Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем та програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: «Розробка веб-сайту каталогу книг з використанням технологій JS React та Google Authenticator»

Виконав(ла): студент(ка) 4 курсу, групи СП-41
спеціальності 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

(підпис) Климкович С. М. (прізвище та ініціали)

Керівник (підпис) Мудрик І. Я. (прізвище та ініціали)
Нормоконтроль (підпис) Стоянов Ю. М. (прізвище та ініціали)
Завідувач кафедри (підпис) Петрик М. Р. (прізвище та ініціали)
Рецензент (підпис) (прізвище та ініціали)

Тернопіль 2024

РЕФЕРАТ

Дана дипломна робота на тему «Розробка веб-сайту для каталогу книг з використанням React.js та Google Authenticator» розроблена для здобуття ступеня бакалавра та виконана студентом Тернопільського національного технічного університету імені Івана Пулюя, Факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра програмної інженерії, групи СП-41

Обсяг становить 55 сторінок, 25 рисунків, 3 таблиці

Ключові слова: книги, веб-сайт, сервер, клієнт, база даних, двофакторна автентифікація, безпека даних, React, MongoDB, Express, Node.js.

Головною метою цієї кваліфікаційної роботи є створення сучасного веб-сайту каталогу книг, який забезпечує користувачам зручний доступ до великого асортименту літературних видань. Додаток дозволяє користувачам реєструватися, входити в систему, шукати книги, переглядати деталі, додавати книги до кошика, залишати відгуки та оцінки. Особлива увага приділялася безпеці користувацьких даних шляхом впровадження двофакторної автентифікації.

Розроблено клієнтську та серверну частини використовуючи стек технологій MERN (MongoDB, Express, React, Node.js) та Firebase для зберігання медіа-даних.

ANNOTATION

This bachelor's thesis titled "Development of a Book Catalog Website Using React.js and Google Authenticator" has been prepared for the attainment of a bachelor's degree and is completed by a student of Ternopil Ivan Puluj National Technical University, Faculty of Computer Information Systems and Software Engineering, Department of Software Engineering, group SP-41.

The work comprises 55 pages, 25 figures, and 3 tables.

Keywords: books, website, server, client, database, two-factor authentication, data security, React, MongoDB, Express, Node.js.

The main objective of this qualification work is to create a modern book catalog website that provides users with convenient access to a large assortment of literary editions. The application allows users to register, log in, search for books, view details, add books to the cart, leave reviews, and rate books. Special attention was paid to the security of user data through the implementation of two-factor authentication.

Both the client-side and server-side parts were developed using the MERN stack (MongoDB, Express, React, Node.js) and Firebase for media data storage.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

MERN - Стек технологій для розробки веб-додатків, що включає MongoDB (база даних), Express (веб-фреймворк), React (бібліотека для створення інтерфейсу) і Node.js (середовище виконання JavaScript).

JWT - Стандарт для створення безпечних токенів, які можуть передаватися між сторонами для автентифікації та обміну інформацією.

2FA - Метод захисту, який вимагає два незалежних способи підтвердження особи для доступу до системи.

HTTP - Протокол передачі гіпертексту, який використовується для передачі даних у мережі Інтернет.

DB - Організована сукупність даних, яка забезпечує зберігання, обробку та пошук інформації.

JSON - Легкий формат обміну даними, який легко читається людьми і зручно використовується в програмах.

HTML - Основна мова для створення веб-сторінок та додатків, яка визначає структуру контенту.

CSS - Мова стилів, яка використовується для опису зовнішнього вигляду документів, написаних мовами розмітки, такими як HTML.

JS - Мова програмування, яка використовується для створення інтерактивних ефектів у веб-браузерах.

TOTP - Одноразовий пароль, який генерується на основі поточного часу і використовується для двофакторної автентифікації.

ЗМІСТ

ВСТУП

У сучасному цифровому світі інтернет відіграє ключову роль у забезпеченні доступу до інформації, товарів та послуг. Книги завжди були важливим джерелом знань, розваг та культурного збагачення, і з появою онлайн-книжкових магазинів доступ до літератури став ще зручнішим. Створення веб-сайту для каталогу книг з використанням сучасних технологій дозволяє забезпечити зручну та ефективну взаємодію користувачів з контентом. Це особливо актуально в умовах зростаючої популярності електронних та аудіокниг, а також необхідності забезпечення безпеки та надійності даних.

Основною метою цієї роботи є розробка веб-сайту для каталогу книг з використанням технологій MERN (MongoDB, Express, React, Node.js) та Firebase для зберігання медіа-даних. Сайт має забезпечити зручний доступ до великої кількості книг, надавати користувачам можливість швидко знаходити потрібні видання, переглядати їх деталі, залишати відгуки та оцінки. Важливою складовою є забезпечення високого рівня безпеки користувацьких даних через інтеграцію двофакторної автентифікації.

Для досягнення поставленої мети необхідно виконати ряд завдань. По-перше, потрібно провести аналіз вимог до системи, визначити основні функціональні та нефункціональні вимоги. По-друге, розробити архітектуру системи, включаючи клієнтську та серверну частини, та інтегрувати їх між собою. По-третє, створити зручний та інтуїтивно зрозумілий інтерфейс користувача, який забезпечить легкий доступ до всіх функцій сайту. Крім того, необхідно впровадити механізми забезпечення безпеки даних та протестувати систему для виявлення та усунення можливих помилок.

РОЗДІЛ 1. АНАЛІТИЧНА ЧАСТИНА.

1.1 Актуальність теми

Сьогодні інформаційні технології розвиваються дуже швидко і змінюють всі аспекти нашого життя, включаючи те, як ми отримуємо і поширюємо інформацію. Інтернет став частиною нашого щоденного життя, задовольняючи наші інформаційні та комунікаційні потреби. Особливо популярними стають книжкові інтернет-магазини, оскільки вони значно спрощують процес купівлі книг та надають доступ до широкого асортименту видань з будь-якої точки світу.

Книги завжди були важливим джерелом знань, культури та розваг для людей. З розвитком цифрових технологій доступ до книг став ще зручнішим і швидшим. Книжкові інтернет-магазини пропонують можливість купівлі не тільки паперових книг, але й електронних видань, аудіо-книг та інших форматів, що відповідають сучасним потребам споживачів. Це робить книжкові інтернет-магазини привабливими для різних категорій користувачів, включаючи студентів, науковців, професіоналів та звичайних читачів.

Крім того, книжкові інтернет-магазини дозволяють швидко знаходити і купувати рідкісні або спеціалізовані видання, які можуть бути недоступні у звичайних магазинах книг. Завдяки зручним інструментам пошуку та рекомендацій користувачі можуть легко знаходити книги, що відповідають їхнім інтересам та потребам. Інтеграція з соціальними мережами та можливість залишати рецензії також додає до популярності таких платформ, сприяючи формуванню спільнот читачів та обміну думками про книги.

1.2 Аналіз вимог

Створення веб-магазину книг починається з детального аналізу вимог. Це допомагає зрозуміти, які функції потрібні для правильної роботи системи. Аналіз вимог важливий, оскільки дозволяє визначити необхідні компоненти системи та їх взаємодію.

Функціонал веб-магазину включає конкретні можливості та функції. До них належать каталог книг, опис та рецензії, реєстрація та авторизація, кошик та підтримку користувачів. Каталог книг повинен бути добре структурованим, щоб користувачі могли легко знаходити потрібні видання за жанром, автором, роком видання чи форматом. Важливо забезпечити зручний пошук та фільтрацію.

Кожна книга повинна мати детальний опис, включаючи анотацію, інформацію про автора, ціну, рецензії та оцінки користувачів. Це допоможе покупцям зробити обґрунтований вибір. Система повинна підтримувати реєстрацію нових користувачів та авторизацію зареєстрованих. Інтеграція з системою безпеки Google Authenticator забезпечить надійну безпеку. Функціонал кошика дозволить зберігати обрані книги перед оформленням замовлення. Наявність системи підтримки користувачів, яка включає електронну пошту та телефонну лінію, забезпечить оперативне вирішення проблем та відповідь на запитання.

Системні вимоги – це сукупність характеристик, які повинна мати система для забезпечення її коректної, ефективної та безпечної роботи. Вони включають такі аспекти, як продуктивність, безпека, надійність, масштабованість та зручність використання. Висока швидкість завантаження сторінок та обробки запитів користувачів є ключовою для продуктивності системи. Надійність передбачає безперебійну роботу веб-магазину, мінімізацію простоїв та швидке відновлення після збоїв за допомогою резервного копіювання даних [1].

Захист персональних даних користувачів та забезпечення безпечної передачі інформації між клієнтом та сервером є дуже важливими. Впровадження SSL-сертифікатів та шифрування даних допоможе захистити конфіденційність інформації. Забезпечення сумісності веб-магазину з різними пристроями та браузерами, включаючи мобільні пристрої та планшети, гарантує коректне відображення та функціонування системи на всіх платформах.

Врахування всіх цих аспектів допоможе створити зручний та конкурентоспроможний продукт.

1.3 Опис постановки задачі

Вибір стеку MERN (MongoDB, Express, React, Node.js) та Firebase для зберігання медіа даних обумовлений необхідністю створення масштабованої, надійної та високопродуктивної системи, яка забезпечить зручний доступ користувачів до великого обсягу інформації про книги та можливість інтерактивної взаємодії з ресурсами сайту.

Основною метою проекту є розробка комплексного рішення для керування інформацією про книги, що включає можливість:

- додавання нових книг
- редагування існуючих записів
- видалення книг з каталогу
- перегляд детальної інформації про кожну книгу
- пошук товарів за фільтрами
- додавання книг у кошик
- система оцінювання

Крім того, система повинна забезпечити реєстрацію та автентифікацію користувачів, а також адміністрування прав доступу, що дозволить розмежувати функції звичайних користувачів та адміністраторів сайту.

1.4 Аналіз конкурентів

Одним з головних конкурентів нашого веб-магазину книг є Yakaboo.ua. Це один з найбільших онлайн-магазинів книг в Україні, який пропонує широкий асортимент літератури різних жанрів та мов. Yakaboo має значну кількість переваг, що робить його популярним серед книголюбів. Перш за все, це великий каталог книг, який включає як художню, так і наукову літературу, підручники, книги для дітей, а також рідкісні видання. Веб-сайт Yakaboo регулярно оновлюється, додаючи нові книги та серії, що робить його привабливим для постійних клієнтів. Окрім того, магазин активно проводить акції та надає знижки, що дозволяє клієнтам економити при купівлі. Важливою перевагою Yakaboo є також наявність різноманітних бонусних програм та програми лояльності, які заохочують клієнтів до повторних покупок. Зручність доставки також є вагомою перевагою Yakaboo. Вони пропонують різні способи доставки, включаючи кур'єрську доставку. Магазин також співпрацює з різними кур'єрськими службами, що дозволяє забезпечити швидку та надійну доставку [2].

Однак, не обходиться і без недоліків. Інтерфейс сайту, хоча і функціональний, іноді може бути не дуже інтуїтивно зрозумілим для нових користувачів. Деякі клієнти також скаржаться на проблеми з технічною підтримкою, зокрема на затримки у відповідях або невирішені питання, що можуть виникати під час оформлення замовлень або повернення товарів. Це може створювати певні незручності для

клієнтів, особливо у випадках, коли потрібне термінове вирішення проблеми. На рисунку 1.1 зображено головну сторінку магазину Yakaboo:

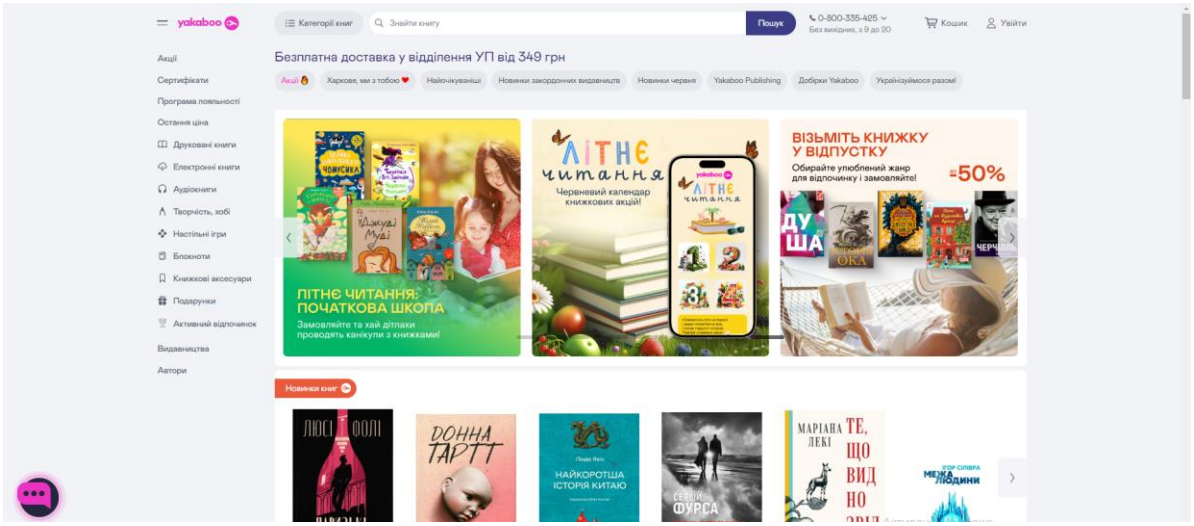


Рисунок 1.1 – Веб-магазин «Yakaboo»

Іншим важливим конкурентом є Книгарня «Є». Це мережа книжкових магазинів, яка активно розвиває свій онлайн-продаж. Книгарня «Є» має міцну репутацію завдяки своєму акценту на українську літературу. Вони пропонують широкий вибір книг українських авторів, включаючи класиків та сучасних письменників, а також переклади іноземної літератури українською мовою. Цей магазин відомий своєю підтримкою місцевих авторів та видавців, що є вагомим аргументом для патріотично налаштованих читачів. Крім того, Книгарня «Є» регулярно організовує літературні заходи, зустрічі з авторами та презентації нових видань, що робить їх платформу більш привабливою для поціновувачів книжок. Ці заходи дозволяють клієнтам безпосередньо спілкуватися з улюбленими авторами, отримувати автографи та брати участь у літературних дискусіях, що створює особливу атмосферу навколо бренду, також Книгарня «Є» пропонує самовивіз із пунктів видачі, що охоплюють більшість міст України [3]. На рисунку 1.2 зображено один з фізичних магазинів мережі Книгарня «Є»:

Рисунок 1.2 – Магазин Книгарня «Є» в місті Львів

Асортимент літератури на інших мовах може бути дещо обмеженим у порівнянні з конкурентами, що може зменшити інтерес з боку клієнтів, які читають іншими мовами. Крім того, ціни на деякі видання можуть бути досить високими, що може відлякати частину потенційних покупців. Це може бути зумовлено тим, що Книгарня «Є» співпрацює з багатьма малими видавництвами, які не завжди можуть дозволити собі великі тиражі, а отже, і низькі ціни. Водночас, цей недолік компенсується високою якістю видань та унікальними пропозиціями, які важко знайти в інших магазинах. Незважаючи на це, Книгарня «Є» залишається вагомим гравцем на ринку завдяки своїй спеціалізації та активній культурній діяльності.

Третім конкурентом можна назвати Book24.ua. Це сучасний онлайн-магазин книг, який пропонує широкий вибір літератури різних жанрів. Перевагами Book24 є швидка доставка, зручний і зрозумілий інтерфейс сайту, а також можливість замовлення електронних книг та аудіокниг. Веб-сайт Book24 добре структурований,

що дозволяє легко знаходити потрібні книги за жанром, автором або іншими критеріями. Магазин також пропонує добре розроблену систему лояльності для постійних клієнтів, яка включає бонусні програми та спеціальні пропозиції, що сприяє утриманню аудиторії. Завдяки сучасному дизайну та високій швидкості роботи сайту, користувачі можуть зручно та швидко оформляти замовлення, що є значним плюсом.

Book24 має обмежену кількість фізичних магазинів, що може бути незручним для тих, хто віддає перевагу традиційній купівлі книг. Крім того, іноді виникають проблеми з доступністю певних видань, особливо нових чи ексклюзивних книг, які не завжди є в наявності. Це може створювати певні незручності для клієнтів, які шукають конкретні новинки або рідкісні книги. Незважаючи на це, Book24 залишається популярним завдяки своїм сучасним підходам та широкому вибору літератури. Вони також активно працюють над розширенням асортименту та покращенням сервісу, що дозволяє їм утримувати високий рівень задоволеності клієнтів [4].

Таким чином, кожен з цих конкурентів має свої унікальні переваги та недоліки. Yakaboo.ua відомий своїм широким асортиментом і ексклюзивними виданнями, що робить його привабливим для великої аудиторії. Вони також мають добре розвинену систему доставки та регулярні акції, які залучають багато клієнтів. Книгарня "Є" акцентує увагу на українській літературі та культурних заходах, що залучає патріотично налаштованих читачів та створює особливу атмосферу навколо бренду. Book24.ua пропонує сучасні рішення для купівлі книг онлайн, включаючи електронні та аудіокниги, що робить його привабливим для технологічно обізнаних користувачів. Вони також мають зручний інтерфейс і швидку доставку, що є важливим фактором для багатьох покупців.

1.5 Google Authenticator

Google Authenticator – це мобільний додаток, який забезпечує додатковий рівень безпеки для ваших облікових записів. Він працює на основі двофакторної автентифікації (2FA), що означає, що для входу в обліковий запис необхідно ввести не тільки пароль, але і одноразовий код, згенерований додатком. Коли ви налаштовуєте Google Authenticator для свого облікового запису, система генерує секретний ключ, який ви вводите або скануєте за допомогою QR-коду в додатку. Цей секретний ключ є унікальним для вашого облікового запису і використовується для генерації одноразових кодів.

Після налаштування Google Authenticator генерує новий одноразовий код кожні 30 секунд. Коли ви намагаєтеся увійти в свій обліковий запис, окрім пароля, система запитає вас про одноразовий код з додатка. Цей код синхронізований з конкретним секретним ключем і часом, що забезпечує його унікальність та безпеку [5, 6].

На рисунку 1.3 зображено додаток з згенерованими кодами:

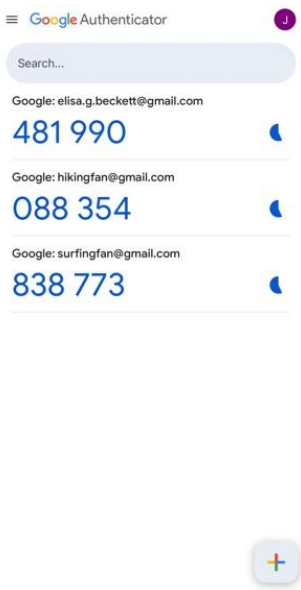


Рисунок 1.3 – Додаток Google Authenticator

Основна перевага використання Google Authenticator полягає в тому, що навіть якщо ваш пароль буде викрадено, зловмисники не зможуть отримати доступ до вашого облікового запису без одноразового коду. Це значно підвищує рівень захисту ваших даних. Google Authenticator працює автономно, без необхідності підключення до Інтернету, що робить його дуже зручним для використання в будь-яких умовах. Він підтримує множинні облікові записи, тому ви можете використовувати його для захисту різних сервісів і додатків.

Однак варто пам'ятати, що якщо ви втратите свій телефон або видалите додаток, доступ до облікових записів, захищених Google Authenticator, може бути втрачено. Тому важливо зберігати резервні копії секретних ключів або використовувати резервні коди, надані різними сервісами під час налаштування двофакторної автентифікації [5, 6].

Таким чином, Google Authenticator є простим, зручним і надійним засобом підвищення безпеки ваших облікових записів, забезпечуючи додатковий рівень захисту за допомогою одноразових кодів, які генеруються на вашому мобільному пристрої. На рисунку 1.4 зображено логотип програми Google Authenticator:

Рисунок 1.4 – Логотип Google Authenticator

РОЗДІЛ 2. ПРОЕКТУВАННЯ, КОНСТРУЮВАННЯ НА ТЕСТУВАННЯ ВЕБ-ДОДАТКУ

2.1 Загальна архітектура веб-сайту

Основною архітектурою проекту є трирівнева архітектура розподілу додатку на клієнтську, серверну та базу даних частини, що дозволяє зручно та ефективно працювати з кожним компонентом системи. Трирівнева архітектура зображена на рисунку 2.1 нижче:

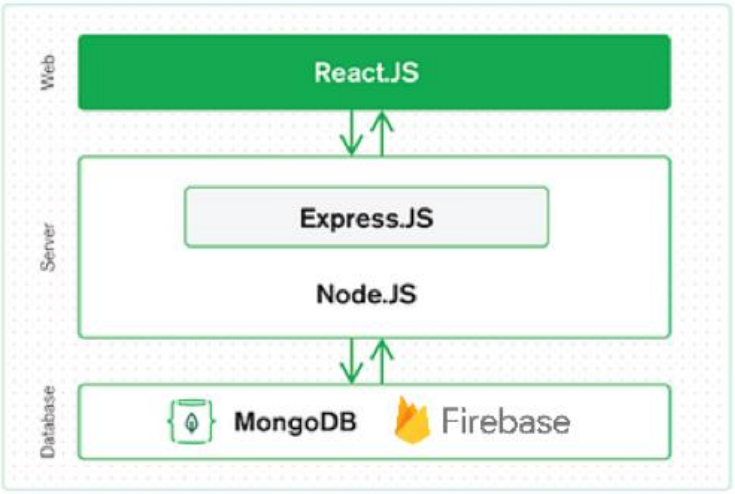


Рисунок 2.1 – Трирівнева архітектура MERN + Firebase

Презентаційний рівень (front-end) відповідає за взаємодію з користувачем і включає все, що бачить і з чим взаємодіє користувач. Це інтерфейс користувача, форми введення даних, таблиці, кнопки та інші елементи. У нашому випадку, для реалізації презентаційного рівня використовується бібліотека React, яка дозволяє створювати динамічні та інтерактивні інтерфейси користувача. React як бібліотека для побудови інтерфейсів користувача забезпечує створення динамічних та інтерактивних веб-сторінок [7].

Серверний рівень (back-end) відповідає за обробку запитів від клієнта, виконання необхідних операцій з даними та повернення відповідей назад клієнту. Для реалізації цього рівня використовується Node.js та Express. Node.js є серверним середовищем виконання, яке дозволяє запускати JavaScript на сервері. Використання Node.js дозволяє використовувати JavaScript поза браузером та мати єдину мову програмування для обох фронт-енд та бек-енд частин додатку, що значно спрощує процес розробки та підтримки [8]. Express, у свою чергу, є мінімалістичним веб-фреймворком для Node.js, який забезпечує потужні інструменти для побудови веб-додатків та API. Він дозволяє легко створювати маршрути для обробки запитів HTTP, інтегрувати middleware для обробки запитів та відповідей та багато іншого [9].

Рівень даних відповідає за зберігання і управління даними додатку. Для зберігання даних використовується MongoDB, яка є документо-орієнтованою NoSQL базою даних, та Firebase для зберігання медіа даних. MongoDB, як документно-орієнтована NoSQL база даних, забезпечує гнучке та ефективне зберігання даних [10]. Використання Firebase дозволяє надійно зберігати та швидко отримувати доступ до медіа файлів, таких як зображення обкладинок книг та фотографії користувачів [11].

Отже, для розробки системи було обрано клієнт-серверну архітектуру, де клієнт (веб-браузер) взаємодіє з сервером, який надає необхідні дані. Така архітектура забезпечує чіткий поділ обов'язків між клієнтською та серверною частинами, що підвищує масштабованість і гнучкість системи [12]. На клієнтській стороні використовується бібліотека React. Серверна частина реалізована за допомогою Node.js та Express.js. Для зберігання та обробки даних використовується MongoDB. Окрім MongoDB, для зберігання медіа файлів використовується Firebase.

2.2 Актори та варіанти використання

Провівши детальний аналіз предметної області та структури проекту, ми визначили основні ролі (актори) та їх варіанти використання в рамках системи. На основі визначених акторів та варіантів використання була створена діаграма варіантів використання, яка ілюструє взаємодію між акторами та системою.

Діаграма демонструє основні сценарії використання системи, включаючи дії як незареєстрованих, так і авторизованих користувачів. Ця структура допомагає зрозуміти, які функції доступні різним типам користувачів та як вони взаємодіють із системою. Діаграму використання зображено на рисунку 2.2:

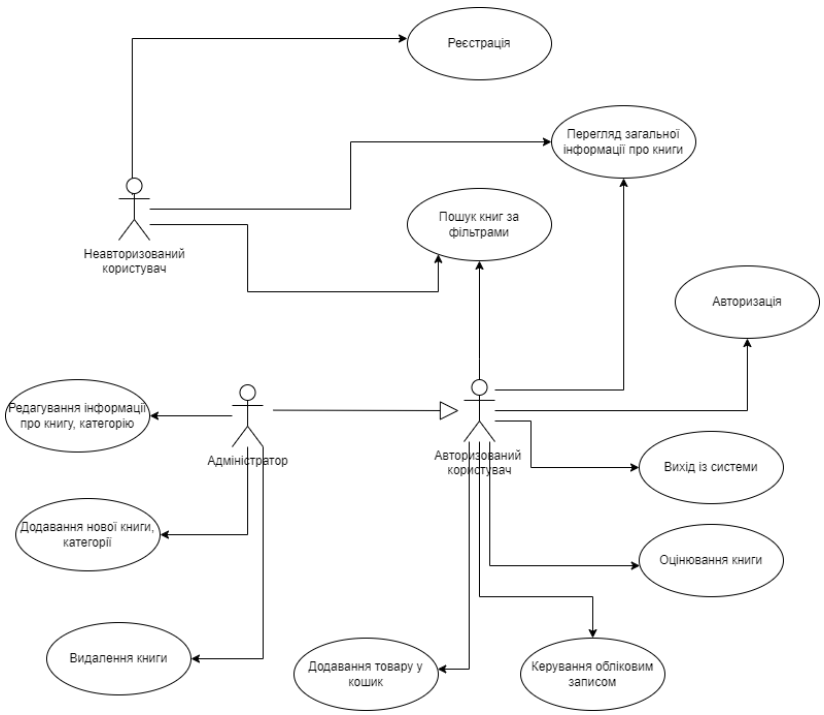


Рисунок 2.2 – Діаграма варіантів використання

З отриманого рисунку маємо результат на існування двох основних акторів:

Незареєстрований користувач: Це користувач, який ще не створив обліковий запис у системі або не увійшов у свій аккаунт. Йому доступний обмежений

функціонал, що включає перегляд загальної інформації про книги, пошук книг за фільтрацією та саму реєстрацію. Це доволі маленький обсяг дій, збільшення якого можливе тільки при реєстрації користувача.

Авторизований користувач: Це користувач, який успішно зареєструвався і увійшов у систему. Цей актор має розширений доступ до функціоналу, включаючи керування обліковим записом, додавання товару у кошик, оцінювання товару.

Адміністратор: Це користувач з правами адміністратора, включаючи в себе можливості звичайного авторизованого користувача, адміністратор додаткові права, такі як керування категоріями/книгами, додавання та видалення книг, редагування поточних категорій/книг

Серед цих акторів лише авторизований користувач та адміністратор можуть виступати як головний актор, адже незареєстрованому користувачеві не будуть доступні розширені можливості системи.

Для кращого розуміння створеної діаграми використання, розглянемо основні варіанти використання системи для кожного актора.

- Перегляд інформації про книги: Користувач може переглядати загальну інформацію про книги, включаючи оцінки, опис тощо.
- Пошук книг за фільтрами: Користувач може виконати пошук конкретних книг за відповідними фільтрами.
- Реєстрація: Користувач може створити новий обліковий запис, ввівши необхідні дані тим самим зареєструвавши себе
- Авторизація: Користувач може увійти у свій обліковий запис за допомогою введення логіну та паролю.
- Додавання нової книги: Адміністратор може додавати нові книги до каталогу.
- Додавання нової категорії: Адміністратор може додавати нові книги до каталогу.

- Редагування інформації про книгу: Адміністратор може змінювати інформацію про книги, такі як назва, автор, жанр, опис тощо.
- Видалення книги: Адміністратор може видаляти книги з каталогу.
- Видалення категорії: Адміністратор може видаляти категорію книг.
- Керування обліковим записом: Користувач може змінювати налаштування свого облікового запису, включаючи зміну паролю та інших персональних даних.
- Вихід із системи: Авторизований користувач може вийти з системи, завершивши сесію.
- Оцінювання книги: Авторизований користувач може додавати оцінку з відгуком до відповідної книги.
- Додавання товару у кошик: Авторизований користувач може додати товар у кошик.

2.3 Архітектура бази даних

База даних проекту складається з кількох взаємопов’язаних колекцій, які створюють цілісну інформаційну структуру для зберігання та отримання доступу до даних. Кожна колекція має свою унікальну структуру, що дозволяє зберігати специфічну інформацію, необхідну для функціонування системи.

Головною колекцією є "Users", де зберігається основна інформація про користувачів, включаючи їх особисті дані та вподобані продукти. Колекція "Products" містить детальну інформацію про книги, включаючи їх опис, категорії, рік випуску та рейтинг. Колекція "Categories" використовується для організації книг за жанрами.

2.4 Колекції бази даних

Для кожної колекції створено відповідні структури на мові JavaScript, що відображають таблиці бази даних в об'єктній формі і використовуються для доступу до даних. База даних реалізована за допомогою MongoDB, яка забезпечує гнучке та ефективне зберігання даних.

Колекція "Users" містить основну інформацію про користувачів. Наведемо її структуру в таблиці 2.1:

Таблиця 2.1. Структура колекції “Users”

Ім’я поля	Тип поля	Опис поля
_id	Mongo.ObjectId	Унікальний ідентифікатор користувача
fullName	String	Повне ім'я користувача
email	String	Адреса електронної пошти
password	String	Хешований пароль для безпечного зберігання
isAdmin	Boolean	Вказує, чи є користувач адміністратором
likedProducts	Array<ObjectId>	Масив ідентифікаторів продуктів, які сподобалися
totpSecret	String	Секретний ключ для двофакторної автентифікації
is2FAEnabled	Boolean	Вказує, чи ввімкнено двофакторну автентифікацію
createdAt	Date	Дата створення облікового запису

Продовження таблиці 2.1

updatedAt	Date	Дата останнього оновлення облікового запису
image	String	URL зображення профілю користувача

Колекція "Products" містить інформацію про книги, представлені на веб-сайті. Наведемо її структуру в таблиці 2.2:

Таблиця 2.2. Структура колекції “Products”

Ім’я поля	Тип поля	Опис поля
_id	Mongo.ObjectId	Унікальний ідентифікатор користувача
userId	Mongo.ObjectId	Ідентифікатор користувача, який додав продукт
name	String	Назва книги
desc	String	Детальний опис книги
titleImage	String	URL зображення обкладинки
image	String	URL додаткових зображень
category	String	Категорія книги
language	String	Мова книги
year	Number	Рік випуску книги
time	Number	Час, необхідний для прочитання книги
rate	Number	Рейтинг книги

Продовження таблиці 2.2

numberOfReviews	Number	Кількість відгуків на книгу
casts	Array<Object>	Список акторів або інших учасників, пов'язаних з книгою
reviews	Array<Object>	Список відгуків на книгу
createdAt	Date	Дата створення запису про книгу
updatedAt	Date	Дата останнього оновлення запису про книгу

Колекція "Categories" зберігає інформацію про жанри книг. Наведемо її структуру цієї колекції в таблиці 2.3:

Таблиця 2.3. Структура колекції "Categories"

Ім’я поля	Тип поля	Опис поля
_id	Mongo.ObjectId	Унікальний ідентифікатор категорії
title	String	Назва категорії
createdAt	Date	Дата створення запису про категорію
updatedAt	Date	Дата останнього оновлення запису про категорію

2.5 Проектування бази даних

База даних відіграє ключову роль у зберіганні та обробці користувацьких даних та інформації про книги в нашій системі. Як зазначено вище, для зберігання даних була обрана документо-орієнтована база даних MongoDB та база даних Firebase для зберігання медіа-даних (обкладинки книг, фото користувачів). Вибір цих баз даних був обумовлений її високою швидкодією, гнучкістю у зберіганні даних та здатністю масштабуватися в залежності від потреб проекту. Перевагою MongoDB над реляційними базами даних є можливість уникнення дороговартісних операцій з'єднання завдяки зберіганню всіх необхідних даних в межах одного документу.

Після проведення аналізу вимог інформаційної системи, було розроблено діаграму баз даних, що зображує основні колекції та взаємозв'язки між ними. Основними колекціями в нашій базі даних є "Users", "Products" та "Categories". Ці колекції зберігають відповідні дані про користувачів, продукти (книги) та категорії книг.

На наведеній нижче діаграмі на рисунку 2.3 зображені основні колекції бази даних та їх взаємозв'язки.

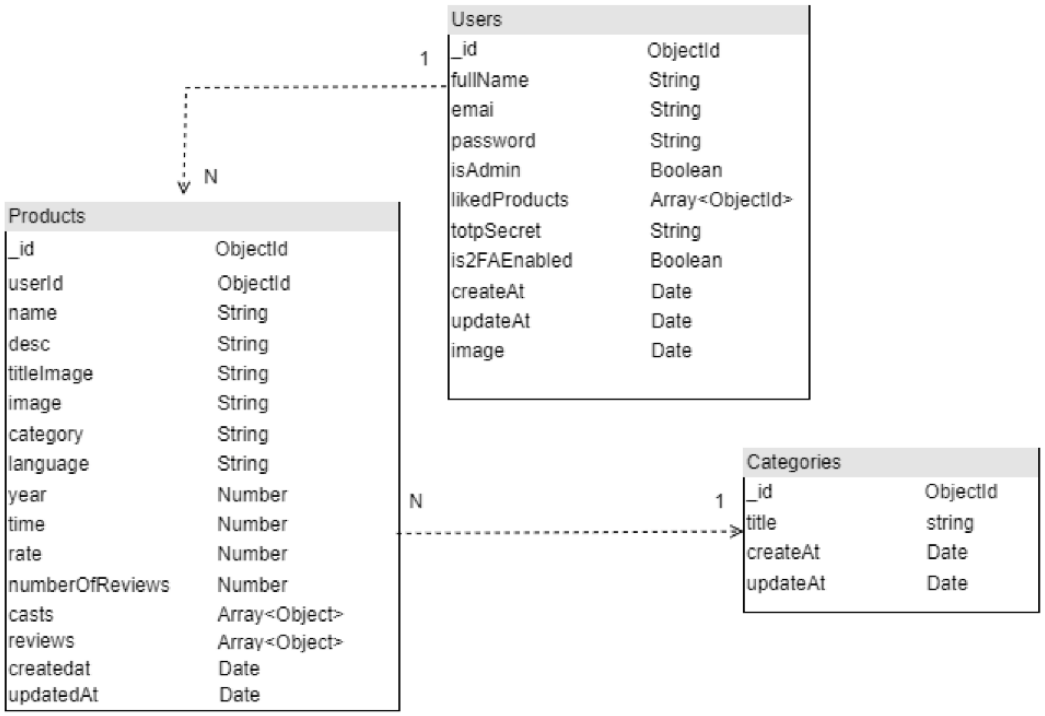


Рисунок 2.3 – Діаграма баз даних

Розглянемо детальніше кожну з колекцій та їх взаємозв'язки.

- Лінія між "Users" та "Products" з позначками 1 і N вказує на зв'язок один-до-багатьох: один користувач може додати багато продуктів (книг), але кожен продукт належить лише одному користувачу.
- Лінія між "Products" та "Categories" з позначками N і 1 вказує на зв'язок багато-до-одного: багато продуктів (книг) можуть належати одній категорії, але кожна категорія може містити багато продуктів.

Колекція "Users" зберігає інформацію про зареєстрованих користувачів. Кожен користувач має унікальний ідентифікатор _id, що дозволяє однозначно ідентифікувати його в системі. Користувачі можуть додавати книги до своїх вподобань, що відображається через зв'язок з колекцією "Products". Також кожен користувач має поля для зберігання особистих даних, таких як повне ім'я, електронна пошта, хешований пароль, статус адміністратора, тощо.

Колекція "Products" зберігає інформацію про книги, які представлені на веб-сайті. Кожен продукт (книга) має унікальний ідентифікатор _id, а також поля для зберігання назви, опису, категорії, мови, року випуску, рейтингу, кількості відгуків тощо. Продукти можуть бути пов'язані з користувачами через поле userId, яке вказує на ідентифікатор користувача, що додав книгу. Крім того, кожен продукт може мати кілька відгуків, що зберігаються у вигляді масиву об'єктів.

Колекція "Categories" зберігає інформацію про різні жанри книг. Кожна категорія має унікальний ідентифікатор _id та назву категорії. Ці категорії використовуються для організації книг у каталозі та полегшення їх пошуку користувачами.

2.6 Розробка серверної частини

Сервер забезпечує управління даними, а також їх збереження, обробку та передачу між клієнтською частиною та базою даних [1]. У цьому розділі буде детально описано структуру серверної частини, її компоненти та їх взаємодію, а також наведено приклади коду для кращого розуміння реалізації.

Серверна частина проекту складається з конфігураційніих файлів, контролерів. моделей, роутів, мідлверів (проміжні обробники), конфігурації бази даних

Кожен з цих компонентів має свою функціональність і відповідає за певні задачі у загальному процесі обробки запитів. Структура серверної частини наведена на рисунку 2.4:

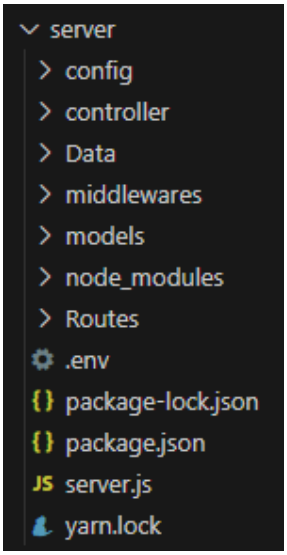


Рисунок 2.4 – Структура серверу

Як бачимо з рисунку, серверна частина складається з кількох ключових компонентів: конфігураційних файлів, моделей, контролерів, роутів та мідлверів.

Файл package.json визначає списки пакетів, які необхідні для роботи проекту, це дозволяє автоматично встановлювати всі необхідні пакети за допомогою команд npm install або yarn install.

Файл .env використовується для зберігання конфіденційних налаштувань та змінних середовища, які потрібні для роботи додатку. Це включає API-ключі, налаштування бази даних, порти для сервера та інші конфігураційні параметри, які не бажано зберігати безпосередньо в коді.

Папка controllers зберігає контролери, які відповідають за правильну обробку запитів, що надходять з клієнтської частини. Контролери містять логіку додатку, яка включає створення, читання, оновлення та видалення даних. Вони взаємодіють з моделями для виконання операцій над даними і формують відповіді, що повертаються клієнту. Для прикладу на рисунку 2.5 наведено структура файлу UploadFile.js:

server > controller > JS UploadFile.js > default

```
import express from 'express'
import multer from 'multer'
import path from 'path'
import { v4 as uuidv4 } from 'uuid';
import storage from '../config/firebaseStorage.js'

const Uploadrouter = express.Router();
const upload = multer({
  storage: multer.memoryStorage(),
})

Uploadrouter.post("/", upload.single("file"), async (req, res) => {
  try {
    //get file from request
    const file = req.file;
    //create new filename
    if (file) {
      const fileName = `${uuidv4()}${path.extname(file.originalname)}`;

      const blob = storage.file(fileName);
      const blobStream = blob.createWriteStream({
        resumable: false,
        metadata: {
          contentType: file.mimetype,
        },
      });
      // if error
      blobStream.on("error", (error) => {
        res.status(400).json({ message: error.message });
      })
      //if success
      blobStream.on("finish", () => {
        //get the public URL
        const publicUrl = `https://firebasestorage.googleapis.com/v0/b/${storage.name}/o/${fileName}?alt=media`
        //return the file name and its public URL
        res.status(200).json(publicUrl);
      })
      blobStream.end(file.buffer);
      //when there is no file
    } else {
      res.status(400).json({ message: "Please upload a file" })
    }
  } catch (error) {
    res.status(400).json({ message: error.message });
  }
})

export default Uploadrouter
```

Рисунок 2.5 – Обробка завантаження файлів на сервер

Опис алгоритму:

1. Отримання запиту: Клієнт надсилає запит з файлом на сервер.
2. Перевірка файлу: Сервер перевіряє, чи відповідає файл встановленим вимогам.
3. Збереження файлу: Якщо файл відповідає вимогам, сервер зберігає його у визначеному місці.
4. Відповідь клієнту: Сервер відправляє клієнту відповідь з підтвердженням завантаження або повідомленням про помилку.

Папка models зберігає моделі даних, що визначають структуру документів у базі даних. Моделі використовують Mongoose для визначення схем, валідації та взаємодії з MongoDB. Вони забезпечують узгодженість та цілісність даних, а також надають методи для створення, читання, оновлення та видалення записів у базі даних.

Папка routes містить файли маршрутизації, які визначають шляхи (URL) та методи HTTP-запитів (GET, POST, PUT, DELETE). Роутинг перенаправляє запити до відповідних контролерів, забезпечуючи організацію та структурованість обробки запитів. Він дозволяє легко додавати нові маршрути та розширювати функціональність додатку.

Папка middlewares зберігає мідлвери, які виконують проміжну обробку HTTP-запитів перед передачею їх до контролерів або після їх обробки. Мідлвери можуть виконувати задачі автентифікації, авторизації, логування, обробки помилок та інших додаткових операцій. Вони забезпечують додатковий рівень контролю та безпеки під час обробки запитів.

Папка config містить файли конфігурації для підключення до бази даних та інших налаштувань додатку. Наприклад, файл db.js відповідає за підключення до MongoDB за допомогою Mongoose. Конфігураційні файли дозволяють централізовано управляти налаштуваннями додатку та забезпечують їх зручне налаштування для різних середовищ (розробка, тестування, продакшн).

Папка data зберігає початкові або фіктивні дані для наповнення бази даних під час розробки та тестування додатку. Ці файли містять зразки даних, такі як списки категорій, продуктів чи користувачів, які можуть бути використані для початкового налаштування додатку. Використання таких даних дозволяє розробникам швидко створити робоче середовище для тестування функціональності додатку без необхідності вручну вводити дані в базу.

Серверна частина веб-додатку забезпечує обробку запитів клієнтів, управління даними та їх безпечне збереження в базі даних. Коли клієнт надсилає HTTP-запит, роутинг визначає, який контролер повинен обробити цей запит. Контролер отримує

дані з запиту, взаємодіє з моделями для доступу до бази даних. Моделі, реалізовані за допомогою Mongoose, забезпечують структуру та валідацію даних, а також зберігають їх у MongoDB. Мідлвери виконують додаткові задачі, такі як перевірка автентичності користувача за допомогою токенів та обробка помилок, що виникають під час виконання запитів. Після обробки контролером, сервер формує відповідь і надсилає її клієнту, що дозволяє клієнтській частині отримати необхідні дані чи підтвердження виконання операції. Усе це забезпечує узгоджену роботу додатку, зберігаючи дані цілісними та захищеними.

2.7 Розробка клієнтської частини

Клієнтська частина веб-додатку відповідає за взаємодію з користувачами, відображення інтерфейсу та обробку введених даних. Вона використовує сучасні технології, такі як React і Tailwind CSS, для створення динамічних і інтерактивних веб-сторінок. Клієнтська частина організована у кілька папок, кожна з яких має свою специфічну роль. Основна структура зображена на рисунку 2.6

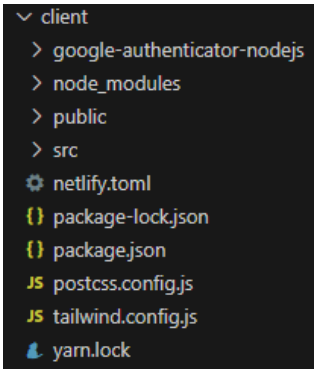


Рисунок 2.6 – Клієнт-структура

Папка public містить статичні файли, які не змінюються під час роботи додатку. Це файли, які браузер завантажує одразу після відкриття сторінки. Основним файлом у цій папці є index.html. Він є точкою входу для всього додатку. Цей HTML-файл містить базову структуру сторінки та елемент , куди буде вставлений весь React-додаток. У цьому файлі також є метатеги для пошукових систем, посилання на файли стилів та іконки. Окрім index.html, у папці public зберігаються інші статичні ресурси, такі як зображення, файли маніфесту (manifest.json) та файли для пошукових систем (robots.txt). Наприклад, логотипи, фонові зображення та іконки.

Папка src містить весь вихідний код клієнтської частини додатку. Тут знаходяться компоненти, сторінки, стилі та інші необхідні файли для роботи додатку. Основний файл у цій папці - App.js, який є головним компонентом додатку. Він відповідає за структуру додатку та навігацію між різними сторінками. App.js визначає, які компоненти будуть показані для кожного маршруту, використовуючи бібліотеку react-router-dom. Файл index.js є точкою входу JavaScript. Він відповідає за вставку компонента App у кореневий елемент HTML. У цьому файлі також підключаються глобальні стилі та налаштовуються контексти (наприклад, контекст автентифікації).

2.7.1 Компоненти клієнт частини

Компоненти системи є основними будівельними блоками клієнтської частини веб-додатку. Вони відповідають за різні аспекти взаємодії з користувачами, відображення інтерфейсу та обробку даних. У папці src зберігаються основні компоненти, які забезпечують роботу додатку. Кожна папка та файл має свою специфічну роль і функціональність. Основні компоненти систем изображено на рисунку 2.7.1:

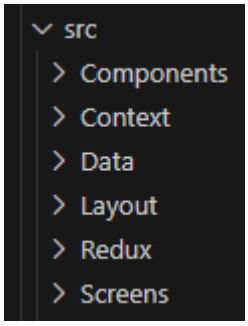


Рисунок 2.7.1 – Основні компоненти системи

У структуру додатку входять різноманітні папки, які містять різні компоненти та файли для ефективного управління та організації коду. Папка "Components" включає багаторазові елементи інтерфейсу, такі як кнопки, форми, модальні вікна та інші подібні компоненти, які можуть бути використані в різних частинах додатку. Папка "Context" містить файли, призначені для управління глобальним станом додатку за допомогою React Context API, що дозволяє зручно передавати дані між компонентами без необхідності пропускати їх через кожен рівень компонентного дерева. У папці "Layouts" зберігаються компоненти, які визначають загальний макет сторінок, такі як навігаційні меню, заголовки та футери, забезпечуючи єдиний стиль і структуру для всіх сторінок додатку. Папка "Redux" містить файли, що використовуються для управління станом додатку за допомогою Redux, надаючи потужний інструмент для керування складними станами додатку і полегшуючи роботу з даними. Папка "Screens" включає компоненти, які представляють цілі сторінки додатку, такі як сторінки входу, реєстрації, профілю користувача та інші основні розділи додатку. Нарешті, папка "Styles" містить файли стилів, які використовуються для оформлення компонентів та сторінок, забезпечуючи узгоджений і привабливий вигляд інтерфейсу додатку.

2.8 Загальна робота клієнт та серверної частин

Для прикладу опису роботи файлів при розробці, візьмемо процес входу користувача в систему та опишемо кожен пункт:

1. Користувач вводить облікові дані у формі входу (Login.js) та натискає кнопку входу.
2. Функція loginUser у файлі api.js надсилає запит з обліковими даними на сервер.
3. Серверний маршрут /api/auth/login обробляє запит за допомогою контролера authController.js.
4. Контролер перевіряє дані користувача у базі даних за допомогою моделі userModel.js.
5. Сервер генерує JWT-токен для автентифікованого користувача та надсилає його назад клієнту.
6. Клієнтська частина зберігає токен у локальному сховищі (localStorage).
7. Клієнт використовує токен для автентифікації при наступних запитах до захищених маршрутів.
8. Сервер перевіряє токен за допомогою проміжної функції authMiddleware.js перед обробкою запиту.

Розробка клієнтської частини включає створення динамічного та інтерактивного інтерфейсу користувача за допомогою React. Основна структура клієнтської частини організована у кілька папок, кожна з яких відповідає за певний аспект роботи додатку. У папці Components створюються багаторазові елементи інтерфейсу, такі як кнопки, поля введення та модальні вікна. Ці компоненти використовуються на різних сторінках для забезпечення єдиного стилю та функціональності.

Папка Screens містить компоненти, які представляють цілі сторінки додатку, такі як сторінки входу, реєстрації, профілю користувача та інші. Кожна сторінка включає кілька компонентів і має свою логіку для обробки даних та взаємодії з сервером. Наприклад, сторінка Login.js включає форму для введення облікових даних користувача і використовує функцію loginUser з файлу api.js для надсилання запиту на сервер.

Для управління глобальним станом додатку використовуються React Context API та Redux. Папка Context містить файли для управління станом автентифікації та тем оформлення додатку, а папка Redux включає файли для налаштування store, ред'юсерів та дій. Це забезпечує централізоване управління станом додатку та полегшує його підтримку.

Серверна частина відповідає за обробку запитів клієнтів, перевірку даних, управління базою даних та забезпечення безпеки додатку. Основні компоненти серверної частини включають контролери, моделі, маршрути та проміжні функції.

Контролери, розміщені в папці controllers, обробляють логіку запитів. Наприклад, контролер authController.js відповідає за логіку автентифікації користувачів, включаючи вхід, реєстрацію та перевірку токенів. Моделі, розміщені в папці models, визначають структуру даних і методи для взаємодії з базою даних. Наприклад, модель userModel.js визначає схему користувача та методи для роботи з колекцією користувачів у базі даних.

Маршрути, розміщені в папці routes, визначають кінцеві точки API, через які клієнтська частина взаємодіє з сервером. Наприклад, маршрут authRoutes.js визначає маршрути для автентифікації, такі як вхід, реєстрація та перевірка токена. Проміжні функції, розміщені в папці middleware, виконують додаткові задачі, такі як перевірка автентичності користувача за допомогою токенів та обробка помилок, що виникають під час виконання запитів.

2.9 Розробка Google Authenticator

Для забезпечення двофакторної автентифікації (2FA) за допомогою Google Authenticator на веб-сайті було додано кілька важливих функцій та файлів. Основні кроки цього процесу включають генерацію секретного ключа та QR-коду, збереження секретного ключа у базі даних, перевірку одноразового пароля під час входу користувача, а також використання JWT-токена для захищених запитів. Ця реалізація забезпечує додатковий рівень безпеки для користувачів, дозволяючи використовувати двофакторну автентифікацію.

Першим кроком було додавання маршруту для увімкнення двофакторної автентифікації. У файлі routes/authRoutes.js додано маршрут /enable-2fa, який викликає функцію enable2FA з файлу controllers/authController.js. Далі, для генерації секретного ключа та QR-коду у функції generateTOTPSecret, що знаходиться у файлі controllers/UserController.js, використовувалася бібліотека speakeasy для генерування секретного ключа TOTP та бібліотека qrcode для створення QR-коду.

Секретний ключ зберігається у полі totpSecret моделі користувача у файлі models/UserModel.js. Наступним кроком було додавання маршруту для входу користувача. У файлі routes/userRoutes.js додано маршрут /login, який викликає функцію loginUser з файлу controllers/UserController.js. Для перевірки одноразового пароля під час входу у функції loginUser використовується бібліотека speakeasy. Вона перевіряє введений користувачем одноразовий пароль на основі збереженого секретного ключа.

Якщо одноразовий пароль правильний, сервер генерує JWT-токен за допомогою бібліотеки jsonwebtoken і надсилає його назад клієнту. Клієнтська частина зберігає отриманий JWT-токен у локальному сховищі браузера (localStorage). Токен додається до заголовків запиту при зверненні до захищених маршрутів, що дозволяє серверу перевіряти автентичність запитів [5].

На рисунку 2.8 зображено діаграму роботи двофакторної автентифікації, яка ілюструє весь процес, включаючи генерацію секретного ключа, збереження його у базі даних, перевірку одноразового пароля, генерацію та збереження JWT-токена, а також використання токена для захищених запитів. Така реалізація підвищує безпеку користувачів, забезпечуючи додатковий рівень захисту їхніх облікових записів.

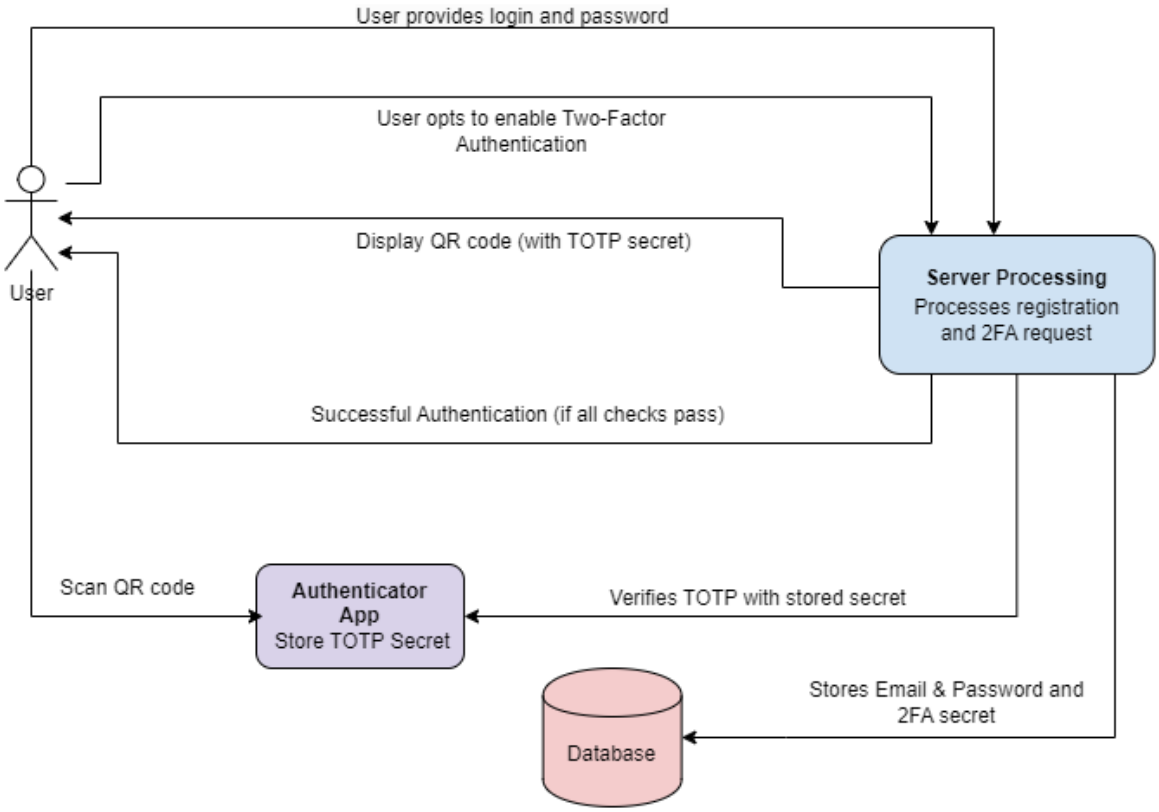


Рисунок 2.8 – Робота Google Authenticator

З діаграми можемо побачити роботу Google Authenticator на нашому веб-сайті. Нижче описано роботу по пунктам.

1. Користувач обирає увімкнення двофакторної автентифікації у своєму профілі. Сервер генерує секретний ключ TOTP та QR-код, який відображається користувачу для сканування за допомогою Google Authenticator.

2. Збереження секретного ключа у базі даних: Після сканування QR-коду користувачем, секретний ключ зберігається у полі totpSecret в моделі користувача у базі даних.
3. Процес входу користувача: Користувач вводить свої облікові дані (логін і пароль) та одноразовий пароль, згенерований Google Authenticator, у форму на сторінці входу.
4. Перевірка даних на сервері: Сервер отримує запит на вхід, перевіряє облікові дані користувача у базі даних та порівнює збережений секретний ключ з одноразовим паролем, наданим користувачем.
5. Генерація JWT-токена: Якщо одноразовий пароль правильний, сервер генерує JWT-токен і надсилає його назад клієнту.
6. Збереження токена на клієнті та використання для подальших запитів: Отриманий токен зберігається на клієнті, що дозволяє використовувати його для автентифікації при наступних запитах на сервер.

2.10 Тестування

У цьому підрозділі ми розглянемо різні аспекти тестування нашого веб-додатку, що включає як функціональне, так і нефункціональне тестування. Також ми детально опишемо процес тестування різних компонентів нашого додатку. Зокрема, ми зосередимося на тестуванні авторизації користувачів, перевірці роботи головної сторінки та інших важливих функцій.

Коли користувач вперше потрапляє на сайт, його зустрічає головна сторінка, яка забезпечує перше враження про веб-додаток.

На головній сторінці веб-сайту користувач зустрічається з привітним та інтуїтивно зрозумілим інтерфейсом, який надає доступ до основних функцій сайту.

Головна сторінка включає навігаційне меню, яке дозволяє користувачам швидко переходити між різними розділами сайту, такими як каталог продуктів, сторінка про нас та контакти.

У верхній частині сторінки розміщено логотип компанії, що веде користувача на головну сторінку при натисканні. Поруч з логотипом розташована форма пошуку, яка дозволяє користувачам швидко знайти потрібні товари за ключовими словами. Користувач може ввести запит у пошукове поле та натиснути кнопку пошуку, щоб отримати результати.

Головна сторінка також включає рекламні банери та спеціальні пропозиції, які привертають увагу користувачів до нових або акційних продуктів. Ці елементи допомагають залучити користувачів до перегляду продукції та підвищити інтерес до покупок.

Головну сторінку ми можемо побачити на рисунку 2.10.1:

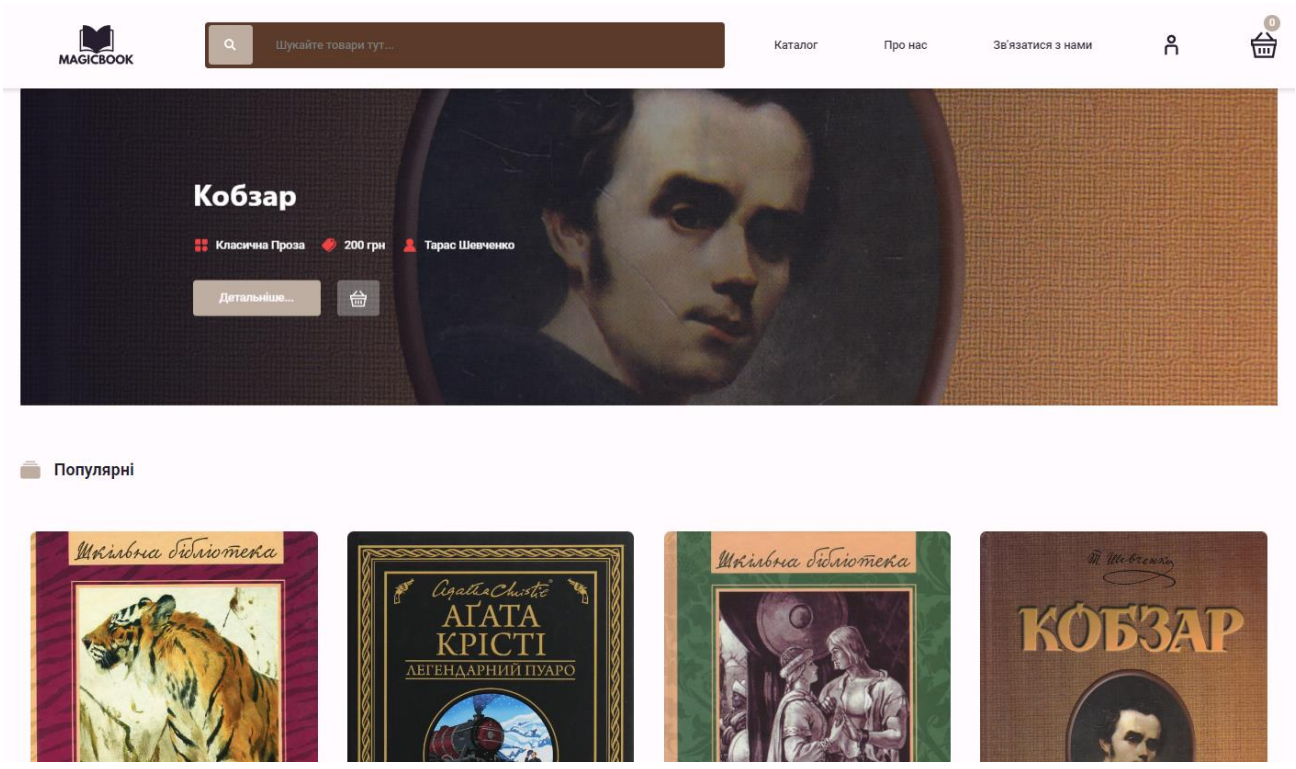


Рисунок 2.10.1 – Головна сторінка додатку

Для авторизованих користувачів головна сторінка може відображати доступ до їхнього профілю. Користувачі можуть легко перейти до свого облікового запису, щоб переглянути або змінити свої дані, а також переглянути список улюблених продуктів.

Коли користувач переходить на сторінку входу, його зустрічає форма для введення облікових даних. Вона містить три основні поля: для електронної пошти, пароля та двофакторну автентифікацію (2FA) для введення одноразового пароля (TOTP), згенерованого додатком Google Authenticator. Користувач вводить свою електронну пошту та пароль, а також одноразовий пароль, якщо в налаштуваннях була увімкнена відповідна функція, після чого натискає кнопку "Увійти". Форма надсилає запит на сервер для перевірки введених даних. На рисунку 2.10.2 зображено сторінку авторизації:

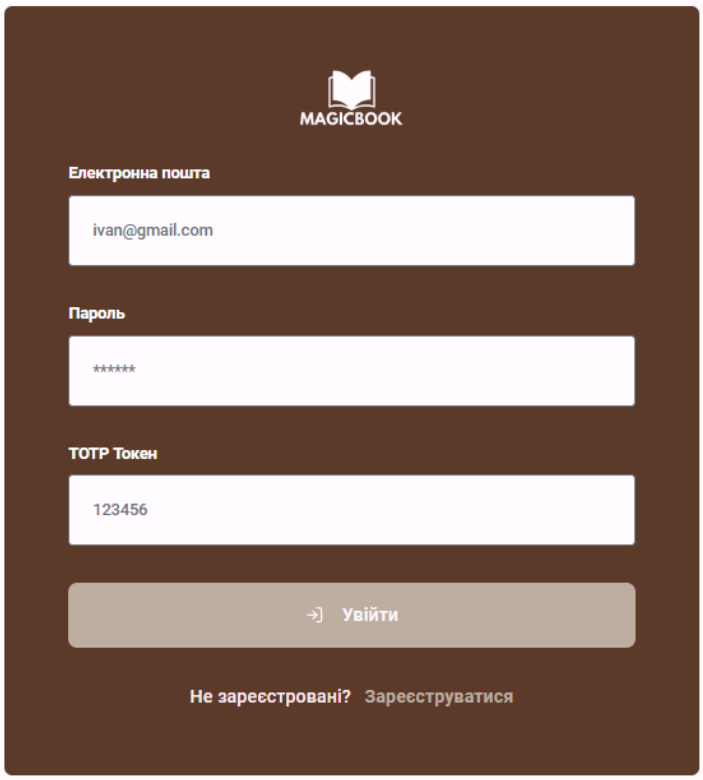


Рисунок 2.10.2 – Сторінка авторизації

Коли користувач натискає кнопку "Увійти" після введення своїх облікових даних, система надсилає запит на сервер для перевірки введеної електронної пошти та пароля. Сервер отримує цей запит і починає процес перевірки. Спочатку сервер шукає користувача в базі даних за введеною електронною поштою. Якщо користувача з такою електронною поштою не знайдено, сервер негайно повертає відповідь з повідомленням про помилку, вказуючи, що пошта або пароль невірні. Те ж саме відбувається із паролем. Помилка зображена на рисунку 2.10.3:

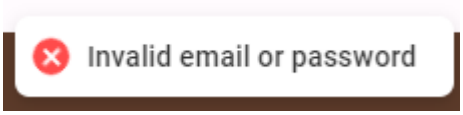


Рисунок 2.10.3 – Помилка при авторизації

Для нових користувачів, які ще не мають акаунту, на сторінці входу є "Зареєструватися", яка веде на сторінку реєстрації. Тут користувач може створити новий обліковий запис, заповнивши форму реєстрації. Форма включає поля для введення повного імені, електронної пошти, пароля. Після заповнення всіх необхідних полів користувач натискає кнопку "Зареєструватися", що надсилає запит на сервер для створення нового облікового запису. Сторінка реєстрації зображена на рисунку 2.10.4:

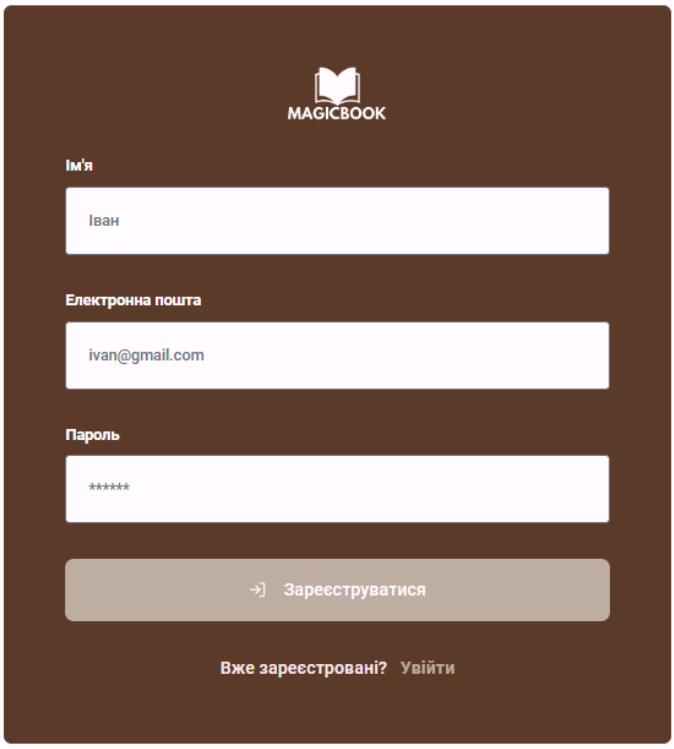


Рисунок 2.10.4 – Сторінка реєстрації

Аналогічно з авторизацією, сторінка реєстрація видає помилки при відповідних випадках, наприклад:

- Користувач ввів пароль менше 6 символів
- Користувач ввів вже існуючу пошту від існуючого акаунту
- Користувач не ввів пошту, пароль або ім’я

Подібну помилку зображено на рисунку 2.10.5:

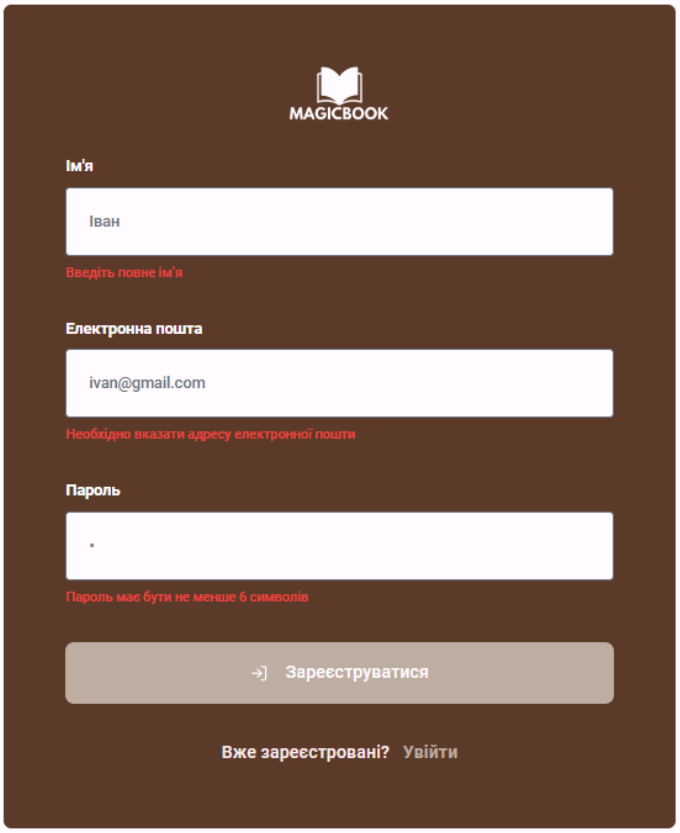


Рисунок 2.10.5 – Помилка при реєстрації

Після успішної реєстрації, в базі даних MongoDB зберігається інформація про нового користувача, вигляд нового користувача в базі даних зображено на рисунку 2.10.6:


```
_id: ObjectId('6668bbeebd14c9742151777c')
fullName : "Serhii"
email : "serhii@gmail.com"
password : "$2a$10$MfO8YT5AyQdVVBO5jmS8bOl6RhKWje4rpGMaXPIVPBDkLKw8x9LNi"
isAdmin : false
likedProducts : Array (empty)
totpSecret : ""
is2FAEnabled : false
createdAt : 2024-06-11T21:04:46.418+00:00
updatedAt : 2024-06-11T21:04:46.418+00:00
__v : 0
```


Рисунок 2.10.6 – Дані користувача в MongoDB

Після успішної реєстрації користувач автоматично перенаправляється на сторінку профілю, де він може переглянути та налаштувати свої персональні дані, переглянути кошик, змінити пароль або фото профілю та увімкнути двофакторну автентифікацію. Сторінка профілю зображена на рисунку 2.10.7:

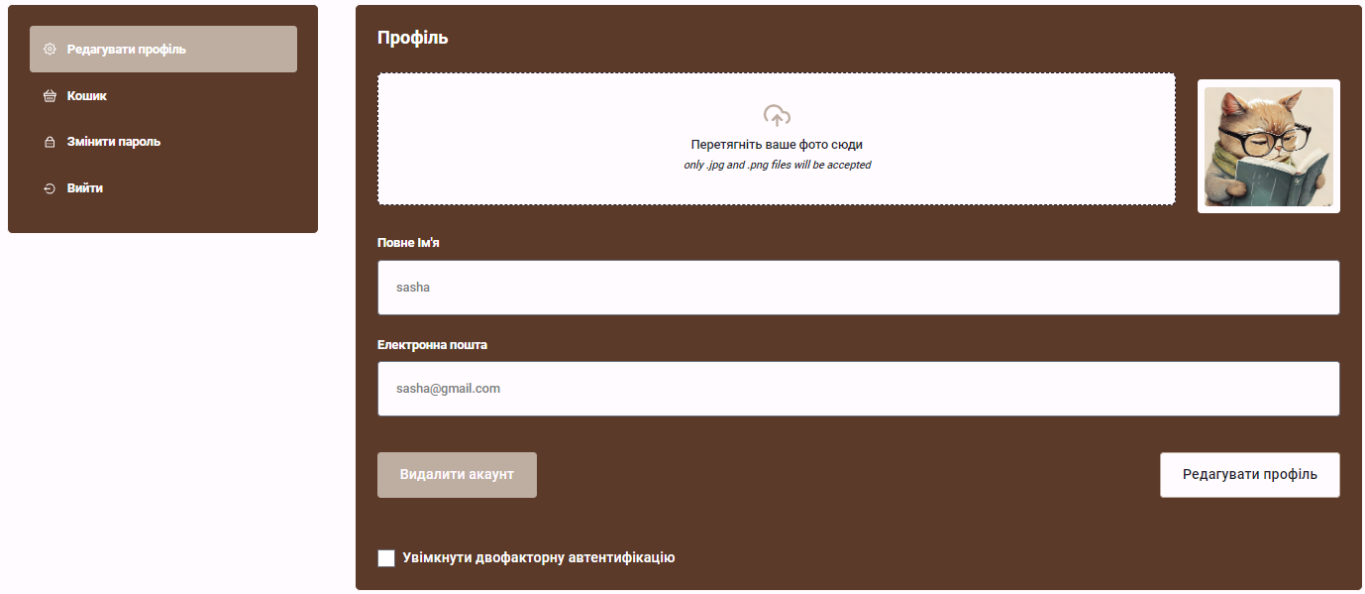


Рисунок 2.10.7 – Профіль користувача

На головній сторінці, користувач може перейти на окрему сторінку товару, переглядаючи опис, ціну та інші пункти товару. Також на сторінці товару, зареєстрований користувач може залишити відгук про товар, з оцінкою та самим описом відгуку. Сам відгук буде відображатись в окремому вікні. Додатково внизу сторінки зображено схожі товари до поточного. Сторінку товару та відгуків зображено на рисунках 2.10.8-2.10.9:

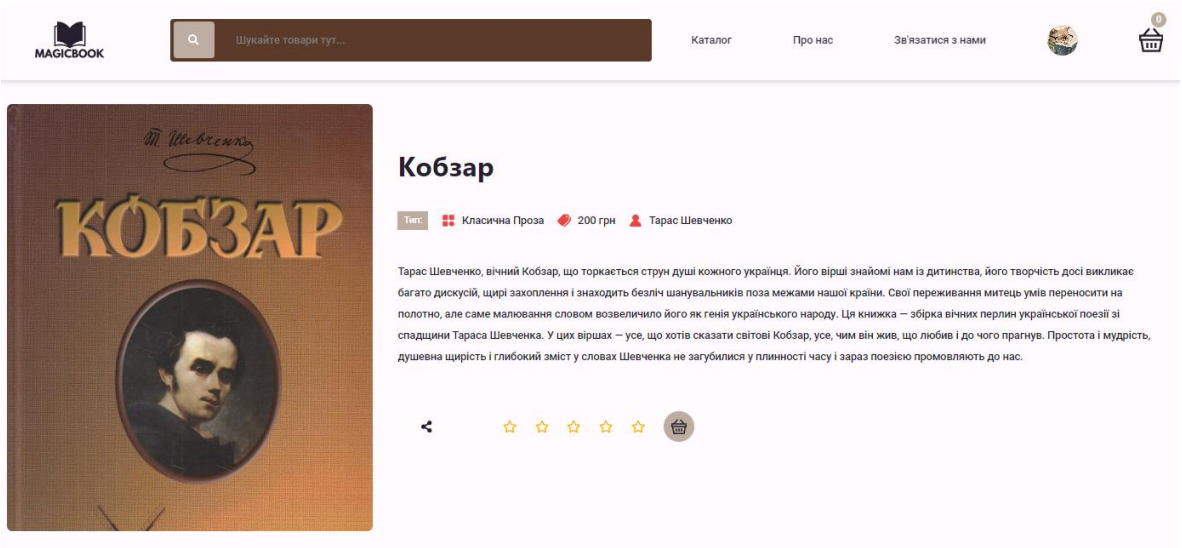


Рисунок 2.10.8 – Сторінка товару

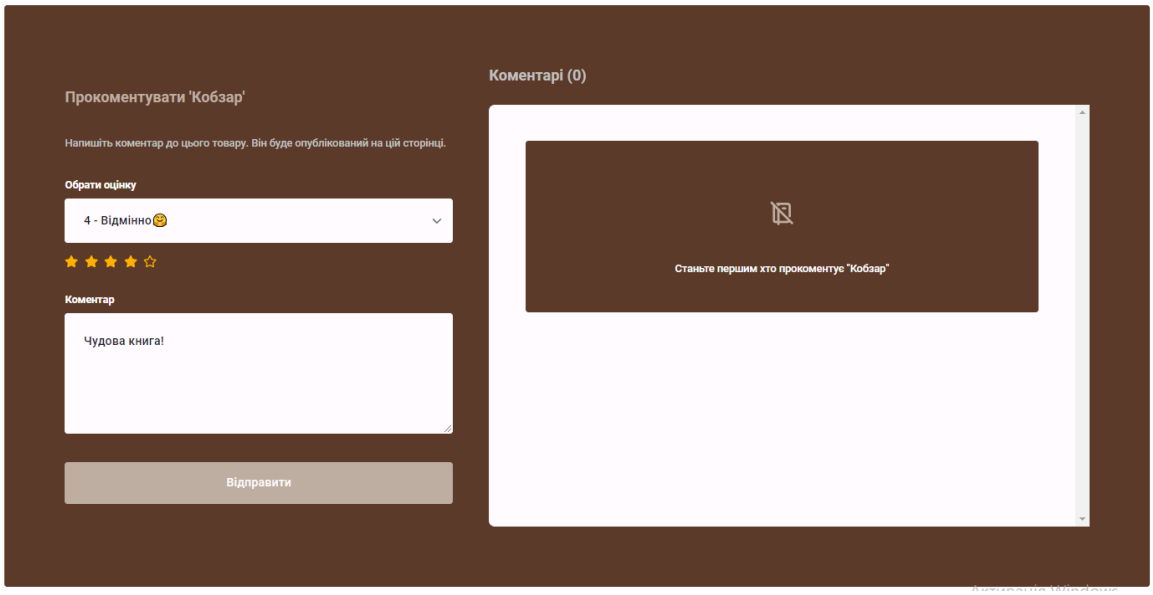


Рисунок 2.10.9 – Меню відгуків

Головна сторінка містить форму пошуку, де користувач може вводити ключові слова для пошуку книг. Форма пошуку допомагає швидко знайти потрібний продукт. Як і сама книга, фільтр має відповідні пункти, такі як: Категорія, Автор, Рік видання, Ціна та рейтинг згідно відгуків. По цим фільтрам користувач може легко знайти відповідний товар. Сторінку пошуку по фільтрам зображено на рисунку 2.10.10:

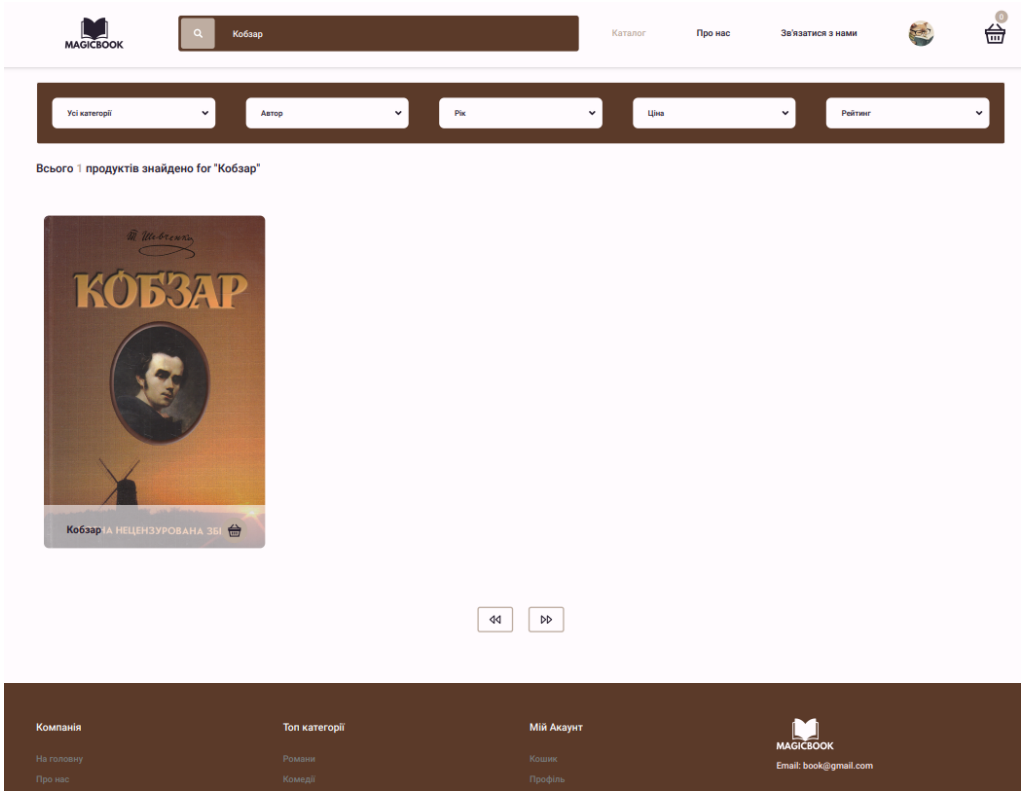
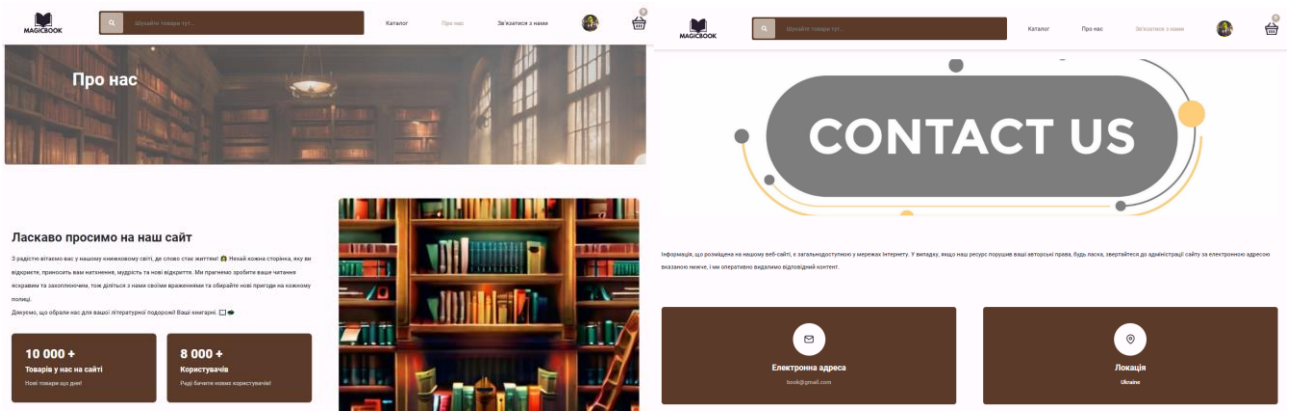


Рисунок 2.10.10 – Сторінка пошуку по фільтрам

На рисунках 2.10.11-2.10.12 зображено сторінки «Про нас» та «Зв'язатися з нами»



Рисунки 2.10.11-2.10.12 – Сторінки «Про нас» та «Зв'язатися з нами»

Панель адміністратора забезпечує інтерфейс для управління контентом та користувачами на сайті. Адміністратор має доступ до розширених функцій, таких як

перегляд та управління користувачами, перегляд та редагування книг. Крім того, адміністратор може добавляти книги та категорії. Панель адміністратора спрощує виконання цих завдань завдяки централізованому доступу до всіх необхідних інструментів та даних, що забезпечує ефективне управління та підтримку сайту, що зображено на рисунку 2.10.13:

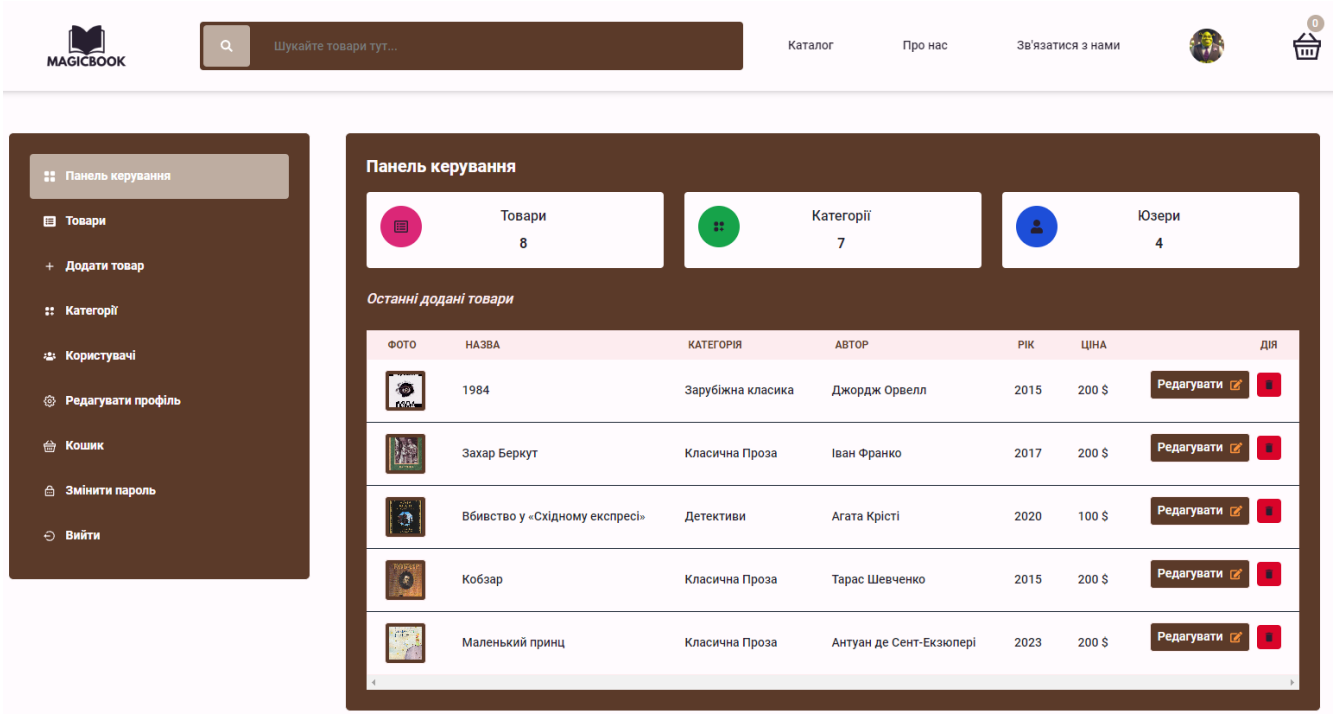


Рисунок 2.10.13 – Панель адміністратора

Загалом, інтерфейс сайту створює позитивний користувацький досвід, сприяючи легкому доступу до інформації та забезпечуючи ефективну взаємодію з усіма функціями сайту.

РОЗДІЛ 3. БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ ТА ОСНОВИ ОХОРОНИ ПРАЦІ

3.1 Долікарська допомога при кровотечах

При наданні долікарської допомоги при кровотечах першочерговим завданням є зупинка кровотечі. Для цього застосовуються різні методи, такі як пальцеве притиснення артерії до кістки, згинання кінцівки в ліктьовому, колінному або тазостегновому суглобах, а також накладання джгута або закрутки. Джгут накладається вище рани на 10 см, і при цьому обов'язково вкладається записка із зазначенням точного часу накладання джгута, оскільки максимальний час його використання становить 1,5 години. Після зупинки кровотечі необхідно обробити краї рани спиртовим розчином йоду або зеленкою. Допускається заливати неглибокі рани перекисом водню, уникаючи попадання бруду в рану. Потім на рану накладається туга пов'язка, що складається з кількох шарів стерильного бинта, валика зі стерильної вати і зовнішнього шару бинта. Пов'язка повинна бути такою, щоб джгут можна було зняти без відновлення кровотечі. Якщо кровотеча відновиться, джгут потрібно накласти знову [13].

Кровотечі можуть бути різних типів, включаючи капілярні, венозні та артеріальні. Капілярні кровотечі зазвичай менш небезпечні і легко зупиняються за допомогою простого бинтування. Венозні кровотечі характеризуються повільним, але постійним витіканням темної крові і потребують більш серйозних заходів, таких як накладання давлючого бинта. Артеріальні кровотечі найнебезпечніші через швидкий витік яскраво-червоної крові під тиском, що може швидко призвести до значної крововтрати і навіть смерті. У такому випадку необхідно негайно накласти джгут вище місця пошкодження для зупинки кровотечі.

Перед початком надання допомоги потрібно оцінити стан постраждалого і визначити тип кровотечі. Важливо залишатися спокійним і діяти рішуче, дотримуючись основних принципів першої допомоги: правильність, доцільність,

швидкість, продуманість, рішучість і спокій. Послідовність надання першої допомоги включає усунення впливу небезпечних факторів, зупинку кровотечі і забезпечення подальшого догляду до прибуття медиків.

Для зупинки кровотечі при капілярних пораненнях використовують стерильний бинт або чисту тканину, притискаючи її до рани. Венозні кровотечі вимагають накладання давлючої пов'язки, а артеріальні — джгута. Джгут накладається вище рани, і час його накладання обов'язково слід зафіксувати, щоб уникнути довготривалого перекриття кровотоку.

Окрім фізичних дій, постраждалий також потребує психологічної підтримки. Увага, щирість і турбота допомагають зменшити стрес і тривогу потерпілого, що є важливою частиною долікарської допомоги. Відповідальність і швидка реакція оточуючих можуть значно підвищити шанси на виживання та успішне лікування постраждалого після прибуття професійної медичної допомоги [14].

Долікарська допомога при кровотечах є важливим елементом, що може врятувати життя потерпілого. Надання першої допомоги при кровотечах включає кілька основних кроків.

Кровотеча може бути артеріальною, венозною або капілярною. Артеріальна кровотеча характеризується яскраво-червоною кров’ю, що витікає сильними поштовхами. Венозна кров темно-червона і витікає безперервно, повільніше. Капілярна кровотеча характеризується кровоточивістю по всій поверхні рани або витіканням крові краплями.

Для зупинки кровотечі важливо діяти швидко. При капілярній кровотечі досить накласти стисну пов’язку, попередньо обробивши шкіру навколо рани антисептичним розчином. Якщо в рані є сторонній предмет, його не можна витягувати, а слід закріпити в рані, зробивши отвір у пов’язці, щоб уникнути додаткового пошкодження тканин.

Венозну кровотечу можна зупинити, піднявши кінцівку та наклавши стисну пов’язку. Якщо це не допомагає, кінцівку згинають у суглобі і забинтовують.

Артеріальна кровотеча зупиняється притискуванням артерії до кістки вище від місця поранення. Якщо притискування не ефективне, застосовують джгут або закрутку з підручних матеріалів (пояс, рушник тощо). Джгут накладають вище рани на кінцівку, з приміткою про час накладання, оскільки його можна тримати не більше двох годин, а в зимовий період - одну годину. Найважливіше при наданні першої допомоги — забезпечити потерпілому спокій та якнайшвидше доставити його до медичного закладу для подальшої професійної допомоги [15].

3.2 Інформація в системі управління охороною праці

Інформаційна система управління охороною праці є ключовою складовою забезпечення безпеки праці на підприємствах, установах і організаціях. Вона охоплює широкий спектр функцій, спрямованих на збереження здоров'я і працездатності працівників. Основні складові такої системи включають правові та організаційні основи, систему управління охороною праці на підприємствах, гігієнічне нормування умов праці, навчання та інструктажі з питань охорони праці, а також заходи з профілактики професійних захворювань та безпеки під час роботи з обчислювальною технікою.

Правові та організаційні основи охорони праці охоплюють відносини між роботодавцем і працівниками. Важливою частиною є забезпечення працівників знаннями про їх права на охорону праці та функціонування системи державного управління і нагляду за охороною праці. Охорона праці як складова безпеки життєдіяльності передбачає створення належних умов для збереження здоров'я працівників через впровадження необхідних заходів і стандартів.

Система управління охороною праці на підприємстві (СУОПП) є важливою частиною загальної системи управління, яка спрямована на зниження ризиків

виникнення нещасних випадків та професійних захворювань, що передбачає організацію та координацію заходів для забезпечення безпеки працівників. Вона включає аналіз причин травматизму та професійних захворювань, методи їх попередження, а також розслідування нещасних випадків і аварій. Фонд соціального страхування здійснює розслідування і компенсацію шкоди, завданої працівникам у разі нещасних випадків на виробництві. Вона також включає в себе моніторинг умов праці, аналіз ризиків, розробку та впровадження заходів для поліпшення безпеки на робочих місцях. Гігієнічне нормування умов праці є невід'ємною частиною цієї системи, оскільки воно забезпечує відповідність робочих місць санітарно-гігієнічним стандартам та вимогам, що сприяє зниженню ризиків для здоров’я працівників.

Навчання та інструктажі з питань охорони праці спрямовані на підвищення рівня обізнаності працівників про можливі небезпеки та методи їх уникнення. Це включає регулярне проведення навчальних занять, семінарів та тренінгів, що дозволяє працівникам бути готовими до реагування на надзвичайні ситуації та знати, як діяти у випадку небезпеки. Безпека під час роботи з обчислювальною технікою передбачає дотримання правил ергономіки, запобігання шкідливим впливам на зір, поставу та загальний стан здоров’я працівників, що працюють з комп’ютерами та іншими електронними пристроями.

Професійні захворювання та їх попередження є важливим аспектом охорони праці. Це включає ідентифікацію причин, які можуть призводити до виникнення професійних захворювань, та впровадження заходів для їх запобігання. Важливим елементом є також система державного соціального страхування, яка забезпечує соціальний захист працівників у випадку захворювань чи нещасних випадків на виробництві. Пожежна безпека підприємств, установ, організацій охоплює заходи для запобігання виникненню пожеж, забезпечення наявності необхідних засобів пожежогасіння та навчання персоналу діям у випадку пожежі [13].

Однією з основних складових інформації в СУОП є облік показників стану охорони праці, аналіз одержаних даних та узагальнення причин недодержання вимог

законодавчих та інших нормативних актів про охорону праці. Це включає аналіз матеріалів про нещасні випадки та професійні захворювання, результати всіх видів контролю за станом охорони праці, дані паспортів санітарно-технічного стану умов праці, матеріали спеціальних обстежень будівель, споруд, приміщень, обладнання та інших елементів виробничого середовища. На основі цієї інформації здійснюються коригування оперативних, поточних та перспективних планів роботи з охорони праці, а також визначаються кращі серед структурних підрозділів, служб, працівників за досягнутими показниками з охорони праці.

Особливої уваги заслуговує аналіз матеріалів про нещасні випадки та професійні захворювання, результати всіх видів контролю за станом охорони праці, дані паспортів санітарно-технічного стану умов праці в цеху (на дільниці), матеріали спеціальних обстежень будівель, споруд, приміщень, обладнання та інших елементів виробничого середовища. За результатами обліку, аналізу та оцінки стану охорони праці вносяться доповнення та уточнення до оперативних, поточних та перспективних планів роботи з охорони праці, а також визначаються кращі серед структурних підрозділів, служб, працівників за досягнутими показниками з охорони праці.

Таким чином, інформація є центральним елементом СУОП, забезпечуючи своєчасне і точне прийняття рішень, підвищення рівня обізнаності працівників і створення культури безпеки на підприємстві. Без ефективного управління інформацією неможливо забезпечити високий рівень безпеки та здоров'я на робочому місці, що робить інформаційний компонент ключовим аспектом СУОП [16].

ВИСНОВКИ

В результаті розробки веб-сайту каталогу книг на основі технологій MERN (MongoDB, Express, React, Node.js) та Firebase було створено зручну платформу для користувачів. основна мета якої полягала в забезпеченні легкого доступу до великої кількості книг та інтерактивної взаємодії з користувачами.

Проведений аналіз вимог дозволив визначити ключові функції та особливості, які необхідні для створення ефективного веб-додатку. Було вивчено потреби користувачів, що допомогло розробити функціонал для реєстрації, авторизації, пошуку та фільтрації книг, перегляду книг та додавання їх до кошика а також залишення відгуків. Виявлені вимоги стали основою для подальшого проектування та розробки системи.

На етапі проектування архітектури було створено структуру системи, що включає клієнтську та серверну частини. Було розроблено моделі бази даних, що забезпечують зберігання та управління даними користувачів та книг. Архітектура була спроектована таким чином, щоб забезпечити високу продуктивність, масштабованість та безпеку системи.

Клієнтська частина була розроблена з використанням React, що забезпечило створення динамічного та інтерактивного інтерфейсу. Серверна частина була розроблена з використанням Node.js та Express, що забезпечило стабільну та швидку обробку запитів. База даних MongoDB дозволила ефективно зберігати та управляти даними про користувачів і книги.

Загалом, проект успішно досяг поставлених цілей, створивши зручну та функціональну платформу для користувачів. Проект демонструє важливість використання сучасних технологій для створення зручних та цікавих веб-додатків, що відповідають потребам сучасності.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Інженерія програмного забезпечення: навч. посібник / Д. П. Кучеров, Є. Б. Артамонов. – К. : НАУ, 2017. – 388 с.
2. Yakaboo - інтернет-магазин книг, дитячих товарів і подарунків в Україні [Електронний ресурс] – Режим доступу до ресурсу: https://www.yakaboo.ua/
3. Книгарня «Є» - українська мережа книгарень [Електронний ресурс] – Режим доступу до ресурсу: https://book-ye.com.ua/
4. Book24 - книжковий інтернет-магазин [Електронний ресурс] – Режим доступу до ресурсу: https://book24.ua/ua/
5. Speakeasy two-factor authenticaon for node,js [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/speakeasyjs/speakeasy
6. Google блог безпеки [Електронний ресурс] – Режим доступу до ресурсу: https://security.googleblog.com/2023/04/google-authenticator-now-supports.html
7. ReactJS - JavaScript-фреймворк [Електронний ресурс] – Режим доступу до ресурсу: https://cases.media/article/sho-take-react-js-yak-pochati-vivchati-reakt-navichki-dlya-react-developer
8. NodeJS - середовище виконання JavaScript [Електронний ресурс] – Режим доступу до ресурсу: https://nodejs.org/en/about
9. Веб-фреймворк Express [Електронний ресурс] – Режим доступу до ресурсу: https://expressjs.com/
10. Introduction to MongoDB [Електронний ресурс] – Режим доступу до ресурсу: https://www.mongodb.com/docs/manual/introduction/
11. FireBase - хмарне сховище [Електронний ресурс] – Режим доступу до ресурсу: https://www.geeksforgeeks.org/introduction-to-firebase-cloud-storage/
12. Архітектура клієнт-сервер [Електронний ресурс] – Режим доступу до ресурсу: http://inter.ptngu.com/kompyuterni-merezhi/arhitektura-kliyent-server

13. Безпека життєдіяльності: навчальний посібник / Т.Є Стиценко, Г.В. Пронюк, Н.М. Сердюк, І.І. Хондак. – Харків: ХНУРЕ, 2018. – 336 с.
14. Безпека життєдіяльності та цивільний захист: підручник / О.Г. Левченко, О.В. Землянська, Н.А. Праховнік, В.В. Зацарний; КПІ ім. Ігоря Сікорського. – Київ: КПІ ім. Ігоря Сікорського, 2019. – 267 с.
15. Безпека життєдіяльності: навчальний посібник / В.В. Зацарний, О.В. Зацарна, О.В. Землянська, Н.А. Праховнік. – Київ: НТУУ «КПІ ім. І. Сікорського», 2016. – 267 с.
16. Основи хорони праці : підручник / О.В. Третьяков, Є.В. Доронін, Р.В. Пономаренко, В.Л. Безсонний. Харків, ТОВ «Планета-Прінт», 2020. – 588 с.
17. Проблеми аналізу унікальності контенту веб-сайтів у роботі SEO-оптимізатора // І. Мудрик, Р. Карагодін, ТНТУ, 2023, – 161 с.

ДОДАТКИ

ДОДАТОК А

Лістинг коду розробленого веб-сервісу зображено на лістингах 1 – 1ч.

Лістинг 1 – Файл server.js

```
import express from 'express';
import cors from 'cors';
import dotenv from 'dotenv';
import { connectDB } from './config/db.js';
import { errorHandler } from './middlewares/errorMiddleware.js';
import userRouter from './Routes/UserRouter.js'
import productsRouter from './Routes/ProductsRouter.js'
import categoriesRouter from './Routes/CategoriesRouter.js'
import Uploadrouter from './controller/UploadFile.js';

dotenv.config();

const app = express();
app.use(cors());
app.use(express.json());
//Connect DB
connectDB();


// Main route
app.get('/', (req, res) => {
res.send('API is running...');
});
//other routes
app.use('/api/users', userRouter);
app.use('/api/products', productsRouter);
app.use('/api/categories', categoriesRouter);
app.use('/api/upload', Uploadrouter);

//error handling middleware
app.use(errorHandler);

const PORT = process.env.PORT || 5000;

app.listen(PORT, () => {
console.log(`Server running in http://localhost/${PORT}`);
});
```

Лістинг 2 – Файл App.js

```
import React, { useEffect } from 'react';
import { BrowserRouter, Route, Routes } from 'react-router-dom';
import HomeScreen from './Screens/HomeScreen';
import AboutUs from './Screens/AboutUs';
import NotFound from './Screens/NotFound';
import ContactUs from './Screens/ContactUs';
import ProductsPage from './Screens/Products';
import SingleProduct from './Screens/SingleProduct';
import Login from './Screens/Login';
import Register from './Screens/Register';
import Profile from './Screens/Dashboard/Profile';
import Aos from 'aos';
import Password from './Screens/Dashboard/Password';
import FavoritesProducts from './Screens/Dashboard/FavoritesProducts';
import ProductList from './Screens/Dashboard/Admin/ProductList';
import Dashboard from './Screens/Dashboard/Admin/Dashboard';
import Categories from './Screens/Dashboard/Admin/Categories';
import Users from './Screens/Dashboard/Admin/Users';
import AddProducts from './Screens/Dashboard/Admin/AddProducts';
import ScrollOnTop from './ScrollOnTop';
import ToastContainer from './Components/Notification/ToastContainer';
import { AdminProtectionRouter, ProtectedRouter } from './ProtectedRouter';
import { useDispatch, useSelector } from 'react-redux';
import { getAllCategoriesAction } from './Redux/Actions/CategoriesActions';
import { getAllProductsAction } from './Redux/Actions/ProductsActions';
import { getFavoriteProductsAction } from './Redux/Actions/userActions';
import toast from 'react-hot-toast';
import EditProduct from './Screens/Dashboard/Admin/EditProduct';
import DrawerContext from './Context/DrawerContext';

function App() {
Aos.init();
const dispatch = useDispatch();
const { userInfo } = useSelector((state) => state.userLogin);
const { isError, isSuccess } = useSelector((state) => state.userLikeProduct);
const { isError: catError } = useSelector((state) => state.categoryGetAll);

useEffect(() => {
dispatch(getAllCategoriesAction());
dispatch(getAllProductsAction({}));
if (userInfo) {
dispatch(getFavoriteProductsAction());
}
if (isError || catError) {
toast.error(isError || catError);
dispatch({ type: "LIKE_PRODUCT_RESET" });
}
if (isSuccess) {
dispatch({ type: "LIKE_PRODUCT_RESET" });
}
}, [dispatch, userInfo, isError, catError, isSuccess]);

return (
<>
```

```
<BrowserRouter>
<ToastContainer />
<DrawerContext>
<ScrollOnTop>
<Routes>
{ /************* * PUBLIC ROUTERS **************/ }
<Route path='/' element={ <HomeScreen /> } />
<Route path='/about-us' element={ <AboutUs /> } />
<Route path='/contact-us' element={ <ContactUs /> } />
<Route path='/products' element={ <ProductsPage /> } />
<Route path="/products/:search" element={ <ProductsPage /> } />
<Route path='/product/:id' element={ <SingleProduct /> } />
<Route path='/login' element={ <Login /> } />
<Route path='/register' element={ <Register /> } />
<Route path='*' element={ <NotFound /> } />
{ /************* * PRIVATE PUBLIC ROUTERS **************/ }
<Route element={ <ProtectedRouter /> }>
<Route path='/profile' element={ <Profile /> } />
<Route path='/password' element={ <Password /> } />
<Route path='/favorites' element={ <FavoritesProducts /> } />
{ /************* * ADMIN ROUTERS **************/ }
<Route element={ <AdminProtectionRouter /> }>
<Route path='/productslist' element={ <ProductList /> } />
<Route path='/dashboard' element={ <Dashboard /> } />
<Route path='/categories' element={ <Categories /> } />
<Route path='/users' element={ <Users /> } />
<Route path='/addproduct' element={ <AddProducts /> } />
<Route path='/edit/:id' element={ <EditProduct /> } />
</Route>
</Route>
</Routes>
</ScrollOnTop >
</DrawerContext>
</BrowserRouter>
</>
);
}

export default App;
```

Лістинг 3 – Файл Index.js

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import "swiper/css";
import "swiper/css/pagination";
import "swiper/css/navigation";
import "aos";
import "aos/dist/aos.css"
import { BrowserRouter } from 'react-router-dom';
import { Provider } from 'react-redux';
```

```
import { store } from './Redux/store';

import { GoogleOAuthProvider } from '@react-oauth/google'

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <Provider store={ store } >
      <BrowserRouter />
      <GoogleOAuthProvider clientId="290090792949-
j2bq2ug2i3rdgm66b78ek4g7mhp51dm2.apps.googleusercontent.com">
        <App />
      </GoogleOAuthProvider>
      <BrowserRouter />
    </Provider>
  </React.StrictMode >
);
```

Лістинг 4 – Файл Auth.js

```
import jwt from 'jsonwebtoken'
import User from '../models/UserModels.js';
import asyncHandler from 'express-async-handler';

// generate a token
// @desc Generate token
// @route GET /api/users/:id

const generateToken = (id) => {
    return jwt.sign({ id }, process.env.JWT_SECRET, {
        expiresIn: '1d',
    });
}

// protection middleware
const protect = asyncHandler(async (req, res, next) => {
    let token;
    // check if token exists in headers
    if (
        req.headers.authorization &&
        req.headers.authorization.startsWith('Bearer')
    ) {
        // set token from Bearer token in header
        try {
            token = req.headers.authorization.split(' ')[1];
            // verify token and get user id
            const decoded = jwt.verify(token, process.env.JWT_SECRET);
            // get user id from decoded token
            req.user = await User.findById(decoded.id).select("-password")
            next();
        } catch (error) {
            console.error(error);
            res.status(401);
            throw new Error('Not authorized, token failed');
```

```
        }
    }
    // if token doesn't exist in headers send error
    if (!token) {
        res.status(401);
        throw new Error('Not authorized, no token');
    }
})

//admin middleware
const admin = (req, res, next) => {
    if (req.user && req.user.isAdmin) {
        next();
    } else {
        res.status(401);
        throw new Error("Not authorized as an admin");
    }
};

export { generateToken, protect, admin }
```

Лістинг 5 – Файл Login.js

```
import React, { useEffect } from 'react';
import Layout from '../Layout/Layout';
import { Input } from '../Components/UsedInputs';
import { Link, useNavigate } from 'react-router-dom';
import { FiLogIn } from 'react-icons/fi';
import { useDispatch, useSelector } from 'react-redux';
import { useForm } from 'react-hook-form';
import { LoginValidation } from '../Components/Validation/UserValidation';
import { yupResolver } from '@hookform/resolvers/yup';
import { InlineError } from './../Components/Notification/Error';
import { loginAction } from '../Redux/Actions/userActions';
import toast from 'react-hot-toast';

function Login() {
  const dispatch = useDispatch();
  const navigate = useNavigate();

  const { isLoading, isError, userInfo, isSuccess } = useSelector(
    (state) => state.userLogin
  );

  //validate user
  const {
    register,
    handleSubmit,
    formState: { errors }
  } = useForm({
    resolver: yupResolver(LoginValidation),
  });
```

```
  // on submit
  const onSubmit = (data) => {
    dispatch(loginAction(data));
  }

  // useEffect
  useEffect(() => {
    if (userInfo?.isAdmin) {
      navigate("/dashboard");
    }
    else if (userInfo) {
      navigate("/profile");
    }
    if (isSuccess) {
      toast.success(`${userInfo?.fullName} Раді бачити знову 😁`);
    }
    if (isError) {
      toast.error(isError);
      dispatch({ type: "USER_LOGIN_RESET" });
    }
  }, [userInfo, isSuccess, isError, navigate, dispatch]);

  return (
    <Layout>
      <div className='container mx-auto px-2 my-24 flex-colo'>
        <form onSubmit={handleSubmit(onSubmit)} className='w-full 2xl:w-2/5
gap-8 flex-colo p-8 sm:p-14 md:w-3/5 bg-dry rounded-lg border border-main'>
          <img src='/images/logo1.png' alt='logo' className='w-full h-12
object-contain' />

          <div className='w-full'>
            <Input label='Електронна пошта' placeholder='ivan@gmail.com'
type='email' name='email' register={register('email')} bg={true} />
            {
              errors.email && <InlineError text={errors.email.message} />
            }
          </div>

          <div className='w-full'>
            <Input label='Пароль' placeholder='******' type='password'
name='password' register={register('password')} bg={true} />
            {
              errors.password && <InlineError text={errors.password.message}
/>
            }
          </div>

          <div className='w-full'>
            <Input label='TOTP Токен' placeholder='123456' type='text'
name='token' register={register('token')} bg={true} />
            {
              errors.token && <InlineError text={errors.token.message} />
```

```
            }
          </div>

          <button
            type='submit'
            disabled={isLoading}
            className='bg-subMain transitions hover:bg-main flex-rows gap-4
text-main hover:text-mainText p-4 rounded-lg w-full'>
            {
              // if loading show loading
              isLoading ? (
                "Завантаження..."
              ) : (
                <>
                  <FiLogIn /> Увійти
                </>
              )
            }
          </button>

          <p className='text-center text-border'>
            Не зареєстровані?{ ' ' }
            <Link to='/register' className='text-subMain font-semibold ml-2'>
              Зареєструватися
            </Link>
          </p>
        </form>
      </div>
    </Layout>
  );
}

export default Login;
```

Лістинг 6 – Файл Profile.js

```
import React, { useEffect, useState } from 'react';
import SideBar from './SideBar';
import Uploder from '../../Components/Uploder';
import { Input } from '../../Components/UsedInputs';
import toast from 'react-hot-toast';
import { yupResolver } from '@hookform/resolvers/yup';
import { useForm } from 'react-hook-form';
import { useDispatch, useSelector } from 'react-redux';
import { ProfileValidation } from
'../../Components/Validation/UserValidation';
import { InlineError } from '../../Components/Notification/Error';
import { ImagePreview } from '../../Components/imagePreview';
import { deleteProfileAction, updateProfileAction } from
'../../Redux/Actions/userActions';
import axios from 'axios';

function Profile() {
```

```
  const dispatch = useDispatch();
  const { userInfo } = useSelector((state) => state.userLogin);
  const [imageUrl, setImageUrl] = useState(userInfo ? userInfo.image : "");
  const { isLoading, isError, isSuccess } = useSelector(
    (state) => state.userUpdateProfile
  );
  const { isLoading: deleteLoading, isError: deleteError } = useSelector(
    (state) => state.userDeleteProfile
  );
  const [totpSecret, setTotpSecret] = useState('');
  const [totpURL, setTotpURL] = useState('');
  const [is2FAEnabled, setIs2FAEnabled] = useState(userInfo ?
userInfo.is2FAEnabled : false);

  //validate user
  const {
    register,
    handleSubmit,
    setValue,
    formState: { errors }
  } = useForm({
    resolver: yupResolver(ProfileValidation),
  });

  // update profile
  const onSubmit = (data) => {
    dispatch(updateProfileAction({ ...data, image: imageUrl }));
  };

  //delete profile
  const deleteProfile = () => {
    window.confirm("Ви впевнені, що хочете видалити свій профіль?") &&
      dispatch(deleteProfileAction());
  };

  //generate TOTP secret and URL
  const generateTOTP = async () => {
    try {
      const response = await axios.get('/api/users/generate-qr-code', {
        headers: { Authorization: `Bearer
${localStorage.getItem('authToken')}` }
      });
      setTotpSecret(response.data.secret);
      setTotpURL(response.data.url);
    } catch (error) {
      toast.error('Error generating TOTP secret');
    }
  };

  const handle2FAToggle = async () => {
    try {
      const response = await axios.post('/api/users/enable-2fa', { enable:
!is2FAEnabled }, {
```

```
        headers: { Authorization: `Bearer
${localStorage.getItem('authToken')}` }
      });
      setIs2FAEnabled(!is2FAEnabled);
      toast.success(response.data.message);
    } catch (error) {
      toast.error('Error toggling 2FA');
    }
  };

  useEffect(() => {
    if (userInfo) {
      setValue("fullName", userInfo?.fullName);
      setValue("email", userInfo?.email);
      setIs2FAEnabled(userInfo.is2FAEnabled);
    }
    if (isSuccess) {
      dispatch({ type: 'USER_UPDATE_PROFILE_RESET' });
    }
    if (isError || deleteError) {
      toast.error(isError || deleteError);
      dispatch({ type: 'USER_UPDATE_PROFILE_RESET' });
      dispatch({ type: 'USER_DELETE_PROFILE_RESET' });
    }
  }, [userInfo, setValue, isSuccess, isError, dispatch, deleteError]);

  return (
    <SideBar>
      <form onSubmit={handleSubmit(onSubmit)} className='flex flex-col gap-
6'>
        <h2 className='text-xl font-bold text-white'>Профіль</h2>
        <div className="w-full grid lg:grid-cols-12 gap-6">
          <div className='col-span-10'>
            <Uploder setImageUrl={setImageUrl} />
          </div>
          {/* image preview */}
          <div className='col-span-2'>
            <ImagePreview
              image={imageUrl}
              name={userInfo ? userInfo.fullName : ""} />
          </div>
        </div>

        <div className='w-full'>
          <Input label="Повне Ім'я" placeholder='ivan' type='text'
name='fullName' register={register('fullName')} bg={true} />
          {errors.fullName && <InlineError text={errors.fullName.message} />}
        </div>

        <div className='w-full'>
          <Input label='Електронна пошта' placeholder='ivan@gmail.com'
type='email' name='email' register={register('email')} bg={true} />
          {errors.email && <InlineError text={errors.email.message} />}
        </div>
```

```
        <div className='flex gap-2 flex-wrap flex-col-reverse sm:flex-row
justify-between items-center my-4'>
          <button
            onClick={deleteProfile}
            disabled={deleteLoading || isLoading}
            className='bg-subMain font-medium transitions hover:bg-main
border border-subMain text-main hover:text-mainText py-3 px-6 rounded w-full
sm:w-auto'>
            {deleteLoading ? "Видалення..." : "Видалити акаунт"}
          </button>
          <button
            type="submit"
            disabled={deleteLoading || isLoading}
            className='bg-main font-medium transitions hover:text-main
hover:bg-subMain border border-subMain text-mainText py-3 px-6 rounded w-full
sm:w-auto'>
            {isLoading ? "Редагування..." : "Редагувати профіль"}
          </button>
        </div>

        <div className='flex flex-col gap-4 mt-4'>
          <label className='flex items-center gap-2'>
            <input
              type="checkbox"
              className='form-checkbox h-5 w-5 text-main'
              checked={is2FAEnabled}
              onChange={handle2FAToggle}
            />
            <span className='text-white'>Увімкнути двофакторну
автентифікацію</span>
          </label>
          {is2FAEnabled && !totpSecret && (
            <button
              type="button"
              onClick={generateTOTP}
              className='bg-main font-medium transitions hover:text-main
hover:bg-subMain border border-subMain text-mainText py-3 px-6 rounded w-full
sm:w-auto'>
              Згенерувати секретний ключ для двофакторної автентифікації
            </button>
          )}
          {totpSecret && (
            <div className='bg-gray-800 p-4 rounded-lg'>
              <p className='text-white'>Секретний ключ: {totpSecret}</p>
              <p className='text-white'>URL для додатку: {totpURL}</p>
            </div>
          )}
        </div>
      </form>
    </SideBar>
  );
}
```

```
export default Profile;
```

Лістинг 7 – Файл AboutUs.js

```
import React from 'react'
import Layout from './../Layout/Layout'
import Head from '../Components/Head'

function AboutUs() {
  return (
    <Layout>
      <div className='min-height-screen container mx-auto px-2 my-6'>
        <Head title='Про нас' />
        <div className='xl:py-20 py-10 px-4'>
          <div className='grid grid-flow-row xl:grid-cols-2 gap-4 xl:gap-16
items-center'>
            <div>
              <h3 className='text-xl lg:text-3xl mb-4 font-semibold'>
                Ласкаво просимо на наш сайт
              </h3>
              <div className='mt-3 text-sm leading-8 text-mainText'>
                <p>
                  З радістю вітаємо вас у нашому книжковому світі, де слово
стає життям! 🌟 Нехай кожна сторінка, яку ви відкриєте, приносить вам
натхнення, мудрість та нові відкриття. Ми прагнемо зробити ваше читання
яскравим та захоплюючим, тож діліться з нами своїми враженнями та обирайте
нові пригоди на кожному полиці.     </p>
                <p>
                  Дякуємо, що обрали нас для вашої літературної подорожі!
Ваші книгарні. 📖🔗
                </p>
              </div>
              <div className='grid md:grid-cols-2 gap-6 mt-8'>
                <div className='p-8 bg-dry rounded-lg'>
                  <span className='text-3xl text-white block font-extrabold'>
                    10 000 +
                  </span>
                  <h4 className='text-lg text-white font-semibold my-
1'>Товарів у нас на сайті</h4>
                  <p className='mb-0 text-text leading-7 text-sm '>
                    Нові товари що дня!
                  </p>
                </div>
                <div className='p-8 bg-dry rounded-lg'>
                  <span className='text-3xl text-white block font-extrabold'>
                    8 000 +
                  </span>
                  <h4 className='text-lg  text-white font-semibold my-
1'>Користувачів</h4>
                  <p className='mb-0 text-text leading-7 text-sm '>
```

```
                  Раді бачити нових користувачів!
                </p>
              </div>
            </div>
          </div>
          <div className='mt-10 lg:mt-0'>
            <img src='/images/descAbout.jpg' alt='aboutus' className='w-
full xl:block hidden h-header rounded-lg object-cover' />
          </div>
        </div>
      </div>
    </div>
  </Layout>
 )
}

export default AboutUs
```

Лістинг 8 – Файл ContactUs.js

```
import React from 'react'
import Layout from '../Layout/Layout'
import { FiMail, FiMapPin } from 'react-icons/fi'
import HeadContact from '../Components/HeadContact'

function ContactUs() {
  const ContactData = [
    {
      id: 1,
      title: "Електронна адреса",
      info: "",
      icon: FiMail,
      contact: "book@gmail.com",
    },
    {
      id: 2,
      title: "Локація",
      info: "Ukraine",
      icon: FiMapPin,
      contact: "",
    },
  ]
  return (
    <Layout>
      <div className='min-height-screen container mx-auto px-2 my-6'>
        <HeadContact title="" />
        <div className='mt-20 text-sm leading-8 text-mainText'>
          <p>
            Інформація, що розміщена на нашому веб-сайті, є загальнодоступною
у мережах Інтернету.
            У випадку, якщо наш ресурс порушив ваші авторські права, будь
ласка, звертайтеся до адміністрації
```

```
            сайту за електронною адресою вказаною нижче, і ми оперативно
видалимо відповідний контент.
          </p>
        </div>
        <div className='grid mg:grid-cols-2 gap-6 lg:my-20 my-10 lg:grig-
cols-2 xl:gap-8'>
          <div className='grid grid-flow-row xl:grid-cols-2 text-white gap-4
xl:gap-16 items-center'>
            {
              ContactData.map((item) => (
                <div key={ item.id } className="border border-main flex-colo
p-10 bg-dry rounded-lg text-center">
                  <span className='flex-colo w-20 h-20 mb-4 rounded-full bg-
main text-mainText text-2xl'>
                    <item.icon />
                  </span>
                  <h5 className='text-xl font-semibold mb-2'>{ item.title
}</h5>
                  <p className='mb-0 text-sm text-white leading-7'>
                    <a href={ `mailto:{item.contact}` } className='text-
subMain'>
                      { item.contact }
                    </a>
                    { item.info }
                  </p>
                </div>
              ))
            }
          </div>
        </div>
      </div>
    </Layout>
  )
}

export default ContactUs
```

Лістинг 9 – Файл Product.js

```
import React from 'react'
import { Link } from 'react-router-dom'
import { GrBasket } from "react-icons/gr";
import { useDispatch, useSelector } from 'react-redux';
import { IfProductLiked, LikeProduct } from '../Context/Functionalities';

function Product({ product }) {
  const { isLoading } = useSelector((state) => state.userLikeProduct);
  const dispatch = useDispatch();
  const { userInfo } = useSelector((state) => state.userLogin);

  // if liked function
  const isLiked = IfProductLiked(product);
```

```
  return (
    <>
      <div className='p-3 bg-main rounded-lg hover:scale-95 transitions
relative overflow-hidden'>
        <Link to={ `/product/${product?._id}` } className='w-full'>
          <img src={ product?.image ? product?.image : "/images/user.jpg" }
alt={ product?.name } className='h-86 w-full rounded-lg object-cover' />
        </Link>
        <div className='absolute flex-btn gap-2 bottom-0 right-0 left-2 bg-
main bg-opacity-60 text-mainText px-10 py-5'>
          <h3 className='font-semibold truncate'>{ product?.name }</h3>

          <button
            onClick={ () => LikeProduct(product, dispatch, userInfo) }
            disabled={ isLiked || isLoading }
            className={ `min-h-10 min-w-10  text-sm flex-colo transitions
           ${isLiked ? "bg-red-500 border-white" : "bg-subMain text-mainText
"}
            hover:bg-transparent rounded-full border-2 border-subMain text-
mainText`}
          >
            <div className='text-md'>< GrBasket className='h-5 w-5' /></div>
          </button>

        </div>
      </div>
    </>
  )
}

export default Product
```

Лістинг 10 – Файл Banner.js

```
import React from 'react';
import { Swiper, SwiperSlide } from 'swiper/react';
import { Autoplay } from 'swiper/modules';
import FlexProductItems from '../FlexProductItems';
import { Link } from 'react-router-dom';
import { GrBasket } from "react-icons/gr";
import Loader from '../../../src/Components/Notification/Loader';
import { TbNotebookOff } from "react-icons/tb";
import { useDispatch, useSelector } from 'react-redux';
import { IfProductLiked, LikeProduct } from '../../Context/Functionalities';

const Swipper = ({ sameClass, products }) => {
  const { isLoading } = useSelector((state) => state.userLikeProduct);
  const dispatch = useDispatch();
  const { userInfo } = useSelector((state) => state.userLogin);

  // if liked function
  const isLiked = (product) => {
    return IfProductLiked(product);
```

```
  };

  return (
    <Swiper
      direction="vertical"
      slidesPerView={ 1 }
      loop={ true }
      speed={ 1000 }
      modules={ [Autoplay] }
      autoplay={ { delay: 4000, disableOnInteraction: false } }
      className={ sameClass }
    >
      { products?.slice(0.6).map((product, index) => (
        <SwiperSlide key={ index } className='relative rounded overflow-
hidden'>
          <img src={ product?.titleImage ? product.titleImage :
"/images/user.jpg" } alt={ product?.name } className='w-full h-full object-
cover' />
          <div className='absolute linear-bg xl:pl-52 sm:pl-32 pl-8 top-0
bottom-0 right-0 left-0 flex flex-col justify-center lg:gap-8 md:gap-5 gap-
4'>
            <h1 className='xl:text-4xl truncate text-white capitaliz font-
sans sm:text-2xl text-xl font-bold'>
              { product?.name }
            </h1>
            <div className='flex gap-5 items-center text-white'>
              <FlexProductItems product={ product } />
            </div>
            <div className='flex gap-5 items-center text-white'>
              <Link to={ `/product/${product?._id}` } className='bg-subMain
hover:text-main transitions text-main px-8 py-3 rounded font-medium sm:text-
sm text-xs'>
                Детальніше...
              </Link>
              <button
                onClick={ () => LikeProduct(product, dispatch, userInfo) }
                disabled={ isLiked(product) || isLoading }
                className={ `bg-white
              ${isLiked(product) ? "text-red-500" : "text-white"}
               hover:text-red-500 transitions px-4 py-3 rounded text-sm bg-
opacity-30`}
              >
                <GrBasket className='h-5 w-5' />
              </button>
            </div>
          </div>
        </SwiperSlide>
      ))
      }
    </Swiper>
  )
}

function Banner({ products, isLoading }) {
```

```
  const sameClass = 'w-full flex-colo xl:h-128 bg-dry lg:h-96 h-64'
  return (
    <div className="relative w-full">
      { isLoading ? (
        <div className={ sameClass }>
          <Loader />
        </div>
      ) : products?.length > 0 ? (
        <Swipper sameClass={ sameClass } products={ products } />
      ) : (
        <div className={ sameClass }>
          <div className='flex-colo w-24 h-24 p-5 mb-4 rounded-full bg-dry
text-subMain text-4xl'>
            <TbNotebookOff />
          </div>
          <p className='text-main text-sm'>На цьому сайті не має товарів</p>
        </div>
      ) }
    </div >
  )
}

export default Banner
```

Лістинг 11 – Файл CategoriesActions.js

```
import * as CategoriesConstants from '../Constants/CategoriesConstants'
import * as CategoriesAPIs from '../APIs/CategoriesServices'
import toast from 'react-hot-toast'
import { ErrorsAction, tokenProtection } from '../Protection'

// Get all Categories action
export const getAllCategoriesAction = () => async (dispatch) => {
  try {
    dispatch({ type: CategoriesConstants.GET_ALL_CATEGORIES_REQUEST });
    const data = await CategoriesAPIs.getCategoriesService();
    dispatch({ type: CategoriesConstants.GET_ALL_CATEGORIES_SUCCESS, payload:
data });
  } catch (error) {
    ErrorsAction(error, dispatch,
CategoriesConstants.GET_ALL_CATEGORIES_FAIL)
  }
}

// Create Category action
export const createCategoryAction = (title) => async (dispatch, getState) =>
{
  try {
    dispatch({ type: CategoriesConstants.CREATE_CATEGORIES_REQUEST });
    await CategoriesAPIs.createCategoryService(title,
tokenProtection(getState));
    dispatch({ type: CategoriesConstants.CREATE_CATEGORIES_SUCCESS });
    toast.success("Категорія успішно створена");
```

```
  } catch (error) {
    ErrorsAction(error, dispatch,
CategoriesConstants.CREATE_CATEGORIES_FAIL);
  }
}

// Update Category action
export const updateCategoryAction = (id, title) => async (dispatch, getState)
=> {
  try {
    dispatch({ type: CategoriesConstants.UPDATE_CATEGORIES_REQUEST });
    await CategoriesAPIs.updateCategoryService(id, title,
tokenProtection(getState)
    );
    dispatch({ type: CategoriesConstants.UPDATE_CATEGORIES_SUCCESS });
    toast.success("Категорія успішно редагована")
    dispatch(getAllCategoriesAction())
  } catch (error) {
    ErrorsAction(error, dispatch,
CategoriesConstants.UPDATE_CATEGORIES_FAIL);
  }
}

// Delete Category action
export const deleteCategoryAction = (id) => async (dispatch, getState) => {
  try {
    dispatch({ type: CategoriesConstants.DELETE_CATEGORIES_REQUEST });
    await CategoriesAPIs.deleteCategoryService(id,
tokenProtection(getState));
    dispatch({ type: CategoriesConstants.DELETE_CATEGORIES_SUCCESS });
    toast.success("Категорія успішно видалена");
    dispatch(getAllCategoriesAction())
  } catch (error) {
    ErrorsAction(error, dispatch,
CategoriesConstants.DELETE_CATEGORIES_FAIL);
  }
}
```

Лістинг 12 – Файл ProductsActions.js

```
import * as productsConstants from '../Constants/ProductsConstants';
import * as productsAPIs from '../APIs/ProductsServices'
import { ErrorsAction, tokenProtection } from '../Protection'
import toast from 'react-hot-toast';


// get all product action
export const getAllProductsAction =
  ({
    category = "",
    time = "",
    language = "",
    rate = "",
```

```
    year = "",
    search = "",
    pageNumber = "",
  }) =>
    async (dispatch) => {
      try {
        dispatch({ type: productsConstants.PRODUCTS_LIST_REQUEST });
        const response = await productsAPIs.getAllProductsService(
          category,
          time,
          language,
          rate,
          year,
          search,
          pageNumber
        );
        dispatch({
          type: productsConstants.PRODUCTS_LIST_SUCCESS,
          payload: response,
        });
      } catch (error) {
        ErrorsAction(error, dispatch, productsConstants.PRODUCTS_LIST_FAIL);
      }
    };

// get random product action
export const getRandomProductsAction = () => async (dispatch) => {
  try {
    dispatch({ type: productsConstants.PRODUCTS_RANDOM_REQUEST });
    const response = await productsAPIs.getRandomProductsService();
    dispatch({
      type: productsConstants.PRODUCTS_RANDOM_SUCCESS,
      payload: response,
    });
  } catch (error) {
    ErrorsAction(error, dispatch, productsConstants.PRODUCTS_RANDOM_FAIL);
  }
};

// get product by id action
export const getProductByIdAction = (id) => async (dispatch) => {
  try {
    dispatch({ type: productsConstants.PRODUCT_DETAILS_REQUEST });
    const response = await productsAPIs.getProductByIdService(id);
    dispatch({
      type: productsConstants.PRODUCT_DETAILS_SUCCESS,
      payload: response,
    });
  } catch (error) {
    ErrorsAction(error, dispatch, productsConstants.PRODUCT_DETAILS_FAIL);
  }
};

// get top rated product action
```

```
export const getTopRatedProductAction = () => async (dispatch) => {
  try {
    dispatch({ type: productsConstants.PRODUCT_TOP_RATED_REQUEST });
    const response = await productsAPIs.getTopRatedProductService();
    dispatch({
      type: productsConstants.PRODUCT_TOP_RATED_SUCCESS,
      payload: response,
    });
  } catch (error) {
    ErrorsAction(error, dispatch, productsConstants.PRODUCT_TOP_RATED_FAIL);
  }
};

// review product action
export const reviewProductAction = ({ id, review }) => async (dispatch,
getState) => {
  try {
    dispatch({ type: productsConstants.CREATE_REVIEW_REQUEST });
    const response = await productsAPIs.reviewProductService(
      tokenProtection(getState),
      id,
      review
    );
    dispatch({
      type: productsConstants.CREATE_REVIEW_SUCCESS,
      payload: response,
    });
    toast.success("Коментар успішно створено✨😈");
    dispatch({ type: productsConstants.CREATE_REVIEW_RESET });
    dispatch(getProductByIdAction(id));
  } catch (error) {
    ErrorsAction(error, dispatch, productsConstants.CREATE_REVIEW_FAIL);
  }
};

// delete product action
export const deleteProductAction = (id) => async (dispatch, getState) => {
  try {
    dispatch({ type: productsConstants.DELETE_PRODUCT_REQUEST });
    const response = await productsAPIs.deleteProductService(
      tokenProtection(getState),
      id
    );
    dispatch({
      type: productsConstants.DELETE_PRODUCT_SUCCESS,
      payload: response,
    });
    toast.success("Товар успішно видалено");
    dispatch(getAllProductsAction({}));
  } catch (error) {
    ErrorsAction(error, dispatch, productsConstants.DELETE_PRODUCT_FAIL);
  }
};
```

```
// delete all products action
export const deleteAllProductsAction = () => async (dispatch, getState) => {
  try {
    dispatch({ type: productsConstants.DELETE_ALL_PRODUCTS_REQUEST });
    const response = await productsAPIs.deleteAllProductsService(
      tokenProtection(getState)
    );
    dispatch({
      type: productsConstants.DELETE_ALL_PRODUCTS_SUCCESS,
      payload: response,
    });
    toast.success("Всі товари видалено😈");
    dispatch(getAllProductsAction({}));
  } catch (error) {
    ErrorsAction(error, dispatch,
productsConstants.DELETE_ALL_PRODUCTS_FAIL);
  }
};

// create product action
export const createProductAction = (product) => async (dispatch, getState) =>
{
  try {
    dispatch({ type: productsConstants.CREATE_PRODUCT_REQUEST });
    const response = await productsAPIs.createProductService(
      tokenProtection(getState),
      product
    );
    dispatch({
      type: productsConstants.CREATE_PRODUCT_SUCCESS,
      payload: response,
    });
    toast.success("Товар успішно створено👍");
    dispatch(deleteAllCastAction());
  } catch (error) {
    ErrorsAction(error, dispatch, productsConstants.CREATE_PRODUCT_FAIL);
  }
};

// *******CASTS**********

// add cast
export const addCastAction = (cast) => async (dispatch, getState) => {
  dispatch({ type: productsConstants.ADD_CAST, payload: cast });
  localStorage.setItem("casts", JSON.stringify(getState().casts.casts));
};

// remove cast
export const removeCastAction = (id) => async (dispatch, getState) => {
  dispatch({ type: productsConstants.DELETE_CAST, payload: id });
  localStorage.setItem("casts", JSON.stringify(getState().casts.casts));
};
```

```
// update cast
export const updateCastAction = (cast) => async (dispatch, getState) => {
  dispatch({ type: productsConstants.EDIT_CAST, payload: cast });
  localStorage.setItem("casts", JSON.stringify(getState().casts.casts));
};

// delete all cast
export const deleteAllCastAction = () => async (dispatch) => {
  dispatch({ type: productsConstants.RESET_CAST });
  localStorage.removeItem("casts");
};

// Update product action
export const updateProductAction = (id, product) => async (dispatch,
getState) => {
  try {
    dispatch({ type: productsConstants.UPDATE_PRODUCT_REQUEST });
    const response = await productsAPIs.updateProductService(
      tokenProtection(getState),
      id,
      product
    );
    dispatch({
      type: productsConstants.UPDATE_PRODUCT_SUCCESS,
      payload: response,
    });
    toast.success("Товар відредаговано успішно");
    dispatch(getProductByIdAction(id));
    dispatch(deleteAllCastAction());
  } catch (error) {
    ErrorsAction(error, dispatch, productsConstants.UPDATE_PRODUCT_FAIL);
  }
};
```

Лістинг 13 – Файл ProductsRates.js

```
import React, { useEffect } from 'react'
import Titles from '../Titles'
import { BsBookmarkStarFill } from 'react-icons/bs'
import { Message, Select } from '../UsedInputs';
import Rating from '../Stars';
import { Empty } from './../Notification/Empty';
import { useDispatch, useSelector } from 'react-redux';
import { yupResolver } from '@hookform/resolvers/yup';
import { useForm } from 'react-hook-form';
import { ReviewValidation } from '../Validation/ProductValidation';
import toast from 'react-hot-toast';
import { InlineError } from '../Notification/Error';
import { Link } from 'react-router-dom';
import { reviewProductAction } from '../../Redux/Actions/ProductsActions';

const Ratings = [
```

```
  {
    title: '0 - Жахливо😒',
    value: 0,
  },
  {
    title: '1 - Посередньо😑',
    value: 1,
  },
  {
    title: '2 - Добре😌',
    value: 2,
  },
  {
    title: '3 - Дуже Добре😊',
    value: 3,
  },
  {
    title: '4 - Відмінно😁',
    value: 4,
  },
  {
    title: '5 - Шедевр😍',
    value: 5,
  },
];

function ProductsRates({ product }) {
  const dispatch = useDispatch();
  // use Selector
  const { isLoading, isError } = useSelector(
    (state) => state.createReview
  );
  const { userInfo } = useSelector(
    (state) => state.userLogin
  );

  //validate review
  const {
    register,
    handleSubmit,
    watch,
    formState: { errors }
  } = useForm({
    resolver: yupResolver(ReviewValidation),
  })

  // on submit
  const onSubmit = (data) => {
    dispatch(
      reviewProductAction({
        id: product?._id,
        review: data,
```

```
      })
    );
  };

  useEffect(() => {
    if (isError) {
      toast.error(isError)
      dispatch({ type: "CREATE_REVIEW_RESET" })
    }
  }, [isError, dispatch])

  return (
    <div className='my-12'>
      <Titles title='Враження після покупки' Icon={ BsBookmarkStarFill } />
      <div className='mt-10 xl:grid flex-colo grid-cols-5 gap-12 bg-dry xs:p-
10 py-10 px-2 sm:p-20 rounded'>
        {/* write review */ }
        <form onSubmit={ handleSubmit(onSubmit) } className='xl:col-span-2 w-
full flex flex-col gap-8'>
          <h3 className='text-xl text-subMain font-semibold'>
            Прокоментувати '{ product?.name }'
          </h3>
          <p className='text-sm leading-7 font-medium text-text'>
            Напишіть коментар до цього товару. Він буде опублікований на цій
сторінці.
          </p>
          <div className='text-sm text-mainText w-full'>
            <Select
              label='Обрати оцінку'
              options={ Ratings }
              name="rating"
              register={ { ...register("rating") } }
            />
            <div className='flex mt-4 text-lg gap-2 text-star'>
              <Rating value={ watch("rating", false) } />
            </div>
            { errors.rating && <InlineError message={ errors.rating.message }
/> }
          </div>
          {/* message */ }
          <div className="w-full">
            <Message
              name="comment"
              register={ { ...register("comment") } }
              label="Коментар"
              placeholder="Залишити коментар..."
            />
            { errors.comment && <InlineError text={ errors.comment.message }
/> }
          </div>

          {/* submit */ }
          { userInfo ? (
            <button
```

```
              disabled={ isLoading }
              type="submit"
              className="bg-subMain text-main py-4 w-full flex-colo rounded"
            >
              { isLoading ? "Завантаження..." : "Відправити" }
            </button>
          ) : (
            <Link
              to="/login"
              className="bg-main border border-dashed border-border text-
subMain py-4 w-full flex-colo rounded"
            >
              Ділитися враженнями можуть лише зареєстровані користувачі
            </Link>
          ) }
        </form>
        {/* Reviwers */ }
        <div className='col-span-3 flex w-full flex-col gap-6'>
          <h3 className='text-xl text-text font-semibold'>Коментарі ({
product?.numberOfReviews })</h3>
          <div className='w-full flex flex-col bg-main gap-6 rounded-lg md:p-
12 p-6 h-header overflow-y-scroll'>
            { product?.reviews?.length > 0 ? product?.reviews?.map((review)
=> (
              <div key={ review?._id } className='md:grid flex flex-col w-
full grid-cols-12 gap-6 bg-dry p-4 border border-main rounded-lg'>
                <div className='col-span-2 bg-dry hidden md:block'>
                  <img src={ review?.userImage ? review?.userImage :
'user.jpg' } alt={ review?.userName } className='w-16 h-16 rounded-lg object-
cover' />
                </div>
                <div className='col-span-7 flex flex-col gap-2'>
                  <h2>{ review?.userName }</h2>
                  <p className='text-xs leading-6 font-medium text-text'>
                    { review?.comment }
                  </p>
                </div>
                {/* rates */ }
                <div className='col-span-3 flex-rows border-l border-border
text-xs gap-1 text-star'>
                  <Rating value={ review?.rating } />
                </div>
              </div>
            )) : <Empty message={ `Станьте першим хто прокоментує
"${product?.name}"` } />
            }
          </div>
        </div>
      </div>
    </div>
  )
}

export default ProductsRates
```

ДОДАТОК Б

Диск з роботою