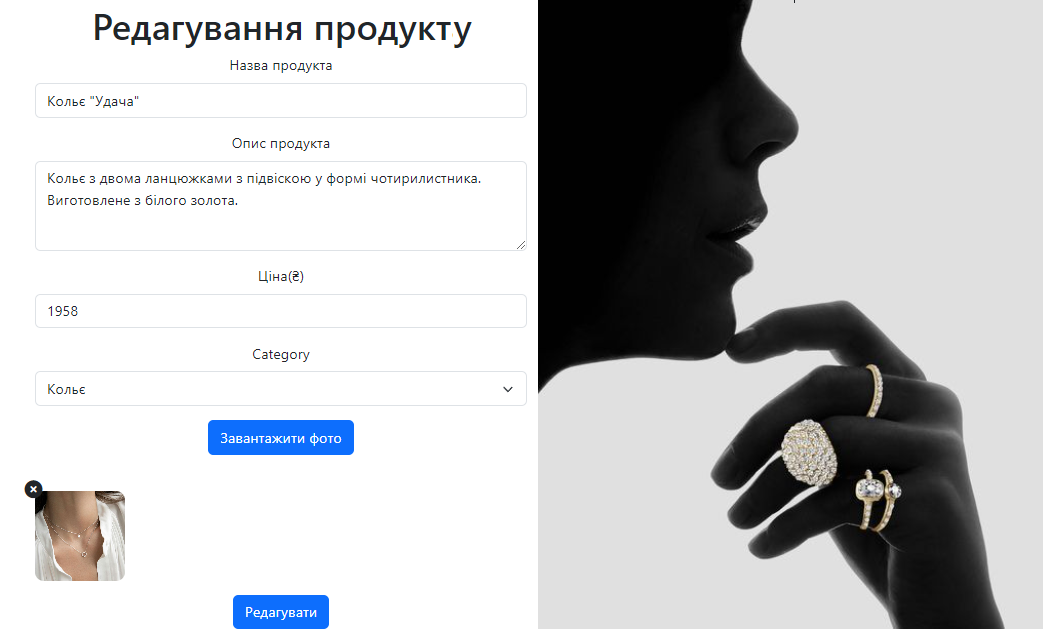


Рисунок 2.23 – Редагування властивостей продукту

Лістинг коду компоненту <EditProductPage /> поданий в Додатку Й дипломного проекту.

2.4.2 Написання серверної частини

Серверна частина інтернет-магазину «Atna» реалізована на основі фреймовку ExpressJS [9]. Для початку потрібно встановити та налаштувати середовище розробки. Тобто встановити Node.js, якщо цього ще не зроблено. Завантажити його можна з офіційного веб-сайту Node.js [2].

Для створення нового проекту потрібно створити нову папку для проекту. Після цього в командному рядку потрібно перейти до створеної папки та ініціалізувати новий проект за допомогою команди npm init. Це створить файл package.json, в якому будуть зберігатись залежності та інші налаштування проекту.

Після цього потрібно встановити Express.js, виконавши команду

npm install express

Це додасть Express.js як залежність до проекту.

Далі потрібно створити новий файл, server.js, в кореневій папці проекту.

У цьому файлі потрібно імпортувати Express.js за допомогою оголошення: const express = require('express').

На наступному етапі потрібно створити новий екземпляр Express.js за допомогою оголошення const app = express().

Далі будемо визначати маршрути, обробники запитів та інші налаштування сервера.

Запустити роботу сервера, можна виконавши команду node server.js або npm start, якщо налаштований скрипт start у файлі package.json.

Після цих кроків сервер на Express.js буде запущено, і буде можливість почати розробляти серверну частину інтернет-магазину, додавати маршрути, обробники запитів, з'єднання з базою даних та інший функціонал, необхідний для проекту.

Окрім express в проекті серверної частини будуть використані такі пакети:

* "bcrypt" – бібліотека хешування паролів, яка надає безпечний спосіб зберігання та перевірки паролів в зашифрованому вигляді. Використовується для забезпечення безпеки паролів користувачів у веб-додатках та системах, де важливо зберігати паролі в захищеному вигляді
* "cloudinary" - бібліотека, яка надає інтерфейс для взаємодії з хмарним сервісом Cloudinary. Будемо використовувати для збереження зображень (images) товарів магазину;
* "cors" – використовується для керування політикою обмеження доступу до ресурсів між різними доменами (origin) у веб-додатках. Оскільки клієнтстка частина та серверна частина будуть реалізовані як окремі проекти, то пакет CORS буде використовуватися для дозволу запитів з клієнтської частини;
* "dotenv" – використовується для налаштування змінних оточення (environment variables) з файлу .env. Змінні оточення використовуються для зберігання конфіденційної або змінної інформації, такої як налаштування бази даних, секретні токени тощо, в окремому файлі, щоб уникнути розміщення цих даних безпосередньо у вихідному коді додатку;
* "mongoose" - об'єктно-реляційна бібліотека (ORM) для Node.js, яка надає спрощений спосіб взаємодії з базою даних MongoDB. Основна мета пакету mongoose полягає в полегшенні розробки бази даних MongoDB, надаючи зручний інтерфейс для виконання операцій створення, читання, оновлення і видалення даних.
* "nodemon" - інструмент розробки для Node.js, який допомагає автоматично перезапускати сервер після змін у вихідному коді. Є корисним під час розробки серверних додатків, коли при зміні коду є потреба бачити оновлення без необхідності вручну перезапускати сервер кожного разу;
* "validator" - є набором функцій, які допомагають валідувати і перевіряти різні типи даних у JavaScript, такі як рядки, електронні адреси, URL-адреси, числа, дати тощо. Забезпечує зручні методи для валідації вхідних даних і перевірки їх на відповідність певним правилам або форматам. Будемо використовувати для валідації електронних адрес.

Дані пакети будуть автоматично прописуватися у залежностях проекту (файл package.json, секція dependencies).

Також в файлі server.js будуть визначені проміжні обробники express.urlencoded({extended: true}) та express.json(), які будуть використовуватися для обробки даних, які надходять в запитах HTTP:

* express.urlencoded({extended: true}) - проміжний обробник, який дозволяє обробляти дані форми, що надсилаються методом POST або PUT, та декодувати їх у розумний об'єкт JavaScript. Параметр extended: true дозволяє обробляти розширені об'єкти форми, які містять гніздовані об'єкти або масиви. Після використання цього проміжного обробника, дані форми будуть доступні через об'єкт req.body у визначеному маршруті Express;
* express.json() - проміжний обробник, який дозволяє обробляти дані, що надходять у форматі JSON у запитах HTTP. Він парсить вхідні дані JSON і розташовує їх у властивості req.body об'єкта запиту (req). Після використання цього проміжного обробника, дані JSON будуть доступні через об'єкт req.body у визначеному маршруті Express.

Обидва ці проміжні обробники дозволяють зручно отримувати та обробляти дані, що надходять у вхідних запитах HTTP, без необхідності ручного розбору та перетворення даних. Вони спрощують роботу з формами та JSON-даними у веб-додатку на базі Express.js.

За обробку запитів HTTP по певних ендпойнтах та визначення поведінки для кожного маршруту реалізуємо маршрутизатор за допомогою функції express.Router().

Додамо маршрути до маршрутизатора (див. рис. 2.24), визначаючи метод HTTP та шляхи, для яких потрібно встановити обробник.

Визначимо маршрути за допомогою методів маршрутизатора, таких як get(), post(), put(), delete(). Кожен метод відповідає HTTP-запиту, який обробляється для певного шляху.

Підключимо маршрутизатор до веб-додатку за допомогою методу app.use().

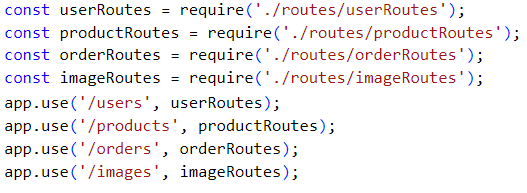


Рисунок 2.24 – Структура маршрутизатора

Лістинг файлу server.js поданий в Додатку К дипломного проекту.

Маршрутизатор визначає спосіб обробки запитів HTTP для конкретного шляху. Визначимо маршрути для різних типів запитів (GET, POST, PUT, DELETE) та будемо виконувати необхідні дії відповідно до цих запитів. Тобто створимо точки доступу до різних частин додатку і будемо обробляти запити до цих маршрутів. Тобто використаємо маршрутизатор для побудови API інтернет-магазину «Atna».

Як видно з рисунку 2.23 структура маршрутизатора буде реалізована у папці routes.

Для обробки запитів, маршрут яких починаються з «/users» буде реалізовано частину маршрутизатора у файлі routes\userRoutes.js. В даному файлі реалізований функціонал для обробки:

* POST-запитів по маршруту '/users/signup' (реєстрація);
* POST-запитів по маршруту '/users/login' (авторизація);
* GET-запитів по маршруту '/users ' (список користувачів);
* GET-запитів по маршруту '/users/:id/orders' (замовлення користувача).

Лістинг файлу routes\userRoutes.js поданий в Додатку Л дипломного проекту.

Для обробки запитів, маршрут яких починаються з «/products» буде реалізовано частину маршрутизатора у файлі routes\productRoutes.js. В даному файлі реалізований функціонал для обробки:

* GET-запитів по маршруту '/products/' (усі продукти);
* POST-запитів по маршруту '/products/'(створення продукту)
* PATCH-запитів по маршруту '/products/:id' (оновлення властивостей продукту);
* DELETE-запитів по маршруту '/products/:id' (видалення продукту);
* GET-запитів по маршруту '/products/:id' (властивості продукту);
* GET-запитів по маршруту '/products/category/:category' (сортування продуктів по категоріях);
* POST-запитів по маршруту '/products/add-to-cart' (додавання продукту в корзину);
* POST-запитів по маршруту '/products/increase-cart' (збільшення числа продуктів в корзині);
* POST-запитів по маршруту '/products/decrease-cart' (зменшення числа продуктів в корзині);
* POST-запитів по маршруту '/products/remove-from-cart (видалення продукту з корзини).

Лістинг файлу routes\productRoutes.js поданий в Додатку М дипломного проекту.

Оскільки зображення на клієнтській стороні завантажуються за домогою відповідного віджету (UploadWidget), то видалення зображень з хмарного сховища реалізовано частину функціоналу маршрутизатора у файлі routes\imageRoutes.js. В даному файлі реалізований функціонал для обробки DELETE-запиту по маршруту '/images/:public\_id' (видалення зображення продукту з хмарного сховища).

Для обробки запитів, маршрут яких починаються з «/orders» буде реалізовано частину функціоналу маршрутизатора у файлі routes\orderRoutes.js. В даному файлі реалізований функціонал для обробки:

* POST-запитів по маршруту '/orders' (створення замовлення);
* GET-запитів по маршруту '/orders' (виведення усіх замовлення)
* PATCH-запитів по маршруту '/orders/:id/mark-shipped' (зміна статусу замовлення).

Лістинг файлу routes\orderRoutes.js поданий в Додатку Н дипломного проекту.

Для роботи з базою даних використовується бібліотека mongoose. Вона надає високорівневий інтерфейс для зручної роботи з базою даних MongoDB.

В проекту використовуються такі методи mongoose [10]:

* connect() - для підключення до бази даних MongoDB за допомогою mongoose. Він приймає рядок підключення до бази даних та набір параметрів;
* Schema() - використовується для визначення схеми (структури) колекції в MongoDB;
* model() - використовується для створення моделі, яка базується на певній схемі.

Підключення до бази даних реалізовано у файлі connection.js

Схеми та моделі визначені у файлах:

* models\User.js – для запису у базу даних інформації про користувачів;
* models\Product.js – для запису у базу даних інформації про продукти;
* models\Order.js – для запису у базу даних інформації про замовлення.

Дані файли подані в Додатку О дипломного проекту.

2.5 Тестування вебсайту

Клієнтська частина інтернет-магазину «Atna» реалізовано на сонові бібліотеки react. Для тестування додатку, написаного на React, можна використовувати різні інструменти і підходи.

Використання Unit-тестів дозволяє перевірити окремі функціональні частини додатку, такі як компоненти, хуки і утиліти. Для цього можна можна використовувати бібліотеки, такі як Jest та React Testing Library, для написання і виконання unit-тестів. Unit-тести допомагають виявити проблеми на ранній стадії розробки і забезпечують стабільність окремих компонентів.

Для проведення Unit-тестування на початку розробки клієнтської частино був реалізований тест компоненту <App/> (файл src\App.test.js). Лістинг коду даного файлу поданий в додатку П дипломного проекту.

Також можна виконати інтеграційні тести, які перевіряють взаємодію між різними компонентами додатку та перевіряють, чи працюють вони разом належним чином. Для цього можна використовувати бібліотеки, такі як Jest, React Testing Library або Enzyme, для написання і виконання інтеграційних тестів.

Приклад інтеграційного тесту поданий в Додатку Р дипломного проекту. У цьому прикладі тестуємо компонент App, рендеримо його за допомогою render функції з React Testing Library, знаходимо елементи за допомогою різних методів, таких як getByTestId, getByRole та getByText, та перевіряємо, чи вони відображені на екрані за допомогою функції toBeInTheDocument. Далі ми виконуємо клік на кнопку за допомогою userEvent.click і перевіряємо, чи змінилось значення тексту після кліку.

Також можна провести UI тести, які перевіряють, чи відповідає візуальний вигляд вашого додатку заданим очікуванням. Ви можете використовувати інструменти, такі як Cypress або Puppeteer, для автоматизації UI тестування. Ці інструменти дозволяють симулювати взаємодію користувача з додатком і перевірити, чи відображається контент правильно та чи працюють взаємодії коректно.

Для тестування серверної частини інтернет-магазину «Atna» реалізованої на Express, ви можете використовувати Unit-тести для маршрутів, тобто можна тестувати кожен маршрут окремо, перевіряючи, чи повертає він очікувані дані або виконує очікувані дії. Для цього можна використовувати бібліотеки, такі як Mocha або Jest, для написання і виконання unit-тестів для маршрутів. Для цього можна створити макети (mocks) для об'єктів req і res, щоб перевірити, як сервер обробляє запити.

Також можна провести інтеграційні тести, які перевіряють взаємодію між різними компонентами сервера, такими як маршрути, контролери, сервіси та база даних. Для цього можна використати бібліотеки, такі як Supertest або Jest, для створення запитів HTTP до сервера та перевірки очікуваних відповідей та стану сервера. Це допоможе перевірити, чи працюють всі компоненти сервера разом належним чином.

Для тестування бази даних можна написати тести для перевірки правильності зберігання, отримання та оновлення даних. Для цього використовують бібліотеку Mongoose, що дозволяє підключатися до бази даних, створювати моделі, виконувати запити та перевіряти результати.

Приклад використання Jest та Supertest для написання простого інтеграційного тесту для маршруту на Express поданий в Додатку С дипломного проекту.

У цьому прикладі ми використовуємо supertest для створення запиту GET до маршруту /api/users на Express-сервері, а потім перевіряємо статус відповіді (200) та формат тіла відповіді (масив).

3 СПЕЦІАЛЬНИЙ РОЗДІЛ

3.1 Інструкція з розміщення сайту в Інтернеті

Інтернет-магазин «Atna» реалізований як веб-додаток , який складається з клієнтської частини, написаної на react, та серверної частини, написаної на express. По суті це два різних проекти, поєднаних через проміжний обробник cors.

Розгортати ці два проекти будемо на платформі render.com. Render - це платформа хостингу, яка спеціалізується на розгортанні веб-додатків із відкритим вихідним кодом. Вона надає простий і зручний інтерфейс для розміщення клієнтського і серверного коду, а також для керування налаштуваннями проекту. Render пропонує інтуїтивно зрозумілий інтерфейс, який дозволяє легко створювати, керувати та масштабувати веб-додатки. Інструкції та документація на платформі допоможуть швидко розпочати роботу. Render підтримує багато мов програмування, фреймворків та інструментів розробки, включаючи Node.js. Дана платформа дозволяє легко масштабувати додатки залежно від потреб. Тут можна налаштувати автоматичне масштабування на основі трафіку або вручну налаштовувати кількість розміщених копій додатку. Підтримується інтеграція з GitHub, тобто можна підключити свій репозиторій на GitHub до Render.com і налаштувати автоматичне розгортання (continuous deployment). При кожному новому коміті коду, Render.com автоматично оновить веб-додаток. Також платформа Render.com надає високу надійність та безпеку для додатків. Код буде розгортатися на інфраструктурі з вбудованими механізмами автоматичного моніторингу, відновлення та захисту від атак. Сервіс Render.com надає широкі можливості налаштування середовища. Тут можна налаштувати змінні середовища, налаштування мережі, баз даних та багато іншого, у відповідності до вимог проекту [11].

Щоб розмістити серверну частину проекту на render.com, слід дотриматися наступних кроків:

1. Зареєструватися на render.com та увійти до свого облікового запису.

2. Створити новий проект на платформі render.com та надати йому назву.

3. Обрати підтримувану платформу "Custom" під час створення проекту.

4. Підключити свій репозиторій з серверною частиною до проекту. Для цього обрати "Connect to GitHub" та вибрати відповідний репозиторій. Надати render.com доступ до репозиторію.

5. Налаштувати основні параметри проекту, такі як кількість розміщених копій, регіон розгортання тощо.

6. Відкрити файл render.yaml (або render.yml) у кореневій директорії серверного проекту. Якщо цього файлу немає, створити його. Додайте наступну конфігурацію:

buildCommand: npm install && npm run build

startCommand: npm start

Тобто вказуємо команду npm install для встановлення залежностей, команду npm run build для збірки серверного коду та команду npm start для запуску сервера.

Зберегти зміни у файлі render.yaml та надіслати їх у репозиторій на GitHub.

7. Натиснути кнопку "Deploy" на платформі render.com, щоб розпочати процес розгортання серверної частини. Render.com автоматично збудує та розгорне серверний проект.

Після успішного розгортання, буде отримана URL-адреса, за допомогою якої можна звертатися до сервера.

Для розміщення клієнтської частини спочатку потрібно зробити збірку (build) клієнтської частини, написаної на React. Для цього слід дотриматися таких кроків:

1. Відкрити командний рядок або термінал у кореневій директорії клієнтської частини.

2. Виконати команду для збірки проекту:

npm run build

Ця команда запустить скрипт build, визначений у файлі package.json. React виконає збірку проекту та згенерує оптимізовані версії коду, які готові для розгортання на сервері. Після завершення збірки з'явиться нова тека з назвою build, що містить оптимізований код клієнтської частини. Цю збірку проекту React будемо розгортати на render.com

3. Створити новий проект на render.com, надайте йому назву і обрати підтримувану платформу "Static" під час створення проекту.

4. Підключити свій репозиторій з клієнтською частиною до проекту, аналогічно до серверної частини. Вибрати "Connect to GitHub" та надати доступ до репозиторію.

5. Налаштуйте основні параметри проекту, такі як кількість розміщених копій, регіон розгортання тощо.

6. Відкрити файл render.yaml (або render.yml) у кореневій директорії вашого клієнтського проекту. Якщо цього файлу немає, створити його. Додати наступну конфігурацію:

buildCommand: npm install && npm run build

publishDirectory: path/to/build/folder

Тобто вказуємо команду npm install для встановлення залежностей, команду npm run build для збірки клієнтського коду та шлях до теки зі збудованими файлами за допомогою параметра publishDirectory.

7. Зберегти зміни у файлі render.yaml та надіслати їх у репозиторій на GitHub.

8. Натиснути кнопку "Deploy" на платформі render.com, щоб розпочати процес розгортання вашої клієнтської частини (див. рис. 3.1). Render автоматично збудує та розгорне клієнтський проект.

Після успішного розгортання Render повідомить URL-адресу, за допомогою якої можна звертатися до клієнтського додатку.

В результаті розгортання клієнтської частини було отримано URL-адресу: https://atna-menr.onrender.com

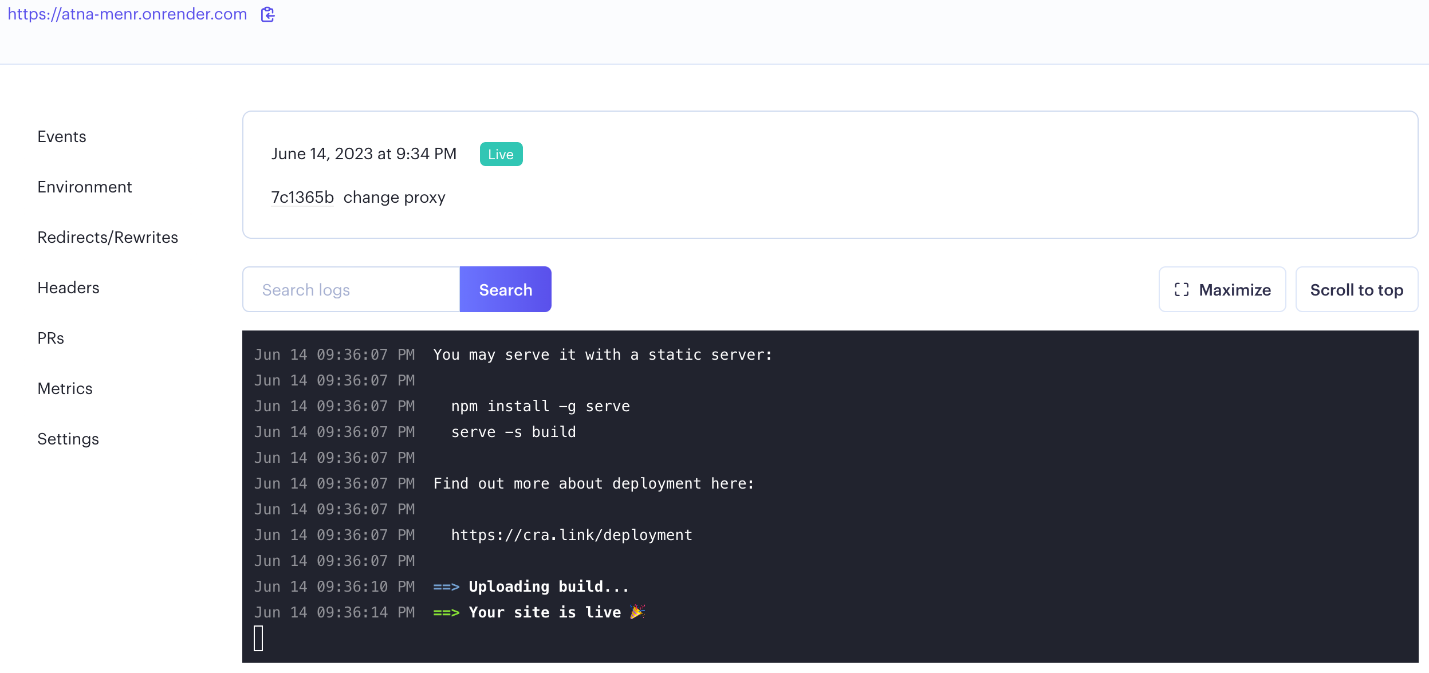


Рисунок 3.1 – Розгортання клієнтської частини додатку

Після окремого розміщення клієнтської та серверної частини інтернет магазину «Atna» на платформі render.com за допомогою проміжного обробника CORS можна з'єднати дві частини і забезпечити взаємодію між ними.

3.2 Інструкція з обслуговування та наповнення сайту

Для наповнення інтернет-магазину «Atna» продуктами розроблений відповідний функціонал, доступний з профіля адміністратора.

Для створення нових продуктів магазину потрібно скористатися функціоналом компоненту <NewProduct /> (див. рис. 2.18). Тобто перейти у відповідне навігаційне меню «Створити продукт» (див. рис. 2.9) і у відповідній формі ввести та вибрати назву продукту, опис продукту, ціну продукту, категорію продукту, зображення продукту.

Для редагування та видалення продуктів потрібно відкрити меню «Інформаційна панель», доступне в профілі адміністратора (див. рис. 2.9) і вибрати відповідну дію (див. рис. 2.21).

Функціонал інтернет-магазину «Atna» не передбачає створення користувача з роллю «Адміністратор». Змінити роль будь-якого зареєстрованого користувача на «Адміністратор» можна безпосередньо зміною відповідного параметру у базі даних, яка розміщена у хмарному сервісі, доступитися до якого можна за допомогою функціоналу MongoDB Atlas. MongoDB Atlas - це повністю керований хмарний сервіс бази даних, який надає можливість легко розгортати, керувати і масштабувати MongoDB - одну з найпопулярніших NoSQL баз даних [12]. Для цього потрібно:

1. Відкрити колекцію users;

2. В документі відповідного профілю, який потрібно перевести у роль «Адміністратор», змінити значення властивості isAdmin з «false», яке встановлюється по замовчуванню (див. рис. 3.2), на «true» (див. рис. 3.3).



Рисунок 3.2 – Обліковий запис користувача у базі даних



Рисунок 3.3 – Обліковий запис адміністратора у базі даних

3.3 Інструкція з популяризації та підтримки сайту

Існує кілька ефективних способів популяризувати інтернет-магазин і залучити більше клієнтів. Для цього потрібно:

1. Визначити свою цільову аудиторію та розробити маркетингові стратегії, спрямовані саме на цю групу. Проаналізувати конкурентів, визначити свої унікальні продажні пропозиції і розробити маркетинговий план.

2. Зробіть інтернет-магазин привабливим і зручним для користувачів. Розробити зручну навігацію, привабливий дизайн, інтуїтивно зрозумілі форми замовлення і забезпечити швидку та зручну оплату.

3. Виконати пошукову оптимізацію інтернет-магазину «Atna», щоб забезпечити його високу видимість в пошукових системах. Для цього потрібно використовувати ключові слова, оптимізувати метатеги, створити якісний контент і розробити стратегію посилань.

4. Активно використовувати соціальні медіа для просування інтернет-магазину «Atna». Створювати цікавий контент, який зацікавить цільову аудиторію. Використовувати блогінг, гостьові статті, відео та інші формати контенту для привертання уваги до інтернет-магазину.

5. Розглянути можливість запуску рекламних кампаній, таких як Google Ads, Facebook Ads або Instagram Ads

6. Збирати адреси електронної пошти від клієнтів і створювати ефективні email-кампанії для просування продуктів та послуг магазину. Надсилати персоналізовані пропозиції, знижки, новини та інформаційні бюлетені, щоб залучити і утримувати клієнтів.

7. Запросити своїх задоволених клієнтів залишати відгуки про свої покупки та досвід користування магазином. Ці позитивні відгуки можуть привернути увагу нових клієнтів та підвищити довіру до магазину.

8. Використовувати інструменти аналітики, такі як Google Analytics, для вивчення поведінки користувачів у інтернет-магазині та ідентифікації можливих зон покращень. Оптимізувати інтернет-магазин та маркетингові кампанії на основі зібраних даних для забезпечення більш ефективного просування.

Клієнтська частина інтернет-магазину «Atna» реалізована на основі бібліотеки ReactJS. SEO-оптимізація для односторінкових додатків (SPA - Single Page Applications), таких як додатки на основі ReactJS, вимагає особливого підходу через їх динамічну природу. Важливі кроки для SEO-оптимізації інтернет-магазину «Atna»:

використовувати роутер, який підтримує обробку маршрутів на стороні клієнта (React Router). Привести URL-адреси до семантично-значущих назв, які відображають структуру контенту інтернет-магазину;

розглянути можливість використання SSR (Server-Side Rendering), яке дозволяє генерувати HTML-код на сервері перед відправкою його на клієнт. Це допомагає покращити індексацію пошуковими системами, оскільки вони можуть бачити повний HTML-код з контентом;

забезпечити швидке завантаження, оскільки швидкість завантаження є важливим фактором для SEO. Зменшити розмір файлів, використовувати кешування, оптимізувати зображення та інші ресурси;

Також потрібно створити карту сайту інтернет-магазину. Карта сайту є важливим елементом SEO-оптимізації, особливо для веб-додатків на основі ReactJS та інших SPA. Карта сайту - це файл або сторінка, яка містить структуровану ієрархію всіх доступних сторінок та ресурсів вашого сайту. Вона дозволяє пошуковим системам більш ефективно сканувати та індексувати ваш контент [13].

Для SPA на основі ReactJS, які мають динамічну природу, існують кілька підходів до створення карт сайту:

* використовуйте JavaScript на стороні клієнта, щоб динамічно генерувати карту сайту під час завантаження інтернет магазину, тобто сканувати структуру маршрутів (наприклад, за допомогою React Router) і створити відповідний XML-файл або сторінку, яка містить всі доступні маршрути;
* скористатися інструментами генерації карт сайту які підтримують SPA. Наприклад, можете скористатися бібліотекою react-router-sitemap, яка автоматично генерує карту сайту на основі структури маршрутів React Router.

4 ЕКОНОМІЧНИЙ РОЗДІЛ

Метою економічної частини кваліфікаційної роботи є здійснення економічних розрахунків, спрямованих на визначення економічної ефективності проектування веб-сайту магазину аксесуарів «Atna» і прийняття рішення щодо його подальшого розвитку та впровадження або ж недоцільність проведення відповідної розробки.

Для розрахунку вартості НДР необхідно виконати наступні етапи:

* описати технологічний процес розробки із зазначенням трудомісткості кожної операції;
* визначити суму витрат на оплату праці основного і допоміжного персоналу, включаючи відрахування на соціальні заходи;
* визначити суму матеріальних затрат;
* обчислити витрати на електроенергію для науково-виробничих цілей;
* розрахувати транспортні витрати;
* нарахувати суму амортизаційних відрахувань;
* визначити суму накладних витрат;
* скласти кошторис та визначити собівартість НДР;
* розрахувати ціну НДР;
* визначити економічну ефективність та термін окупності продукту.

4.1 Визначення стадій технологічного процесу та загальної тривалості проведення НДР

Для визначення загальної тривалості проведення науково-дослідних робіт доцільно дані витрат часу по окремих операціях технологічного процесу звести у таблицю 4.1.

## Таблиця 4.1 - Середній час виконання НДР та стадії (операції) технологічного процесу

|  |  |  |  |
| --- | --- | --- | --- |
| №  п/п | Назва операції (стадії) | Виконавець | Середній час вико-нання операції, год. |
| 1. | Підготовка | Керівник проекту | 7 |
| 2 | Розробка алгоритму веб-додатку | Веб-розробник | 7 |
| 3 | Написання текстів веб-додатку | Веб-розробник | 73 |
| 4 | Тестування веб-додатку | Лаборант | 15 |
| 5 | Розміщення веб-додатку на хостингу | Веб-розробник | 5 |
| 6 | Створення інструкції по експлуатації веб-додатку | Лаборант | 5 |
|  | Р а з о м | – | 112 |

Сумарний час виконання операцій технологічного процесу становить 112 годин.

4.2 Визначення витрат на оплату праці та відрахувань на соціальні заходи

Відповідно до Закону України “Про оплату праці” заробітна плата – це “винагорода, обчислена, як правило, у грошовому виразі, яку власник або уповноважений ним орган виплачує працівникові за виконану ним роботу”.

Розмір заробітної плати залежить від складності та умов виконуваної роботи, професійно-ділових якостей працівника, результатів його праці та господарської діяльності підприємства. Заробітна плата складається з основної та додаткової оплати праці.

Основна заробітна плата розраховується за формулою:

*Зосн.=Тс⋅ Кг*, (4.1)

де *Тс* – тарифна ставка, грн. (приймаємо для керівника – 98 грн./год, веб-розробника – 90 грн./год., лаборанта – 55 грн./год. );;

*Кг* – кількість відпрацьованих годин.

Отже основна заробітна плата для:

керівника проекту *Зосн2*= 98\*7= 686,00 грн.

веб-розробника *Зосн3*= 90\*85= 7650,00 грн.

лаборанта *Зосн4*= 55\*20= 1100,00 грн.

Сумарна основна заробітна плата становить:

*Зосн*=686,00+7650,00+1100,00=9436,00 грн.

Додаткова заробітна плата становить 10–15 % від суми основної заробітної плати.

*Здод.= Зосн.⋅ Кдопл.*, (4.2)

де *Кдопл.*– коефіцієнт додаткових виплат працівникам.

*Здод.=* 9436,00 \*0,15=1415,40 грн.

Звідси загальні витрати на оплату праці (Во.п.) визначаються за формулою:

*Во.п.*= *Зосн.+ Здод.*, (4.3)

*Во.п.*= 9436,00 +1415,40 = 10851,40 грн.

Крім того, слід визначити відрахування на заробітну плату:

* єдиний соціальний внесок – 22 %.

Отже, сума відрахувань на соціальні заходи буде становити:

*Вc.з.=ФОП ⋅ 0,22*, (4.4)

де *ФОП* – фонд оплати праці, грн.

*Вc.з.=* 10851,40 \*0,22=2387,31 грн.

Проведені розрахунки зведемо у наступну таблицю 4.2.

Таблиця 4.2 - Зведені розрахунки витрат на оплату праці

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| №  п/п | Категорія працівників | Основна заробітна плата, грн. | | | Додатко-ва заробітна плата, грн. | Нарах. на ФОП, грн. | Всього витрати на оплату праці, грн.  6=3+4+5 |
| Тарифна ставка, грн. | К-сть від-працьов. год. | Фактично нарах. з/пл., грн. |
| А | Б | 1 | 2 | 3 | 4 | 5 | 6 |
| 1 | Керівник проекту | 98 | 7 | 686,00 | 102,90 | - | - |
| 2 | Веб-розробник | 90 | 85 | 7650,00 | 1147,50 | - | - |
| 3 | Лаборант | 55 | 20 | 1100,00 | 165,00 | - | - |
|  | Разом | - | - | 9436,00 | 1415,40 | 2387,31 | 13238,71 |

4.3 Розрахунок матеріальних витрат

Матеріальні витрати визначаються як добуток кількості витрачених матеріалів та їх ціни:

*МВі= qi ⋅ рі* , (4.5)

де qi – кількість витраченого матеріалу і-го виду;

рі – ціна матеріалу і-го виду.

Звідси, загальні матеріальні витрати можна визначити:

*Зм.в.=Σ МВі* (4.6)

*Зм.в.*=215+192+280= 687 грн.

Проведені розрахунки занесемо у таблицю 4.3.

Таблиця 4.3 - Зведені розрахунки матеріальних витрат

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| №  п/п | Найменування матеріальних ресурсів | Од. виміру | Факт. витр. матеріалів | Ціна 1-ці, грн. | Заг. сума витрат, грн. |
| 1 | USB флеш-накопичувач 32GB Transcend JetFlash 700 32GB | шт. | 1 | 215 | 215 |
| 2 | Друк документації | арк. | 120 | 1,6 | 192 |
| 3 | Друк плакатів | листів | 4 | 70 | 280 |
|  | Р а з о м |  |  |  | 687,00 |

4.4 Розрахунок витрат на електроенергію

Затрати на електроенергію 1-ці обладнання визначаються за формулою:

*Зе=W⋅ T⋅ S*, (4.7)

де *W* – необхідна потужність, кВт;

*T* – кількість годин роботи обладнання;

*S* – вартість кіловат-години електроенергії.

Усі процеси розробки та впровадження веб-проекту виконуються на одному ПК. Витрати на електроенергію для цього комп’ютера обчислимо, взявши за основу час виконуваних робіт (згідно табл.4.1) і споживану потужність комп’ютера – 0,35 кВт/год.

Зе=0,35\*112\*1,68=65,86 грн.

4.5 Визначення транспортних затрат

Транспортні витрати слід прогнозувати у розмірі 8–10 % від загальної суми матеріальних затрат.

*Тв=Зм.в.⋅* 0,08 … 0,1, (4.8)

де *ТВ* – транспортні витрати.

*Тв=*687,00\*0,08=54,96 грн

4.6 Розрахунок суми амортизаційних відрахувань

Характерною особливістю застосування основних фондів у процесі виробництва є їх відновлення. Для відновлення засобів праці у натуральному виразі необхідне їх відшкодування у вартісній формі, яке здійснюється шляхом амортизації.

Амортизація – це процес перенесення вартості основних фондів на вартість новоствореної продукції з метою їх повного відновлення.

Для визначення амортизаційних відрахувань застосовуємо формулу:

, (4.9)

де А – амортизаційні відрахування за звітний період, грн.;

БВ – балансова вартість групи основних фондів на початок звітного періоду, грн.;

НА – норма амортизації, %.

Для проектування даної комп’ютерної мережі використовується один комп’ютер (вартість якого становить 31550 грн.), який працює 112 години.

А= 31550\*0,04\*112/150=942,29 грн.

4.7 Обчислення накладних витрат

Накладні витрати пов’язані з обслуговуванням виробництва, утриманням апарату управління підприємства та створення необхідних умов праці.

В залежності від організаційно-правової форми діяльності господарюючого суб’єкта, накладні витрати можуть становити 20–60 % від суми основної та додаткової заробітної плати працівників.

*Нв=Во.п. ⋅* 0,2 … 0,6, (4.10)

де НВ – накладні витрати.

*НВ*=10851,40\*0,2=2170,28 грн.

4.8 Складання кошторису витрат та визначення собівартості НДР

Результати проведених вище розрахунків зведемо у таблицю 4.4.

Таблиця 4.4 - Кошторис витрат на НДР

|  |  |  |
| --- | --- | --- |
| Зміст витрат | Сума, грн. | В % до заг. суми |
| Витрати на оплату праці | 10851,4 | 63,24 |
| Відрахування на соціальні заходи | 2387,31 | 13,91 |
| Матеріальні витрати | 687 | 4,00 |
| Витрати на електроенергію | 65,86 | 0,38 |
| Транспортні витрати | 54,96 | 0,32 |
| Амортизаційні відрахування | 942,29 | 5,49 |
| Накладні витрати | 2170,28 | 12,65 |
| Собівартість | 17159,10 | 100,00 |

Собівартість (СВ) НДР розрахуємо за формулою:

*Св*= *Во.п.*+ *Вс.з.*+ *Зм.в.*+ *Зе* + *Тв* +*А+ Нв* (4.11)

*Св*=10851,4+2387,31+687+65,86+54,96+942,29+2170,28=17159,10 грн.

4.9 Розрахунок ціни НДР

Ціну НДР можна визначити за формулою:

, (4.12)

де Ррен. – рівень рентабельності,%;

Ві.н. – вартість носія інформації, грн. ;

ПДВ – ставка податку на додану вартість, (20 %).

*Ц*=17159,10\*(1+0,25)\*(1+0,2)= 25738,65 грн.

4.10 Визначення економічної ефективності і терміну окупності капітальних вкладень

Ефективність виробництва – це узагальнене і повне відображення кінцевих результатів використання робочої сили, засобів та предметів праці на підприємстві за певний проміжок часу.

Обчислимо значення прибутку за наступною формулою:

*П=Ц - СВ*(4.13)

*П*=25738,65-17159,10=8579,55 грн.

Для визначення ефективності продукту розраховують чисту теперішню вартість (ЧТВ) і термін окупності ().

, (4.14)

де КВ – затрати на проект;

Гп – грошовий потік за t–ий рік;

 – відповідний рік проекту;

 - величина дисконтної ставки (10…15%).

*ЧТВ* = −17159,10+8579,55/(1+0,1)+ 8579,55/(1+0,1)2 + 8579,55/(1+0,1)3=

=−17159,10+7799,59+7090,54+6445,94=4176,97 грн.

Якщо ЧТВ ≥ 0, то проект може бути рекомендований до впровадження.

Термін окупності визначається за формулою:

, (4.15)

де *ТПВ* – період до повного відшкодування витрат, років;

*НВ* – невідшкодовані витрати на початок року, грн.;

*ГПР* – грошовий потік на початок року, грн.

Тепер необхідно знайти значення початкових інвестицій (Нв) за формулою:

2268,97

*ТОК=*2+2268,97/8579,55=2,3

Таблиця 4.5 - Економічні показники НДР

|  |  |  |
| --- | --- | --- |
| № п/п | Показник | Значення |
| 1. | Собівартість, грн. | 17159,10 |
| 2. | Плановий прибуток, грн. | 8579,55 |
| 3. | Ціна, грн. | 25738,65 |
| 4. | Чиста теперішня вартість, грн | 4176,97 |
| 5. | Термін окупності, рік | 2,3 |

Загальна вартість розробленого веб-додатку становить 25738 грн.65 коп. Чистої теперішньої вартості ми досягаємо на третій рік, що є позитивним показником. Тому проводити проектні роботи варто і вкладені кошти окупляться за 2,3 року.

5 ОХОРОНА ПРАЦІ, ТЕХНІКА БЕЗПЕКИ ТА ЕКОЛОГІЧНІ ВИМОГИ

5.1 Гігієнічні вимоги до організації і обладнання робочих місць з ВДТ

Розглянемо якими документами необхідно користуватися при обладнанні робочих місць користувачів ПК, як правильно і на якій відстані необхідно знаходитися користувачу ПК від екрану ВДТ.

Обладнання і організація робочого місця з ВДТ мають забезпечувати відповідність конструкції всіх елементів робочого місця та їх взаємного розташування ергономічним вимогам з урахуванням характеру і особливостей трудової діяльності (ГОСТ 12.2.032-78, ГОСТ 22.269-76, ГОСТ 21.889-76) [14].

Конструкція робочого місця користувача ВДТ має забезпечити підтримання оптимальної робочої пози.

Робочі місця з ВДТ слід так розташовувати відносно світових прорізів, щоб природне світло падало збоку, переважно зліва.

При розміщенні робочих столів з ВДТ слід дотримуватись таких відстаней: між бічними поверхнями ВДТ - 1,2 м; від тильної поверхні одного ВДТ до екрана іншого - 2,5 м.

Екран ВДТ має розташовуватися на оптимальній відстані від очей користувача, що становить 600...700 мм, але не ближче ніж за 600 мм з урахуванням розміру літерно-цифрових знаків і символів. Розташування екрана ВДТ має забезпечувати зручність зорового спостереження у вертикальній площині під кутом +30° до нормальної лінії погляду працюючого.

Клавіатуру слід розташовувати на поверхні столу на відстані 100...300 мм від краю, звернутого до працюючого. У конструкції клавіатури має передбачатися опорний пристрій (виготовлений із матеріалу з високим коефіцієнтом тертя, що перешкоджає мимовільному її зсуву), який дає змогу змінювати кут нахилу поверхні клавіатури у межах 5... 15°.

Для забезпечення захисту і досягнення нормованих рівнів комп'ютерних випромінювань необхідно застосовувати приекранні фільтри, локальні світлофільтри (засоби індивідуального захисту очей) та інші засоби захисту, що пройшли випробування в акредитованих лабораторіях і мають щорічний гігієнічний сертифікат.

При оснащенні робочого місця з ВДТ лазерним принтером параметри лазерного випромінювання повинні відповідати вимогам СанПіН № 5804-91.

5.2 Факти впливу ВДТ на людину

Утручання в життя мільйонів людей інформаційних технологій породжує багато проблем, у першу чергу пов'язаних з безпечністю використання інформаційного обладнання [14].

Негативні наслідки комп'ютерних технологій виявляються в наступному [14]:

* інтенсифікації темпу роботи та її монотонності;
* ізоляції працівника у виробничому середовищі, обмеженні його контактів з іншими людьми;
* розвитку несприятливих психічних станів;
* великих нервових навантаженнях при незначних фізичних;
* перенапруженні органів зору;
* розладі стану здоров'я, спричиненого дією шкідливих факторів, джерелом яких є ВДТ.

У зв'язку з цим праця професійних користувачів ВДТ має свої особливості. Вони полягають у відмінності розумового і науково-емоційного компонентів праці, ступені включення в діяльність тих чи інших органів і систем. Функціональні розлади діяльності аналізаторів, захворювання опорно-рухового апарату, нервової, серцево-судинної та інших систем організму є виробничо зумовленими.

Серед користувачів ВДТ значного поширення набуло специфічне захворювання, яке отримало назву синдром комп'ютерного стресу (СКС). СКС супроводжується головним болем, запаленням очей, алергією, роздратованістю, млявістю і депресією. Дослідження, проведені в США, Німеччині, Швейцарії та інших країнах, показали, що робота з обслуговування ВДТ супроводжується підвищеним напруженням зору, інтенсивністю і монотонністю праці, збільшенням статичних навантажень, нервово-психічним напруженням, впливом різного виду випромінювань та ін. Вважається, що стан організму операторів ВДТ визначається комплексним впливом факторів трудового процесу і середовища, значення яких є неоднаковим. На операторів з малим стажем роботи на ВДТ домінуючий вплив чинять фактори середовища, а на операторів зі стажем понад 5 років - фактори трудового процесу.

Результати медико-біологічних досліджень впливу комп'ютерів на користувачів подані в таблиці 5.1.

Таблиця 5.1 - Результати впливу комп'ютерів на користувачів

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Симптоми | Кількість користувачів (%), що повідомили про симптоми від загальної чисельності опитаних | | | |
| залежно від | | | |
| режиму роботи | | стажу роботи | |
| 12 місяців при неповній зміні | 12 місяців при повній зміні | понад 1 рік | понад 2 роки |
| 1 | 2 | 3 | 4 | 5 |
| Головний біль і біль в очах | 8 | 35 | 51 | 76 |
| Втома і запаморочення | 5 | 32 | 41 | 69 |

Продовження таблиці 5.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 |
| Порушення нічного сну | - | 8 | 15 | 50 |
| Сонливість протягом Дня | 11 | 22 | 48 | 76 |
| Зміна настрою | 8 | 24 | 27 | 50 |
| Підвищена роздратованість | 8 | 11 | 22 | 51 |
| Депресія | 3 | 16 | 22 | 50 |

Характер захворювань користувачів значним чином зумовлений типом і умовами виконання роботи з ВДТ.

Таким чином, небезпечність використання інформаційного обладнання зумовлена: недосконалістю організації праці користувачів ВДТ; емісіями, джерелом яких є комп'ютери; особливостями праці на ВДТ; умовами праці.

Захворювання очей та порушення зору. Ці захворювання є найбільш поширеними скаргами персоналу ВДТ.

У операторів ВДТ "очні" симптоми трапляються частіше, ніж "зорові", причому частота проявів астенопії вища у жінок, ніж у чоловіків і більше виражена в осіб середнього і старшого віку. Причиною вважається електромагнітне випромінювання від ВДТ.

Робота за комп'ютером характеризується також тим, що постійний напружений погляд на екран монітора зменшує частоту моргання. При цьому погіршується зволоження поверхні очного яблука сльозовою рідиною, яка захищає рогівку ока від висихання, пилу та інших забруднень. Це може призвести до виникнення так званого синдрому Сікка: рогівка висихає і мутніє, і як наслідок — сліпота.

Також при напруженій зоровій роботі за ЕОМ можуть бути не лише порушення функції зору, а й виникнення головного болю, посилення нервово-психічного напруження, зниження працездатності.

Виникнення та розвиток патології зорової функції зумовлені:

* умовами зорової роботи на ВДТ (зменшення вільного руху очей, зменшення функціонального поля сітківки та ін.);
* змінами умов, характерних для традиційного зорового процесу читання (темні знаки на світлому фоні при падаючому світловому потоці), а також демонстрування зображення на майже вертикальній поверхні, що випромінює світловий потік, а отже, потребує пониженого загального освітлення на робочому місці;
* змінами умов, характерних для традиційного зорового процесу читання (темні знаки на світлому фоні при падаючому світловому потоці), а також демонстрування зображення на майже вертикальній поверхні, що випромінює світловий потік, а отже, потребує пониженого загального освітлення на робочому місці;
* світлотехнічною різнорідністю об'єктів зорової роботи (екран, клавіатура, документація), розташованих у різних зонах спостереження, що потребує багаторазового переведення лінії зору від одного до іншого;
* робота з пульсуючим самосвітним об'єктом, який постійно перебуває у центрі поля зору, що не відповідає нормативним вимогам щодо обмеження пульсації та засліпленості. Наявність пульсації яскравості знаків викликає дискомфорт і втому, загальну й здорову;
* несприятливим розподілом яскравості у полі зору (стеля, стіни, меблі тощо можуть виявитися світлішими, ніж центр поля зору - темний, обмежено освітлений та іноді малозаповнений знаками екран монітора);
* засліплюючою дією світильників, які освітлюють приміщення на робочому місці та ін.

Отже, порушення зорових функцій користувачів ВДТ пов'язані, головним чином, з чотирма групами факторів:

* параметрами освітлення робочого місця;
* характеристиками дисплея;
* специфікою роботи на ВДТ;
* неправильною організацією робочого місця.

Порушення опорно-рухового апарату. Інтенсивна і тривала робота на ВДТ може стати джерелом професійних захворювань, пов'язаних з травмою монотонних навантажень. Це так звані ергономічні захворювання. Ці захворювання проявляються у вигляді втоми, скутості, болю, судоми, оніміння та інших симптомів, що локалізуються у різних частинах тіла (шиї, спині, ногах, руках тощо).

До найтиповіших симптомів, характерних для таких захворювань, належать:

* больові відчуття різної сили в суглобах та м'язах кистей рук;
* оніміння та повільна рухомість пальців;
* судоми м'язів кісті;
* поява нічного болю в зап'ясті.

Перенапруження опорно-рухового апарату, головним чином, спричиняється:

* нераціональною позою, яка ускладнюється нераціональною організацією робочого місця;
* однотипними циклічними навантаженнями, викликаними роботою на клавіатурі або "миші";
* обмеженістю загальної рухової активності (гіподинамією).

Захворювання шкіри. У науковій літературі наводяться численні дані про захворювання шкіри у користувачів ВДТ, які виявляються у вигляді папульозного висипання, свербежу, лущення шкіри, перорального та себорального дерматитів, рожевих вугрів тощо.

Ураження шкіри пов'язують із впливом на користувачів ВДТ електромагнітного поля, що генерується дисплеєм комп'ютера. Воно посилює електростатичний заряд на тілі користувача. Це сприяє відкладенню аерозольних часток на обличчі й може у чутливих осіб викликати різноманітні шкірні реакції.

Є повідомлення про те, що робота на ВДТ протягом 2-6 і більше годин на день викликає екзему, яка, на думку фахівців, розвивається під впливом статичного, а можливо, електромагнітного полів.

Нервово-психічні захворювання. Робота професійних користувачів ВДТ пов'язана з такими психологічними особливостями:

* інформаційним перевантаженням мозку в поєднанні з дефіцитом часу;
* тривожним очікуванням інформації, особливо тієї, що викликає необхідність прийняття рішення;
* високою відповідальністю за кінцевий результат;
* ізоляцією у спілкуванні та ін.

Під впливом цих факторів відбуваються зміни у співвідношенні процесів збудження і гальмування в корі головного мозку. При цьому функціональна активність центральної нервової системи знижується, основні нервові процеси гальмуються. В організмі розвивається психічна втома, яка характеризується:

* зниженням здатності концентрувати увагу, сприймати інформацію;
* уповільненням мислення з витратою його гнучкості та широти;
* зниженням здатності до запам'ятовування та згадування;
* змінами в емоційному стані (депресія, роздратування, втрата емоційної рівноваги);
* сповільненням сенсомоторних функцій, унаслідок чого час реакції користувачів збільшується, рухи стають метушливими і неточними тощо.

Психічна втома спричиняє виникнення неврозів, основними симптомами яких є зниження працездатності користувачів ВДТ, збайдужіння до навколишнього життя, звуження кола зацікавлень.

Тривале перебування людини у стані стресу може призвести до розвитку серцево-судинних захворювань.

Основними факторами розвитку неврозів у користувачів ВДТ е:

* особливості характеру трудового процесу та умов праці;
* організація робочих місць;
* мотивація праці;
* особливості приладового та програмного забезпечення;
* соціальні фактори.

Таким чином, ВДТ можуть суттєво впливати на здоров'я користувачів. Серед причин формування ергономічно зумовлених патологій провідне місце займають організаційні особливості праці користувачів ВДТ та характер трудового процесу. Разом з тим негативний вплив на стан здоров'я користувачів пов'язують з невідповідністю окремих моделей ВДТ гігієнічним та ергономічним вимогам. Ступінь цього впливу залежить від технічних характеристик ВДТ, інтенсивності впливу шкідливих факторів, організації праці на робочих місцях, упровадження засобів безпеки обслуговування ВДТ, тобто від рівня небезпечності інформаційного обладнання,

5.3 Способи та засоби пожежогасіння

Офісні приміщення оснащуються великою кількістю комп’ютерної та оргтехніки, електроприладами, меблями, виготовленими з легкозаймистих матерів. В них одночасно працюють по кілька десятків людей [14].

Одним з елементів забезпечення пожежної безпеки в офісі є первинні засоби пожежогасіння. Необхідно утримувати їх в належному стані та навчити персонал користуватися ними у випадку виникнення надзвичайної ситуації.

До первинних засобів пожежогасіння належать: вогнегасники, кошма (покривало з негорючого теплоізоляційного полотна), ящики з піском, бочки з водою, пожежні відра, багри, ломи, сокири тощо. Найбільш зручними для використання в умовах офісу є вогнегасники. Попри обладнання будівель будь-якими типами установок пожежогасіння, пожежної сигналізації або внутрішніми пожежними кранами, офісні приміщення також мають бути забезпечені первинними засобами пожежогасіння.

Відповідальними за своєчасне та повне оснащення об’єктів засобами пожежогасіння, забезпечення їх технічного обслуговування, навчання працівників правил користування ними є власники або орендарі об’єктів.

В кожній організації наказом або розпорядженням керівника повинна бути призначена особа, відповідальна за експлуатацію вогнегасників. Це може бути особа відповідальна за дотримання вимог пожежної безпеки на об’єкті або спеціаліст відповідної категорії з іншої організації, наприклад, пункту технічного обслуговування вогнегасників.

Успішне гасіння пожежі залежить від правильного вибору типу та виду вогнегасника. Вибір типу та необхідна кількість вогнегасників здійснюється відповідно до Правил експлуатації та типових норми належності вогнегасників, затверджених наказом Міністерства внутрішніх справ України від 15 січня 2018 р. № 25.

Експлуатація вогнегасників без призначення відповідального за організацію цієї роботи не допускається.

Згідно з Правилами, будинки адміністративного призначення на кожному поверсі повинні мати не менше двох переносних (порошкових, водопінних або водяних) вогнегасників з масою заряду вогнегасної речовини 5 кг і більше.

Крім того, на 20 м² площі підлоги в офісних приміщеннях з оргтехнікою, слід передбачати по одному газовому вогнегаснику з величиною заряду вогнегасної речовини 3 кг і більше. Приміщення, у яких розміщено оргтехніку, слід оснащувати переносними газовими вогнегасниками з розрахунку один вогнегасник ВВК-1,4 чи ВВК-2, але не менше ніж один вогнегасник зазначених типів на приміщення.

Додатково будинки та офісні приміщення можуть оснащуватися пристроєм вогнегасним водопінним аерозольним (ВВПА), з масою заряду вогнегасної речовини 400 г і більше.

Для гасіння пожежі в початковій стадії в офісах, крім вогнегасників доречно мати ще кошму. Пожежні покривала повинні бути розміром не менше ніж 1 x 1 м. У місцях застосування та зберігання ЛЗР та ГР1 мінімальні розміри пожежних покривал збільшуються до величин: 2 x 1,5 м і 2 x 2 м відповідно.

Необхідна кількість первинних засобів пожежогасіння повинна визначатися відповідальним за пожежну безпеку на об’єкті окремо для кожного поверху та приміщення з урахуванням специфіки даного офісу.

Купувати вогнегасники слід лише в спеціалізованих організаціях, які мають ліцензію на такий вид діяльності й продукція яких сертифікована в Україні.

Перед розміщенням вогнегасників на об’єкті особі, відповідальній за пожежну безпеку, необхідно обов’язково провести їх огляд. Після проведення огляду вогнегасникам присвоюються облікові (інвентарні) номери за прийнятою на об’єкті системою нумерації.

Особі, відповідальній за пожежну безпеку на об’єкті, необхідно вести журнал обліку вогнегасників встановленого зразка (додаток 2 до Правил).

Не рідше одного разу на місяць особою, відповідальною за пожежну безпеку має проводитись огляд вогнегасників при їх експлуатації.

Особа, що відповідає за пожежну безпеку, зобов’язана організувати технічне обслуговування вогнегасників у таких випадках:

* пошкодження або відсутність маркування, пломб або пристроїв блокування на них;
* наявність механічних пошкоджень і слідів корозії на їх корпусах або запірно-пускових пристроях;
* відсутність робочого тиску в корпусі та (або) наявність надмірного тиску (для вогнегасників закачного типу);
* після використання за призначенням;
* після закінчення гарантійного терміну експлуатації, передбаченого експлуатаційною документацією виробника.

Технічне обслуговування вогнегасників слід довіряти лише пунктам технічного обслуговування вогнегасників (далі — ПТОВ), що мають відповідну ліцензію з надання послуг і виконання робіт протипожежного призначення відповідно до вимог ДСТУ 4297:2004 «Пожежна техніка. Технічне обслуговування вогнегасників. Загальні технічні вимоги». Приймання вогнегасників після технічного обслуговування оформлюється актом, який складається не менше ніж у двох примірниках і підписується представниками споживача послуг та ПТОВ. Під час огляду вогнегасників, після надходження з технічного обслуговування, відповідальний за пожежну безпеку має перевірити наявність на корпусі вогнегасника етикетки ПТОВ, встановленого зразка (додаток 3 до Правил).

Рекомендовано ознайомитись зі статтею «Вимоги до первинних засобів пожежогасіння на підприємстві» в ОППБ № 8-2018 та спецвипуском з охорони праці та пожежної безпеки «Пожежна безпека підприємства: мінімізуємо ризики» № 3-2020.

Враховуючи те, що в офісних приміщеннях багато апаратури, приладів та документів, щоб запобігти їх псуванню при гасінні, краще користуватись газовими (вуглекислотними) вогнегасниками. Застосування порошкових вогнегасників для гасіння таких пожеж прийнятне лише за відсутності газових вогнегасників.

Під час застосування газових або порошкових вогнегасників для гасіння електрообладнання, що перебуває під напругою до 1000 В, необхідно дотримуватися рекомендацій, зазначених у паспортах вогнегасників.

Забороняється гасити обладнання, що перебуває під напругою, водяними та водопінними вогнегасниками.

При користуванні газовими вогнегасниками необхідно враховувати можливість зниження концентрації кисню в повітрі приміщення, особливо якщо воно невелике за об’ємом. Якщо через використання газових вогнегасників може створитись небезпечна для життя людини концентрація газів у повітрі слід використовувати засоби індивідуального захисту органів дихання.

Усі працівники офісу повинні знати, як користуватися первинними засобами пожежогасіння.

Раз на пів року повинні проводитись практичні заняття на яких персонал офісу навчається та відпрацьовує дії на випадок пожежі.

Під час проведення практичних занять працівники мають засвоїти:

* будову та принцип роботи вогнегасників;
* тактику застосування вогнегасників;
* гасіння умовної пожежі за допомогою первинних засобів пожежогасіння

ВИСНОВКИ

В даному дипломному проекті розроблено веб-сайту магазину аксесуарів «Atna». Даний веб-сайт являє собою повноцінний інтернет-магазин.

В першому розділі дипломного проекту здійснено аналіз основних підходів для реалізації інтернет магазинів, обґрунтовано переваги розробки власного програмного продукту та сформовано технічне завдання.

В другому розділі описано основні вимоги до функціоналу магазину.

На основі бібліотеки ReactJS реалізовано клієнтську частину додатку. Тобто реалізовано функціонал: відображення товарів магазину, пошуку, фільтрації товарі, пагінації для виведення певної кількості товарів на сторінці адміністратора. Також реалізований механізм реєстрації та автентифікації користувачів магазину. Для адміністратора магазину реалізовано функціонал наповнення магазину товарами, редагування властивостей товарів, видалення товарів. Для покупця реалізовано функціонал корзини. Реалізований механізм авторизації користувачів.

Серверна частина веб-сайту реалізована з використанням фреймворку ExpressJS для платформи NodeJS. У ній реалізоване API-магазину. Тобто реалізовано обробники маршрутів, моделі записів бази даних MongoDB та контроллери роботи з цими моделями. Також реалізовано механізм шифрування паролів.

В третьому розділі описані інструкції з розміщення проекту у Інтернеті, інструкція з обслуговування та наповнення сайт, а також інструкція з популяризації та підтримки сайту

В економічному розділі розраховано вартість впровадження проекту,

В останньому розділі розглянуті питання охорони праці та безпеки життєдіяльності.

Магазин розміщений на вільному хостингу render.com за доменною адресною https://atna-menr.onrender.com

ПЕРЕЛІК ПОСИЛАНЬ

1. Start a New React Project [Електронний ресурс] – Режим доступу до ресурсу: https://react.dev/learn/start-a-new-react-project – Дата доступу: 26.04.2023.

2. Node.js open-source, cross-platform JavaScript runtime environment [Електронний ресурс] – Режим доступу до ресурсу: https://nodejs.org/en – Дата доступу: 26.04.2023.

3. What is npm [Електронний ресурс] – Режим доступу до ресурсу: https://www.w3schools.com/whatis/whatis\_npm.asp – Дата доступу: 26.04.2023.

4. Create React App Set up a modern web app by running one command [Електронний ресурс] – Режим доступу до ресурсу: https://create-react-app.dev/ – Дата доступу: 26.04.2023.

5. Understanding Redux: A tutorial with examples [Електронний ресурс] – Режим доступу до ресурсу: https://blog.logrocket.com/understanding-redux-tutorial-examples/ – Дата доступу: 26.04.2023.

6. Data Modeling Introduction [Електронний ресурс] – Режим доступу до ресурсу: https://www.mongodb.com/docs/manual/core/data-modeling-introduction/ – Дата доступу: 26.04.2023.

7. Everything as a Component in ReactJS [Електронний ресурс] – Режим доступу до ресурсу: https://www.pluralsight.com/guides/everything-as-a-component-in-reactjs – Дата доступу: 26.04.2023.

8. Image transformations for Developers [Електронний ресурс] – Режим доступу до ресурсу: https://cloudinary.com/documentation/image\_transformations – Дата доступу: 26.04.2023.

9. Express Fast, unopinionated, minimalist web framework for Node.js [Електронний ресурс] – Режим доступу до ресурсу: https://expressjs.com/ – Дата доступу: 26.04.2023.

10. Mongoose Methods and Statics [Електронний ресурс] – Режим доступу до ресурсу: https://mongoosejs.com/docs/2.7.x/docs/methods-statics.html – Дата доступу: 26.04.2023.

11. Web Services - Render Cloud Hosting for Developers [Електронний ресурс] – Режим доступу до ресурсу: view-source:https://render.com/docs/web-services – Дата доступу: 26.04.2023.

12. MongoDB Atlas. Fully managed MongoDB in the cloud [Електронний ресурс] – Режим доступу до ресурсу: https://www.mongodb.com/cloud/atlas/ lp/try4 – Дата доступу: 26.04.2023

13. Single Page Application SEO: a checklist of what you need to be aware of [Електронний ресурс] – Режим доступу до ресурсу: https://kruschecompany.com/seo-tips-and-tricks-for-single-page-web-applications/4 – Дата доступу: 26.04.2023

14. Охорона праці – Москальова В.М. [Електронний ресурс] – Режим доступу до ресурсу: http://studentbooks.com.ua/content/view/1327/76/ – Дата доступу: 19.04.2022.

ДОДАТКИ

Додаток А – Лістинг файлу src\index.js

import React from "react";

import ReactDOM from "react-dom/client";

import "./index.css";

import App from "./App";

import reportWebVitals from "./reportWebVitals";

//setup store

import store from "./store";

import { Provider } from "react-redux";

import { PersistGate } from "redux-persist/integration/react";

import persistStore from "redux-persist/es/persistStore";

// store to persit

const persistedStore = persistStore(store);

const root = ReactDOM.createRoot(document.getElementById("root"));

root.render(

<Provider store={store}>

<PersistGate loading={<div>Loading...</div>} persistor={persistedStore}>

<App />

</PersistGate>

</Provider>

);

reportWebVitals();

Додаток Б – Лістинг коду компоненту <App/>

import "./App.css";

import "bootstrap/dist/css/bootstrap.min.css";

import { BrowserRouter, Routes, Route } from "react-router-dom";

import Navigation from "./components/Navigation";

import Home from "./pages/Home";

import Login from "./pages/Login";

import Signup from "./pages/Signup";

import { useDispatch, useSelector } from "react-redux";

import NewProduct from "./pages/NewProduct";

import ProductPage from "./pages/ProductPage";

import CategoryPage from "./pages/CategoryPage";

import ScrollToTop from "./components/ScrollToTop";

import CartPage from "./pages/CartPage";

import OrdersPage from "./pages/OrdersPage";

import AdminDashboard from "./pages/AdminDashboard";

import EditProductPage from "./pages/EditProductPage";

import { useEffect } from "react";

function App() {

const user = useSelector((state) => state.user);

const dispatch = useDispatch();

return (

<div className="App">

<BrowserRouter>

<ScrollToTop />

<Navigation />

<Routes>

<Route index element={<Home />} />

{!user && (

<>

<Route path="/login" element={<Login />} />

<Route path="/signup" element={<Signup />} />

</>

)}

{user && (

<>

<Route path="/cart" element={<CartPage />} />

<Route path="/orders" element={<OrdersPage />} />

</>

)}

{user && user.isAdmin && (

<>

<Route path="/admin" element={<AdminDashboard />} />

<Route path="/product/:id/edit" element={<EditProductPage />} />

</>

)}

<Route path="/product/:id" element={<ProductPage />} />

<Route path="/category/:category" element={<CategoryPage />} />

<Route path="/new-product" element={<NewProduct />} />

<Route path="\*" element={<Home />} />

</Routes>

</BrowserRouter>

</div>

);

}

export default App;

Додаток В – Лістинг коду компоненту <Navigation />

import axios from "../axios";

import React, { useRef, useState } from "react";

import { Navbar, Button, Nav, NavDropdown, Container } from "react-bootstrap";

import { useDispatch, useSelector } from "react-redux";

import { LinkContainer } from "react-router-bootstrap";

import { logout, resetNotifications } from "../features/userSlice";

import "./Navigation.css";

function Navigation() {

const user = useSelector((state) => state.user);

const dispatch = useDispatch();

const bellRef = useRef(null);

const notificationRef = useRef(null);

const [bellPos, setBellPos] = useState({});

function handleLogout() {

dispatch(logout());

}

const unreadNotifications = user?.notifications?.reduce((acc, current) => {

if (current.status === "unread") return acc + 1;

return acc;

}, 0);

function handleToggleNotifications() {

const position = bellRef.current.getBoundingClientRect();

setBellPos(position);

notificationRef.current.style.display = notificationRef.current.style.display === "block" ? "none" : "block";

dispatch(resetNotifications());

if (unreadNotifications > 0) axios.post(`/users/${user.\_id}/updateNotifications`); }

return (

<Navbar bg="light" expand="lg">

<Container>

<LinkContainer to="/">

<Navbar.Brand>ATNA</Navbar.Brand>

</LinkContainer>

<Navbar.Toggle aria-controls="basic-navbar-nav" />

<Navbar.Collapse id="basic-navbar-nav">

<Nav className="ms-auto">

{/\* if no user \*/}

{!user && (

<LinkContainer to="/login">

<Nav.Link>Логін</Nav.Link>

</LinkContainer>

)}

{user && !user.isAdmin && (

<LinkContainer to="/cart">

<Nav.Link>

<i className="fas fa-shopping-cart"></i>

{user?.cart.count > 0 && (

<span className="badge badge-warning" id="cartcount">

{user.cart.count}

</span>

)}

</Nav.Link>

</LinkContainer> )}

{/\* if user \*/}

{user && (

<>

<Nav.Link style={{ position: "relative" }} onClick={handleToggleNotifications}>

<i className="fas fa-bell" ref={bellRef} data-count={unreadNotifications || null}></i>

</Nav.Link>

<NavDropdown title={`${user.email}`} id="basic-nav-dropdown">

{user.isAdmin && (

<>

<LinkContainer to="/admin">

<NavDropdown.Item>Інформаційна Панель</NavDropdown.Item>

</LinkContainer>

<LinkContainer to="/new-product">

<NavDropdown.Item>Створити Продукт</NavDropdown.Item>

</LinkContainer>

</>

)}

{!user.isAdmin && (

<>

<LinkContainer to="/cart">

<NavDropdown.Item>Корзина</NavDropdown.Item>

</LinkContainer>

<LinkContainer to="/orders">

<NavDropdown.Item>Мої замовлення</NavDropdown.Item>

</LinkContainer>

</>)}

<NavDropdown.Divider />

<Button variant="danger" onClick={handleLogout} className="logout-btn">

Вийти

</Button>

</NavDropdown>

</>

)}

</Nav>

</Navbar.Collapse>

</Container>

{/\* notifications \*/}

<div className="notifications-container" ref={notificationRef} style={{ position: "absolute", top: bellPos.top + 30, left: bellPos.left, display: "none" }}>

{user?.notifications.length > 0 ? (

user?.notifications.map((notification) => (

<p className={`notification-${notification.status}`}>

{notification.message}

<br />

<span>{notification.time.split("T")[0] + " " + notification.time.split("T")[1]}</span>

</p> ))

) : (<p>Сповіщень ще немає</p>)}

</div>

</Navbar> );}

export default Navigation;

Додаток Г – Лістинг коду компоненту <Login />

import React, { useState } from "react";

import { Button, Col, Container, Form, Row, Alert } from "react-bootstrap";

import { Link } from "react-router-dom";

import { useLoginMutation } from "../services/appApi";

function Login() {

const [email, setEmail] = useState("");

const [password, setPassword] = useState("");

const [login, { isError, isLoading, error }] = useLoginMutation();

function handleLogin(e) {

e.preventDefault();

login({ email, password });

}

return (

<Container>

<Row>

<Col md={6} className="login\_\_form--container">

<Form style={{ width: "100%" }} onSubmit={handleLogin}>

<h1>УВІЙДІТЬ У СВІЙ АКАУНТ</h1>

{isError && <Alert variant="danger">{error.data}</Alert>}

<Form.Group>

<Form.Label>Електронна адреса</Form.Label>

<Form.Control type="email" placeholder="Електронна адреса" value={email} required onChange={(e) => setEmail(e.target.value)} />

</Form.Group>

<Form.Group className="mb-3">

<Form.Label>Пароль</Form.Label>

<Form.Control type="password" placeholder="Пароль" value={password} required onChange={(e) => setPassword(e.target.value)} />

</Form.Group>

<Form.Group>

<Button type="submit" disabled={isLoading}>

Увійти

</Button>

</Form.Group>

<p className="pt-3 text-center">

Не маєте акаунта? <Link to="/signup">Створити аккаунт</Link>{" "}

</p>

</Form>

</Col>

<Col md={6} className="login\_\_image--container"></Col>

</Row>

</Container>

);

}

export default Login;

Додаток Д – Лістинг коду компоненту <Signup />

import React, { useState } from "react";

import { Container, Row, Col, Form, Button, Alert } from "react-bootstrap";

import { Link } from "react-router-dom";

import "./Signup.css";

import { useSignupMutation } from "../services/appApi";

function Signup() {

const [email, setEmail] = useState("");

const [password, setPassword] = useState("");

const [name, setName] = useState("");

const [signup, { error, isLoading, isError }] = useSignupMutation();

function handleSignup(e) {

e.preventDefault();

signup({ name, email, password }).then((response) => {

if (response.data) {

window.location.replace('/')

}})}

return (

<Container>

<Row>

<Col md={6} className="signup\_\_form--container">

<Form style={{ width: "100%" }} onSubmit={handleSignup}>

<h1>СТВОРИТИ АКАУНТ</h1>

{isError && <Alert variant="danger">{error.data}</Alert>}

<Form.Group>

<Form.Label>Іm'я</Form.Label>

<Form.Control type="text" placeholder="Ваше іm'я" value={name} required onChange={(e) => setName(e.target.value)} />

</Form.Group>

<Form.Group>

<Form.Label>Електронна адреса</Form.Label>

<Form.Control type="email" placeholder="Електронна адресаl" value={email} required onChange={(e) => setEmail(e.target.value)} />

</Form.Group>

<Form.Group className="mb-3">

<Form.Label>Пароль</Form.Label>

<Form.Control type="password" placeholder="Пароль" value={password} required onChange={(e) => setPassword(e.target.value)} />

</Form.Group>

<Form.Group>

<Button type="submit" disabled={isLoading}>

Створити акаунт

</Button>

</Form.Group>

<p className="pt-3 text-center">

Вже маєте акаунт? <Link to="/login">Увійдіть</Link>{" "}

</p>

</Form>

</Col>

<Col md={6} className="signup\_\_image--container"></Col>

</Row>

</Container>

);

}

export default Signup;

Додаток E – Лістинг коду компоненту <appAPI />

import { createApi, fetchBaseQuery } from "@reduxjs/toolkit/query/react";

// create the api

export const appApi = createApi({

reducerPath: "appApi",

baseQuery: fetchBaseQuery({ baseUrl: "https://atna-menr-api.onrender.com" }),

endpoints: (builder) => ({

signup: builder.mutation({

query: (user) => ({

url: "/users/signup",

method: "POST",

body: user,

}),

}),

login: builder.mutation({

query: (user) => ({

url: "/users/login",

method: "POST",

body: user,

}),

}),

// creating product

createProduct: builder.mutation({

query: (product) => ({

url: "/products",

body: product,

method: "POST",

}),

}),

deleteProduct: builder.mutation({

query: ({ product\_id, user\_id }) => ({

url: `/products/${product\_id}`,

body: {

user\_id, },

method: "DELETE",

}),

}),

updateProduct: builder.mutation({

query: (product) => ({

url: `/products/${product.id}`,

body: product,

method: "PATCH",

}),

}),

// add to cart

addToCart: builder.mutation({

query: (cartInfo) => ({

url: "/products/add-to-cart",

body: cartInfo,

method: "POST",

}),

}),

// remove from cart

removeFromCart: builder.mutation({

query: (body) => ({

url: "/products/remove-from-cart",

body,

method: "POST",

}),

}),

// increase cart

increaseCartProduct: builder.mutation({

query: (body) => ({

url: "/products/increase-cart",

body,

method: "POST",

}),}),

// decrease cart

decreaseCartProduct: builder.mutation({

query: (body) => ({

url: "/products/decrease-cart",

body,

method: "POST",

}),

}),

// create order

createOrder: builder.mutation({

query: (body) => ({

url: "/orders",

method: "POST",

body,

}),

}),

}),

});

export const {

useSignupMutation,

useLoginMutation,

useCreateProductMutation,

useAddToCartMutation,

useRemoveFromCartMutation,

useIncreaseCartProductMutation,

useDecreaseCartProductMutation,

useCreateOrderMutation,

useDeleteProductMutation,

useUpdateProductMutation,

} = appApi;

export default appApi;

Додаток Є – Лістинг коду компоненту <Home />

import axios from "../axios";

import React, { useEffect } from "react";

import { Col, Row } from "react-bootstrap";

import { LinkContainer } from "react-router-bootstrap";

import { Link } from "react-router-dom";

import categories from "../categories";

import "./Home.css";

import { useDispatch, useSelector } from "react-redux";

import { updateProducts } from "../features/productSlice";

import ProductPreview from "../components/ProductPreview";

function Home() {

const dispatch = useDispatch();

const products = useSelector((state) => state.products);

const lastProducts = products.slice(0, 8);

useEffect(() => {

axios.get("/products").then(({ data }) => dispatch(updateProducts(data)));

}, [dispatch]);

return (

<div>

<img src="https://res.cloudinary.com/lvmcloud/image/upload/c\_pad,b\_auto:predominant,fl\_preserve\_transparency/v1685985245/banner\_mtpdad.jpg?\_s=public-apps" alt="" className="home-banner" />

<div className="recent-products-container container mt-4">

<h2>КАТЕГОРІЇ</h2>

<div>

<Link to="/category/all" style={{ textAlign: "right", display: "block", textDecoration: "none" }}>

{"Всі продукти >>"}

</Link>

</div>

<Row>

{categories.map((category, index) => (

<LinkContainer to={`/category/${category.name.toLocaleLowerCase()}`} key={index}>

<Col md={4}>

<div style={{ backgroundImage: `linear-gradient(rgba(0, 0, 0, 0.5), rgba(0, 0, 0, 0.5)), url(${category.img})`, gap: "10px" }} className="category-tile">

{category.label}

</div>

</Col>

</LinkContainer>

))}

</Row>

</div>

<div className="featured-products-container container mt-4">

{lastProducts.length ? <h2>Нові продукти</h2> : null}

<div className="d-flex justify-content-center flex-wrap">

{lastProducts.map((product, index) => (

<ProductPreview {...product} key={index} />

))}

</div>

</div>

</div>

);

}

export default Home;

Додаток Ж – Лістинг коду компоненту <ProductPage />

import axios from "../axios";

import React, { useEffect, useState } from "react";

import AliceCarousel from "react-alice-carousel";

import "react-alice-carousel/lib/alice-carousel.css";

import { Container, Row, Col, Badge, ButtonGroup, Form, Button } from "react-bootstrap";

import { useSelector } from "react-redux";

import { useParams } from "react-router-dom";

import Loading from "../components/Loading";

import SimilarProduct from "../components/SimilarProduct";

import "./ProductPage.css";

import { LinkContainer } from "react-router-bootstrap";

import { useAddToCartMutation } from "../services/appApi";

import ToastMessage from "../components/ToastMessage";

import categories from "../categories";

function ProductPage() {

const { id } = useParams();

const user = useSelector((state) => state.user);

const [product, setProduct] = useState(null);

const [similar, setSimilar] = useState(null);

const [addToCart, { isSuccess }] = useAddToCartMutation();

const handleDragStart = (e) => e.preventDefault();

useEffect(() => {

axios.get(`/products/${id}`).then(({ data }) => {

setProduct(data.product);

setSimilar(data.similar);

});

}, [id]);

if (!product) {

return <Loading />;

}

const responsive = {

0: { items: 1 },

568: { items: 2 },

1024: { items: 3 },

};

const images = product.pictures.map((picture) => <img className="product\_\_carousel--image" alt='' src={picture.url} onDragStart={handleDragStart} />);

let similarProducts = [];

if (similar) {

similarProducts = similar.map((product, idx) => (

<div className="item" data-value={idx}>

<SimilarProduct {...product} />

</div>

));

}

return (

<Container className="pt-4" style={{ position: "relative" }}>

<Row>

<Col lg={6}>

<AliceCarousel mouseTracking items={images} controlsStrategy="alternate" />

</Col>

<Col lg={6} className="pt-4">

<div className='info\_container'>

<div className='info\_name'>

<h1>{product.name}</h1>

<p>

<Badge bg="primary">{categories.filter((item) => item.name === product.category)[0].label}</Badge>

</p>

</div>

<p className="product\_\_price">₴{product.price}</p>

</div>

<p style={{ textAlign: "justify" }} className="py-3">

<strong>Опис:</strong> {product.description}

</p>

{user && !user.isAdmin && (

<ButtonGroup style={{ width: "90%" }}>

<Button size="lg" onClick={() => addToCart({ userId: user.\_id, productId: id, price: product.price, image: product.pictures[0].url })}>

Додати до корзини

</Button>

</ButtonGroup>

)}

{user && user.isAdmin && (

<LinkContainer to={`/product/${product.\_id}/edit`}>

<Button size="lg">Редагувати продукт</Button>

</LinkContainer>

)}

{isSuccess && <ToastMessage bg="info" title="Added to cart" body={`${product.name} is in your cart`} />}

</Col>

</Row>

<div className="my-4">

<h2>Схожі продукти</h2>

<div className="d-flex justify-content-center align-items-center flex-wrap">

<AliceCarousel mouseTracking items={similarProducts} responsive={responsive} controlsStrategy="alternate" />

</div>

</div>

</Container>

);

}

export default ProductPage;

Додаток З – Лістинг коду компоненту <CartPage />

import { Elements } from "@stripe/react-stripe-js";

import { loadStripe } from "@stripe/stripe-js";

import React, { useEffect, useState } from "react";

import { Alert, Col, Container, Row, Table } from "react-bootstrap";

import { useSelector } from "react-redux";

import CheckoutForm from "../components/CheckoutForm";

import { useIncreaseCartProductMutation, useDecreaseCartProductMutation, useRemoveFromCartMutation } from "../services/appApi";

import "./CartPage.css";

const stripePromise = loadStripe("your\_stripe\_publishable\_key");

function CartPage() {

const user = useSelector((state) => state.user);

const products = useSelector((state) => state.products);

const userCartObj = user.cart;

let cart = products.filter((product) => userCartObj[product.\_id] != null);

const [increaseCart] = useIncreaseCartProductMutation();

const [decreaseCart, { isLoading: isDecreaseLoading }] = useDecreaseCartProductMutation();

const [removeFromCart, { isLoading }] = useRemoveFromCartMutation();

function handleDecrease(product) {

const quantity = user.cart.count;

if (quantity <= 0) return alert("Can't proceed");

decreaseCart(product);

}

return (

<Container style={{ minHeight: "95vh" }} className="cart-container">

<Row>

<Col>

<h1 className="pt-2 h3">Корзина</h1>

{cart.length === 0 ? (

<Alert variant="info">Корзина пуста. Додайте продукти до корзини</Alert>

) : (

<Elements stripe={stripePromise}>

<CheckoutForm />

</Elements>

)}

</Col>

{cart.length > 0 && (

<Col md={5}>

<>

<Table responsive="sm" className="cart-table">

<thead>

<tr>

<th>&nbsp;</th>

<th>Назва</th>

<th>Ціна</th>

<th>Кількість</th>

<th>Сума</th>

</tr>

</thead>

<tbody>

{/\* loop through cart products \*/}

{cart.map((item) => (

<tr>

<td>&nbsp;</td>

<td>

{!isLoading && <i className="fa fa-times" style={{ marginRight: 10, cursor: "pointer" }} onClick={() => removeFromCart({ productId: item.\_id, price: item.price, userId: user.\_id })}></i>}

<img src={item.pictures[0].url} alt="" style={{ width: 100, height: 100, objectFit: "cover" }} />

</td>

<td>₴{item.price}</td>

<td><span className="quantity-indicator">

<i className="fa fa-minus-circle"

onClick={() => {

if (!isDecreaseLoading) {

const data = { productId: item.\_id, price: item.price, userId: user.\_id }

if (user.cart[item.\_id] > 1) {

handleDecrease(data)

} else if (user.cart[item.\_id] === 1) {

removeFromCart(data)

}}}}></i>

<span>{user.cart[item.\_id]}</span>

<i className="fa fa-plus-circle" onClick={() => increaseCart({ productId: item.\_id, price: item.price, userId: user.\_id })}></i>

</span>

</td>

<td>₴{item.price \* user.cart[item.\_id]}</td>

</tr>

))}

</tbody>

</Table>

<div>

<h3 className="h4 pt-4">Сума: ₴{user.cart.total}</h3>

</div>

</>

</Col>

)}

</Row>

</Container>

);

}

export default CartPage;

Додаток И – Лістинг коду компоненту <CheckoutForm />

import { CardElement, useElements, useStripe } from "@stripe/react-stripe-js";

import React, { useState } from "react";

import { Alert, Button, Col, Form, Row } from "react-bootstrap";

import { useSelector } from "react-redux";

import { useCreateOrderMutation } from "../services/appApi";

function CheckoutForm() {

const stripe = useStripe();

const elements = useElements();

const user = useSelector((state) => state.user);

const [alertMessage, setAlertMessage] = useState("");

const [createOrder, { isLoading, isError, isSuccess }] = useCreateOrderMutation();

const [country, setCountry] = useState("");

const [address, setAddress] = useState("");

const [paying, setPaying] = useState(false);

async function handlePay(e) {

e.preventDefault();

if (!stripe || !elements || user.cart.count <= 0) return;

setPaying(true);

const { client\_secret } = await fetch("https://atna-menr-api.onrender.com/create-payment", {

method: "POST",

headers: {

"Content-Type": "application/json",

Authorization: "Bearer ", },

body: JSON.stringify({ amount: user.cart.total }),

}).then((res) => res.json());

const { paymentIntent } = await stripe.confirmCardPayment(client\_secret, {

payment\_method: {

card: elements.getElement(CardElement),

}, });

setPaying(false);

if (paymentIntent) {

createOrder({ userId: user.\_id, cart: user.cart, address, country }).then((res) => {if (!isLoading && !isError) {

setAlertMessage(`Payment ${paymentIntent.status}`);

setTimeout(() => {

// navigate("/orders");

}, 3000);

}});}}

return (

<Col className="cart-payment-container">

<Form onSubmit={handlePay}>

<Row>

{alertMessage && <Alert>{alertMessage}</Alert>}

<Col md={6}>

<Form.Group className="mb-3">

<Form.Label>Імʼя</Form.Label>

<Form.Control type="text" placeholder="Імʼя" value={user.name} disabled />

</Form.Group>

</Col>

<Col md={6}>

<Form.Group className="mb-3">

<Form.Label>Електронна адреса</Form.Label>

<Form.Control type="text" placeholder="Електронна адреса" value={user.email} disabled />

</Form.Group>

</Col>

</Row>

<Row>

<Col md={7}>

<Form.Group className="mb-3">

<Form.Label>Адреса</Form.Label>

<Form.Control type="text" placeholder="Адреса" value={address} onChange={(e) => setAddress(e.target.value)} required />

</Form.Group>

</Col>

<Col md={5}>

<Form.Group className="mb-3">

<Form.Label>Країна</Form.Label>

<Form.Control type="text" placeholder="Країна" value={country} onChange={(e) => setCountry(e.target.value)} required />

</Form.Group>

</Col>

</Row>

<label htmlFor="card-element">Номер карти</label>

<CardElement id="card-element" />

<Button className="mt-3" type="submit" disabled={user.cart.count <= 0 || paying || isSuccess}>

{paying ? "Обробка..." : "Оплатити"}

</Button>

</Form>

</Col>

);

}

export default CheckoutForm;

Додаток І – Лістинг коду компоненту <NewProduct />

import React, { useState } from "react";

import { Alert, Col, Container, Form, Row, Button } from "react-bootstrap";

import { useNavigate } from "react-router-dom";

import { useCreateProductMutation } from "../services/appApi";

import axios from "../axios";

import "./NewProduct.css";

function NewProduct() {

const [name, setName] = useState("");

const [description, setDescription] = useState("");

const [price, setPrice] = useState("");

const [category, setCategory] = useState("");

const [images, setImages] = useState([]);

const [imgToRemove, setImgToRemove] = useState(null);

const navigate = useNavigate();

const [createProduct, { isError, error, isLoading, isSuccess }] = useCreateProductMutation();

function handleRemoveImg(imgObj) {

setImgToRemove(imgObj.public\_id);

axios

.delete(`/images/${imgObj.public\_id}/`)

.then((res) => {

setImgToRemove(null);

setImages((prev) => prev.filter((img) => img.public\_id !== imgObj.public\_id));

})

.catch((e) => console.log(e)); }

function handleSubmit(e) {

e.preventDefault();

if (!name || !description || !price || !category || !images.length) {

return alert("Please fill out all the fields"); }

createProduct({ name, description, price, category, images }).then(({ data }) => { if (data.length > 0) {

setTimeout(() => {navigate("/");}, 1500); }}); }

function showWidget() {

const widget = window.cloudinary.createUploadWidget(

{ cloudName: "lvmcloud",

uploadPreset: "ml\_default",},

(error, result) => {

if (!error && result.event === "success") {

setImages((prev) => [...prev, { url: result.info.url, public\_id: result.info.public\_id }]);}});

widget.open(); }

return (

<Container>

<Row>

<Col md={6} className="new-product\_\_form--container">

<Form style={{ width: "100%" }} onSubmit={handleSubmit}>

<h1 className="mt-4">НОВИЙ ПРОДУКТ</h1>

{isSuccess && <Alert variant="success">Продукт успішео створено</Alert>}

{isError && <Alert variant="danger">{error.data}</Alert>}

<Form.Group className="mb-3">

<Form.Label>Назва продукта</Form.Label>

<Form.Control type="text" placeholder="Назва продукта" value={name} required onChange={(e) => setName(e.target.value)} />

</Form.Group>

<Form.Group className="mb-3">

<Form.Label>Опис продукта</Form.Label>

<Form.Control as="textarea" placeholder="Опис продукта" style={{ height: "100px" }} value={description} required onChange={(e) => setDescription(e.target.value)} />

</Form.Group>

<Form.Group className="mb-3">

<Form.Label>Ціна(₴)</Form.Label>

<Form.Control type="number" placeholder="Ціна (₴)" value={price} required onChange={(e) => setPrice(e.target.value)} />

</Form.Group>

<Form.Group className="mb-3" onChange={(e) => setCategory(e.target.value)}>

<Form.Label>Категорія</Form.Label>

<Form.Select>

<option disabled selected>Категорія</option>

<option value="necklace">Кольє</option>

<option value="earring">Сережки</option>

<option value="bracelet">Браслети</option>

</Form.Select>

</Form.Group>

<Form.Group className="mb-3">

<Button type="button" onClick={showWidget}>

Завантажити фото</Button>

<div className="images-preview-container">

{images.map((image) => (

<div className="image-preview">

<img src={image.url} alt="" />

{imgToRemove !== image.public\_id && <i className="fa fa-times-circle" onClick={() => handleRemoveImg(image)}></i>}

</div>))}

</div>

</Form.Group>

<Button type="submit" disabled={isLoading || isSuccess}>

Створити продукт

</Button>

</Form>

</Col>

<Col md={6} className="new-product\_\_image--container"></Col>

</Row>

</Container> );}

export default NewProduct;

Додаток Ї – Лістинг коду компоненту <AdminDashboard />

import React from "react";

import { Container, Nav, Tab, Col, Row } from "react-bootstrap";

import ClientsAdminPage from "../components/ClientsAdminPage";

import DashboardProducts from "../components/DashboardProducts";

import OrdersAdminPage from "../components/OrdersAdminPage";

function AdminDashboard() {

return (

<Container>

<Tab.Container defaultActiveKey="products">

<Row>

<Col sm={3}>

<Nav variant="pills" className="flex-column">

<Nav.Item>

<Nav.Link eventKey="products">Продукти</Nav.Link>

</Nav.Item>

<Nav.Item>

<Nav.Link eventKey="orders">Замовлення</Nav.Link>

</Nav.Item>

<Nav.Item>

<Nav.Link eventKey="clients">Користувачі</Nav.Link>

</Nav.Item>

</Nav>

</Col>

<Col sm={9}>

<Tab.Content>

<Tab.Pane eventKey="products">

<DashboardProducts />

</Tab.Pane>

<Tab.Pane eventKey="orders">

<OrdersAdminPage />

</Tab.Pane>

<Tab.Pane eventKey="clients">

<ClientsAdminPage />

</Tab.Pane>

</Tab.Content>

</Col>

</Row>

</Tab.Container>

</Container>

);

}

export default AdminDashboard;

Додаток Й – Лістинг коду компоненту <EditProductPage />

import React, { useEffect, useState } from "react";

import { Alert, Col, Container, Form, Row, Button } from "react-bootstrap";

import { useNavigate, useParams } from "react-router-dom";

import { useUpdateProductMutation } from "../services/appApi";

import axios from "../axios";

import "./NewProduct.css";

function EditProductPage() {

const { id } = useParams();

const [name, setName] = useState("");

const [description, setDescription] = useState("");

const [price, setPrice] = useState("");

const [category, setCategory] = useState("");

const [images, setImages] = useState([]);

const [imgToRemove, setImgToRemove] = useState(null);

const navigate = useNavigate();

const [updateProduct, { isError, error, isLoading, isSuccess }] = useUpdateProductMutation();

useEffect(() => {

axios

.get("/products/" + id)

.then(({ data }) => {

const product = data.product;

setName(product.name);

setDescription(product.description);

setCategory(product.category);

setImages(product.pictures);

setPrice(product.price);

})

.catch((e) => console.log(e));

}, [id]);

function handleRemoveImg(imgObj) {

setImgToRemove(imgObj.public\_id);

axios

.delete(`/images/${imgObj.public\_id}/`)

.then((res) => {

setImgToRemove(null);

setImages((prev) => prev.filter((img) => img.public\_id !== imgObj.public\_id));

})

.catch((e) => console.log(e));

}

function handleSubmit(e) {

e.preventDefault();

if (!name || !description || !price || !category || !images.length) {

return alert("Please fill out all the fields");

}

updateProduct({ id, name, description, price, category, images }).then(({ data }) => {

if (data.length > 0) {

setTimeout(() => {

navigate("/");

}, 1500);

}

});

}

function showWidget() {

const widget = window.cloudinary.createUploadWidget(

{

cloudName: "lvmcloud",

uploadPreset: "ml\_default",

},

(error, result) => {

if (!error && result.event === "success") {

setImages((prev) => [...prev, { url: result.info.url, public\_id: result.info.public\_id }]);

}

}

);

widget.open();

}

return (

<Container>

<Row>

<Col md={6} className="new-product\_\_form--container">

<Form style={{ width: "100%" }} onSubmit={handleSubmit}>

<h1 className="mt-4">Редагування продукта</h1>

{isSuccess && <Alert variant="success">Продукт відредаговано</Alert>}

{isError && <Alert variant="danger">{error.data}</Alert>}

<Form.Group className="mb-3">

<Form.Label>Назва продукта</Form.Label>

<Form.Control type="text" placeholder="Назва продукта" value={name} required onChange={(e) => setName(e.target.value)} />

</Form.Group>

<Form.Group className="mb-3">

<Form.Label>Опис продукта</Form.Label>

<Form.Control as="textarea" placeholder="Опис продукта" style={{ height: "100px" }} value={description} required onChange={(e) => setDescription(e.target.value)} />

</Form.Group>

<Form.Group className="mb-3">

<Form.Label>Ціна(₴)</Form.Label>

<Form.Control type="number" placeholder="Ціна (₴)" value={price} required onChange={(e) => setPrice(e.target.value)} />

</Form.Group>

<Form.Group className="mb-3" onChange={(e) => setCategory(e.target.value)}>

<Form.Label>Category</Form.Label>

<Form.Select value={category}>

<option disabled selected>Категорія</option>

<option value="necklace">Кольє</option>

<option value="earring">Сережки</option>

<option value="bracelet">Браслети</option>

</Form.Select>

</Form.Group>

<Form.Group className="mb-3">

<Button type="button" onClick={showWidget}>

Завантажити фото

</Button>

<div className="images-preview-container">

{images.map((image) => (

<div className="image-preview">

<img src={image.url} alt="" />

{imgToRemove !== image.public\_id && <i className="fa fa-times-circle" onClick={() => handleRemoveImg(image)}></i>}

</div>

))}

</div>

</Form.Group>

<Form.Group>

<Button type="submit" disabled={isLoading || isSuccess}>

Редагувати

</Button>

</Form.Group>

</Form>

</Col>

<Col md={6} className="new-product\_\_image--container"></Col>

</Row>

</Container>

);

}

export default EditProductPage;

Додаток К – Лістинг файлу server.js

const express = require('express');

const cors = require('cors');

const app = express();

const http = require('http');

require('dotenv').config();

const stripe = require('stripe')(process.env.STRIPE\_SECRET);

require('./connection')

const server = http.createServer(app);

const {Server} = require('socket.io');

const io = new Server(server, {

cors: 'http://localhost:3001', methods: ['GET', 'POST', 'PATCH', "DELETE"]})

const User = require('./models/User');

const userRoutes = require('./routes/userRoutes');

const productRoutes = require('./routes/productRoutes');

const orderRoutes = require('./routes/orderRoutes');

const imageRoutes = require('./routes/imageRoutes');

app.use(cors());

app.use(express.urlencoded({extended: true}));

app.use(express.json());

app.use('/users', userRoutes);

app.use('/products', productRoutes);

app.use('/orders', orderRoutes);

app.use('/images', imageRoutes);

app.post('/create-payment', async(req, res)=> {

const {amount} = req.body;

try {const paymentIntent = await stripe.paymentIntents.create({amount, currency: 'uah', payment\_method\_types: ['card'] });

res.status(200).json(paymentIntent)} catch (e) { console.log(e.message);

res.status(400).json(e.message); }})

server.listen(8080, ()=> { console.log('server running at port', 8080)}) app.set('socketio', io);

Додаток Л – Лістинг файлу routes\userRoutes.js

const router = require('express').Router();

const User = require('../models/User');

const Order = require('../models/Order');

router.post('/signup', async(req, res)=> {

const {name, email, password} = req.body;

try { const user = await User.create({name, email, password});

res.json(user);

} catch (e) {

if(e.code === 11000) return res.status(400).send('Email already exists');

res.status(400).send(e.message) }})

router.post('/login', async(req, res) => {

const {email, password} = req.body;

try { const user = await User.findByCredentials(email, password);

res.json(user)

} catch (e) {

res.status(400).send(e.message) }})

router.get('/', async(req, res)=> {

try { const users = await User.find({ isAdmin: false }).populate('orders');

res.json(users);

} catch (e) {

res.status(400).send(e.message); }})

router.get('/:id/orders', async (req, res)=> {

const {id} = req.params;

try { const user = await User.findById(id).populate('orders');

res.json(user.orders);

} catch (e) { res.status(400).send(e.message); }})

module.exports = router;

Додаток М – Лістинг файлу routes\productRoutes.js

const router = require('express').Router();

const Product = require('../models/Product');

const User = require('../models/User');

//get products;

router.get('/', async(req, res)=> {

try {

const sort = {'\_id': -1}

const products = await Product.find().sort(sort);

res.status(200).json(products);

} catch (e) {

res.status(400).send(e.message); }})

//create product

router.post('/', async(req, res)=> {

try {

const {name, description, price, category, images: pictures} = req.body;

const product = await Product.create({name, description, price, category, pictures});

const products = await Product.find();

res.status(201).json(products);

} catch (e) {

res.status(400).send(e.message); }})

// update product

router.patch('/:id', async(req, res)=> {

const {id} = req.params;

try {

const {name, description, price, category, images: pictures} = req.body;

const product = await Product.findByIdAndUpdate(id, {name, description, price, category, pictures});

const products = await Product.find();

res.status(200).json(products);

} catch (e) {

res.status(400).send(e.message); }})

// delete product

router.delete('/:id', async(req, res)=> {

const {id} = req.params;

const {user\_id} = req.body;

try {

const user = await User.findById(user\_id);

if(!user.isAdmin) return res.status(401).json("You don't have permission");

await Product.findByIdAndDelete(id);

const products = await Product.find();

res.status(200).json(products);

} catch (e) {

res.status(400).send(e.message); }})

router.get('/:id', async(req, res)=> {

const {id} = req.params;

try {

const product = await Product.findById(id);

const similar = await Product.find({category: product.category}).limit(5);

res.status(200).json({product, similar})

} catch (e) {

res.status(400).send(e.message); }});

router.get('/category/:category', async(req,res)=> {

const {category} = req.params;

try { let products;

const sort = {'\_id': -1}

if(category == "all"){

products = await Product.find().sort(sort);

} else {

products = await Product.find({category}).sort(sort) }

res.status(200).json(products)

} catch (e) {

res.status(400).send(e.message); }})

// cart routes

router.post('/add-to-cart', async(req, res)=> {

const {userId, productId, price} = req.body;

try {

const user = await User.findById(userId);

const userCart = user.cart;

if(user.cart[productId]){

userCart[productId] += 1;

} else {

userCart[productId] = 1; }

userCart.count += 1;

userCart.total = Number(userCart.total) + Number(price);

user.cart = userCart;

user.markModified('cart');

await user.save();

res.status(200).json(user);

} catch (e) {

res.status(400).send(e.message); }})

router.post('/increase-cart', async(req, res)=> {

const {userId, productId, price} = req.body;

try {

const user = await User.findById(userId);

const userCart = user.cart;

userCart.total += Number(price);

userCart.count += 1;

userCart[productId] += 1;

user.cart = userCart;

user.markModified('cart');

await user.save();

res.status(200).json(user);

} catch (e) {

res.status(400).send(e.message); }});

router.post('/decrease-cart', async(req, res)=> {

const {userId, productId, price} = req.body;

try {

const user = await User.findById(userId);

const userCart = user.cart;

if (userCart[productId] > 1) {

userCart.total -= Number(price);

userCart.count -= 1;

userCart[productId] -= 1;

user.cart = userCart;

user.markModified('cart');

await user.save();

res.status(200).json(user);

} else {

res.status(200).json(user);

}

} catch (e) {

res.status(400).send(e.message);

}})

router.post('/remove-from-cart', async(req, res)=> {

const {userId, productId, price} = req.body;

try {

const user = await User.findById(userId);

const userCart = user.cart;

userCart.total -= Number(userCart[productId]) \* Number(price);

userCart.count -= userCart[productId];

delete userCart[productId];

user.cart = userCart;

user.markModified('cart');

await user.save();

res.status(200).json(user);

} catch (e) {

res.status(400).send(e.message);

}

})

module.exports = router;

Додаток Н – Лістинг файлу routes\orderRoutes.js

const router = require('express').Router();

const Order = require('../models/Order');

const User = require('../models/User');

//creating an order

router.post('/', async(req, res)=> {

const io = req.app.get('socketio');

const {userId, cart, country, address} = req.body;

try {

const user = await User.findById(userId);

const order = await Order.create({owner: user.\_id, products: cart, country, address});

order.count = cart.count;

order.total = cart.total;

await order.save();

user.cart = {total: 0, count: 0};

user.orders.push(order);

const notification = {status: 'unread', message: `New order from ${user.name}`, time: new Date()};

io.sockets.emit('new-order', notification);

user.markModified('orders');

await user.save();

res.status(200).json(user)

} catch (e) {

res.status(400).json(e.message)

}

})

// getting all orders;

router.get('/', async(req, res)=> {

try {

const orders = await Order.find().populate('owner', ['email', 'name']);

res.status(200).json(orders);

} catch (e) {

res.status(400).json(e.message)

}

})

//shipping order

router.patch('/:id/mark-shipped', async(req, res)=> {

const io = req.app.get('socketio');

const {ownerId} = req.body;

const {id} = req.params;

try {

const user = await User.findById(ownerId);

await Order.findByIdAndUpdate(id, {status: 'shipped'});

const orders = await Order.find().populate('owner', ['email', 'name']);

const notification = {status: 'unread', message: `Order ${id} shipped with success`, time: new Date()};

io.sockets.emit("notification", notification, ownerId);

user.notifications.push(notification);

await user.save();

res.status(200).json(orders)

} catch (e) {

res.status(400).json(e.message);

}

})

module.exports = router;

Додаток О – Лістинг файлів схем і моделей бази даних

//Файл models\User.js

const mongoose = require('mongoose');

const bcrypt = require('bcrypt');

const UserSchema = mongoose.Schema({

name: {

type: String,

required: [true, 'is required']

},

email: { type: String,

required: [true, 'is required'],

unique: true,

index: true,

validate: { validator: function(str){

return /^[\w-\.]+@([\w-]+\.)+[\w-]{2,4}$/g.test(str); },

message: props => `${props.value} is not a valid email` } },

password: { type: String,

required: [true, 'is required'] },

isAdmin: { type: Boolean,

default: false },

cart: { type: Object, default: {

total: 0,

count: 0 } },

notifications: { type: Array,

default: [] },

orders: [{type: mongoose.Schema.Types.ObjectId, ref: 'Order'}]

}, {minimize: false});

UserSchema.statics.findByCredentials = async function(email, password) {

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

if(!user) throw new Error('invalid credentials');

const isSamePassword = bcrypt.compareSync(password, user.password);

if(isSamePassword) return user;

throw new Error('invalid credentials');}

UserSchema.methods.toJSON = function(){

const user = this;

const userObject = user.toObject();

delete userObject.password;

return userObject;}

//before saving => hash the password

UserSchema.pre('save', function (next) {

const user = this;

if(!user.isModified('password')) return next();

bcrypt.genSalt(10, function(err, salt){

if(err) return next(err);

bcrypt.hash(user.password, salt, function(err, hash){

if(err) return next(err);

user.password = hash;

next(); }) })})

UserSchema.pre('remove', function(next){

this.model('Order').remove({owner: this.\_id}, next); })

const User = mongoose.model('User', UserSchema);

module.exports = User;

//Файл models\Product.js

const mongoose = require('mongoose');

const ProductSchema = mongoose.Schema({

name: { type: String,

required: [true, "can't be blank"] },

description: { type: String,

required: [true, "can't be blank"] },

price: { type: String,

required: [true, "can't be blank"] },

category: { type: String,

required: [true, "can't be blank"] },

pictures: { type: Array,

required: true }

}, {minimize: false});

const Product = mongoose.model('Product', ProductSchema);

module.exports = Product;

//Файл models\Order.js

const mongoose = require('mongoose');

const OrderSchema = mongoose.Schema({

products: { type: Object },

owner: {

type: mongoose.Schema.Types.ObjectId,

ref: 'User',

required: true },

status: { type: String,

default: 'processing' },

total : { type: Number,

default: 0 },

count: { type: Number,

default: 0 },

date: { type: String,

default: new Date().toISOString().split('T')[0] },

address: { type: String, },

country: { type: String, }

}, {minimize: false});

const Order = mongoose.model('Order', OrderSchema);

module.exports = Order;

Додаток П – Лістинг файлу src\App.test.js

import { render, screen } from '@testing-library/react';

import App from './App';

test('renders learn react link', () => {

render(<App />);

const linkElement = screen.getByText(/learn react/i);

expect(linkElement).toBeInTheDocument();

});

Додаток Р – Лістинг файлу src\AppInt.test.js

import React from 'react';

import { render, screen } from '@testing-library/react';

import userEvent from '@testing-library/user-event';

import App from './App';

test('renders the app and performs an interaction', () => {

render(<App />);

// Перевіряємо, чи компонент App відображений на екрані

const appElement = screen.getByTestId('app');

expect(appElement).toBeInTheDocument();

// Знаходимо елемент кнопки і виконуємо клік на неї

const buttonElement = screen.getByRole('button');

userEvent.click(buttonElement);

// Перевіряємо, чи змінилося значення тексту після кліку на кнопку

const textElement = screen.getByText('Button clicked!');

expect(textElement).toBeInTheDocument();

});

Додаток С – Приклад інтеграційного тесту для маршруту на Express

const request = require('supertest');

const app = require('../app');

describe('GET /api/users', () => {

test('should return an array of users', async () => {

const response = await request(app).get('/api/users');

expect(response.status).toBe(200);

expect(response.body).toBeInstanceOf(Array);

});

});