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 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА
на здобуття освітнього ступеня бакалавр

зі спеціальності 122 - Комп'ютерних наук,
освітньо-професійної програми «Інформатика»
на тему: «Інтернет магазин з продажу ляльок ручної роботи»
здобувача групи ІН-92 Рудик Олексія Олександровича

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

_____ Олексій РУДИК

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

Суми – 2023

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-92 Рудик Олексія Олександровича

1. Тема роботи: «Інтернет магазин з продажу ляльок ручної роботи»
затверджую наказом по СумДУ від *«01» червня 2023 р. № 0475-VI*

2. Термін здачі здобувачем кваліфікаційної роботи *до 09 червня 2023 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 для продажу ляльок 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*	21.04 - 29.05	

	роботи		
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: 58 стр., 38 рис., 2 додатка, 10 використаних джерел.

Обґрунтування актуальності теми роботи – Тема кваліфікаційної роботи є актуальною, оскільки вона поєднує в собі підтримку місцевого ремесла, унікальність продукції, зростаючий попит на ручні вироби та можливість міжнародного розширення бізнесу.

Об’єкт дослідження — процес продажу ляльок ручної роботи.

Мета роботи — розробка інтернет магазину для продажу ляльок 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 ляльок та придбати їх. Кожна лялька в веб додатку має свій реальний прототип.

HANDMADE, WEB APPLICATION, MERN, REACT, EXPRESS, NODE, MONGODB

ЗМІСТ

ВСТУП

Актуальність. Тема кваліфікаційної роботи є актуальною, оскільки вона поєднує в собі підтримку місцевого ремесла, унікальність продукції, зростаючий попит на ручні вироби та можливість міжнародного розширення бізнесу.

Об’єкт дослідження. Процес продажу ляльок ручної роботи.

Предмет дослідження. Методологія створення додатку та керування ним для продажу ляльок.

Гіпотеза. Створення додатку та керування ним можливе за допомогою особливого стеку технологій, що підходить саме для подібного роду завдань.

Новизна. Створення такого додатку створить можливість майстрам, які виготовляють свої вироби, реалізовувати себе, отримати визнання, та продавати свої продукти.

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

Останнім часом зростає зацікавленість людей до ручної роботи та унікальних виробів. Люди все більше цінують неповторність, індивідуальність та естетичну цінність ляльок, створених вручну. Цей тренд призводить до зростання попиту на інтернет-магазини, які спеціалізуються на таких виробах. Завдяки інтернет-магазинам, майстри ручної роботи мають можливість залучити нових клієнтів з різних країн та розширити свою аудиторію. Завдяки можливості міжнародної доставки, ляльки ручної роботи стають доступними для покупців з усього світу, що відкриває нові перспективи для розвитку бізнесу.

З'явилися спеціальні спільноти, форуми, де люди, які цікавляться ручною роботою та ляльками, можуть обмінюватися досвідом, ідеями та замовленнями. Це створює сприятливе середовище для просування інтернет-магазинів та збільшення свідомості про їх продукцію.

Розвиток технологій сприяє покращенню користувацького досвіду та функціональності інтернет-магазинів. Нові функції, такі як фільтри для пошуку, персоналізація рекомендацій, спрощений процес замовлення та оплати, додаткові можливості для відгуків та інтерактивності, дозволяють забезпечити зручну та швидку покупку для клієнтів.

Ці чинники свідчать про перспективи інтернет-магазинів з продажу ляльок ручної роботи. Вони стають важливим елементом в екосистемі ремісничого мистецтва та сприяють розвитку культури ручної роботи в онлайн-середовищі.

1.2 Аналіз аналогічних проєктів

Інтернет-магазин, спеціалізований на продажу ручної роботи ляльок, є онлайн-платформою, де люди можуть придбати унікальні вироби мистецтва. Цей магазин надає можливість творцям, презентувати та продавати свої ляльки по всьому світу.

В такому інтернет-магазині можна знайти широкий вибір ляльок різних

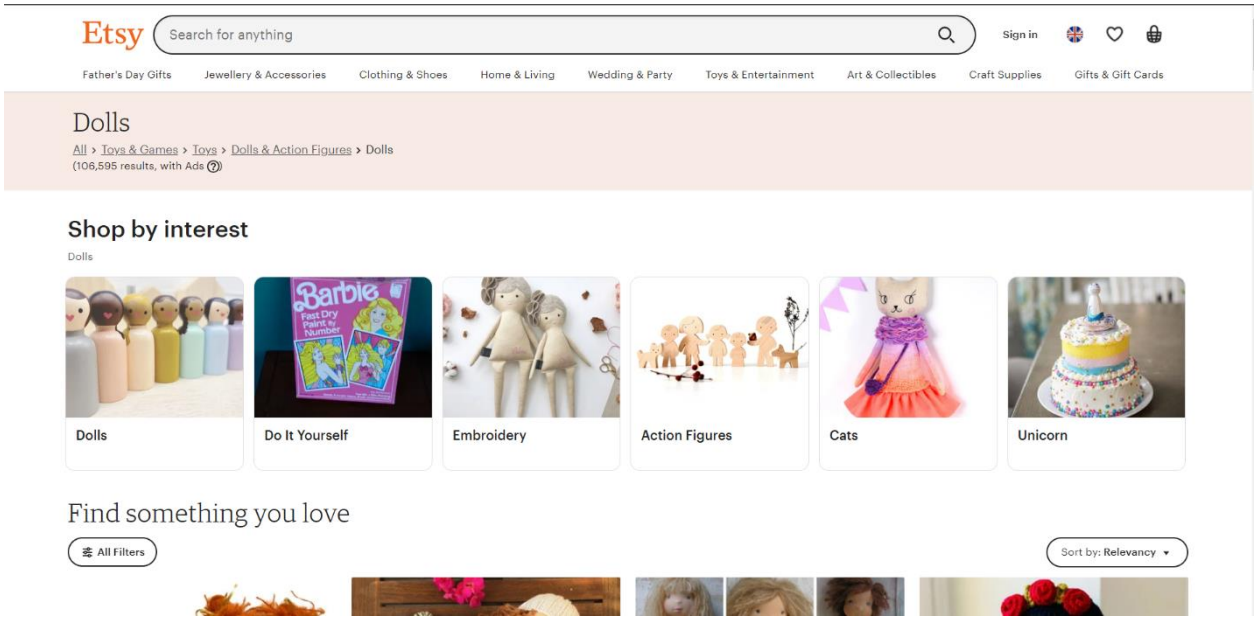
стилів, матеріалів та дизайнів. Покупці мають можливість переглядати фотографії, описи та ціни на ляльки, обирати ті, що їм подобаються, та здійснювати покупки зручним для них способом.

В інтернет-магазинах зазвичай доступні різні функціональні можливості, такі як пошук за категоріями, фільтри, можливість додавання товарів до кошика, онлайн-оплата та доставка. Деякі магазини також дозволяють користувачам залишати відгуки та рейтинги для ляльок, щоб інші покупці могли обрати собі товар, спираючись на той чи інший коментар.

Основна мета інтернет-магазину з продажу ручної роботи ляльок полягає в тому, щоб забезпечити зручний та доступний спосіб для покупців отримати унікальні ляльки, а також допомогти майстрам ручної роботи презентувати та продавати свої вироби широкому колу клієнтів.

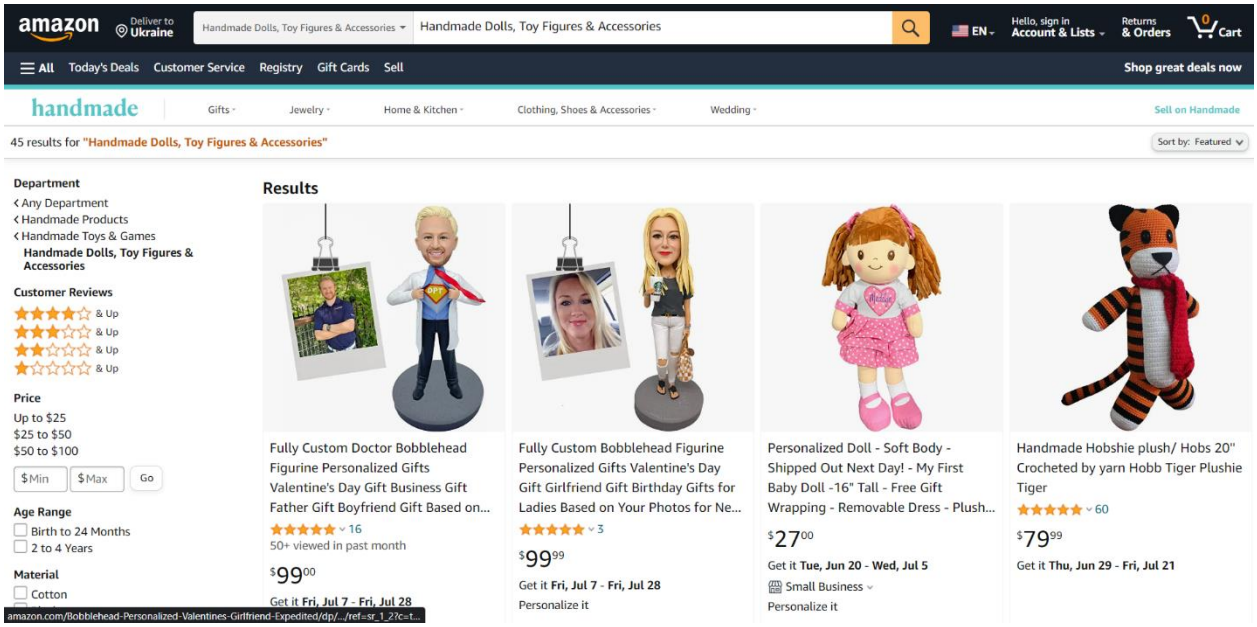
Декілька прикладів подібних інтернет-магазинів:

“Etsy”- Etsy є однією з найвідоміших платформ для продажу ручної роботи. Тут можна знайти велику кількість ляльок ручної роботи, від класичних до унікальних творів мистецтва. Продавці на Etsy мають власні магазини, де представляють свої вироби.



“Handmade at Amazon” - це спеціальна категорія на Amazon, де продавці продають ручну роботу, включаючи ляльки. Платформа пропонує широкий вибір виробів ручної роботи, а також дозволяє фільтрувати за категоріями та

регіонами.



1.3 Постановка задачі

Мета роботи: Розробити інтернет-магазин для продажу унікальних ляльок ручної роботи з метою привернення клієнтів та підтримки місцевих майстрів ручної роботи.

Задачі:

1. Аналіз ринку:
    - Дослідити ринок продажу ляльок ручної роботи та визначити конкурентну ситуацію.
2. Визначення функціональних вимог до інтернет-магазину:
    - Визначити основні функції та можливості інтернет-магазину, необхідні для зручного та ефективного процесу покупок.
    - Врахувати функціональні вимоги до системи оплати та обробки замовлень.
3. Проектування інтерфейсу та дизайну:
    - Розробити зручний та привабливий дизайн інтернет-магазину, який відповідає стилістиці та естетиці ляльок ручної роботи.
    - Врахувати принципи UX-дизайну для покращення користувацького досвіду.

4. Розробка функціоналу та інтеграція:
    - Реалізувати необхідний функціонал, такий як каталог продукції, кошик покупок, система обліку замовлень та оцінок, пошукова система тощо.
    - Інтегрувати систему оплати.
5. Тестування та вдосконалення:
    - Провести тестування функціоналу та коректність роботи інтернет-магазину.
    - Вносити необхідні корективи та вдосконалення на основі отриманих результатів тестування.
6. Розгортання та підтримка:
    - Розгорнути інтернет-магазин на відповідному хостингу або сервері.
    - Забезпечити підтримку та технічне обслуговування інтернет-магазину для безперебійної роботи.
7. Оцінка результатів:
    - Проаналізувати результати та внести корективи для подальшого вдосконалення магазину.

Ці задачі допоможуть розробити та впровадити інтернет-магазин з продажу ляльок ручної роботи з метою успішного функціонування, залучення клієнтів та підтримки майстрів ручної роботи.

2 ВИБІР МЕТОДУ РОЗВ'ЯЗАННЯ ЗАДАЧІ

2.1 Архітектура веб-додатку з використанням стеку MERN

Архітектура веб-додатків з використанням стеку MERN відноситься до моделі розробки, що включає чотири основних компонента: MongoDB, Express.js, React та Node.js. Кожен з цих компонентів виконує свої функції і спільно допомагає побудувати повноцінний веб-додаток.

Далі наведено опис кожного компонента стеку MERN [10]:

MongoDB: MongoDB є документ-орієнтованою, NoSQL базою даних, яка зберігає дані у вигляді JSON-подібних документів. MongoDB дозволяє гнучко зберігати та організовувати дані, а також працювати з ними швидко та ефективно [4].

Додатки, які надають високий пріоритет доступності та масштабованості, отримують вигоду від розподіленої архітектури MongoDB. Вона має вбудовану підтримку високої доступності, горизонтального масштабування за допомогою шардування та масштабованості на рівні декількох центрів обробки даних з різними географічними розподілами [1].

Express.js: Express - це простий серверний веб-фреймворк для створення веб-додатків за допомогою Node. Він доповнює Node шаром елементарних функцій веб-додатків які надають утиліти HTTP і функціональність проміжного програмного забезпечення [6].

У будь-якому веб-додатку, розробленому за допомогою Node, Express можна використовувати як веб-фреймворк для маршрутизації API та проміжного програмного забезпечення. У ланцюжок обробки запитів можна вставити майже будь-яке сумісне проміжне програмне забезпечення на ваш вибір майже в будь-якому порядку, що робить Express дуже гнучким для роботи [1].

React: React є JavaScript бібліотекою для побудови користувацького інтерфейсу. Він дозволяє створювати інтерактивні та динамічні компоненти інтерфейсу, які автоматично оновлюються при зміні даних. React

використовує віртуальний DOM (Virtual DOM) для ефективного відображення змін та реагує на дії користувача швидко та з плавністю. Розробка користувацьких інтерфейсів за допомогою React також змушує фронтенд-програмістів писати обґрунтований модульний код, який є пере використанню і полегшує процес налагодження, тестування та розширення [1].

Node: Node був розроблений як середовище виконання JavaScript, побудоване на JavaScript-движку V8 від Chrome. За допомогою Node з'явилася можливість використовувати JavaScript на стороні сервера для створення різноманітних інструментів і додатків [7].

Node має архітектуру, керовану подіями, здатну до асинхронного, неблокуючого вводу/виводу (скорочено від Input/Output). Унікальна модель неблокуючого вводу/виводу усуває підхід очікування при обслуговуванні запитів. Це дозволяє створювати масштабовані та легкі веб-додатки в режимі реального часу, які можуть ефективно обробляти велику кількість запитів [1].

Загалом, стек MERN забезпечує повний набір інструментів для розробки веб-додатків, починаючи зі зберігання даних в MongoDB, розробки серверної логіки з використанням Express.js і Node.js, а також побудови користувацького інтерфейсу за допомогою React. Цей стек добре поєднується, оскільки всі компоненти використовують JavaScript, що дозволяє розробникам працювати з однією мовою на всіх рівнях додатку [10].

2.2 Здійснення запитів у веб-додатку

У розробці веб-додатків з використанням стеку MERN (MongoDB, Express.js, React, Node.js), здійснення запитів відбувається на різних рівнях компонентів. Ось огляд того, як здійснюються запити на кожному рівні:

Клієнтський рівень (React):

Запити до сервера здійснюються за допомогою функцій або методів, таких як fetch(), axios, fetch API або XMLHttpRequest.

Можна виконувати HTTP-запити, такі як GET, POST, PUT, DELETE і PATCH, для отримання або відправки даних до сервера [9].

Запити можуть бути виконані з використанням асинхронних або промісів для отримання результатів запиту та їх обробки [5].

Серверний рівень (Node.js та Express.js):

На сервері використовується Express.js для створення API та обробки запитів від клієнта.

Визначаються маршрути, які відповідають різним URL та типам запитів.

Запити обробляються за допомогою функцій-обробників, які відправляють відповіді клієнту та здійснюють дії з базою даних (MongoDB).

База даних (MongoDB):

Запити до бази даних MongoDB виконуються за допомогою MongoDB Query Language (MQL).

Вони можуть включати команди для створення, оновлення, видалення або отримання даних з колекцій MongoDB.

Запити можуть бути складнішими, використовуючи фільтри, умови, сортування та інші операції.

Загалом, здійснення запитів у стеку MERN вимагає використання відповідних бібліотек та функцій на кожному рівні компонентів. Запити можуть передавати дані між клієнтом, сервером та базою даних для взаємодії з додатком та отримання необхідної інформації [4].

2.3 State Management у веб-додатку

У веб-додатках з використанням стеку MERN (MongoDB, Express.js, React, Node.js), існує кілька підходів до керування станом (state management). Основними типами state management в таких додатках є:

Локальний state management: Цей підхід використовується для керування станом на рівні компонентів React. Компоненти зберігають свій власний стан, ізольований від інших компонентів. Це може бути досягнуто за допомогою локального стану (state) та функцій setState або за допомогою React

Hooks, таких як useState, useReducer і т.д.

Контекстний state management: Контекст (Context) в React дозволяє передавати дані глибоко вниз по дереву компонентів безпосередньо, без необхідності передавати їх через властивості (props) багатьом компонентам. Використовуючи контекст, можна спільно використовувати стан між багатьма компонентами, що спрощує керування станом на рівні додатку.

Глобальний state management: Цей підхід використовує сторонні бібліотеки для керування глобальним станом додатку. Найпопулярнішою бібліотекою для глобального state management в React є Redux. В Redux стан зберігається в одному місці, називається "store", і кожен компонент може звертатись до нього для отримання або оновлення стану [8].

Зовнішній state management: У додатках MERN можуть використовуватися зовнішні сервіси або API для керування станом. Наприклад, можна використовувати сервіси, такі як GraphQL або Apollo Client для керування станом та взаємодії з сервером та базою даних.

2.4 Архітектурний шаблон (MVC)

Більшість веб-фреймворків для розробки веб-додатків використовують архітектурний патерн "Модель-Вид-Контролер" (MVC), який дозволяє відокремити логіку додатку від його даних та представлення. Протягом довгого часу цей патерн є популярним вибором серед розробників для проектування веб-додатків [2].

Основні компоненти шаблону MVC:

Модель (Model): Модель відповідає за роботу з даними в додатку. Зазвичай це включає операції створення, читання, оновлення та видалення (CRUD) даних в базі даних [2].

Вид (View): Вид відповідає за відображення даних користувачу та взаємодію з ним. Він представляє інтерфейс користувача і може бути представленим у вигляді веб-сторінок, інтерфейсів користувача, графічних елементів тощо. Вид відображає дані з моделі користувачу і дозволяє

користувачу взаємодіяти з додатком, наприклад, вводити дані або натискати кнопки [2].

Контролер (Controller): Контролер є посередниковим класом між класами моделі та представлення. Він контролює потік інформації, приймаючи вхідні дані від користувача з представлення та вказуючи як моделі, так і представленню виконувати дії на основі цієї інформації [2].

Одним із головних принципів шаблону MVC є розділення відповідальностей між компонентами, що сприяє збереженню коду чистим та легко зрозумілим. Він також полегшує роботу над додатком у команді, дозволяючи розробникам працювати над окремими компонентами незалежно один від одного. [2]

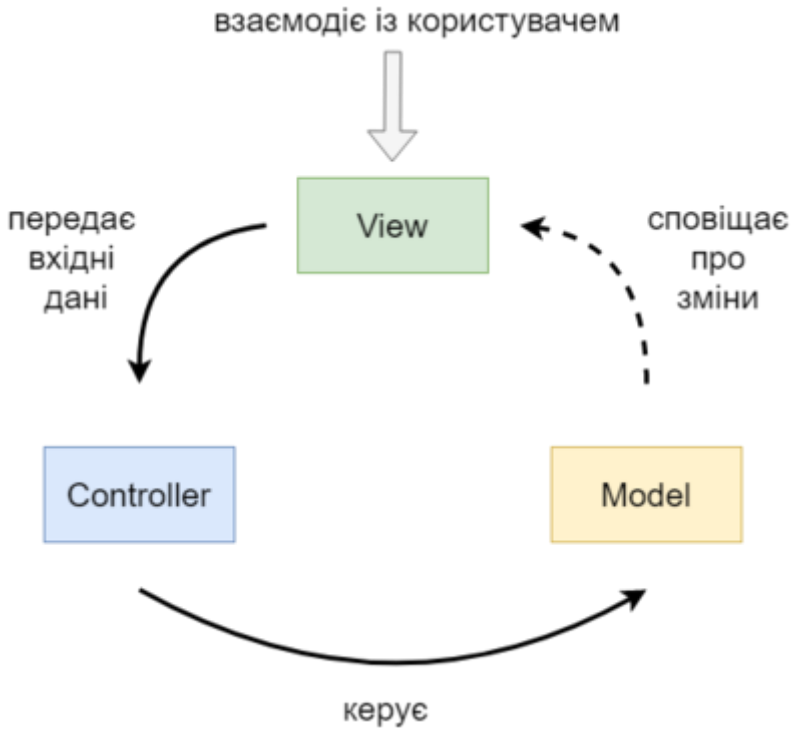


Рисунок 2.1 - Графічне представлення MVC

Було створено архітектуру веб-додатку з використанням MVC

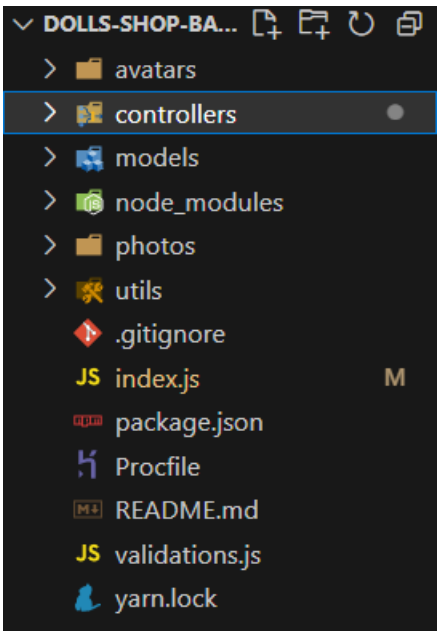


Рисунок 2.2 - Архітектура серверної частини додатка за використанням MVC(Model, Controller)

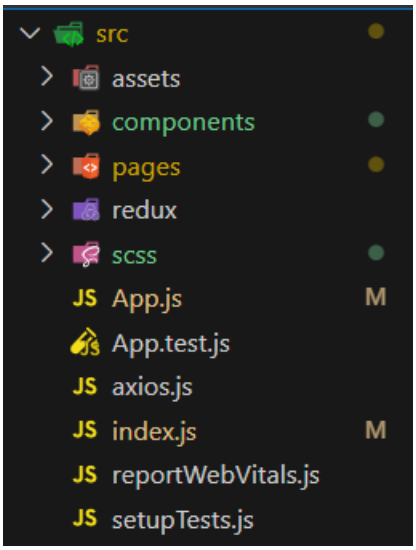


Рисунок 2.3 - Архітектура клієнтської частини додатка за використанням MVC(View)

2.5 State management та обробка й виконання запитів

В моєму додатку я використав глобальний State Management під назвою Redux.

Redux - це бібліотека глобального state management для JavaScript-додатків, зокрема для веб-додатків на базі React.

Redux базується на концепції однієї джерело правди (Single Source of

Truth). Весь стан додатка зберігається в одному об'єкті, відомому як "store" [3].

Для зміни стану в Redux використовуються об'єкти, називані "actions". Actions описують, що відбулося у додатку та передають необхідні дані для оновлення стану [3].

Зміни стану в Redux виконуються за допомогою "reducers". Reducers - це чисті функції, які приймають поточний стан та action і повертають новий стан.

Redux працює за допомогою принципу незмінності (immutability). Це означає, що стан недоступний для безпосередньої зміни, із замість цього створюється новий стан на основі поточного стану та дій [8].

Redux може використовуватись разом з React, але він також може бути використаний з будь-яким іншим JavaScript-фреймворком або бібліотекою.

Для роботи з Redux в React-додатках зазвичай використовуються додаткові бібліотеки, такі як react-redux, які надають інтеграцію між Redux і React.

Redux дозволяє створювати розширені функціональності за допомогою middleware. Middleware може бути використаний для логування, асинхронних операцій, маршрутизації тощо. [3]

Redux також підтримує інструменти розробника, такі як Redux DevTools, які допомагають відстежувати стан додатка, відлагоджувати та аналізувати дії. [3]

Далі наведено архітектуру Redux з додатку (рис. 2.4), та сам store (рис. 2.5):

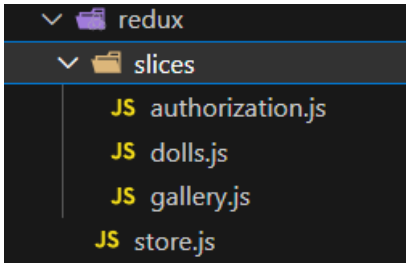


Рисунок 2.4 – архітектура Redux

Рисунок 2.5 – Redux store

Для виконання запитів було використано axios з клієнтської сторони додатку, та express з серверної, для створення маршрутів, які будуть обробляти запити від клієнта. Після приходу запиту на сервер, відповідний обробник запиту (маршрут) виконує необхідну логіку для обробки запиту. Це може включати доступ до бази даних MongoDB, валідацію даних, створення, оновлення або видалення ресурсів [9].

Нижче наведено приклад надсилання запиту за допомогою axios з додатку (рис. 2.6):

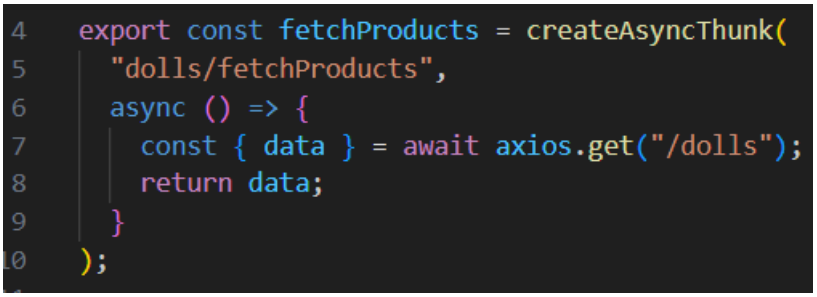


Рисунок 2.6 - надсилання запиту за допомогою axios

Та приклад обробки запиту за допомогою Express з додатку (рис. 2.7)

```
export const getAllDolls = async (res) => {
  try {
    const dolls = await DollModel.find().exec();
    res.json(dolls);
  } catch (err) {
    console.log(err);
    res.status(500).json({
      success: false,
      message: "Can`t get dolls!",
    });
  }
};
```

Рисунок 2.7 - обробки запиту за допомогою Express

2.6 Visual Studio Code як основне середовище розробки

Visual Studio Code (VS Code) є одним з найпопулярніших безкоштовних інтегрованих середовищ розробки **(IDE)**, яке надає широкі можливості для роботи з різними мовами програмування. Воно розроблено компанією Microsoft та доступне для операційних систем Windows, macOS та Linux.

Основні риси і функції Visual Studio Code:

1. Розширюваність: VS Code має потужну систему розширень, яка дозволяє розширити його функціональність для роботи з конкретними мовами програмування, фреймворками, інструментами та іншими розробницькими потребами. Існує широкий вибір розширень, які додають підсвічування синтаксису, автодоповнення, відладку, інтеграцію з системами контролю версій та інші корисні функції.

2. Легкість використання: VS Code має інтуїтивний і зручний інтерфейс користувача, що дозволяє швидко орієнтуватися та виконувати різні дії. Воно має удобну навігацію по файлам і проектам, можливість розгортання терміналу прямо в середовищі та інші зручні функції, що полегшують роботу розробника.

3. Підтримка мов програмування: VS Code надає підтримку для багатьох мов програмування, включаючи JavaScript, TypeScript, Python, C++, Java, PHP, Go, Ruby та багато інших. Він надає можливості підсвічування синтаксису, автодоповнення, перевірки на відповідність стилю коду, відлагодження та інші інструменти для роботи з цими мовами.

4. Інтеграція з Git: VS Code має вбудовану підтримку системи контролю версій Git, що дозволяє виконувати операції комітів, гілок, злиття та інші дії безпосередньо з інтерфейсу редактора. Це полегшує спільну роботу над проектами та забезпечує ефективне використання Git.

5. Налаштування та розширення: VS Code надає широкі можливості для налаштування редактора, включаючи кольорову схему, шрифти, відступи, розширення і багато іншого. Користувачі можуть адаптувати редактор до своїх потреб і вибрати інструменти, які їм підходять.

Різного типу розширення допомогли значно прискорити та спростити роботу з веб-додатком. Далі буде наведено рисунки з розширеннями, які були завантажені для роботи над проєктом (рис. 2.8-2.9):

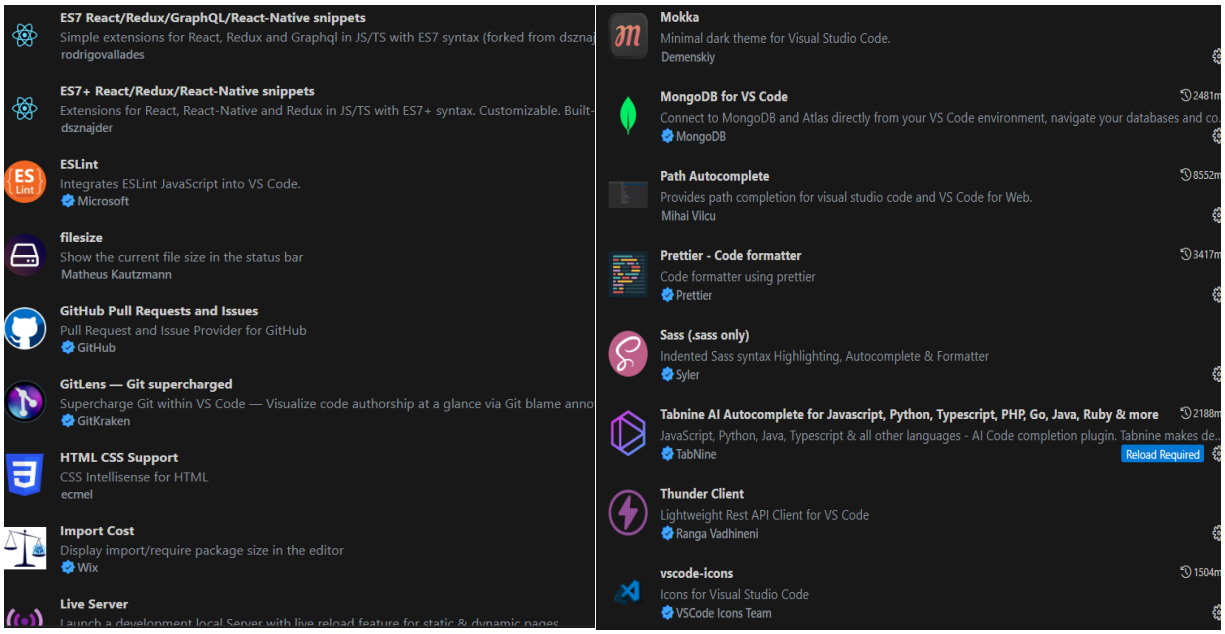


Рисунок. 2.8-2.9 – усі розширення VS Code

3 ІНФОРМАЦІЙНЕ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ СИСТЕМИ

3.1 Реалізація серверної частини проєкту (Back End)

Розробка серверної частини додатку поділяється на дві частини. Перша – проєктування бази даних, створення її структури, моделей. Було створено 5 сутностей, які використовуються в цьому інтернет-магазині, а саме: ляльки, коментарі, галерея, користувачі та замовлення. Ці сутності стали прототипами для основних моделей бази даних:

-Doll – модель, в поля якої зберігаються дані пов’язані з ляльками: ідентифікатор ляльки (_id), назву ляльки (dollName), її зображення (imageUrl), опис ляльки (description), ціну ляльки (price), список коментарів (comments), дату створення товару (createdAt), дату оновлення товару (updatedAt);

-Comment – модель, в поля якої зберігаються дані пов’язані з коментарями до ляльок: ідентифікатор коментаря (_id), текст коментаря (comment), ім’я користувача, що залишив коментар (fullName), аватар користувача, що залишив коментар (avatarUrl), дату створення товару (createdAt), дату оновлення товару (updatedAt).

-Gallery – модель, в поля якої зберігаються дані пов’язані з галереєю зображень: ідентифікатор зображення (_id), зображення (imageUrl).

-User – модель, в поля якої зберігаються дані пов’язані з користувачами: ідентифікатор користувача (_id), повне ім’я (fullName), аватар користувача (avatarUrl), пошту користувача (email), захешований пароль (passwordHash), дату створення акаунту користувача(createdAt), дату оновлення акаунту користувача (updatedAt).

-Order – модель, в поля якої зберігаються дані пов’язані з замовленнями: ідентифікатор замовлення (_id), ідентифікатор замовленого товару (doll_id), електронну адресу покупця (email), адресу доставки (delivery_address), ідентифікатор чеку про оплату товару (payment_id), ідентифікатор користувача, що зробив замовлення (user_id), дату створення замовлення (createdAt), дату оновлення замовлення (updatedAt).

Нижче наведено структуру Back End частини проєкту (рис. 3.1):

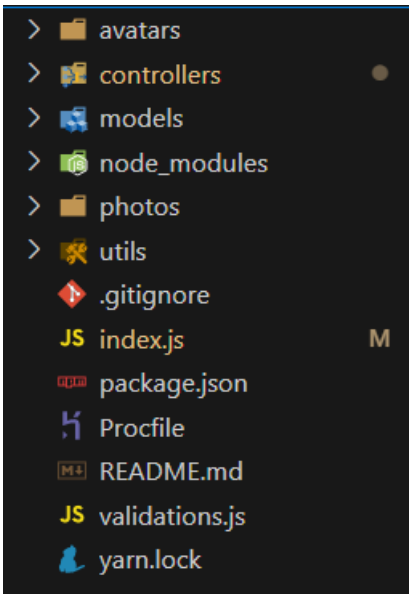


Рис. 3.1 – структура Back End частини проєкту

Також були використані додаткові модулі для Node.js, які дозволяють прискорити та полегшити процес розробки (рис. 3.1.2):

Рис. 3.2 – додаткові залежності в Back End частині проєкту

В кореневому каталозі знаходяться два файли:

-index.js – файл, в якому описані всі маршрути, підключення до бази даних та запуск сервера;

-validations.js – файл, в якому описані перевірки та валідації даних в проєкті;

Також тут знаходяться 5 додаткових каталогів:

Директорія для зберігання аватарів користувачів (**avatars**);

Директорія з контроллерами для кожної моделі в базі даних (**controllers**):

- CommentsController.js – відповідає за обробку запитів, зв’язаних з коментарями;

- DollsController.js – відповідає за обробку запитів, зв’язаних з ляльками;

- Galleryontroller.js - відповідає за обробку запитів, зв’язаних з галереєю зображень;

- UserController.js - відповідає за обробку запитів, зв’язаних з користувачами;

- OrderController.js – відповідає за обробку запитів, зв’язаних з замовленнями ляльок;

Директорія з моделями бази даних (**models**):

- Comment – опис моделі коментарів;

- Doll – опис моделі ляльок;

- Gallery – опис моделі галереї зображень;

- User – опис моделі користувачів;

- Order – опис моделі замовлень;

Директорія для зберігання зображень для галереї (**gallery**);

Директорія з різними допоміжними методами (**utils**):

checkAuth.js – метод, який дозволяє перевірити, чи був авторизований користувач.

handleValidationErrors – файл, який відповідає за обробку помилок валідації.

3.2 Отримання даних з MongoDB за допомогою Express

Express може обробляти усі можливі типи HTTP запитів, такі як GET, POST, PUT, PATCH, DELETE.

На рисунку можна побачити, що в головному index.js файлі представлено маршрути, при переході на які будуть виконуватись певні запити:

```
app.get("/dolls", DollsController.getAllDolls);
app.get("/dolls/:id", DollsController.getOne);
```

Рисунок 3.3 – маршрути для виконання запитів

```
export const getAllDolls = async (res) => {
  try {
    const dolls = await DollModel.find().exec();
    res.json(dolls);
  } catch (err) {
    console.log(err);
    res.status(500).json({
      success: false,
      message: "Can`t get dolls!",
    });
  }
};
```

Рисунок 3.4 – метод виконання першого запиту

```
export const getOne = async (req, res) => {
  try {
    const dollId = req.params.id;

    const doll = await DollModel.findOneAndUpdate(
      {
        _id: dollId,
      },
      {
        $inc: { viewsCount: 1 },
      },
      {
        returnDocument: "after",
      }
    );
    res.json(doll);
  } catch (err) {
    console.log(err);
    res.status(500).json({
      message: "Can't get doll!",
    });
  }
};
```

Рисунок 3.5 – метод виконання другого запиту

3.3 Реалізація клієнтської частини проєкту (Front End)

Робота з клієнтською частиною інтернет магазину складається зі створення інтерфейсу для взаємодії та користування інтернет-магазином та розробки логіки отримання необхідних даних із сервера для подальшого

використання за допомогою здійснення запитів.

Були використані додаткові модулі для Node.js, які дозволяють прискорити та полегшити процес розробки (рис. 3.6):

```
"dependencies": {
  "@emotion/react": "^11.11.0",
  "@emotion/styled": "^11.11.0",
  "@mui/icons-material": "^5.8.0",
  "@mui/material": "^5.13.0",
  "@reduxjs/toolkit": "^1.8.2",
  "@testing-library/jest-dom": "^5.16.5",
  "@testing-library/react": "^13.4.0",
  "@testing-library/user-event": "^13.5.0",
  "axios": "^1.4.0",
  "react": "^18.2.0",
  "react-content-loader": "^6.2.1",
  "react-dom": "^18.2.0",
  "react-hook-form": "^7.43.9",
  "react-multi-carousel": "^2.8.3",
  "react-redux": "^8.0.2",
  "react-responsive-masonry": "^2.1.7",
  "react-router-dom": "^6.11.0",
  "react-scripts": "5.0.1",
  "sass": "^1.62.0",
  "web-vitals": "^2.1.4"
},
```

Рисунок 3.6 – додаткові залежності в Front End частині проєкту

Сама структура проєкту була розбита на окремі директорії, для більш логічного та зручного використання компонентів веб-додатку.

Нижче наведено структуру Front End частини проєкту (рис. 3.7):

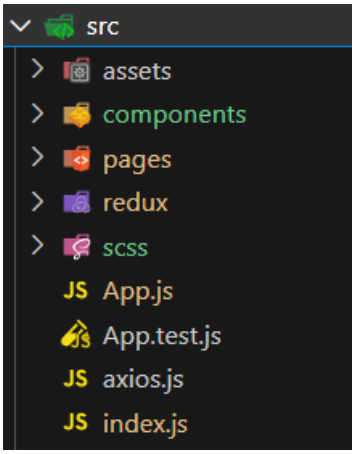


Рисунок 3.7 – структура Front End частини проєкту

Коренева директорія має три основні модулі:

-**App.js** – об’єднує всі файли і дані, та має маршрутизацію;

-**axios.js** – файл, в якому задається стандартне значення посилання на сервер під час запиту;

-**index.js** – файл, який має зв’язок з самою html сторінкою

Також є ще декілька директорій, які також необхідні для проєкту:

-**assets** – директорія для зберігання статичних даних, таких як логотип інтернет-магазину, та інші мультимедійні файли;

-**components** – директорія в якій зберігаються всі основні компоненти, такі як:

- AddComment – компонент, в якому реалізовано та відображено процес додавання коментаря;
- Pagination – компонент, в якому реалізовано та відображено пагінацію;
- Product – компонент, в якому реалізовано відображення інформації про ляльку;
- Search – компонент, в якому реалізована логіка та інтерфейс пошуку ляльки за назвою;
- Sort – компонент, в якому реалізована логіка та інтерфейс сортування відображення ляльок за трьома критеріями, а саме:
    - сортування за ім’ям;
    - сортування за ціною;
    - сортування за популярністю;

Працює як в порядку зростання, так і спадання;

- CartEmpty – компонент, в якому реалізовано інтерфейс порожнього кошика;
- CartItem – компонент, в якому реалізовано логіку та інтерфейс конкретного продукту в кошику;
- CommentsBlock – компонент, в якому реалізовано логіку та інтерфейс блоку коментарів;
- Footer – компонент, в якому реалізовано логіку та інтерфейс блоку нижньої частини веб-додатку;
- Header – компонент, в якому реалізовано логіку та інтерфейс блоку верхньої частини веб-додатку;
- onTopScrollButton – компонент, в якому реалізовано логіку скролу до веб-додатку до початку сторінки.

-**pages** – директорія в якій зберігаються всі основні сторінки, такі як:

- Login – сторінка, авторизації;
- Register – сторінка реєстрації;
- About Us – сторінка «Про нас»;
- Cart – сторінка кошику;
- DollsList – сторінка магазину зі списком ляльок;
- FullProduct – сторінка повної інформації про ляльку;
- Gallery – сторінка галереї зображень;
- Home – домашня сторінка.

-redux – директорія для зберігання глобального стану компонентів, в якій кореневим файлом є файл «сховища» - **store.js,** який зберігає всі необхідні «редьюсери». Також в цій директорії міститься папка slices, в якій знаходяться «слайси», такі як:

- authorization.js – відповідає за стани, пов’язані з користувачем;
- cart.js – відповідає за стани, пов’язані з кошиком;
- comments.js – відповідає за стани, пов’язані з коментарями;
- dolls.js – відповідає за стани, пов’язані з ляльками;
- gallery.js – відповідає за стани, пов’язані з галереєю зображень.

-scss – директорія стилів додатку.

3.4 Зберігання отриманих даних за допомогою Redux

Нижче наведено ініціалізацію сховища store в додатку:

```
import { configureStore } from "@reduxjs/toolkit";
import { authReducer } from "./slices/authorization";
import { dollsReducer } from "./slices/dolls";
import { galleryReducer } from "./slices/gallery";

const store = configureStore({
  reducer: {
    dolls: dollsReducer,
    gallery: galleryReducer,
    auth: authReducer,
  },
});

export default store;
```

Рисунок 3.8 – Ініціалізація store

Це сховище має ‘редьюсери’ – чисті функції, які використовуються для оновлення стану в ньому. Вони приймають поточний стан і дію (action) як аргументи і повертають новий стан.

Далі наведено приклад одного з таких ‘редьюсерів’(reducers):

```
const initialState = {
  dolls: {
    items: [],
    status: "loading",
  },
};
```

Рисунок 3.9 – початковий стан з яким редуктор починає свою роботу

```
4   export const fetchProducts = createAsyncThunk(
5     "dolls/fetchProducts",
6     async () => {
7       const { data } = await axios.get("/dolls");
8       return data;
9     }
10  );
```

Рисунок 3.10 – відправлення запиту за допомогою createAsyncThunk

createAsyncThunk дозволяє створити дію, яка виконує асинхронну операцію. Вона приймає два аргументи: унікальний рядок, що ідентифікує цю дію, і функцію, яка містить логіку асинхронної операції.

```
19  const dollsSlice = createSlice({
20    name: "dolls",
21    initialState,
22    reducer: {},
23    extraReducers: {
24      [fetchProducts.pending]: (state) => {
25        state.dolls.items = [];
26        state.dolls.status = "loading";
27      },
28
29      [fetchProducts.fulfilled]: (state, action) => {
30        state.dolls.items = action.payload;
31        state.dolls.status = "loaded";
32      },
33
34      [fetchProducts.rejected]: (state) => {
35        state.dolls.items = [];
36        state.dolls.status = "error";
37      },
38    },
39  });
40
41  export const dollsReducer = dollsSlice.reducer;
```

Рисунок 3.11 – створення slice стану та обробка

createSlice - це функція, яка надається бібліотекою Redux Toolkit і використовується для створення "срізу" (slice) стану та відповідних дій (actions) та редукторів для цього срізу.

Параметр extraReducers дозволяє додавати додаткові редуктори до срізу (slice), які оброблятимуть дії, які не прив'язані безпосередньо до властивостей стану в срізу.

```
const dispatch = useDispatch();
const { dolls } = useSelector((state) => state.dolls);
const isProductLoading = dolls.status === "loading";

React.useEffect(() => {
  dispatch(fetchProducts());
}, []);
```

Рисунок 3.12 – отримання даних з сервера та подальше відображення на стороні клієнта

dispatch використовується для відправки дій до Redux Store для оновлення стану додатку.

useSelector з Redux Toolkit використовується для вибору певної частини стану зі Store. У цьому випадку, він вибирає властивість dolls зі стану.

React.useEffect використовується для виклику функції під час монтування компонента. У даному випадку, при монтуванні компонента відбувається виклик dispatch(fetchProducts()), що відправляє дію fetchProducts до Redux Store. Ця дія імплементує асинхронну операцію отримання продуктів з сервера.

3.5 Інструкція з використання програмного продукту

З ціллю просування додатку не лише на території України, а й за кордоном, мною було вирішено робити інтерфейс англійською мовою з можливістю в майбутньому змінювати мову додатку на українську або іншу. Коли користувач переходить до веб-додатку, він перед собою бачить домашню сторінку. Нижче буде приведено рисунки домашньої сторінки (рис. 3.13, 3.14):

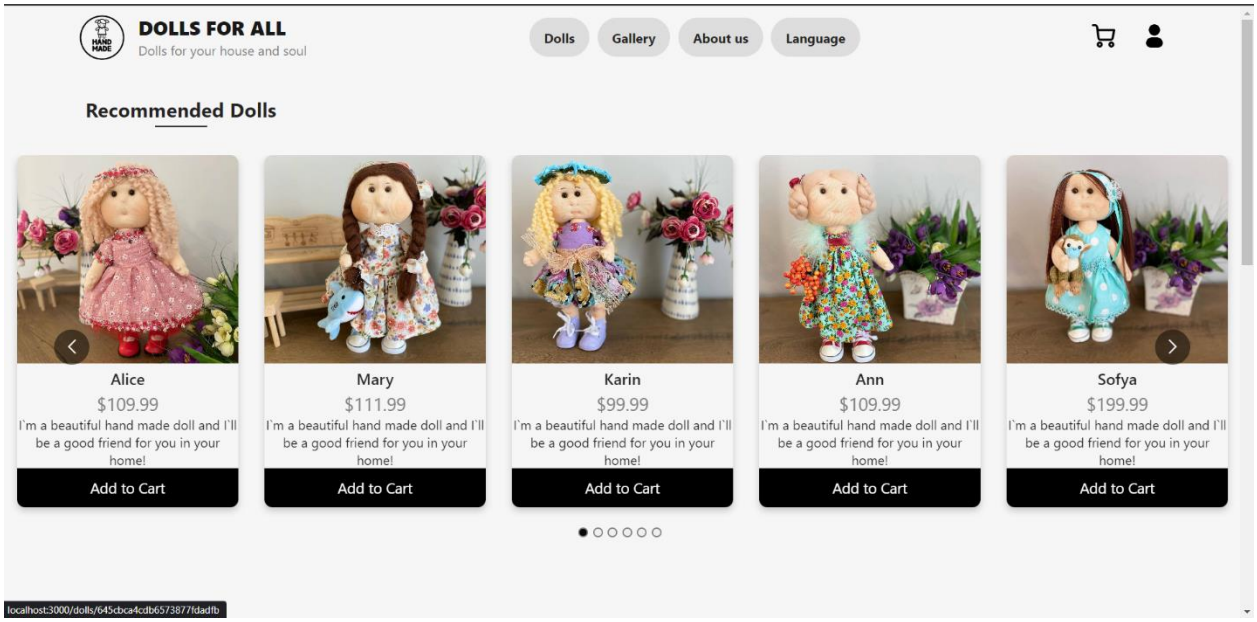


Рисунок 3.13 – домашня сторінка

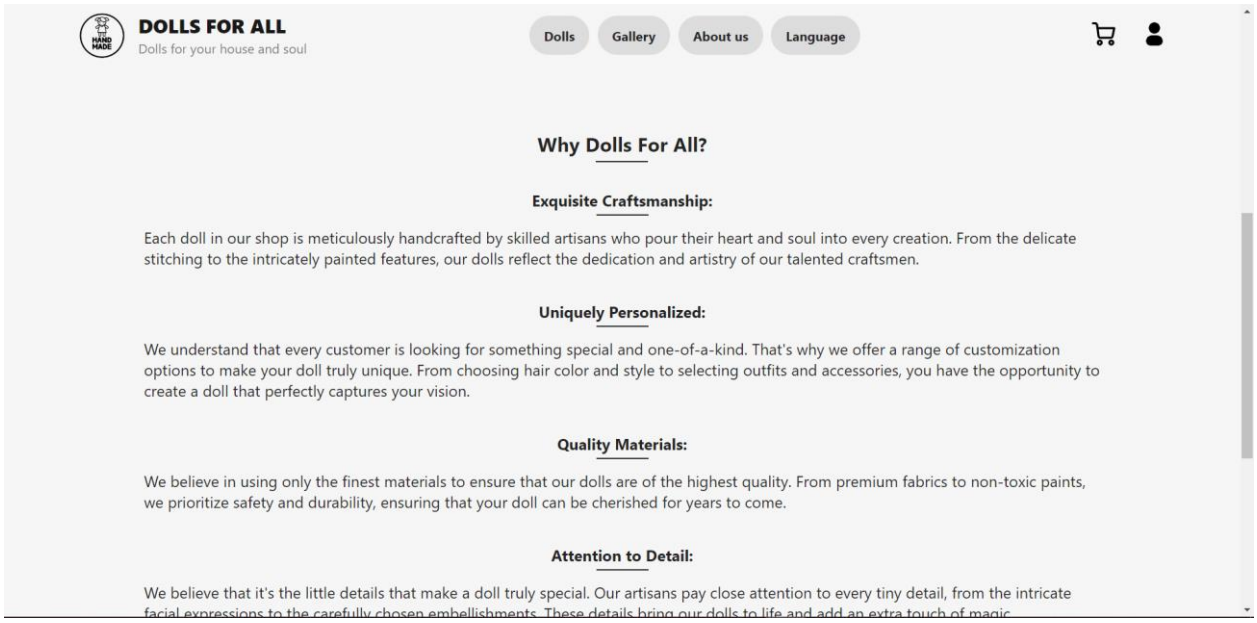


Рисунок 3.14 – домашня сторінка

На цій сторінці є можливість передивитись рекомендовані ляльки, одразу додати їх до кошику, натиснувши відповідну кнопку.

Користувач має можливість увійти в свій обліковий запис, що б мати можливість залишати коментарі, та якщо він був авторизований, переглядати історію своїх замовлень.

Натиснувши на пункт меню авторизації, користувач переходить на сторінку авторизації (рис. 3.15):

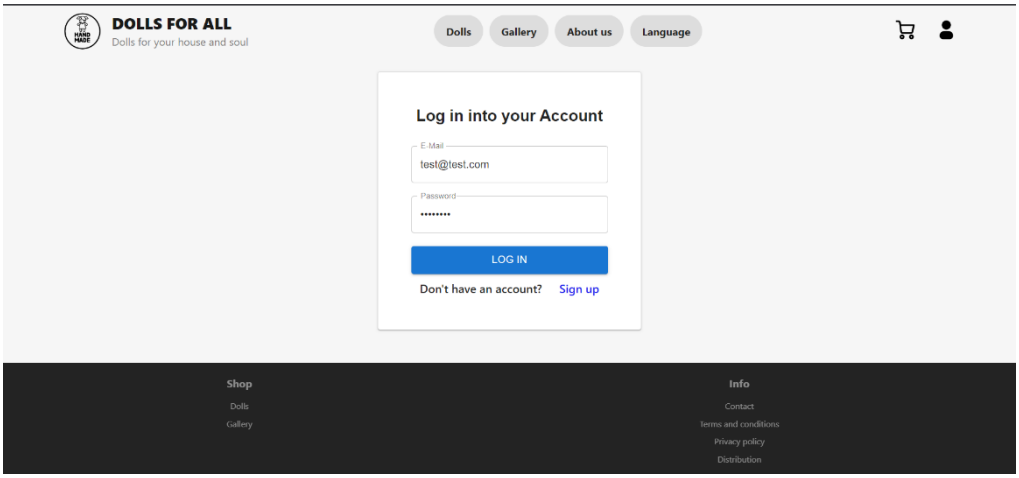


Рисунок 3.15 – форма авторизації

Якщо ж у користувача ще немає облікового запису, він може зареєструватись, натиснувши на посилання “Sign up”, що б перейти на сторінку реєстрації нового користувача (рис. 3.16):

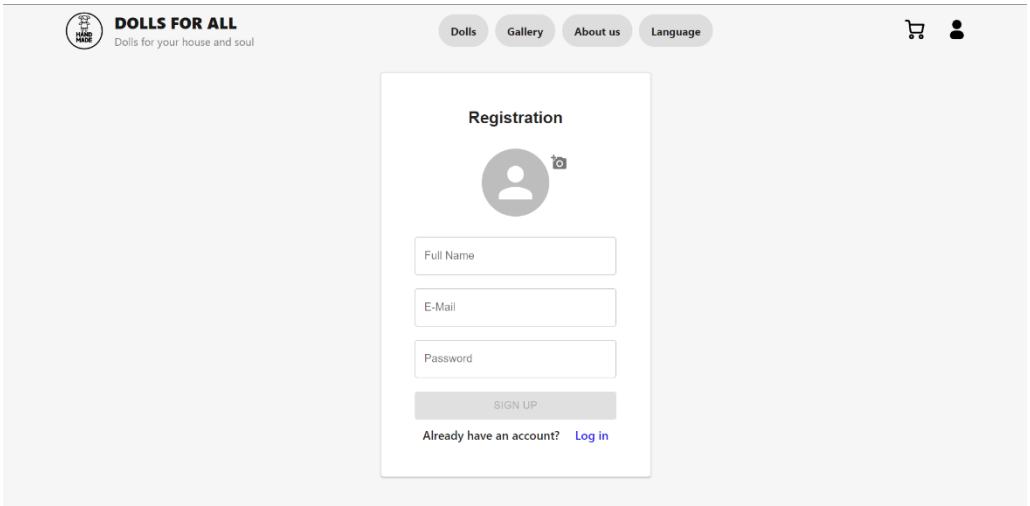


Рисунок 3.16 – форма реєстрації нового облікового запису

Після успішної авторизації чи реєстрації, користувач повертається на головну сторінку, де в шапці змінюється кнопка авторизації на іконку профілю з кнопкою “Log Out” (рис. 3.17):

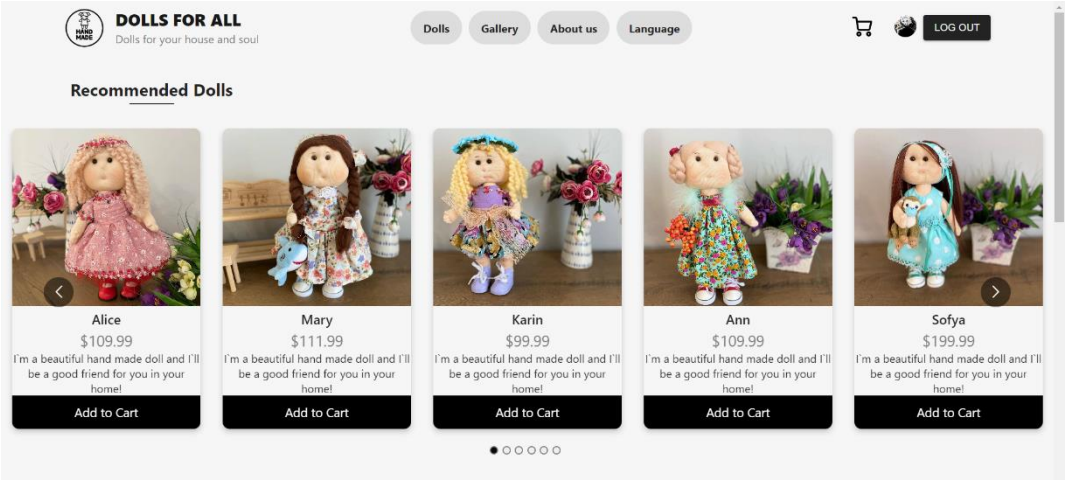


Рисунок 3.17 – успішна авторизація користувача

Також є можливість перейти на сторінку обраної ляльки (рис 3.18). Нижче буде блок інформації, чому користувачі повинні обрати саме нашу продукцію.

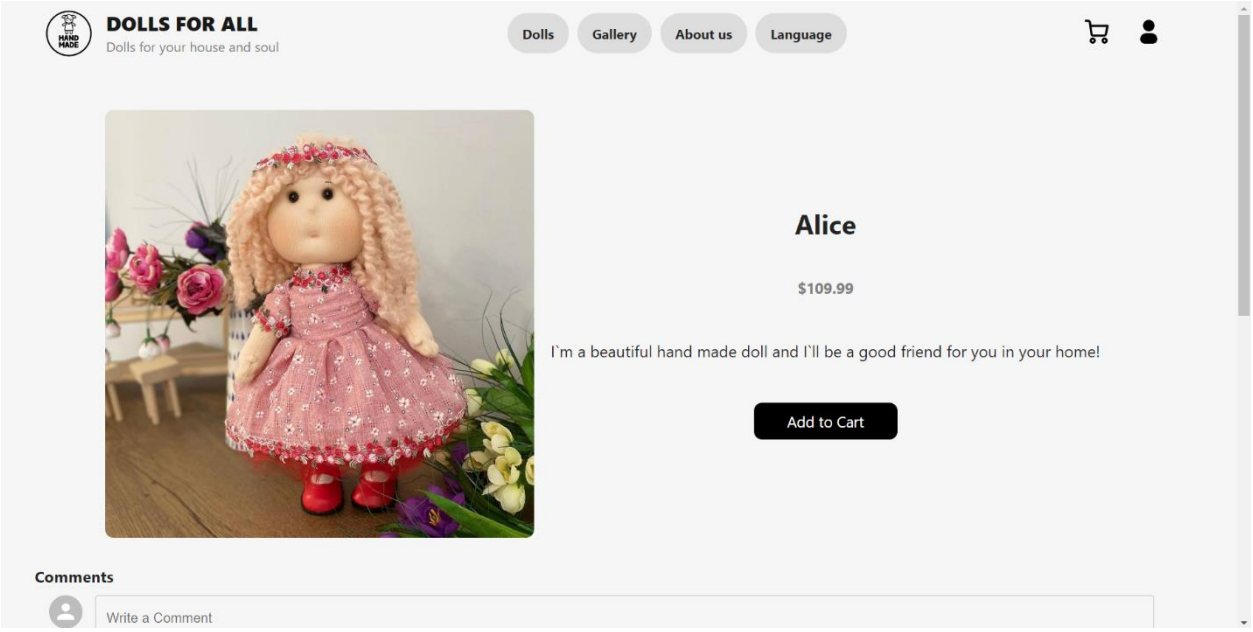


Рисунок 3.18 – сторінка обраної ляльки

На сторінці обраної ляльки, авторизовані користувачі мають можливість залишати коментарі свої коментарі, як це зображено на рисунку нижче (рис. 3.19)

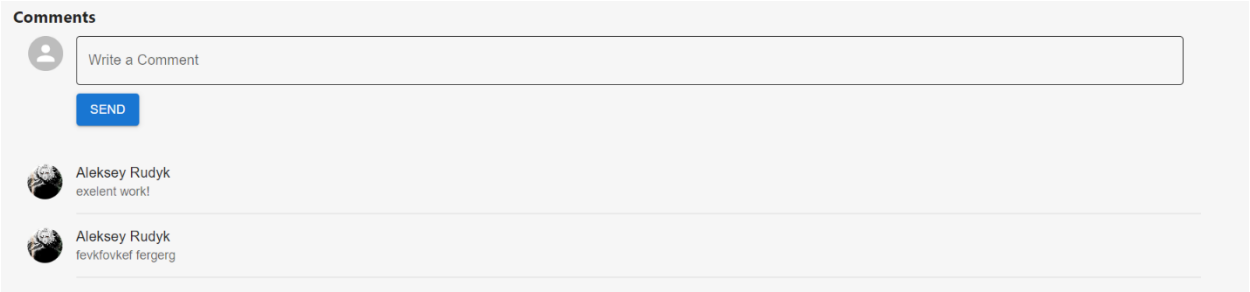


Рисунок 3.19 – блок коментарів

У шапці веб-додатку є список пунктів меню, які користувач може переглянути (рис. 3.20):

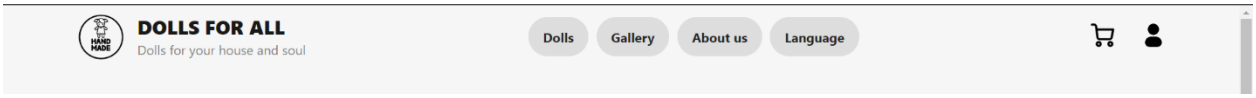


Рисунок 3.20 – шапка веб-додатку

Натиснувши на пункт меню “Dolls”, користувач переходить на сторінку списку ляльок доступних для придбання. Нижче приведено рисунок цієї сторінки (рис. 3.21):

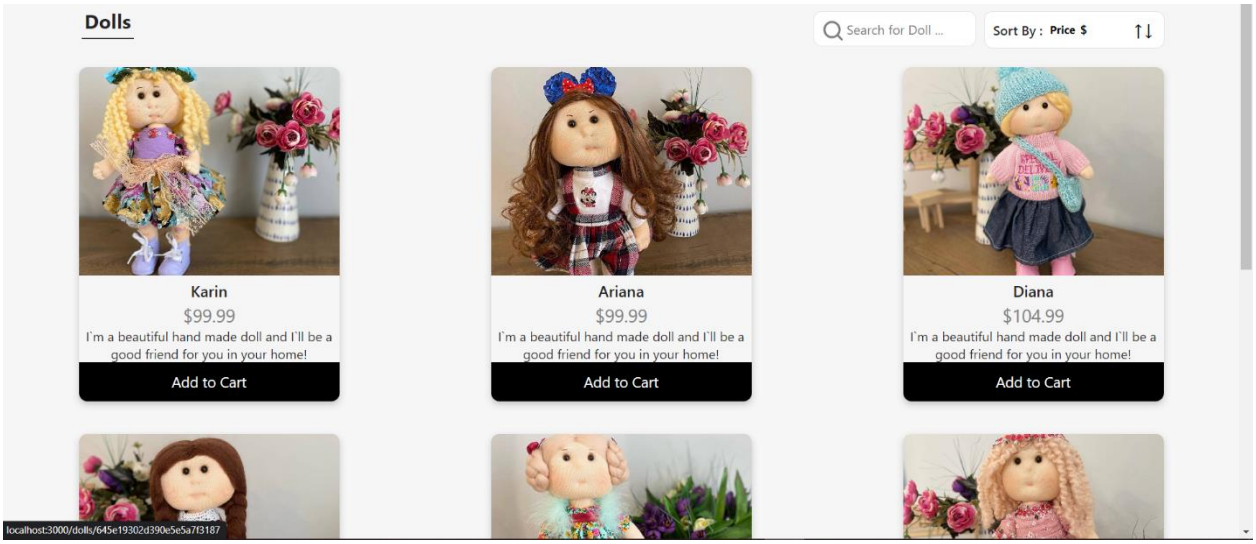


Рисунок 3.21 – сторінка списку ляльок

Тут, як і на домашній сторінці, користувач має можливість передивитись інформацію по обраній ляльці, або додати її до кошику.

Також на цій сторінці, користувачі можуть за допомогою пошукового запиту знайти необхідну їм ляльку за її ім’ям (рис. 3.22),

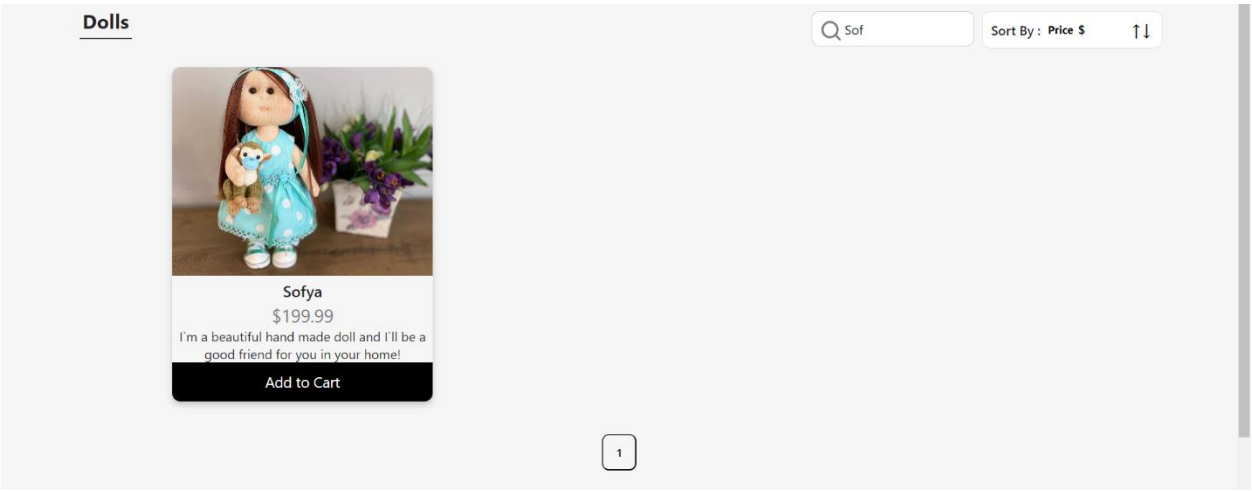


Рисунок 3.22 – пошук за ім’ям

або відсортувати список ляльок за трьома атрибутами (рис. 3.23):

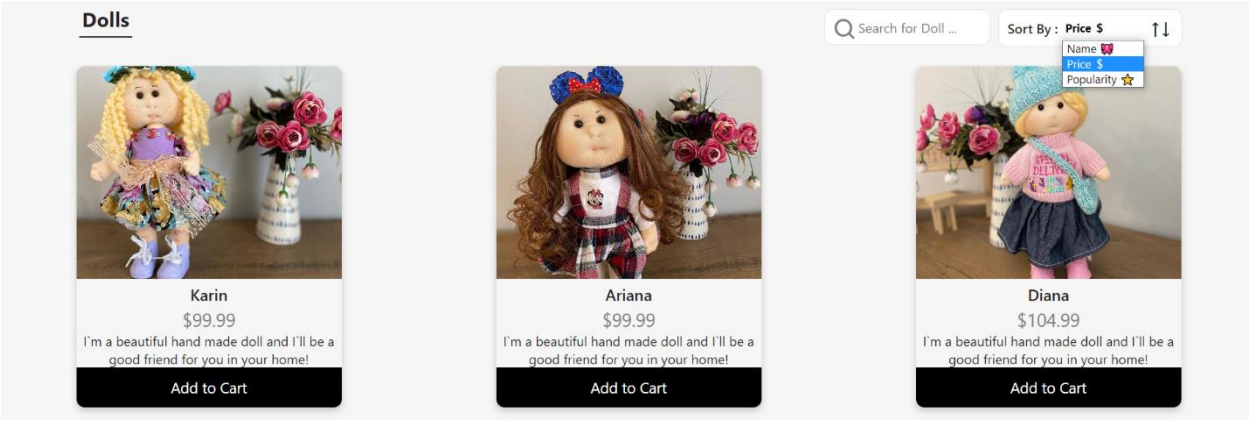


Рисунок 3.23 – сортування ляльок

Натиснувши на пункт меню “Gallery”, користувач переходить на сторінку галереї зображень, де він має можливість переглянути інші роботи майстра у вигляді різних ляльок. Нижче наведено рисунок галереї (рис. 3.24)

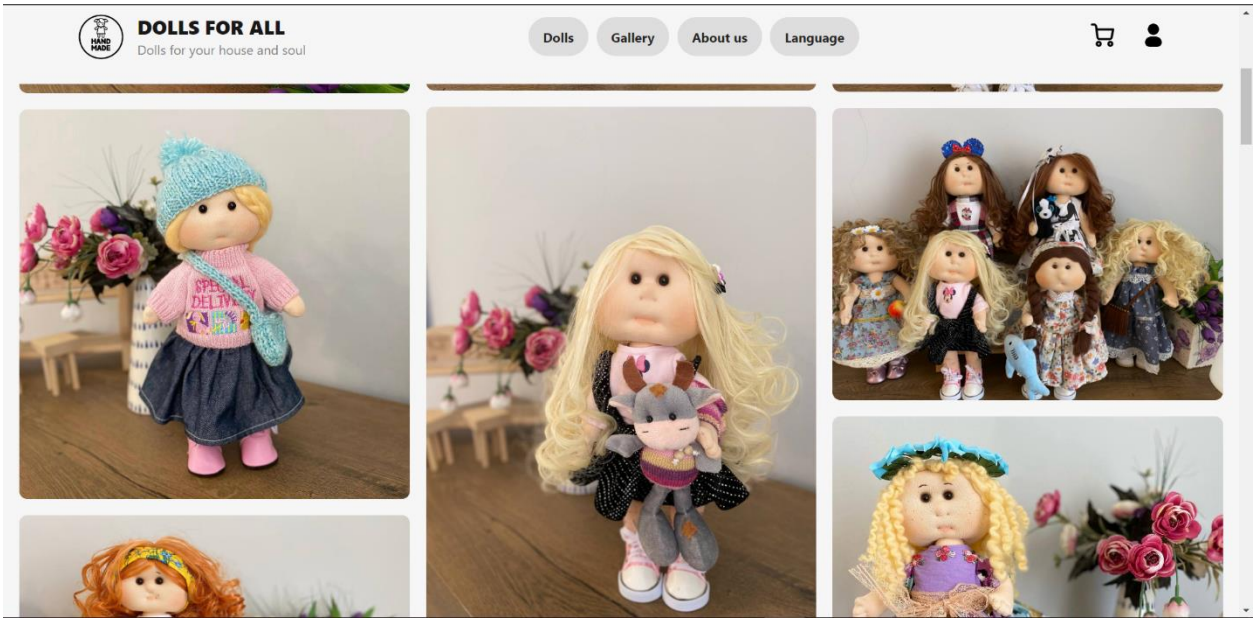


Рисунок 3.24 – галерея зображень

Натиснувши на пункт меню “About Us”, користувач переходить на сторінку про цей інтернет магазин, де власник розповідає про себе, та свої вироби мистецтва (рис. 3.25):

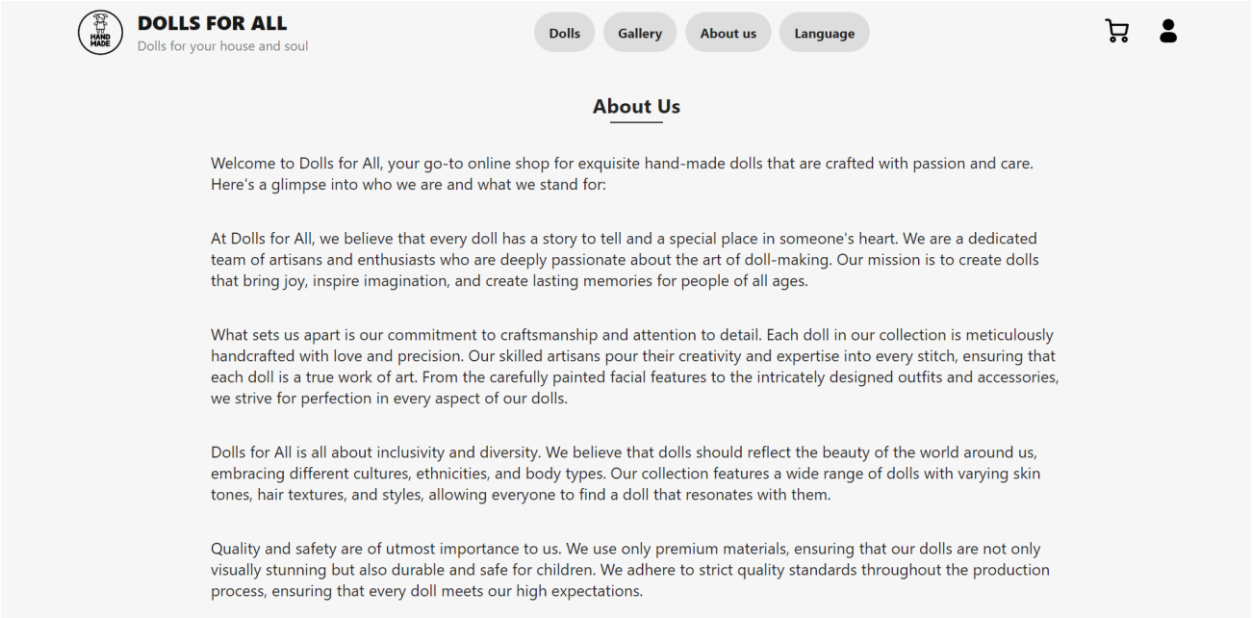


Рисунок 3.25 – сторінка “About us”

Натиснувши кнопку кошику, користувач переходить на відповідну сторінку, де він може переглянути свої замовлення, та придбати ляльки (рис.3.26):

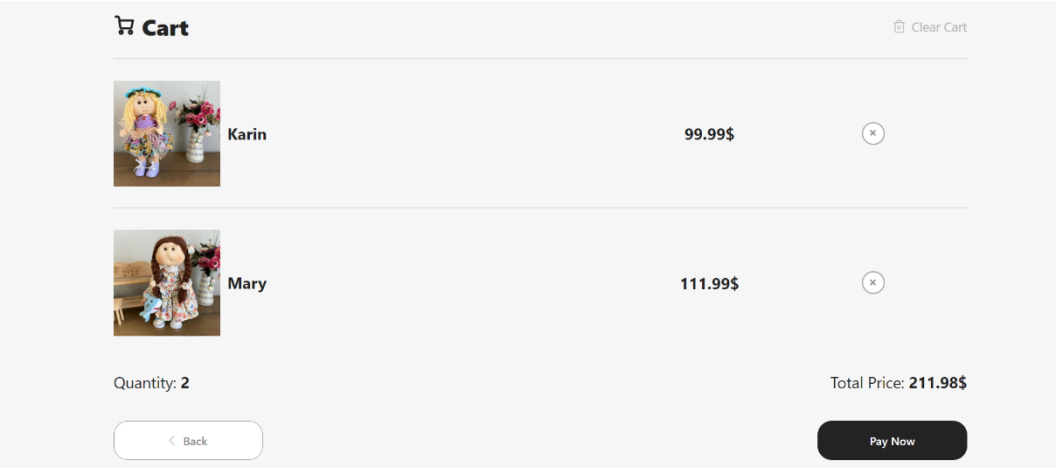


Рисунок 3.26 – Сторінка кошику

Також є можливість повернутись до замовлень, або видалити з кошику обрані ляльки (рис. 3.27):

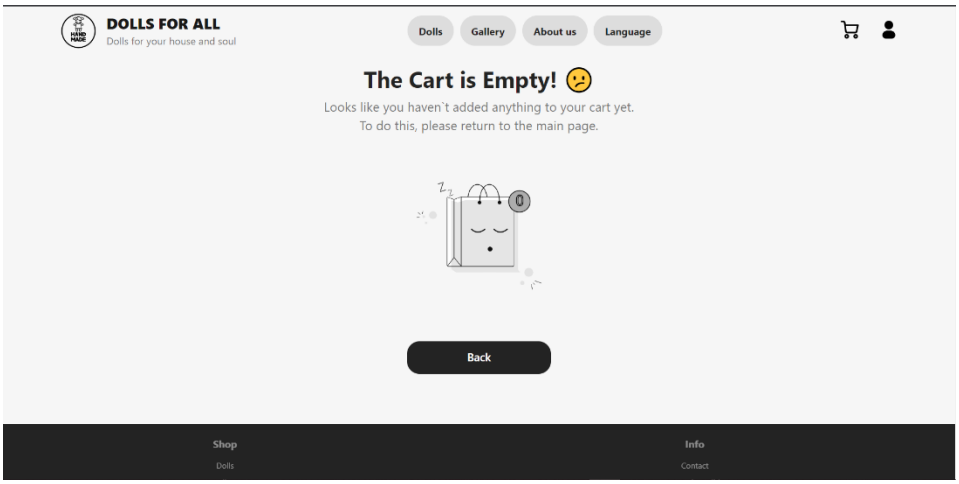


Рисунок 3.27 – порожній кошик після видалення замовлення

ВИСНОВКИ

У результаті виконання дипломного проєкту було розроблено веб-додаток з використанням стеку MERN(MongoDB, Express, React, Node).

Для досягнення бажаного результату були проаналізовані інтернет-джерела та додаткова література, пов’язані з розробкою e-commerce проєктів.

Було визначено основні функції та можливості інтернет-магазину, необхідні для зручного та ефективного процесу покупок.

Розроблено зручний та привабливий дизайн інтернет-магазину, який відповідає стилістиці та естетиці ляльок ручної роботи.

Реалізовано необхідний функціонал, такий як каталог продукції, кошик покупок, система коментарів щодо ляльок, пошукова система тощо.

Інтегровано систему оплати.

Під час роботи над додатком було наведено приклади використання усіх засобів даного стеку. Показано, як за допомогою Express & Node.js відбувається обробка та виконання запитів. Та як за допомогою Redux з’явилася можливість з неймовірною легкістю керувати станом додатка.

Також, за допомогою React, було створено досить швидкий, оптимізований користувацький інтерфейс, що завдяки компонентному підходу, React дозволяє в майбутньому перевикористовувати вже існуючі компоненти.

MongoDB зберігає усі необхідні дані у вигляді моделей, що досить легко дозволяє додавати, редагувати, отримувати або видаляти записи у базі даних.

Надалі планується розробка панелі адміністратора та локалізація веб-додатку іншою мовою(наприклад: українською, та німецькою).

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Full-Stack React Projects 2nd Edition (2020) - https://sd.blackball.lv/library/Full-Stack_React_Projects_2nd_Edition_(2020).pdf
2. MVC - https://drive.google.com/file/d/1hYc6b2qaNvCZ6VimrTzsAEMKFvWkSVg-/view?usp=drive_link
3. Redux - https://drive.google.com/file/d/1sAW2Vxmi2ud8AxKLCoejLhEXq90e3EpX/view?usp=drive_link
4. MongoDB - https://www.mongodb.com/docs/
5. React - https://react.dev/blog/2023/03/16/introducing-react-dev
6. Express - https://expressjs.com/
7. Node - https://nodejs.org/en/docs
8. Redux - https://redux.js.org/
9. Axios - https://axios-http.com/docs/intro
10. MERN - https://blog.logrocket.com/mern-stack-tutorial/

ДОДАТОК А

(A1) // back-end index.js

```
import express from "express";
import multer from "multer";
import mongoose from "mongoose";  Calculating...
import cors from "cors";

import { registerValidation, loginValidation } from "./validations.js";

import {
  UserController,
  DollsController,
  GalleryController,
  CommentsController,
} from "./controllers/index.js";
import { handleValidationErrors, checkAuth } from "./utils/index.js";

mongoose
  .connect(
    "mongodb+srv://admin:Qweasdzxc@cluster0.s7bwyba.mongodb.net/dolls-shop"
  )
  .then(() => console.log("DB connected successfully!"))
  .catch((err) => console.log(err));

const app = express();

const storage = multer.diskStorage({
  destination: (_, __, cb) => {
    cb(null, "avatars");
  },
  filename: (_, file, cb) => {
    cb(null, file.originalname);
  },
});

const upload = multer({ storage });

app.use(express.json());
```

```
app.use(cors());
app.use("/photos", express.static("photos"));
app.use("/avatars", express.static("avatars"));

app.post("/upload", upload.single("image"), (req, res) => {
  res.json({
    url: `avatars/${req.file.originalname}`,
  });
});
app.get("/dolls", DollsController.getAllDolls);
app.get("/dollsCarousel", DollsController.getDollsCarousel);
app.get("/dolls/:id", DollsController.getOne);
app.post("/dolls", checkAuth, DollsController.createDoll);

app.post("/comments/:id", checkAuth, CommentsController.createComment);
app.get("/dolls/comments/:id", DollsController.getComments);

app.get("/gallery", GalleryController.getGallery);
app.post("/addPhoto", GalleryController.addPhoto);

app.post(
  "/auth/login",
  loginValidation,
  handleValidationErrors,
  UserController.login
);
app.post(
  "/auth/register",
  registerValidation,
  handleValidationErrors,
  UserController.register
);
app.get("/auth/me", checkAuth, UserController.getMe);

app.listen(4444, (err) => {      Aleksey Rudyk, 2 months ago • User regi
  if (err) console.log(err);
  console.log(`Server is ok!`);
```

(A2) //CommentsController.js

```
export const createComment = async (req, res) => {
  try {
    const { dollId, comment, userData } = req.body;
    if (!comment)
      return res.status(400).json({
        success: false,
        message: "Comment cannot be empty!",
      });
    const newComment = new CommentModel({        You, 4 d
      comment,
      fullName: userData.fullName,
      avatarUrl: userData.avatarUrl,
    });
    await newComment.save();
    try {
      await DollModel.findByIdAndUpdate(dollId, {
        $push: { comments: newComment._id },
      });
    } catch (err) {
      console.log(err);
      res.status(500).json({
        success: false,
        message: "Can't add comment!",
      });
    }
    res.json(newComment);
  } catch (err) {
    console.log(err);
    res.status(500).json({
      success: false,
      message: "Can't create comment!",
    });
  }
};
```

(A3) // DollsController.js

```
export const getAllDolls = async (req, res) => {
  try {
    const page = parseInt(req.query.page) - 1 || 0;
    const limit = parseInt(req.query.limit) || 6;      You, 11 hours ago • U
    const search = req.query.searchValue || "";
    let sort = req.query.sort || "viewsCount";
    req.query.sort ? (sort = req.query.sort.split(",")) : (sort = [sort]);
    let sortBy = {};
    if (sort[1]) {
      sortBy[sort[0]] = sort[1];
    } else {
      sortBy[sort[0]] = "asc";
    }
    const dolls = await DollModel.find({
      dollName: { $regex: search, $options: "i" },
    })
      .sort(sortBy)
      .skip(page * limit)
      .limit(limit);
    const total = await DollModel.countDocuments({
      dollName: { $regex: search, $options: "i" },
    });

    const response = {
      error: false,
      total,
      page: page + 1,
      limit,
      dolls,
    };
    res.json(response);
  } catch (err) {
    console.log(err);
    res.status(500).json({
      success: false,
      message: "Can`t get dolls!",
    });
  }
```

```
export const getDollsCarousel = async (req, res) => {
  try {
    const dolls = await DollModel.find().exec();
    res.json(dolls);
  } catch (err) {
    console.log(err);
    res.status(500).json({
      success: false,
      message: "Can`t get dolls!",
    });
  }
};

export const getOne = async (req, res) => {
  try {
    const dollId = req.params.id;

    const doll = await DollModel.findOneAndUpdate(
      {
        _id: dollId,
      },
      {
        $inc: { viewsCount: 1 },
      },
      {
        returnDocument: "after",
      }
    );
    res.json(doll);
  } catch (err) {
    console.log(err);
    res.status(500).json({
      message: "Can't get doll!",
    });
  }
};
```

```
export const createDoll = async (req, res) => {
  try {
    const doc = new DollModel({
      dollName: req.body.dollName,
      description: req.body.description,
      price: req.body.price,
      imageUrl: req.body.imageUrl,
      comments: req.body.comments,
    });

    const post = await doc.save();
    res.json(post);
  } catch (err) {
    console.log(err);
    res.status(500).json({
      success: false,
      message: "Can't create doll!",
    });
  }
};

export const getComments = async (req, res) => {
  try {
    const doll = await DollModel.findById(req.params.id);
    const comments = await Promise.all(
      doll.comments.map((comment) => {
        return CommentModel.findById(comment);
      })
    );
    res.json(comments);
  } catch (err) {
    res.status(500).json({
      success: false,
      message: "Can't get comments!",
    });
  }
};
```

(A4) //GalleryController.js

```
import GalleryModel from "../models/Gallery.js";

export const getGallery = async (req, res) => {
  try {
    const gallery = await GalleryModel.find().exec();
    res.json(gallery);
  } catch (err) {
    console.log(err);
    res.status(500).json({
      success: false,
      message: "Can`t get dolls!",
    });
  }
};

export const addPhoto = async (req, res) => {
  try {
    const doc = new GalleryModel({
      imageUrl: req.body.imageUrl,
    });

    const photo = await doc.save();
    res.json(photo);
  } catch (err) {
    console.log(err);
    res.status(500).json({
      success: false,
      message: "Can't add photo!",        You, 3 weeks a
    });
  }
};
```

(A5) //UserController.js

```
export const register = async (req, res) => {
  try {
    const password = req.body.password;
    const salt = await bcrypt.genSalt(10);
    const hash = await bcrypt.hash(password, salt);

    const doc = new UserModel({
      fullName: req.body.fullName,
      email: req.body.email,
      avatarUrl: req.body.avatarUrl,
      passwordHash: hash,
    });

    const user = await doc.save();
    const token = jwt.sign({ _id: user._id }, "secretCode", {
      expiresIn: "1h",
    });

    const { passwordHash, ...userData } = user._doc;

    res.json({
      ...userData,
      token,
    });
  } catch (err) {
    console.log(err);
    res.status(500).json({
      success: false,
      message: "Can't register user!",
    });
  }
};
```

```
export const login = async (req, res) => {
  try {
    const user = await UserModel.findOne({ email: req.body.email });
    if (!user) {
      return res.status(403).json({
        success: false,
        message: "Incorrect login or password!",
      });
    }
    const isMatch = await bcrypt.compare(
      req.body.password,
      user._doc.passwordHash
    );
    if (!isMatch) {
      return res.status(403).json({
        success: false,
        message: "Incorrect login or password!",
      });
    }
    const token = jwt.sign({ _id: user._id }, "secretCode", {
      expiresIn: "1h",
    });
    const { passwordHash, ...userData } = user._doc;
    res.json({
      ...userData,
      token,
    });
  } catch (err) {
    console.log(err);
    res.status(500).json({
      success: false,
      message: "Can't login user!",
    });
  }
};
```

```
export const getMe = async (req, res) => {
  try {
    const user = await UserModel.findById(req.userId);

    if (!user) {
      return res.status(404).json({
        success: false,
        message: "User not found!",
      });
    }
    const { passwordHash, ...userData } = user._doc;

    res.json(userData);
  } catch (err) {
    console.log(err);
    res.status(500).json({
      success: false,

      message: "Can't get user!",
    });
  }
};
```

```
export const update = async (req, res) => {
  try {
    const userId = req.params.id;

    await UserModel.updateOne(
      {
        _id: userId,
      },
      {
        fullName: req.body.fullName,
        email: req.body.email,
        avatarUrl: req.body.avatarUrl,
      }
    );
    res.json({
      success: true,
    });
  } catch (err) {
    console.log(err);
    res.status(500).json({
      success: false,
      message: "Can't update user!",
    });
  }
};
```

(A6) //CommentsModel

```
import mongoose from "mongoose";  839.6k (gzipped: 227.3k)

const CommentSchema = new mongoose.Schema(
  {
    comment: {
      type: String,
      required: true,
    },
    fullName: {
      type: String,
      required: true,
    },
    avatarUrl: String,        You, 4 days ago • comments funct
    author: {
      type: mongoose.Schema.Types.ObjectId,
      ref: "User",
    },
  },
  {
    timestamps: true,
  }
);

export default mongoose.model("Comment", CommentSchema);
```

(A7) //DollsModel

```
import mongoose from "mongoose";  839.6k (gzipped: 227.3k)

const DollSchema = new mongoose.Schema(
  {
    dollName: {
      type: String,
      required: true,
      unique: true,
    },
    description: {
      type: String,
      required: true,
    },
    viewsCount: {
      type: Number,
      default: 0,
    },
    price: {
      type: Number,
      required: true,
    },
    imageUrl: {
      type: String,
      //default: [],
      required: true,
    },
    comments: [{ type: mongoose.Schema.Types.ObjectId, ref: "Comment" }],
  },
  {
    timestamps: true,
  }
);

export default mongoose.model("Doll", DollSchema);
```

(A8) //GalleryModel

```
const GallerySchema = new mongoose.Schema(
  {
    imageUrl: {
      type: String,
      required: true,
      unique: true,
    },
  },
  {
    timestamps: true,
  }       You, 3 weeks ago • gallery backend done …
);

export default mongoose.model("Gallery", GallerySchema);
```

(A9) //UserModel

```
import mongoose from "mongoose";  839.6k (gzipped: 227.3k)

const UserSchema = new mongoose.Schema(
  {
    fullName: {
      type: String,
      required: true,
    },
    email: {       Aleksey Rudyk, 2 months ago • User regis
      type: String,
      required: true,
      unique: true,
    },
    passwordHash: {
      type: String,
      required: true,
    },
    avatarUrl: String,
  },
  {
    timestamps: true,
  }
);

export default mongoose.model("User", UserSchema);
```

(A10) //CheckAuth.js

```
import jwt from "jsonwebtoken";  126.3k (gzipped: 41.3k)

export default (req, res, next) => {
  const token = (req.headers.authorization || "").replace(/Bearer\s?/, "");

  if (!token) {
    return res.status(403).json({ message: "Access denied!" });
  } else {
    try {
      const decoded = jwt.verify(token, "secretCode");
      req.userId = decoded._id;       Aleksey Rudyk, 2 months ago • login, r
      next();
    } catch (err) {
      return res.status(403).json({ message: "Access denied!" });
    }
  }
};
```

(A11) //handleValidationErrors.js

```
import { validationResult } from "express-validator";

export default (req, res, next) => {
  const errors = validationResult(req);
  if (!errors.isEmpty()) {
    return res.status(400).json(errors.array());
  }
  next();
};
```

ДОДАТОК Б

(Б1) front-end // Home.js(Fragment)

```
const Home = () => {
  const [data, setData] = React.useState();
  const [isLoading, setIsLoading] = React.useState(true);

  React.useEffect(() => {
    axios
      .get(`/dollsCarousel`)
      .then((res) => {
        setData(res.data);
        setIsLoading(false);
      })
      .catch((err) => {
        console.warn(err);
        alert("Error getting doll!");
      });
  }, []);
```

(Б2) //Gallery.js(Fragment)

```
export const Gallery = () => {
  const dispatch = useDispatch();
  const { gallery } = useSelector((state) => state.gallery);
  const isGalleryLoading = gallery.status === "loading";

  React.useEffect(() => {       You, 3 weeks ago • gallery frontend done …
    dispatch(fetchGallery());
  }, []);

  return (
    <div style={{ padding: "20px" }}>
      <ResponsiveMasonry columnsCountBreakPoints={{ 350: 1, 750: 2, 900: 3 }}>
        <Masonry gutter="20px">
          {(isGalleryLoading ? [...Array(15)] : gallery.items).map(
            (obj, index) =>
              isGalleryLoading ? (
                <div
                  style={{
                    width: "400px",
                    height: "400px",
                    backgroundColor: "#eee",
                  }}
```

(Б3) //FullProduct.js(Fragment)

```
export const FullProduct = () => {
  const [data, setData] = React.useState();
  const { comments } = useSelector((state) => state.comments);
  const [isLoading, setIsLoading] = React.useState(true);
  const { id } = useParams();
  const dispatch = useDispatch();

  const onClickAdd = () => {
    const item = {       You, 6 days ago • some bug fixes and a
      id: data._id,
      dollName: data.dollName,
      imageUrl: data.imageUrl,
      price: data.price,
    };
    dispatch(addProduct(item));
  };

  React.useEffect(() => {
    axios
      .get(`/dolls/${id}`)
      .then((res) => {
        setData(res.data);
        setIsLoading(false);
      })
      .catch((err) => {
        console.warn(err);
        alert("Error getting doll!");
      });
```

```
  React.useEffect(() => {
    dispatch(fetchComments(id));
  }, []);

  if (isLoading) {
    return (
      <div
        style={{
          display: "flex",
          justifyContent: "center",
          width: "800px",
          height: "800px",
          backgroundColor: "#eee",
        }}
      />
    );
  }
```

(Б4) // DollsList.js(Fragment)

```
export const DollsList = () => {
  const dispatch = useDispatch();

  const { dolls } = useSelector((state) => state.dolls);
  const isProductLoading = dolls.status === "loading";
  const [sort, setSort] = React.useState({ sort: "price", order: "asc" });
  const [searchValue, setSearchValue] = React.useState("");
  const [page, setPage] = React.useState(1);
  React.useEffect(() => {       You, 3 weeks ago • dolls shop menu
    dispatch(fetchProducts({ sort, searchValue, page }));
  }, [sort, searchValue, page]);
  console.log(dolls);
  return (
    <>
      <div className="recommended__header">
        <div className="section_header">
          <h3>Dolls</h3>
        </div>
        <div className="section_right">
          <Search
            setSearchValue={(searchValue) => setSearchValue(searchValue)}
          />
          <Sort sort={sort} setSort={setSort} />
        </div>
```

(Б5) // Cart.js(Fragment)

```
const Cart = () => {
  const dispatch = useDispatch();
  const { items, totalPrice } = useSelector((state) => state.cart);
  console.log(items);       You, 6 days ago • some bug fixes and adding
  const onClickClear = () => {
    dispatch(clearCart());
  };
  const totalCount = items.reduce((sum, item) => sum + item.count, 0);

  React.useEffect(() => {
    localStorage.setItem("cart", JSON.stringify(items));
  }, [items]);

  if (totalPrice === 0) {
    return <CartEmpty />;
  }
```

(Б6) // Register.js(Fragment)

```
export const Register = () => {
  const [imageUrl, setImageUrl] = React.useState("");
  const inputFileRef = React.useRef(null);

  const handleChangeFile = async (event) => {
    try {
      const formData = new FormData();
      const file = event.target.files[0];
      formData.append("image", file);
      const { data } = await axios.post("/upload", formData);
      setImageUrl(data.url);
    } catch (err) {
      console.warn(err);
      alert("Error uploading image!");
    }
  };

  console.log(imageUrl);

  const onClickRemoveImage = () => {
    setImageUrl("");
  };

  const isAuth = useSelector(selectIsAuth);

  const dispatch = useDispatch();
```

```
const {
  register,
  handleSubmit,
  formState: { errors, isValid },
} = useForm({
  defaultValues: {
    fullName: "",
    email: "",
    password: "",
  },
  mode: "onChange",
});

const onSubmit = async (values) => {
  const data = await dispatch(
    fetchRegister({
      ...values,
      avatarUrl: `http://localhost:4444/${imageUrl}`,
    })
  );
  console.log({ ...values, avatarUrl: `http://localhost:4444/${imageUrl}` });
  if (!data.payload) {
    return alert("Couldn't sign up");
  }
  if ("token" in data.payload) {
    window.localStorage.setItem("token", data.payload.token);
  }
};
```

(Б8) // Login.js(Fragment)

```
export const Login = () => {
  const isAuth = useSelector(selectIsAuth);
  const dispatch = useDispatch();
  const {
    register,
    handleSubmit,
    formState: { errors, isValid },
  } = useForm({
    defaultValues: {
      email: "test@test.com",
      password: "12345678",
    },
    mode: "onChange",
  });
      You, 3 weeks ago • gallery frontend done
  const onSubmit = async (values) => {
    const data = await dispatch(fetchLogin(values));
    console.log(data);
    if (!data.payload) {
      return alert("Couldn't login");
    }
    if ("token" in data.payload) {
      window.localStorage.setItem("token", data.payload.token);
    }
  };

  if (isAuth) {
    return <Navigate to="/" />;
```