МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Сумський державний університет

Факультет електроніки та інформаційних технологій

Кафедра комп'ютерних наук

«До захисту допущено»

В.о. завідувача кафедри

_____ Ігор ШЕЛЕХОВ

(підпис)

червня 2024 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня бакалавр

зі спеціальності 122 - Комп'ютерних наук,

освітньо-професійної програми «Інформатика»

на тему: «Інтернет магазин з продажу прикрас ручної роботи»

здобувача групи ІН-01 Картавого Івана Олександровича

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

Іван КАРТАВИЙ

_____

Керівник,
доцент кафедри комп'ютерних наук,
кандидат фізико-математичних наук

Надія ТИРКУСОВА

_____

Суми – 2024

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«Затверджую»
В.о. завідувача кафедри
_____ Ігор ШЕЛЕХОВ
(підпис)

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавра

зі спеціальності 122 - Комп'ютерних наук, освітньо-професійної програми «Інформатика»
здобувача групи ІН-01 Картавого Івана Олександровича

1. Тема роботи: «Інтернет магазин з продажу прикрас ручної роботи»
затверджую наказом по СумДУ від *«00» червня 2024 р. № № 0414-IV*
2. Термін здачі здобувачем кваліфікаційної роботи *до 28 травня 2024 року*
3. Вхідні дані до кваліфікаційної роботи
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) *1) Аналіз проблеми предметної області, постановка й формування завдань дослідження. 2) Огляд технологій, що використовуються для розробки інтернет магазину. 3) Розробка інтернет магазину з продажу прикрас ручної роботи. 4) Аналіз результатів.*

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «____» _____ 20 ___ р.

Завдання прийняв до виконання _____ Керівник _____
(підпис) (підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	*Аналіз проблеми предметної області, постановка й формування завдань дослідження*	15.04 - 17.04	
2	*Огляд технологій, що використовуються для розробки інтернет магазину*	17.04 - 20.04	
3	*Розробка інтернет магазину для продажу ляльок ручної роботи*	21.04 - 29.05	
4	*Аналіз отриманих результатів*	30.05 - 02.06	
5	*Оформлення пояснювальної записки до кваліфікаційної роботи*	03.06 - 08.06	

Здобувач вищої освіти _____ Керівник _____

АНОТАЦІЯ

Записка: 54 стр., 36 рис., 2 додатка, 21 використаних джерел.

Обґрунтування актуальності теми роботи – Тема кваліфікаційної роботи є актуальною, оскільки вона поєднує в собі зростанню електронної комерції та популярності онлайн-покупок, росту інтересу до рукодільництва та унікальних виробів, підтримку малих підприємств і рукодільників.

Об’єкт дослідження — процес продажу прикрас ручної роботи, а саме в'язаних браслетів.

Мета роботи — розробка інтернет магазину для продажу браслетів ручної роботи.

Методи дослідження — алгоритми створення веб додатків(інтернет магазинів) з використанням обраного стеку технологій

Результати — розроблено інтернет магазин у вигляді веб додатку, за допомогою якого користувач має можливість передивитись список доступних браслетів та придбати їх. Кожен браслет в веб додатку має свій реальний прототип.

ЗМІСТ

ВСТУП

Актуальність. Тема кваліфікаційної роботи є актуальною, оскільки вона поєднує в собі зростанню електронної комерції та популярності онлайн-покупок, росту інтересу до рукодільництва та унікальних виробів, підтримку малих підприємств і рукодільників.

Об’єкт дослідження. Процес продажу ручної роботи, а саме в'язаних браслетів.

Предмет дослідження. Розвиток та ефективність інтернет-магазину в'язаних браслетів в сучасних умовах.

Гіпотеза. Створення інтернет-магазину в'язаних браслетів та використання маркетингових стратегій сприяє збільшенню прибутковості та популярності цього виду рукодільництва серед онлайн-покупців.

Наукова новизна. Створення веб додатку дає можливість поєднати технологічний розвиток з підтримкою творчого потенціалу і малих бізнесів, розширюючи можливості для всіх зацікавлених майстрів і сприяти розвитку галузі виготовлення в'язаних браслетів.

Структура. Дана робота складається зі вступу, аналітичного огляду, постановки задачі, вибір методу розв’язання поставленої задачі, опису програмного забезпечення інформаційної системи, висновків, списку використаних джерел та додатків.

1 АНАЛІТИЧНИЙ ОГЛЯД

1.1 Сучасний стан

На сьогоднішній день в'язані браслети є популярними прикрасами, особливо серед молоді та людей, які цінують ручну роботу та індивідуальний стиль. Це своєрідний вид рукодільництва, де кожен виріб унікальний і може виражати той чи інший смак або емоції.

Зростання електронної комерції в останні роки дало поштовх для розвитку інтернет-магазинів, спеціалізованих на в'язаних браслетах. Онлайн-платформи надають можливість покупцям легко знайти та придбати бажані вироби з будь-якого місця та у будь-який час. Інтернет-магазини надають зручну платформу для покупців, які можуть переглядати товари, здійснювати покупки та оплачувати їх онлайн, що робить процес покупок більш доступним і зручним.

Ринок в'язаних браслетів у мережі насичений різноманітними виробниками та магазинами. Це стимулює конкуренцію та робить важливими аспекти, які вирізняють вас від інших, такі як якість виробів, дизайн, цінова стратегія і взаємодія з клієнтами.

Також важливою аспектом є підтримка малих підприємців і рукодільників через підтримку та усвідомленість споживачів. Багато людей виявляють бажання підтримувати такі малі бізнеси, оскільки вони розвивають різноманітність інтернет-торгівлі та рукодільних виробів.

Зважаючи на ці фактори, створення інтернет-магазину в'язаних браслетів може бути перспективним і важливим для розвитку цього бізнесу.

1.2 Аналіз аналогічних проєктів

Інтернет-магазин в'язаних браслетів є частиною ринку fashion-ритейлу, який активно розвивається та продовжує зростати не тільки в Україні, а й в усьому світі. Одним із факторів, що сприяли цьому зростанню, була пандемія COVID-19, яка змусила багатьох споживачів перейти до онлайн-покупок через карантинні обмеження. В'язані браслети належать до категорії аксесуарів, які є

популярними серед споживачів, особливо молодого покоління. Вони мають ряд переваг, таких як:

- **Оригінальність**. В'язані браслети можна створювати в різних кольорах, формах і стилях, що дозволяє виражати свою індивідуальність і настрій.
- **Доступність**. В'язані браслети не вимагають великих витрат на матеріали і виробництво, тому їх ціна зазвичай нижча, ніж у інших аксесуарів.
- **Екологічність**. В'язані браслети можна виготовляти з натуральних або перероблених матеріалів, що зменшує негативний вплив на навколишнє середовище.

Однак, ринок інтернет-магазинів в'язаних браслетів також має деякі виклики, такі як:

- **Конкуренція**. Ринок аксесуарів є дуже насиченим і розмаїтим, тому інтернет-магазинам в'язаних браслетів потрібно виділятися серед інших пропозицій за якістю, дизайном і сервісом.
- **Сезонність**. В'язані браслети можуть бути менш популярними в холодну пору року, коли люди одягаються тепліше і приховують свої аксесуари під одягом.
- **Лояльність**. В'язані браслети є досить простими і швидкими у виготовленні, тому споживачі можуть самостійно створювати або замовляти їх у ручної роботи майстрів, замість купувати їх у інтернет-магазинах.

Ринок інтернет-магазинів рукодільних виробів є невеликою, але перспективною нішею на українському ринку інтернет-торгівлі. Цей сегмент має високий потенціал зростання через попит на унікальні та авторські товари, якими можна показати свою індивідуальність та креативність. Також цей сегмент приваблює споживачів своїми екологічністю та соціальною відповідальністю, оскільки багато рукодільників використовують натуральні матеріали.

Таким чином, ринок інтернет-магазинів в'язаних браслетів є перспективним і цікавим для аналізу.

Серед основних гравців на ринку інтернет-магазинів в’язаних браслетів можна виділити таких:

«*Etsy*» — міжнародна платформа для продажу та покупки товарів ручної роботи та вінтажних предметів. Заснована у 2005 році у США, нараховує понад 81 млн покупців та 4,4 млн продавців у 200 країнах світу. Серед товарів ручної роботи можна знайти велику кількість в'язаних браслетів на різний смак.

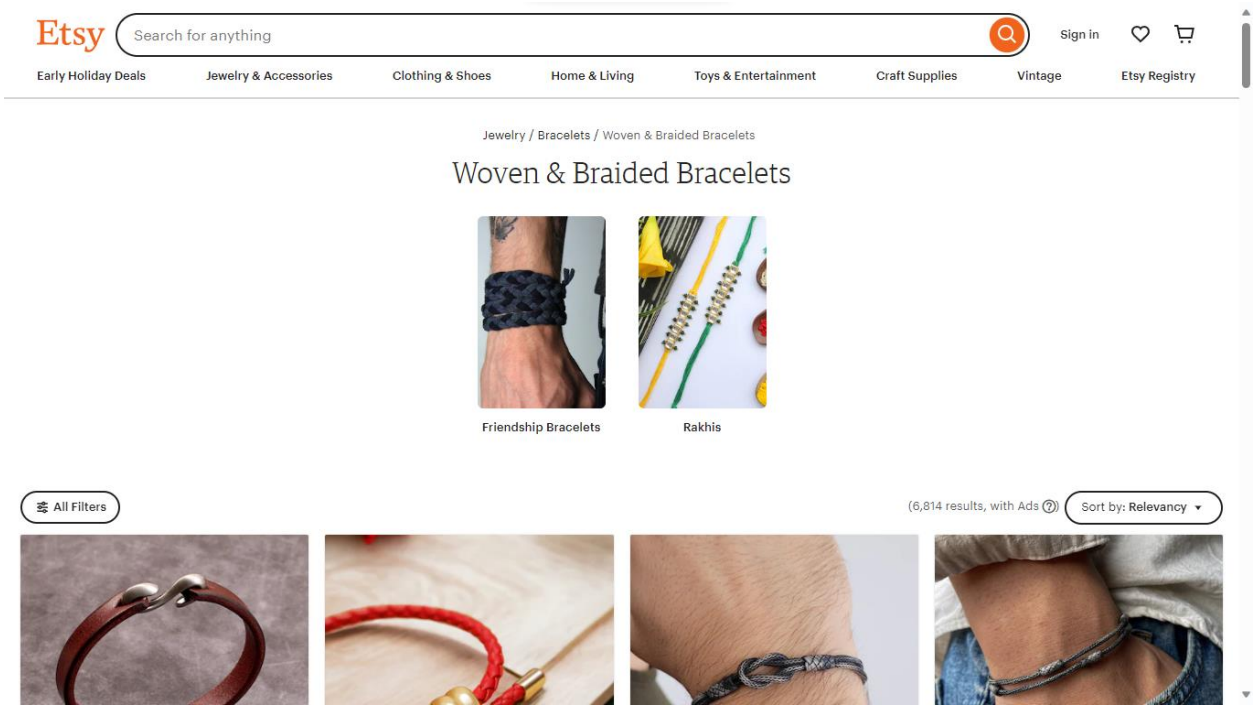

Рисунок 1.1 – Приклад магазину: «Etsy»

«*Crafta*» — це українська онлайн-платформа для продажу авторських виробів та предметів колекціонування. У розділі Хендмейд представлені тисячі авторських товарів українських майстрів та локальних брендів.

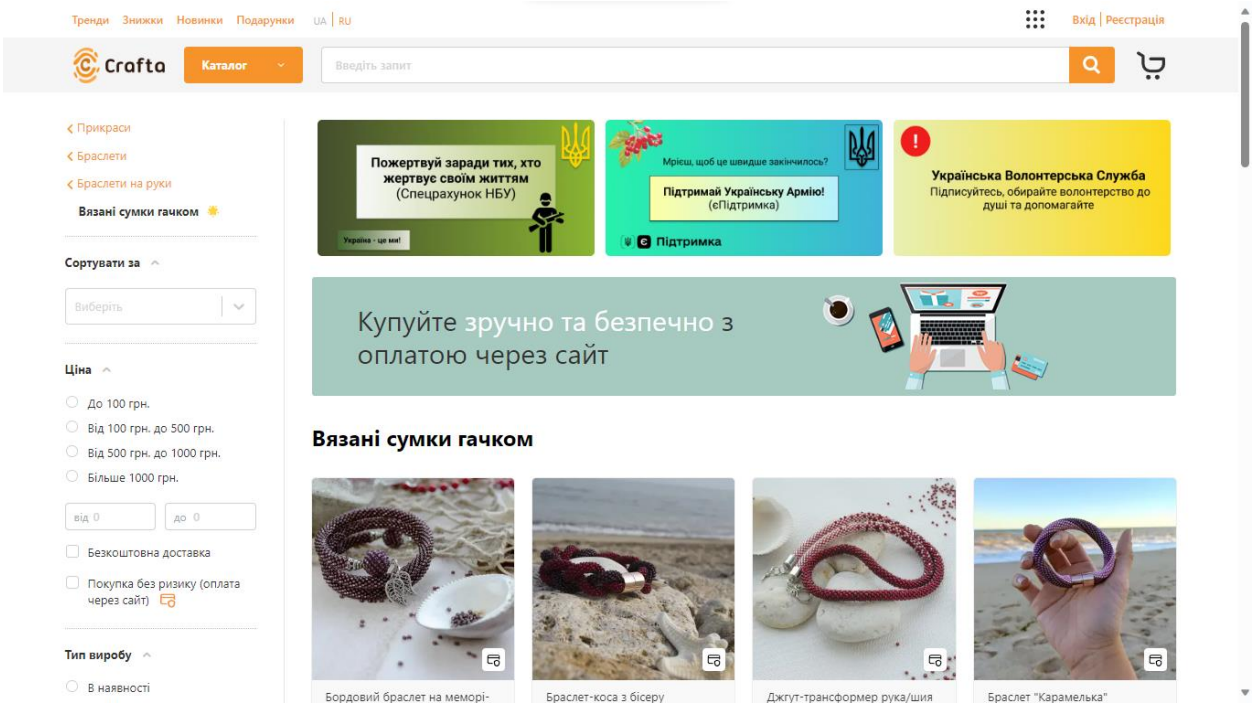

Рисунок 1.2 – Приклад магазину: «Crafta»

На основі цих даних можна скласти таблицю SWOT-аналізу для інтернет-магазину рукодільних виробів:

Сильні сторони	Слабкі сторони
- Висока цінність та унікальність товарів - Екологічність та соціальна відповідальність - Креативність та індивідуальність	- Низька конкурентоспроможність за ціною - Обмежена аудиторія споживачів - Висока залежність від сезонності

Можливості	Загрози
- Зростання попиту на онлайн-покупки через пандемію - Розвиток нових каналів комунікації з клієнтами (соцмережі, блоги, вебінари тощо)	- Сильна конкуренція з боку глобальних платформ - Зменшення доходів споживачів через економічну кризу

- Розширення асортименту та введення нових послуг (доставка, гарантія, консультація тощо) - Співпраця з іншими рукодільниками та організаціями (спільноти, фестивалі, благодійні проекти тощо)	- Порушення прав інтелектуальної власності та підробка товарів - Зміна смаків та потреб споживачів

1.3 Постановка задачі

Мета роботи — розробка інтернет магазину для продажу браслетів ручної роботи та підтримки місцевих майстрів ручної роботи.

Задачі:

1. Визначення цілі і цільової аудиторії:
    - Визначення продуктів для продажу (в'язані браслети).
    - Визначення цінової політику та маржи прибутку.
    - Вибір цільової аудиторії для магазину.
2. Планування проєкту:
    - Визначення, які функціональність і функціональні вимоги повинні бути реалізовані в вашому інтернет-магазині (наприклад, корзина, оплата, адміністративна панель тощо).
3. Вибір технологій:
    - Розгляд використання React для фронтенду інтернет-магазину.
    - Вибір інструментів для створення бекенду та бази даних
4. Розробка фронтенду:
    - Створення користувацького інтерфейсу магазину з використанням React.
    - Розробка сторінки для відображення продуктів, деталей продукту, корзини та оформлення замовлення.

- Забезпечення адаптивності і дружнього інтерфейсу для користувачів.

5. Розробка бекенду:
    - Створення серверу для обробки запитів користувачів (наприклад, додавання товарів у корзину, обробка замовлень).
    - Забезпечення безпеки та аутентифікації користувачів.
    - Інтегрування платіжної системи для обробки платежів.
6. База даних:
    - Створення бази даних для збереження інформації про продукти, користувачів, замовлення тощо.
7. Тестування:
    - Проведення тестування для перевірки роботи інтернет-магазину.
    - Виявлення та виправлення помилок.
8. Розгортання та хостинг:
    - Розгортання свого магазину на веб-сервері.
    - Забезпечення надійного хостингу та безпеки даних користувачів.
9. Підтримка і оновлення:
    - Продовження підтримки та оновлення магазину, враховуючи фідбек користувачів і нові можливості.

Ці кроки утворюють стратегію створення інтернет-магазину в'язаних браслетів, метою якого є забезпечення того, щоб інтернет-магазин був успішним, конкурентоспроможним і здатним задовольняти потреби клієнтів.

2 ВИБІР МЕТОДУ РОЗВ'ЯЗАННЯ ЗАДАЧІ

2.1 Вибір технологій і стеку

Технологічний стек - це набір технологій, які ви вибираєте та використовуєте для створення веб-додатків, мобільних додатків або подібних додатків. Хороший стек технологій має забезпечувати безперебійну роботу користувача, а також бути масштабованим і економічно ефективним. Типовий технологічний стек містить інтерфейс, серверну частину та базу даних і відомий як повний технологічний стек.

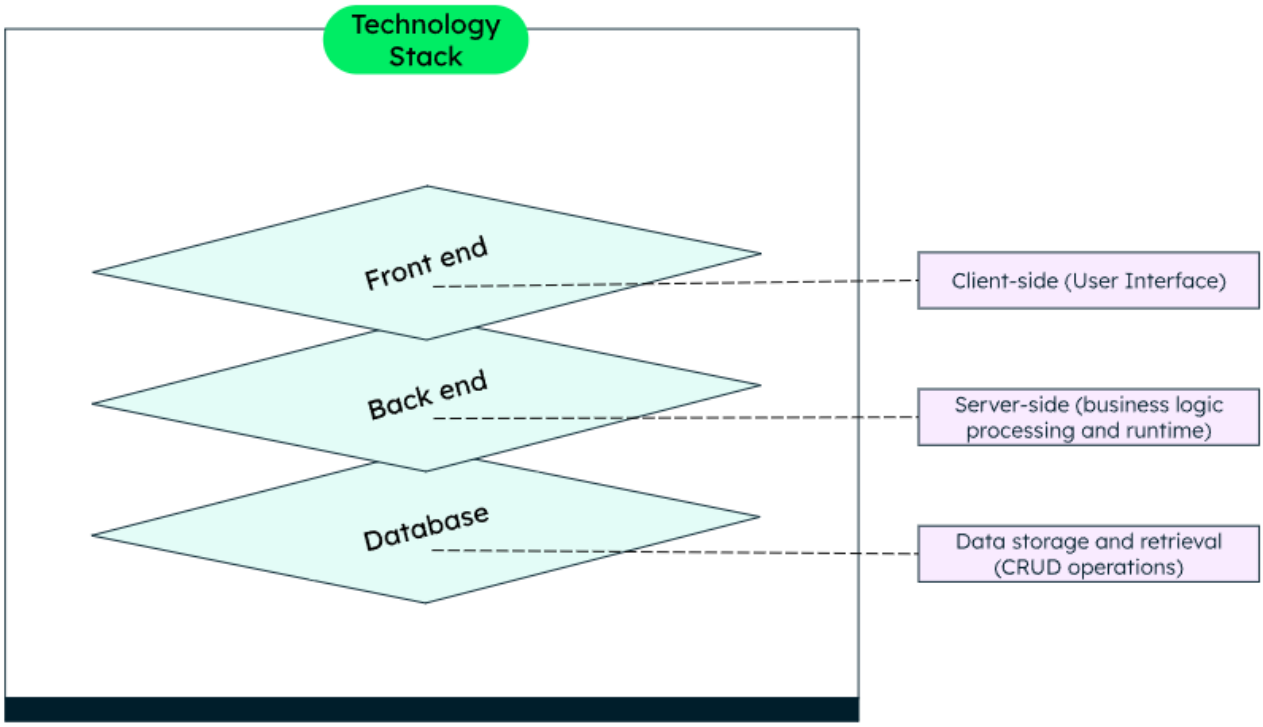

Рисунок 2.1 – Технологічний стек

Стек MERN - це абревіатура, яка включає в себе чотири ключові технології, які часто використовуються для розробки сучасних веб-додатків. MERN означає **M** ongoDB, **E** express, **R** react і **N** ode, після чотирьох ключових технологій, які складають стек.

- MongoDB — база даних документів
- Express(.js) — веб-фреймворк Node.js
- React(.js) — фреймворк JavaScript на стороні клієнта
- Node(.js) — головний веб-сервер JavaScript (час виконання)

Express і Node складають середній (програмний) рівень. Express.js — це веб-фреймворк на стороні сервера, а Node.js — популярна та потужна серверна платформа JavaScript. Незалежно від того, який варіант ви виберете, MERN є ідеальним підходом до роботи з JavaScript і JSON на всьому протязі. Ось розшифровка та пояснення, що представляє собою стек MERN і чому саме його я вибрав для розробки інтернет-магазину.[1]

MongoDB і нелінійна структура даних:

MongoDB - це документ-орієнтована база даних без схеми, яка зберігає дані у вигляді JSON-подібних документів. Вона дозволяє легко впорядковувати та оновлювати дані, коли змінюється структура товарів чи замовлень. Це ідеально підходить для розробки веб-магазину, оскільки є можливість зберігати дані про товари, клієнтів та замовлення у зручному форматі. Як рішення бази даних NoSQL, MongoDB не потребує системи керування реляційною базою даних (RDBMS), тому вона забезпечує еластичну модель зберігання даних, яка дозволяє користувачам легко зберігати та запитувати багатовимірні типи даних. Це не тільки спрощує керування базами даних для розробників, але й створює високомасштабоване середовище для кросплатформних програм і служб.

Документи MongoDB або колекції документів є основними одиницями даних. Ці документи, відформатовані як двійковий JSON (нотація об’єктів Java Script), можуть зберігати різні типи даних і розподілятися між кількома системами. Оскільки MongoDB використовує динамічний дизайн схеми, користувачі мають неперевершену гнучкість під час створення записів даних, запитів до колекцій документів через агрегацію MongoDB та аналізу великих обсягів інформації. Документи MongoDB можуть бути легко адаптовані до змін, що дозволяє швидко реагувати на потреби магазину та покращувати функціональність без значного рефакторингу. Ця база даних також допомагає розвивати проект швидше завдяки своїй гнучкості. [2][3]

React для інтерфейсу користувача:

React - це бібліотека для розробки інтерфейсу користувача, яка дозволяє створювати ефективні та інтерактивні веб-сайти. Він надає можливість

створювати компоненти, які легко підтримувати, розширювати та перевикористовувати. React робить створення інтерактивних інтерфейсів безболісним. Створювати прості представлення для кожного стану програми, і React ефективно оновлюватиме та відтворюватиме потрібні компоненти, коли дані змінюватимуться. У контексті інтернет-магазину, React дозволить створити користувацький інтерфейс, який відображає товари, кошик та інші складові веб-сторінки інтерактивно та ефективно.

Однією з основних переваг React є його віртуальний DOM (Document Object Model). Коли стан додатку змінюється, React спочатку оновлює віртуальний DOM, а потім ефективно порівнює його зі справжнім DOM, щоб вносити мінімальні зміни, що дозволяє значно збільшити продуктивність.

React також підтримує однонаправлене передавання даних, що полегшує керування станом та потоком даних у додатку. Це особливо корисно в масштабних програмах, де складність даних може швидко зрости.

Ще однією важливою рисою React є підтримка JSX (JavaScript XML) - синтаксисного розширення JavaScript, яке дозволяє писати HTML-подібний код у JavaScript. Це робить код більш читабельним і зручним для розробників. [4][5][6]

Node.js з Express.js для серверної частини:

Використання Node.js з Express.js для серверної частини полягає в їх ефективності та сумісності з реактивним підходом, який популярний в розробці сучасних веб-додатків.

Express.js - це фреймворк для розробки веб-додатків на платформі Node.js. Він дозволяє легко створювати сервери та API, обробляти маршрути та запити, і взагалі прискорює розробку серверної частини. В контексті інтернет-магазину, Express.js допоможе створити API для обробки замовлень, аутентифікації користувачів та багато іншого.

Основні переваги Express.js:

- **Легкість у використанні:** Простий синтаксис та мінімум конфігурацій роблять Express.js легким у вивченні та використанні.
- **Гнучкість:** Можливість додавання різних проміжних шарів для обробки запитів дозволяє легко розширювати функціональність вашого додатку.
- **Модульність:** Підтримка модульності дозволяє легко організовувати код та повторно використовувати компоненти.
- **Підтримка шаблонів:** Підтримка різних механізмів шаблонів (наприклад, Pug, EJS), що спрощує генерацію HTML сторінок на сервері.

Node.js - це середовище виконання JavaScript на серверній стороні, яке забезпечує високу продуктивність та можливість обробки багатьох одночасних запитів, що особливо важливо для інтернет-магазину, де важлива реактивність та обробка багатьох одночасних запитів. У поєднанні з Express.js, Node.js дозволить створити швидкий та ефективний сервер для інтернет-магазину.

Основні переваги Node.js:

- **Висока продуктивність:** Завдяки асинхронній моделі та подіям, Node.js може обробляти багато запитів одночасно, що забезпечує швидкий відгук системи навіть під великим навантаженням.
- **Єдиний стек технологій:** Можливість використовувати JavaScript як на клієнтській, так і на серверній частині, що спрощує процес розробки та дозволяє повторно використовувати код.
- **Широкий спектр модулів:** Node.js має величезну екосистему модулів через npm (Node Package Manager), що дозволяє швидко додавати нові функціональні можливості до вашого додатку.
- **Спільнота:** Велика та активна спільнота розробників, що постійно працює над покращенням та розширенням можливостей Node.js.

Використання в інтернет-магазині

Для інтернет-магазину Express.js та Node.js є ідеальним вибором з огляду на наступні аспекти:

- **Аутентифікація та авторизація:** Створення надійної системи реєстрації та входу користувачів, що дозволяє захищати особисті дані та забезпечувати доступ до певних функцій тільки авторизованим користувачам.
- **Обробка замовлень:** Створення API для управління товарами, кошиком, замовленнями та оплатами.
- **Інтеграція з базами даних:** Легка інтеграція з MongoDB або іншими базами даних для зберігання даних про користувачів, товари та замовлення.
- **Висока продуктивність:** Можливість обробки великої кількості одночасних запитів, що забезпечує швидку реакцію системи навіть при великій кількості користувачів.
- **Гнучкість та масштабованість:** Легкість у масштабуванні додатку шляхом додавання нових функціональних можливостей або розподілу навантаження між кількома серверами.

Таким чином, стек MERN об'єднує різні технології, які відмінно доповнюють одна одну і сприяють розробці високоякісних веб-додатків. Вибір MERN для розробки інтернет-магазину може спростити процес розробки, забезпечити гнучкість у зберіганні та обробці даних, а також надати зручність у створенні інтерфейсу користувача та обробці запитів на сервері. [7][8][9]

HTML, CSS, і JavaScript також є важливими частинами розробки веб-додатків і грають ключову роль в успішному створенні інтернет-магазину. Ось як кожна з цих технологій доповнює стек MERN:

HTML (HyperText Markup Language):

HTML є основою будь-якого веб-сайту. HTML — стандартизована мова розмітки документів у Всесвітній павутині за допомогою тегів. Тобто, HTML документ буде складатися з деякої групи елементів, де кожен елемент буде

визначатися (починатися та закінчуватися) певним тегом (Для деяких елементів кінцевий тег не є обов'язковим). Кожен HTML тег має свою унікальну назву з визначеним синтаксисом, яка записується латинськими літерами і не чутлива до регістру. Більшість веб-сторінок мають зміст розмітки мовою HTML (або XHTML). Мова HTML інтерпретується браузерами; отриманий в результаті інтерпретації форматований текст відображається на екрані монітора комп’ютера або мобільного пристрою. TML документ оброблюється браузером та відтворюється на екрані у звичному для людини вигляді.

У контексті інтернет-магазину, HTML дозволяє створювати основну структуру сторінок, таких як головна сторінка, сторінка товарів, сторінка кошика та інші. HTML забезпечує основу для розташування елементів і їх взаємодії на веб-сторінках, що дозволяє розробникам визначити, як буде виглядати і функціонувати інтерфейс магазину. [10][11]

CSS (Cascading Style Sheets):

CSS — спеціальна мова, яка використовується для опису зовнішнього вигляду сторінок, написаних мовами розмітки даних. CSS відповідає за візуальне оформлення веб-сторінок. Найбільш часто CSS використовують для візуальної презентації сторінок, написаних на HTML та XHTML, але формат CSS може застосовуватись і до інших видів XML-документів. CSS використовується для обробки деяких частин веб-сторінки. За допомогою CSS ми можемо керувати кольором тексту та стилем шрифтів, інтервалом між абзацами та багатьма іншими речами. CSS простий для розуміння, але забезпечує потужний контроль над документами Html. CSS поєднується з HTML. [12][13]

У випадку інтернет-магазину, CSS допоможе створити привабливий та зручний для користувачів інтерфейс, що сприятиме кращому користувацькому досвіду. CSS також дозволяє реалізовувати адаптивний дизайн, що робить сайт доступним і зручним для користувачів на різних пристроях, включаючи смартфони, планшети та настільні комп'ютери.

JavaScript:

Мова програмування JavaScript є текстовою і може використовуватися як на стороні клієнта, так і на стороні сервера. Він керує мультимедіа на веб-сторінках і дозволяє їм стати інтерактивними. JavaScript дозволяє розробнику виконувати багато речей, наприклад додавати анімацію до зображень або автоматично оновлювати вміст на сторінці. Функції JavaScript можуть поліпшити зручність взаємодії користувача з веб-сайтом: від оновлення стрічки новин у соціальних мережах і до відображення анімації та інтерактивних карт. JavaScript є мовою програмування під час розроблення скриптів для виконання на стороні клієнта, що робить його однією з базових технологій у всесвітній мережі Інтернет. Наприклад, карусель зображення, меню, що випадає за кліком, і динамічно мінливі кольори елементів на веб-сторінці, які ви бачите під час перегляду сторінок в Інтернеті, виконані за допомогою JavaScript. [14][15]

У контексті інтернет-магазину, JavaScript використовується для реалізації функціональності, такої як динамічне оновлення кошика, обробка форм, перевірка даних користувачів та інтерактивність товарів. JavaScript також є основою для роботи з React, який дозволяє створювати компоненти користувацького інтерфейсу і взаємодіяти з сервером через API, створені на Express.js. Це допомагає забезпечити швидке та плавне користувацьке взаємодію з веб-сайтом.

Інтеграція з MERN стеком:

HTML, CSS та JavaScript відмінно інтегруються з MERN стеком, доповнюючи його можливості. Використовуючи React, розробники створюють компоненти, які генерують HTML та CSS, а також обробляють JavaScript для динамічної взаємодії. Серверна частина на Node.js та Express.js обробляє запити від клієнта, а MongoDB зберігає та надає дані, необхідні для функціонування інтернет-магазину. Таким чином, MERN стек у поєднанні з HTML, CSS та JavaScript створює потужну та гнучку платформу для розробки сучасних веб-додатків, забезпечуючи швидкий розвиток, високу продуктивність та зручність використання.

2.2 Дизайн бази даних

Дизайн бази даних - це ключовий аспект розробки інтернет-магазину, оскільки він визначає, як ваша система буде зберігати та доступ до даних про браслети, клієнтів та замовлення. Ось більш детальне розгляд кожного з підпунктів:

1. Структура та схема бази даних:
    - Структура бази даних повинна бути ретельно спроектованою для забезпечення ефективного зберігання та доступу до інформації. База даних повинна містити таблиці, які відображають основні сутності вашого інтернет-магазину: виріб, клієнти та замовлення.
    - Таблиця "Виріб": У цій таблиці слід зберігати дані про кожен браслет, включаючи ідентифікатор, назву, опис, ціну, наявність на складі тощо.
    - Таблиця "Клієнти": Тут мають знаходитися дані про клієнтів, включаючи їх ідентифікатори, імена, адреси, контактну інформацію та інше.
2. Розгляд можливостей для оптимізації запитів та забезпечення ефективного доступу до даних:
    - Після того, як база даних спроектована, важливо розглянути можливості оптимізації запитів та забезпечення швидкого доступу до даних. Ось кілька способів цього досягнути:
    - *Індексація*: Використання індексів на полях, які часто використовуються у запитах. Наприклад, поля, за якими зазвичай робляться пошукові запити, як ідентифікатори клієнта чи браслета.
    - *Нормалізація даних*: Розгляд можливостей нормалізації даних, щоб уникнути дублювання інформації та зменшити обсяг зберігання даних. Наприклад, клієнти можуть мати свою власну

таблицю, і їх ідентифікатори використовуються в таблиці "Замовлення".

- *Запити та операції*: Вивчення типових запитів та операцій, які будуть виконуватися в інтернет-магазині (наприклад, вибірка товарів, створення замовлень) та переконання, що вони оптимізовані для швидкості та продуктивності.
- *Кешування*: Розгляд можливості використання кешування для запитів, які виконуються часто. Кешування може значно зменшити навантаження на базу даних та покращити продуктивність.
- *Масштабованість*: Передбачення можливості масштабування бази даних на майбутнє, якщо інтернет-магазин зростатиме. Розгляд розподілену базу даних та інші методи масштабування.

Дизайн бази даних - це важливий етап розробки, який впливає на продуктивність та надійність інтернет-магазину. Важливо ретельно розглянути потреби проекту та врахувати можливості оптимізації для досягнення найкращих результатів.

3 ІНФОРМАЦІЙНЕ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ СИСТЕМИ

3.1 Реалізація клієнтської частини проєкту (Front End)

У даному розділі розглянемо реалізацію клієнтської частини проєкту, яка відповідає за взаємодію користувача з інтерфейсом інтернет-магазину браслетів. Клієнтська частина розроблена з використанням технологій HTML, CSS, та JavaScript у рамках бібліотеки React. Основна структура фронтенд-додатку представлена у вигляді дерева файлів та папок, що допомагає організувати код та розділити його на логічні модулі.

Структура проєкту

Проєкт має наступну структуру:

```
bracelet-store/
│
├── src/
│   ├── components/
│   │   ├── UI/
│   │   │   ├── Header.jsx
│   │   │   ├── MyButton.jsx
│   │   ├── BraceletItem.jsx
│   │   ├── BraceletsCarousel.jsx
│   ├── constants/
│   │   ├── bracelets.js
│   │   ├── routes.js
│   ├── pages/
│   │   ├── AboutPage.jsx
│   │   ├── BraceletInfoPage.jsx
│   │   ├── BraceletsCatalogPage.jsx
│   │   ├── CartPage.jsx
│   │   ├── ContactsPage.jsx
│   │   ├── HomePage.jsx
│   │   ├── LoginPage.jsx
│   │   ├── SigninPage.jsx
│   │   ├── ProfilePage.jsx
│   ├── styles/
│   │   ├── AboutPage.css
│   │   ├── App.css
│   │   ├── AuthPage.css
│   │   ├── BraBraceletCarousel.css
│   │   ├── BraceletInfoPage.css
│   │   ├── BraceletItem.css
│   │   ├── BraceletsCatalogPage.css
│   │   ├── Button.css
│   │   ├── CartPage.css
│   │   ├── ContactsPage.css
│   │   ├── Header.css
│   │   ├── HomePage.css
│   │   ├── index.js
│   │   ├── ProfilePage.css
│   ├── api.js
│   ├── App.js
│   ├── index.css
├── .env

```

Рисунок 3.1 – Структура проєкту

Компоненти

Компоненти інтерфейсу користувача (UI)

У папці `*components/UI*` зберігаються компоненти, які використовуються у різних частинах додатку для забезпечення спільного вигляду та поведінки.

- **Header.jsx:** Компонент, який відповідає за відображення заголовку та навігаційного меню інтернет-магазину.
- **MyButton.jsx:** Універсальний кнопковий компонент, який може бути використаний у різних частинах додатку.
- **BraceletsCarousel.jsx:** Компонент, який відповідає за відображення браслетів у вигляді каруселі.

Інші компоненти

У папці `*components*` також міститься `*BraceletItem.jsx*` — компонент, який відповідає за відображення окремого браслету у каталозі.

Сторінки

Усі основні сторінки додатку зберігаються у папці `*pages*`:

- **AboutPage.jsx:** Сторінка "Про нас", де надається інформація про магазин.
- **BraceletInfoPage.jsx:** Сторінка з детальною інформацією про окремий браслет.
- **BraceletsCatalogPage.jsx:** Сторінка каталогу, де відображаються всі браслети.
- **CartPage.jsx:** Сторінка кошика, де користувач може переглядати та редагувати товари, які він додав до кошика.
- **ContactsPage.jsx:** Сторінка з контактною інформацією магазину.
- **HomePage.jsx:** Головна сторінка інтернет-магазину.
- **LoginPage.jsx:** Сторінка для входу користувачів у систему.
- **SigninPage.jsx:** Сторінка для реєстрації нових користувачів.

Константи

У папці constants зберігаються файли зі константами, які використовуються у додатку:

- **bracelets.js:** Файл, який містить дані про браслети.
- **routes.js:** Файл, який містить визначення маршрутів додатку.

Файли зі стилями

У папці styles зберігаються файли з стилями, що визначають зовнішній вигляд додатку: **AboutPage.css**, **App.css**, **AuthPages.css**, **BraceletCarousel.css**, **BraceletInfoPage.css**, **BraceletItem.css**, **BraceletsCatalogPage.css**, **Button.css**, **ContactsPage.css**, **Header.css**, **HomePage.css**, **index.css**

Основні файли

- **App.js:** Головний компонент додатку, який містить логіку маршрутизації та рендерить відповідні сторінки.
- **index.js:** Вхідна точка додатку, яка підключає кореневий компонент App до HTML-документу.
- **App.css та index.css:** Файли стилів, що визначають зовнішній вигляд додатку.

Використання технологій

React

React є основною бібліотекою для побудови користувацького інтерфейсу. Він дозволяє створювати компоненти, які можна повторно використовувати, що значно спрощує розробку та підтримку додатку.

CSS

Для стилізації компонентів використовується CSS. Файли у папці `*styles*` містять загальні стилі, що застосовуються до всього додатку, забезпечуючи консистентний вигляд.

JavaScript

JavaScript використовується для реалізації логіки додатку та взаємодії між компонентами. Файл `*index.js*` є вхідною точкою додатку, що підключає React-компоненти до HTML-документу.

Таким чином, клієнтська частина проєкту є добре структурованою та організованою системою, що забезпечує зручну роботу користувачів з інтернет-магазином браслетів. Кожен компонент відповідає за окремий функціональний блок, що полегшує розробку, тестування та подальшу підтримку додатку.

Опис програмного продукту

У цьому підрозділі наведено опис сторінок інтернет-магазину браслетів з відповідними скріншотами. Кожна сторінка має свою унікальну функцію та призначення, забезпечуючи зручну навігацію та взаємодію користувача з додатком.

Головна сторінка (HomePage)

Головна сторінка є першим екраном, який бачить користувач після відвідування інтернет-магазину. Вона містить привітальний банер, основні категорії товарів, та посилання на популярні колекції браслетів.

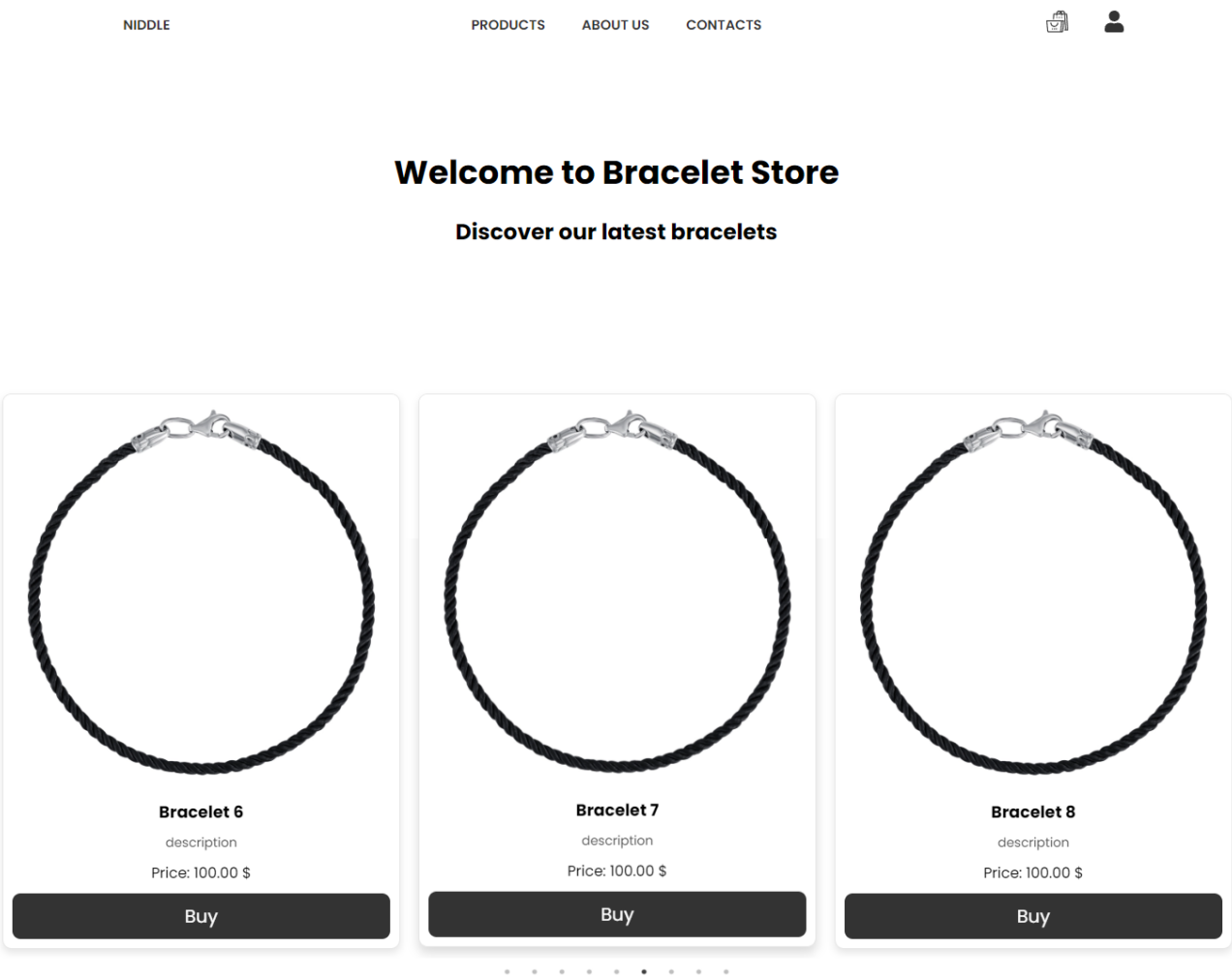

Рисунок 3.2 – Домашня сторінка

Сторінка каталогу браслетів (BraceletsCatalogPage)

На цій сторінці представлений каталог всіх браслетів, доступних у магазині. Користувач може переглядати, фільтрувати та сортувати товари за різними критеріями, такими як ціна, популярність та новинки.

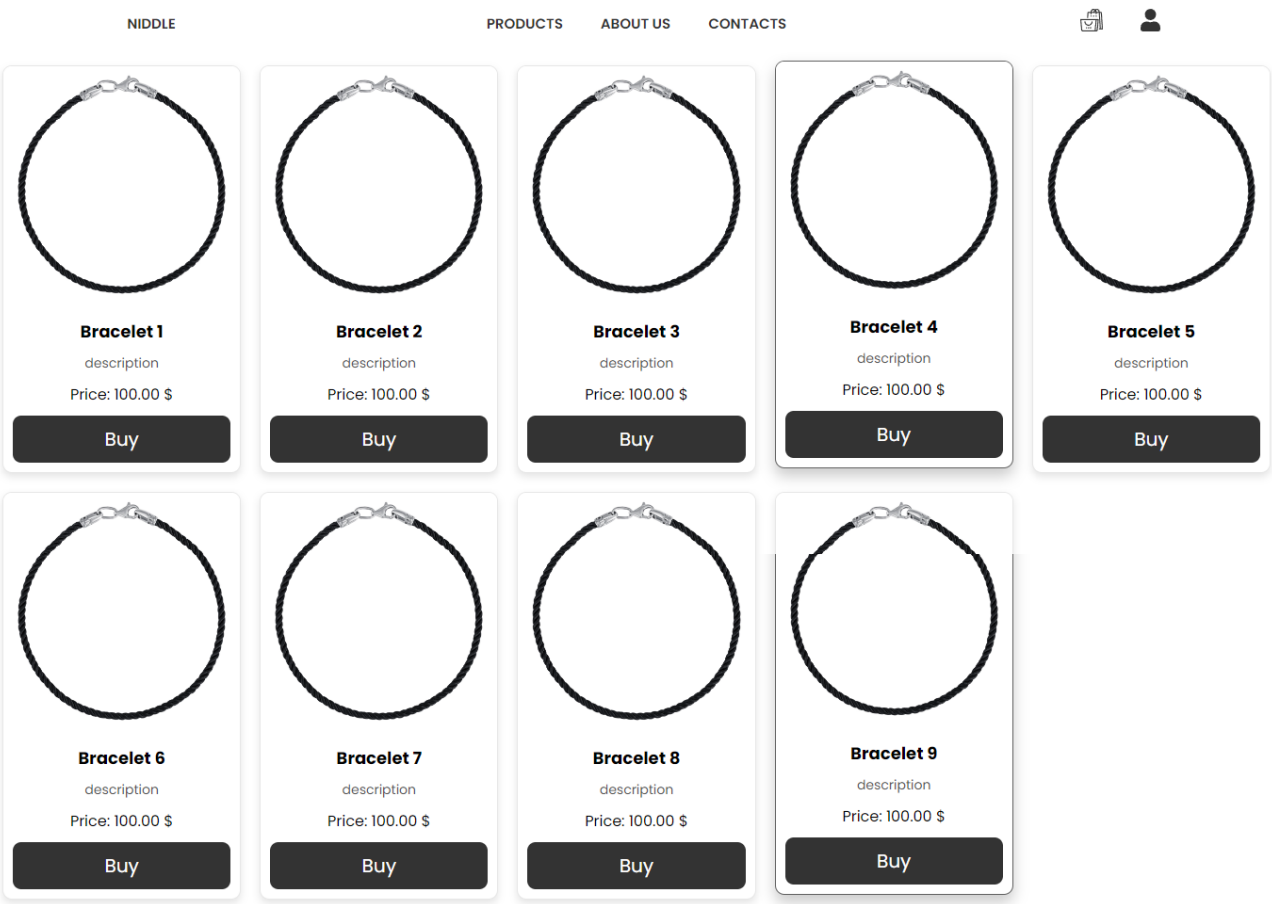

Рисунок 3.3 Сторінка каталогу браслетів

Сторінка інформації про браслет (BraceletInfoPage)

Ця сторінка надає детальну інформацію про обраний браслет, включаючи його опис, матеріали, відгуки користувачів, та можливість додати товар до кошика.

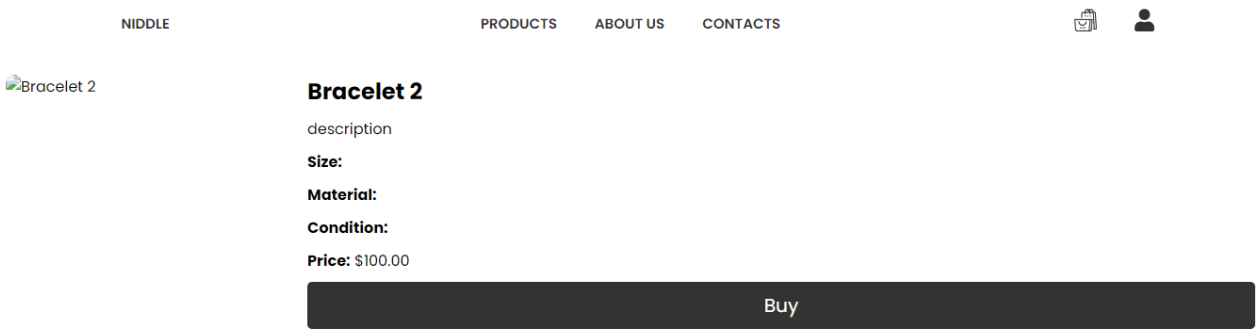

Рисунок 3.4 Сторінка інформації про браслет

Сторінка входу (LoginPage)

Сторінка входу дозволяє зареєстрованим користувачам увійти до системи, використовуючи свої облікові дані (електронна пошта та пароль).

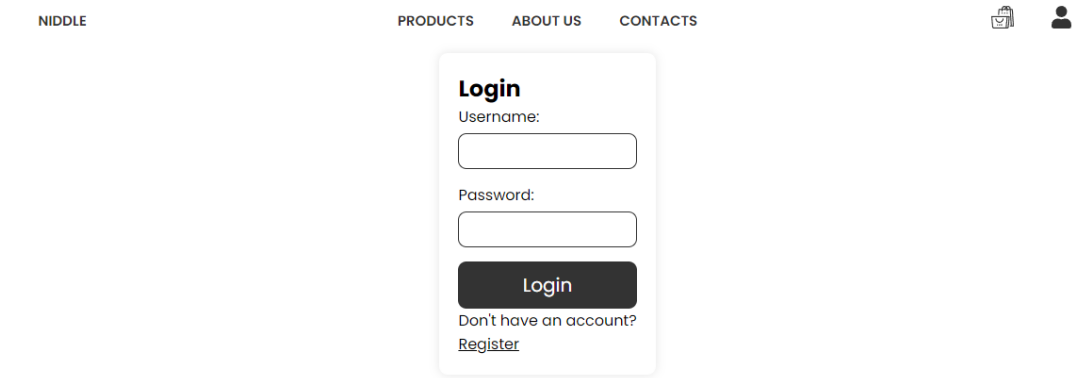

Рисунок 3.5 Сторінка входу

Сторінка реєстрації (SigninPage)

Сторінка реєстрації дозволяє новим користувачам створити обліковий запис, ввівши необхідні дані, такі як ім'я, електронна пошта та пароль.

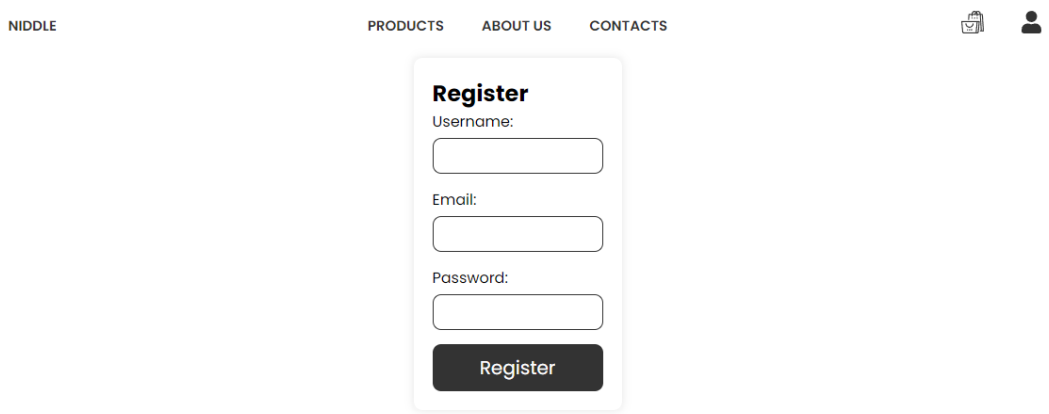

Рисунок 3.6 Сторінка реєстрації

Сторінка профілю користувача (ProfilePage)

Сторінка профілю дозволяє користувачам переглядати та редагувати особисту інформацію, переглядати історію замовлень та змінювати налаштування облікового запису. Це забезпечує персоналізований досвід для кожного користувача.

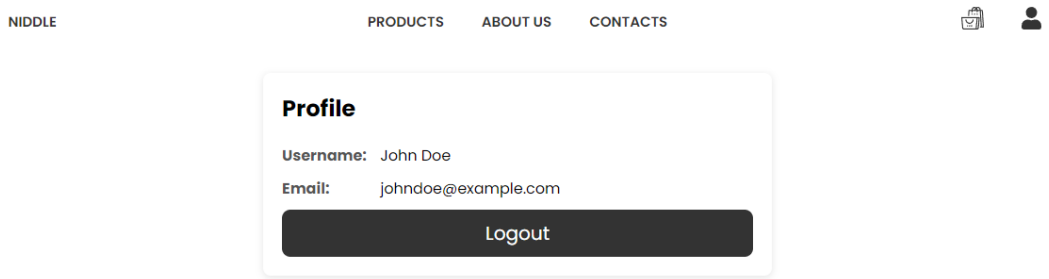

Рисунок 3.7 Сторінка профілю користувача

Сторінка контактів (ContactsPage)

На цій сторінці користувач може знайти контактну інформацію магазину, включаючи адресу, номер телефону, електронну пошту та форму зворотного зв'язку.

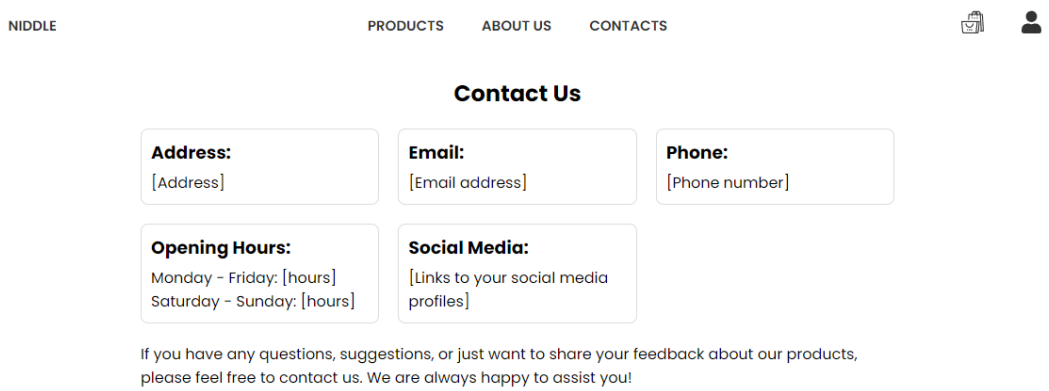

Рисунок 3.8 Сторінка контактів

Сторінка про нас (AboutPage)

Сторінка "Про нас" містить інформацію про історію магазину, його місію, цінності та команду. Це допомагає створити довіру у користувачів та надати їм більше інформації про компанію.

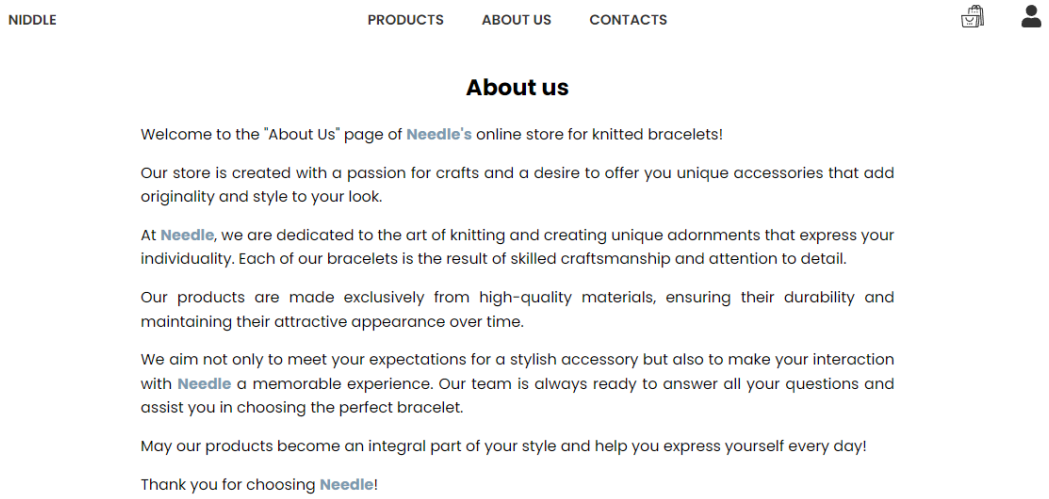

Рисунок 3.9 Сторінка про нас

Сторінка кошика (CartPage)

На сторінці кошика користувач може переглядати товари, які були додані до кошика, змінювати їхню кількість або видаляти непотрібні. Також тут відображається загальна вартість замовлення та можливість перейти до оформлення замовлення.

Таким чином, кожна сторінка інтернет-магазину браслетів виконує свою унікальну функцію, забезпечуючи зручний та інтуїтивний користувацький досвід. Наведені скріншоти і описи допомагають краще зрозуміти структуру та функціональність клієнтської частини проєкту.

3.2 Реалізація серверної частини проєкту (Back End)

Серверна частина (Back End) веб-додатка відповідає за логіку бізнесу, управління базами даних, аутентифікацію користувачів, обробку запитів від клієнтів та взаємодію з іншими сервісами. Реалізація Back End є критичним аспектом будь-якого веб-додатка, оскільки саме тут здійснюється обробка та управління даними, що забезпечує функціональність і продуктивність системи.

Структура Back End

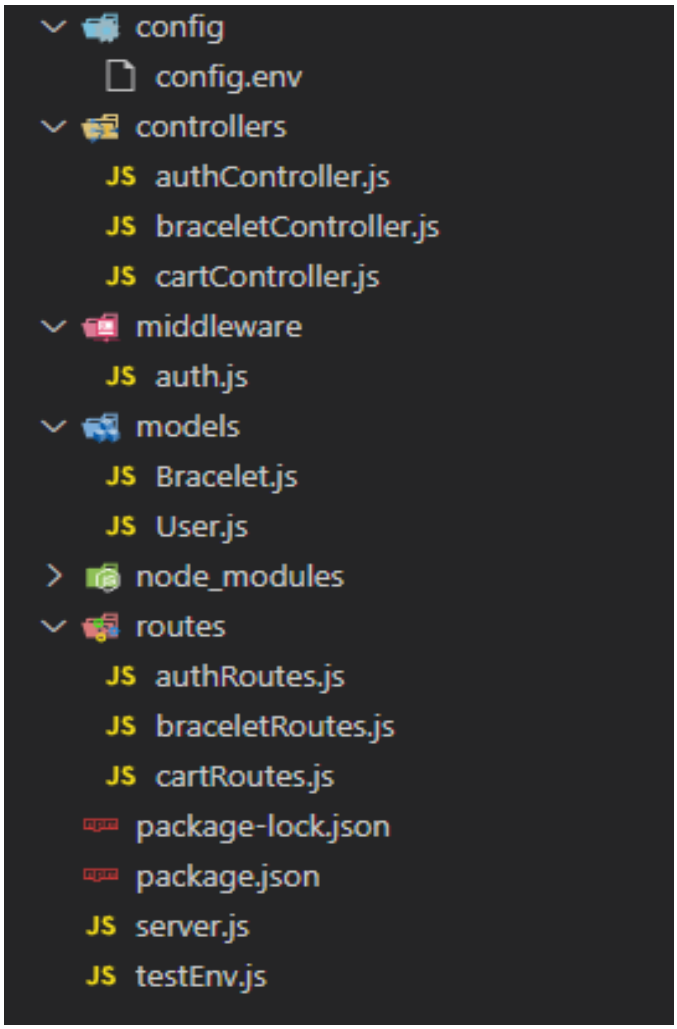

Рисунок 3.10 Структура Back End

Архітектура Back End

Архітектура серверної частини веб-додатка складається з наступних компонентів:

Сервер: Обробляє HTTP-запити, надіслані з фронтенду, та повертає відповідні HTTP-відповіді. У нашому випадку використовуємо Node.js як серверний середовище виконання і Express.js як фреймворк для побудови додатків.

База даних: Зберігає та управляє даними, необхідними для роботи додатка. Використовується MongoDB, яка є NoSQL базою даних, що забезпечує гнучкість у зберіганні даних.

Маршрутизація (Routing): Визначає шляхи для обробки запитів на різні URL-адреси.

- **authRoutes.js** - визначає маршрути для аутентифікації та авторизації. Основні маршрути:
    - **POST /register**: Реєструє нового користувача.
    - **POST /login**: Логін користувача.
    - **GET /profile**: Отримання профілю користувача
- **braceletRoutes.js** - визначає маршрути для роботи з браслетами. Основні маршрути:
    - **GET /**: Повертає список всіх браслетів.
    - **GET /:id**: Повертає браслет за його ID.
    - **POST /**: Створює новий браслет.
    - **PUT /:id**: Оновлює дані браслета.
    - **DELETE /:id**: Видаляє браслет.
- **cartRoutes.js** - визначає маршрути для роботи з кошиком. Основні маршрути:
    - **POST /add**: Додає браслет до кошика.
    - **DELETE /remove**: Видаляє браслет з кошика.

- **GET /**: Повертає всі предмети у кошику користувача.

Моделі (Models): Визначають структуру даних і методи взаємодії з базою даних.

- **Bracelet.js** - визначає структуру даних для браслетів у базі даних та має наступні поля:
  - **image**: Картинка браслета.
  - **name**: Назва браслета.
  - **description**: Опис браслета.
  - **price**: Ціна браслета
- **User.js** - визначає структуру даних для користувачів у базі даних та має наступні поля:
  - **username**: Ім'я користувача.
  - **email**: Електронна пошта користувача.
  - **password**: Хешований пароль користувача.
  - **cart**: Масив об'єктів браслетів, які знаходяться у кошику користувача

Контролери (Controllers): Обробляють бізнес-логіку та виконують взаємодію між моделями і видами (views).

- **authController.js** - контролює аутентифікацію та авторизацію користувачів. Основні функції:
  - **register**: Реєструє нового користувача, хешує пароль перед збереженням.
  - **login**: Перевіряє наявність користувача, порівнює паролі та генерує токен для аутентифікації.
  - **getUserProfile**: Повертає інформацію про профіль користувача на основі токена
- **braceletController.js** - керує функціоналом, пов'язаним з браслетами. Основні функції:
  - **getBracelets**: Повертає список всіх браслетів.
  - **getBraceletById**: Повертає браслет за його ID.

- **createBracelet**: Створює новий браслет.
- **updateBracelet**: Оновлює дані браслета.
- **deleteBracelet**: Видаляє браслет.

- **cartController.js** - керує функціоналом кошика. Основні функції:
  - **addToCart**: Додає браслет до кошика користувача.
  - **removeFromCart**: Видаляє браслет з кошика користувача.
  - **getCart**: Повертає список всіх предметів у кошику користувача.

У проекті структура побудована таким чином, щоб забезпечити чітке розмежування відповідальності між різними компонентами системи. Контролери відповідають за обробку логіки запитів, моделі визначають структуру даних, а маршрути забезпечують доступ до відповідних функцій через API. Це дозволяє легко масштабувати та підтримувати проект, додаючи нові функціональні можливості та оновлюючи існуючі.

ВИСНОВКИ

У даній кваліфікаційній роботі розглянуто розробку інтернет-магазину з продажу виробів ручної роботи. Робота є актуальною через зростання популярності унікальних виробів ручної роботи. Для створення інтернет-магазину обрано стек MERN (MongoDB, Express.js, React, Node.js), що забезпечує високу продуктивність, гнучкість та масштабованість. Використання HTML, CSS, JavaScript у поєднанні з React дозволяє створити зручний та інтуїтивний інтерфейс користувача.

Клієнтська частина проекту реалізована на основі React, що дозволяє створювати багаторазово використовувані компоненти. Серверна частина побудована на Node.js з використанням Express.js для обробки запитів та взаємодії з базою даних MongoDB. Реалізовано основні функції інтернет-магазину: перегляд каталогу, детальна інформація про товари, кошик для покупок, аутентифікація та реєстрація користувачів.

Проведено тестування основних функцій додатку на локальному сервері, що забезпечило його коректну роботу. Подальші перспективи включають розгортання магазину на веб-сервері, підтримку та оновлення з урахуванням фідбеку користувачів, розробку панелі адміністратора.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. What Is The MERN Stack? Introduction & Examples | MongoDB
2. MongoDB
3. MongoDB: The Developer Data Platform | MongoDB
4. React
5. React From the Core to the Interface
6. React
7. Node.js (nodejs.org)
8. Express - Node.js web application framework (expressjs.com)
9. Node.js | Express.js
10. HTML Підручник
11. Довідник по HTML тегам
12. CSS Підручник
13. CSS
14. JavaScript
15. JavaScript
16. Купити вязані сумки гачком - 2023 - в Україні на Crafta.ua
17. Woven & Braided Bracelets - Etsy Ukraine
18. ChatGPT
19. Frontend розробка
20. Back End розробка MERN
21. Full-Stack React Projects 2nd Edition (2020) PDF

ДОДАТОК А

(A1) // back-end server.js

```
const express = require("express");
const mongoose = require("mongoose");  847.1k (gzipped: 228k)
const dotenv = require("dotenv");  6.4k (gzipped: 2.8k)
const bodyParser = require("body-parser");  484.3k (gzipped: 211.6k)
const path = require("path");
const cors = require("cors");  4.5k (gzipped: 1.9k)

// Загрузка переменных окружения из файла config.env
dotenv.config({ path: path.resolve(__dirname, "config", "config.env") })

const app = express();
const PORT = process.env.PORT || 5000;

// Проверка наличия переменной MONGO_URI
if (!process.env.MONGO_URI) {
  console.error("Error: MONGO_URI is not defined");
  process.exit(1);
}

// Middleware
app.use(bodyParser.json());
app.use(cors());

// MongoDB connection
mongoose
  .connect(process.env.MONGO_URI)
  .then(() => console.log("MongoDB connected"))
  .catch((err) => console.log(err));

// Routes
app.use("/api/auth", require("./routes/authRoutes"));
app.use("/api/bracelets", require("./routes/braceletRoutes"));
app.use("/api/cart", require("./routes/cartRoutes"));

// Добавление базового маршрута
app.get("/", (req, res) => {
  res.send("Welcome to the Webstore API");
});

app.listen(PORT, () => console.log(`Server running on port ${PORT}`));

```

(A2) //config.env

```
MONGO_URI=mongodb://localhost:27017/webstore
PORT=5000
JWT_SECRET=BCDBZSGrHvWItphWKnJKWs0cpig1WVhP
```

(A3) //authController.js

```
// controllers/authController.js
const User = require("../models/User");
const bcrypt = require("bcryptjs");  21.6k (gzipped: 9.7k)
const jwt = require("jsonwebtoken");  53k (gzipped: 15.9k)

// Register user
exports.register = async (req, res) => {
  const { name, email, password } = req.body;

  try {
    const user = new User({ name, email, password });
    await user.save();
    res.status(201).json({ message: "User registered successfully" });
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
};

// Login user
exports.login = async (req, res) => {
  const { email, password } = req.body;

  try {
    const user = await User.findOne({ email });
    if (!user) {
      return res.status(400).json({ message: "Invalid credentials" });
    }

    const isMatch = await bcrypt.compare(password, user.password);
    if (!isMatch) {
      return res.status(400).json({ message: "Invalid credentials" });
    }

    const token = jwt.sign({ id: user._id }, process.env.JWT_SECRET, {
      expiresIn: "3h",
    });
    res.json({ token });
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
};

// Get user profile
exports.getUserProfile = async (req, res) => {
  try {
    const user = await User.findById(req.user.id).select("-password");
    if (!user) {
      return res.status(404).json({ message: "User not found" });
    }
    res.status(200).json(user);
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
};

```

(A4) //braceletController.js

```
// controllers/braceletController.js
const Bracelet = require("../models/Bracelet");

// Create a new bracelet
exports.createBracelet = async (req, res) => {
  const { image, name, description, price } = req.body;

  try {
    const bracelet = new Bracelet({ image, name, description, price });
    await bracelet.save();
    res
      .status(201)
      .json({ message: "Bracelet created successfully", bracelet });
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
};

// Get all bracelets
exports.getBracelets = async (req, res) => {
  try {
    const bracelets = await Bracelet.find();
    res.status(200).json(bracelets);
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
};

// Get a single bracelet by ID
exports.getBraceletById = async (req, res) => {
  try {
    const bracelet = await Bracelet.findById(req.params.id);
    if (!bracelet) {
      return res.status(404).json({ message: "Bracelet not found" });
    }
    res.status(200).json(bracelet);
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
};

```

```
41
42  // Update a bracelet by ID
43  exports.updateBracelet = async (req, res) => {
44    const { image, name, description, price } = req.body;
45
46    try {
47      const bracelet = await Bracelet.findByIdAndUpdate(
48        req.params.id,
49        { image, name, description, price },
50        { new: true }
51      );
52      if (!bracelet) {
53        return res.status(404).json({ message: "Bracelet not found" });
54      }
55      res
56        .status(200)
57        .json({ message: "Bracelet updated successfully", bracelet });
58    } catch (error) {
59      res.status(500).json({ error: error.message });
60    }
61  };
62
63  // Delete a bracelet by ID
64  exports.deleteBracelet = async (req, res) => {
65    try {
66      const bracelet = await Bracelet.findByIdAndDelete(req.params.id);
67      if (!bracelet) {
68        return res.status(404).json({ message: "Bracelet not found" });
69      }
70      res.status(200).json({ message: "Bracelet deleted successfully" });
71    } catch (error) {
72      res.status(500).json({ error: error.message });
73    }
74  };
75
```

(A5) //cartController.js

```
const User = require("../models/User");
const Bracelet = require("../models/Bracelet");

// Add bracelet to cart
exports.addToCart = async (req, res) => {
  const { braceletId, quantity } = req.body;

  try {
    const bracelet = await Bracelet.findById(braceletId);
    if (!bracelet) {
      return res.status(404).json({ message: "Bracelet not found" });
    }

    const user = await User.findById(req.user.id);
    const itemIndex = user.cart.findIndex(
      (item) => item.bracelet.toString() === braceletId
    );

    if (itemIndex > -1) {
      // If bracelet already exists in cart, update the quantity
      user.cart[itemIndex].quantity += quantity;
    } else {
      // If bracelet does not exist in cart, add it
      user.cart.push({ bracelet: braceletId, quantity });
    }

    await user.save();
    res.status(200).json(user.cart);
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
};

// Get user's cart
exports.getCart = async (req, res) => {
  try {
    const user = await User.findById(req.user.id).populate("cart.bracelet");
    res.status(200).json(user.cart);
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
};

// Remove bracelet from cart
exports.removeFromCart = async (req, res) => {
  const { braceletId } = req.body;

  try {
    const user = await User.findById(req.user.id);
    user.cart = user.cart.filter(
      (item) => item.bracelet.toString() !== braceletId
    );
    await user.save();
    res.status(200).json(user.cart);
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
};

```

(A6) //auth.js

```
const jwt = require("jsonwebtoken");  53k (gzipped: 15.9k)
const User = require("../models/User");

module.exports = async (req, res, next) => {
  const authHeader = req.header("Authorization");

  if (!authHeader) {
    return res.status(401).json({ message: "No token, authorization denied" });
  }

  const token = authHeader.replace("Bearer ", "");

  if (!token) {
    return res.status(401).json({ message: "No token, authorization denied" });
  }

  try {
    const decoded = jwt.verify(token, process.env.JWT_SECRET);
    req.user = await User.findById(decoded.id).select("-password");
    next();
  } catch (err) {
    res.status(401).json({ message: "Token is not valid" });
  }
};

```

(A7) //Bracelet.js

```
// models/Bracelet.js
const mongoose = require("mongoose");  847.1k (gzipped: 228k)

const BraceletSchema = new mongoose.Schema({
  image: {
    type: String,
    required: true,
  },
  name: {
    type: String,
    required: true,
  },
  description: {
    type: String,
    required: true,
  },
  price: {
    type: Number,
    required: true,
  },
});

module.exports = mongoose.model("Bracelet", BraceletSchema);

```

(A8) //User.js

```
const mongoose = require("mongoose");  847.1k (gzipped: 228k)
const bcrypt = require("bcryptjs");  21.6k (gzipped: 9.7k)

const UserSchema = new mongoose.Schema({
  name: {
    type: String,
    required: true,
  },
  email: {
    type: String,
    required: true,
    unique: true,
  },
  password: {
    type: String,
    required: true,
  },
  cart: [
    {
      bracelet: {
        type: mongoose.Schema.Types.ObjectId,
        ref: "Bracelet",
      },
      quantity: {
        type: Number,
        default: 1,
      },
    },
  ],
});

// Hash password before saving
UserSchema.pre("save", async function (next) {
  if (!this.isModified("password")) {
    return next();
  }
  const salt = await bcrypt.genSalt(10);
  this.password = await bcrypt.hash(this.password, salt);
  next();
});

module.exports = mongoose.model("User", UserSchema);

```

(A9) //authRoutes.js

```
// routes/authRoutes.js
const express = require("express");
const {
  register,
  login,
  getUserProfile,
} = require("../controllers/authController");
const auth = require("../middleware/auth");
const router = express.Router();

router.post("/register", register);
router.post("/login", login);
router.get("/profile", auth, getUserProfile);

module.exports = router;

```

(A10) //braceletRoutes.js

```
// routes/braceletRoutes.js
const express = require("express");
const {
  createBracelet,
  getBracelets,
  getBraceletById,
  updateBracelet,
  deleteBracelet,
} = require("../controllers/braceletController");
const router = express.Router();

router.post("/", createBracelet);
router.get("/", getBracelets);
router.get("/:id", getBraceletById);
router.put("/:id", updateBracelet);
router.delete("/:id", deleteBracelet);

module.exports = router;

```

(A11) //cartRoutes.js

```
const express = require("express");
const {
  addToCart,
  getCart,
  removeFromCart,
} = require("../controllers/cartController");
const auth = require("../middleware/auth");
const router = express.Router();

router.post("/add", auth, addToCart);
router.get("/", auth, getCart);
router.delete("/remove", auth, removeFromCart);

module.exports = router;

```

ДОДАТОК Б

(Б1) front-end // App.js

```jsx
import { BrowserRouter as Router, Route, Routes } from "react-router-dom";
import "./styles/App.css";
import Header from "./components/UI/Header";
import HomePage from "./pages/HomePage";
import AboutPage from "./pages/AboutPage";
import BraceletsCatalogPage from "./pages/BraceletsCatalogPage";
import ContactsPage from "./pages/ContactsPage";
import {
  ABOUT_PAGE,
  CART_PAGE,
  CATALOG_PAGE,
  CONTACTS_PAGE,
  HOME_PAGE,
  LOGIN_PAGE,
  PROFILE_PAGE,
  REGISTRATION_PAGE,
} from "./constants/routs";
import LoginPage from "./pages/LoginPage";
import RegistrationPage from "./pages/RegistrationPage";
import CartPage from "./pages/CartPage";
import BraceletInfoPage from "./pages/BraceletInfoPage";
import ProfilePage from "./pages/ProfilePage";

function App() {
  return (
    <Router>
      <Header />
      <Routes>
        <Route exact path={HOME_PAGE} element={<HomePage />} />
        <Route path={ABOUT_PAGE} element={<AboutPage />} />
        <Route path={CATALOG_PAGE} element={<BraceletsCatalogPage />} />
        <Route path={CONTACTS_PAGE} element={<ContactsPage />} />
        <Route path={LOGIN_PAGE} element={<LoginPage />} />
        <Route path={REGISTRATION_PAGE} element={<RegistrationPage />} />
        <Route path={CART_PAGE} element={<CartPage />} />
        <Route path="/bracelet/:id" element={<BraceletInfoPage />} />
        <Route path={PROFILE_PAGE} element={<ProfilePage />} />
      </Routes>
    </Router>
  );
}

export default App;

```

(Б2) front-end // api.js

```
import axios from "axios";  61.7k (gzipped: 22.7k)

const API_URL = process.env.REACT_APP_API_URL;

// Регистрация пользователя
export const registerUser = async (userData) => {
  try {
    const response = await axios.post(`${API_URL}/auth/register`, userData);
    return response.data;
  } catch (error) {
    throw error.response.data;
  }
};

// Логин пользователя
export const loginUser = async (userData) => {
  try {
    const response = await axios.post(`${API_URL}/auth/login`, userData);
    return response.data;
  } catch (error) {
    throw error.response.data;
  }
};

// Получение данных пользователя
export const getUserProfile = async (token) => {
  try {
    const response = await axios.get(`${API_URL}/auth/profile`, {
      headers: {
        Authorization: `Bearer ${token}`,
      },
    });
    return response.data;
  } catch (error) {
    throw error.response.data;
  }
};

```

```
// Добавление браслета в корзину
export const addToCart = async (token, braceletId, quantity) => {
  try {
    const response = await axios.post(
      `${API_URL}/cart/add`,
      { braceletId, quantity },
      {
        headers: {
          Authorization: `Bearer ${token}`,
        },
      }
    );
    return response.data;
  } catch (error) {
    throw error.response.data;
  }
};

// Получение корзины пользователя
export const getCart = async (token) => {
  try {
    const response = await axios.get(`${API_URL}/cart`, {
      headers: {
        Authorization: `Bearer ${token}`,
      },
    });
    return response.data;
  } catch (error) {
    throw error.response.data;
  }
};

// Удаление браслета из корзины
export const removeFromCart = async (token, braceletId) => {
  try {
    const response = await axios.delete(`${API_URL}/cart/remove`, {
      headers: {
        Authorization: `Bearer ${token}`,
      },
      data: { braceletId },
    });
    return response.data;
  } catch (error) {
    throw error.response.data;
  }
};

// Разлогинивание пользователя
export const logoutUser = () => {
  localStorage.removeItem("token");
};

```

(Б3) front-end // Home.jsx

```jsx
import React from "react";  6.9k (gzipped: 2.7k)
import BraceletsCarousel from "../components/BraceletsCarousel";
import "../styles/HomePage.css";

function HomePage() {
  return (
    <div className="home-page">
      <div className="hero-section">
        <h1>Welcome to Bracelet Store</h1>
        <h2>Discover our latest bracelets</h2>
      </div>
      <div className="bracelet-carousel-container">
        <BraceletsCarousel />
      </div>
    </div>
  );
}
      You, 4 недели назад • Initial commit
export default HomePage;

```

(Б4) front-end // AboutPage.jsx

```jsx
import React from "react";  6.9k (gzipped: 2.7k)
import "../styles/AboutPage.css";

function AboutPage() {
  return (
    <div className="about">
      <h1>About us</h1>
      <p>
        Welcome to the "About Us" page of{" "}
        <span className="store-name">Needle's</span> online store for knitted
        bracelets!
      </p>
      <p>
        Our store is created with a passion for crafts and a desire to offer you
        unique accessories that add originality and style to your look.
      </p>
      <p>
        At <span className="store-name">Needle</span>, we are dedicated to the
        art of knitting and creating unique adornments that express your
        individuality. Each of our bracelets is the result of skilled
        craftsmanship and attention to detail.
      </p>
      <p>
        Our products are made exclusively from high-quality materials, ensuring
        their durability and maintaining their attractive appearance over time.
      </p>
      <p>
        We aim not only to meet your expectations for a stylish accessory but
        also to make your interaction with{" "}
        <span className="store-name">Needle</span> a memorable experience. Our
        team is always ready to answer all your questions and assist you in
        choosing the perfect bracelet.
      </p>
      <p>
        May our products become an integral part of your style and help you
        express yourself every day!
      </p>
      <p>
        Thank you for choosing <span className="store-name">Needle</span>!
      </p>
    </div>
  );
}

export default AboutPage;

```

(Б5) front-end // BraceletsCatalogPage.jsx

```jsx
import React, { useEffect, useState } from "react";  6.9k (gzipped: 2.7k)
// import { Link } from "react-router-dom";
import axios from "axios";  61.7k (gzipped: 22.7k)
import BraceletItem from "../components/BraceletItem";
// import { bracelets } from "../constants/bracelets";
import "../styles/BraceletItem.css";

function BraceletsCatalogPage() {
  const [braceletsList, setBraceletsList] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  console.log("API URL:", process.env.REACT_APP_API_URL);

  useEffect(() => {
    const fetchBracelets = async () => {
      try {
        const response = await axios.get(
          `${process.env.REACT_APP_API_URL}/bracelets`
        );
        setBraceletsList(response.data);
      } catch (error) {
        console.log("Error fetching bracelets:", error);
        setError(error.message);
      } finally {
        setLoading(false);
      }
    };

    fetchBracelets();
  }, []);

  return (
    <div className="bracelets-catalog">
      {loading ? (
        <p>Loading...</p>
      ) : error ? (
        <p>{error}</p>       You, вчера • Uncommitted changes
      ) : braceletsList.length > 0 ? (
        braceletsList.map((bracelet) => (
          <BraceletItem
            key={bracelet._id}
            bracelet={bracelet}
            context="catalog"
          />
        ))
      ) : (
        <p>No bracelets aveilable</p>
      )}
    </div>
  );
}

export default BraceletsCatalogPage;

```

(Б6) front-end // ContactsPage.jsx

```jsx
import React from "react";  6.9k (gzipped: 2.7k)
import "../styles/ContactsPage.css";      You, на прошлой неделе • add new fe

function ContactsPage() {
  return (
    <div className="contacts">
      <h1>Contact Us</h1>
      <div className="contact-info">
        <div className="info-item">
          <h3>Address:</h3>
          <p>[Address]</p>
        </div>
        <div className="info-item">
          <h3>Email:</h3>
          <p>[Email address]</p>
        </div>
        <div className="info-item">
          <h3>Phone:</h3>
          <p>[Phone number]</p>
        </div>
        <div className="info-item">
          <h3>Opening Hours:</h3>
          <p>Monday - Friday: [hours]</p>
          <p>Saturday - Sunday: [hours]</p>
        </div>
        <div className="info-item">
          <h3>Social Media:</h3>
          <p>[Links to your social media profiles]</p>
        </div>
      </div>
      <p className="contact-message">
        If you have any questions, suggestions, or just want to share your
        feedback about our products, please feel free to contact us. We are
        always happy to assist you!
      </p>
    </div>
  );
}

export default ContactsPage;

```

(Б7) front-end // LoginPage.jsx (Фрагмент)

```jsx
function LoginPage() {
  const [email, setEmail] = useState("");
  const [password, setPassword] = useState("");
  const [error, setError] = useState(null);    'error' is assign
  const navigate = useNavigate();

  const handleLogin = async (e) => {
    e.preventDefault();
    try {
      const response = await loginUser({ email, password });
      console.log("User logged in successfully:", response);
      // Store the token and redirect to another page
      localStorage.setItem("token", response.token);
      // Redirect to the home page or other actions
      navigate(HOME_PAGE);
    } catch (err) {
      setError(err.message);
    }
  };

```

(Б8) front-end // RegistrationPage.jsx (Фрагмент)

```jsx
import React, { useState } from "react";  6.9k (gzipped: 2.7k)
import { useNavigate } from "react-router-dom";  4.8k (gzipped: 2k)
import "../styles/AuthPages.css";
import { registerUser } from "../api";
import { LOGIN_PAGE } from "../constants/routs";

function RegistrationPage() {
  const [username, setUsername] = useState("");
  const [email, setEmail] = useState("");
  const [password, setPassword] = useState("");
  const [error, setError] = useState(null);    'error' is assigned a value
  const navigate = useNavigate();

  const handleRegister = async (e) => {
    e.preventDefault();
    try {
      const response = await registerUser({ username, email, password });
      console.log("User registered successfully:", response);
      // Redirect to login page or other actions
      navigate(LOGIN_PAGE);
    } catch (err) {
      setError(err.message);
    }
  };

```

(Б9) front-end // ProfilePage.jsx (Фрагмент)

```
7
8   const ProfilePage = () => {
9     const [user, setUser] = useState(null);
10    const [loading, setLoading] = useState(true);
11    const [error, setError] = useState(null);
12    const token = localStorage.getItem("token");
13    const navigate = useNavigate();
14
15    const handleLogout = () => {
16      localStorage.removeItem("token");
17      navigate(HOME_PAGE);
18    };
19
20    useEffect(() => {
21      const fetchUserProfile = async () => {
22        try {
23          const userProfile = await getUserProfile(token);
24          setUser(userProfile);
25        } catch (err) {
26          setError(err.message);
27        } finally {
28          setLoading(false);
29        }
30      };
31
32      fetchUserProfile();
33    }, [token]);
34
35    if (loading) {
36      return <p>Loading...</p>;
37    }
38
39    if (error) {
40      return <p>Error: {error}</p>;
41    }
42
```

(Б10) front-end // Header.jsx

```jsx
function Header() {
  const token = localStorage.getItem("token");
  const isLoggedIn = !!token;
  const navigate = useNavigate();

  const handleLogout = () => {    'handleLogout' is assigned a value but never used.
    localStorage.removeItem("token");
    navigate(HOME_PAGE);
  };

  return (
    <header>
      <div className="item left">
        <Link to={HOME_PAGE} className="logo">
          Niddle
        </Link>
      </div>
      <div className="item center">
        <ul>
          <li>
            <Link to={CATALOG_PAGE}>Products</Link>
          </li>
          <li>
            <Link to={ABOUT_PAGE}>About us</Link>
          </li>
          <li>
            <Link to={CONTACTS_PAGE}>Contacts</Link>
          </li>
        </ul>
      </div>
      <div className="item right">
        <ul>
          <li>
            <Link to={CART_PAGE}>
              <CartIcon />
            </Link>
          </li>
          <li>
            {isLoggedIn ? (
              <Link to={PROFILE_PAGE}>
                <ProfileIcon />
              </Link>      You, вчера • Uncommitted changes
            ) : (
              <Link to={LOGIN_PAGE}>
                <ProfileIcon />
              </Link>
            )}
          </li>
        </ul>
      </div>
    </header>
  );
}

export default Header;
```