m-math_2025_011

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЧЕРНІВЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ЮРІЯ ФЕДЬКОВИЧА

Факультет математики та інформатики
кафедра математичного моделювання

ОПТИМІЗАЦІЯ ФУНКЦІОНАЛУ ВЕБ-ДОДАТКУ ДЛЯ ПРОДАЖУ АВТОМОБІЛЬНИХ ШИН

Кваліфікаційна робота

Рівень вищої освіти – другий (магістерський)

Виконав:
студент 2 курсу, 601 групи
Резнік Володимир Русланович
Керівник:
кандидат фізико-математичних наук,
доцент **Піддубна Лариса Андріївна**

До захисту допущено
на засіданні кафедри
протокол № 5 від 26 листопада 2024 р.
Зав. кафедрою _____ проф. Черевко І.М.

Чернівці – 2024

АНОТАЦІЯ

У роботі детально розглянуті етапи опти нового функціоналу веб-додатку для продажу автомобільних шин, використовуючи сучасні методи розробки динамічних веб-додатків.

Процес розробки додатку відбувався з використанням стеку розробки “MERN”, основною мовою розробки якого є JavaScript. Протягом розробки було спроектовано та розроблено клієнтську частину веб-додатку, адміністративну панель та деякі серверні команди для зручності роботи.

Ключові слова: JavaScript, веб-додаток, розробка, React.

ABSTRACT

The paper describes in detail the stages of developing a web application for selling car tires using modern methods of developing dynamic web applications.

The application development process was carried out using the MERN development stack, the main development language of which is JavaScript. During the development, the client part of the web application, the administrative panel and some server commands were designed and developed for the convenience of work.

Key words: JavaScript, web application, development, React,

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

_____ В.Р. Резнік

(підпис)

Зміст

ВСТУП

У сучасному світі стрімко зростає популярність онлайн-торгівлі. Пандемія та військові дії значно змінили поведінку споживачів, зробивши дистанційні покупки найбільш безпечним і зручним способом придбання товарів і послуг. Згідно зі статистикою, кожна третя людина здійснює покупки онлайн, що свідчить про високу актуальність дослідження в цій сфері. [1]

Разом із зростанням кількості покупців онлайн зростає і кількість платформ для продажу та моніторингу товарів. Використання маркетплейсів, платформ для оголошень і e-commerce рішень стало невід'ємною частиною сучасної комерції. Проте існуючі платформи мають обмеження, які часто не дозволяють продавцям повністю задовольнити свої потреби в налаштуванні структури, аналітики та SEO-оптимізації.

Актуальність теми полягає в тому, що розробка власного веб-додатку для продажу автомобільних шин може забезпечити необхідний рівень гнучкості, прозорості та ефективності. Такий підхід дозволяє не лише вирізнятись серед конкурентів, але й забезпечувати краще управління процесами продажу, моніторингу та аналізу.

Метою даної роботи є доопрацювання веб-додатку для продажу автомобільних шин, який забезпечує користувачам зручний інтерфейс, адаптивність до потреб бізнесу, а також гнучкі інструменти аналітики та SEO. Для досягнення поставленої мети необхідно вирішити такі завдання:

- проаналізувати існуючі рішення у сфері e-commerce;
- розробити зручний інтерфейс
- впровадити сучасні інструменти для моніторингу та аналітики;
- протестувати розроблену систему та оцінити її ефективність.

Основним методом досліджень є порівняльний аналіз існуючих платформ, а також методи програмної розробки для створення та тестування веб-додатку.

Розділ 1. ПОСТАНОВКА ЗАДАЧІ

Постановку задачі можна розділити на наступні підзадачі:

- актуалізація проекту: оновлення версій плагінів та бібліотек, що використовуються при розробці;
- виправлення помилок попередньої версії, враховуючи баг-репорти;
- розробка нових функцій для клієнтської та серверної частини проекту;
- розробка серверних команд для зручності взаємодії з додатком;
- тестування готового продукту

Задача полягає в створенні кінцевої версії повноцінного веб-додатку для онлайн продажу автомобільних шин: клієнтська версія, панель для адміністраторів, серверні налаштування та команди. В порівнянні з початковою версією цього додатку, поточна версія дозволить користувачам клієнтської частини зручніше працювати з сайтом, уникаючи проблем, які постійно з'являються в початковой версії.

При розробці найважливішу увагу необхідно приділяти оптимізації запитів та зручності користування сайтом. Оскільки фронтен побудований на ReactJS, це дозволяє в реальному часі оновлювати інформацію, тому потрібно грамотно здійснювати та обробляти запити, отримувати та відображати результат, особливо при взаємодії з товарами: пошук, додаванням, видаленням товарів із кошика. Однією з основних проблем початкової версії додатку була відсутність комунікації з користувачами під час запитів на сервер, що приводило до зависань та не передбачуваних результатів роботи додатку.

Після реалізації та оптимізації усього функціоналу треба відповідально підійти до тестування готового додатку, враховуючи усі баги, що були зафіксовані при тестуванні попередньої версії проекту.

Розділ 2. ОПИС ВИКОРИСТАНИХ ТЕХНОЛОГІЙ

2.1. MongoDB

MongoDB — це одна з найпопулярніших нереляційних баз даних, яка належить до категорії NoSQL. Її ключовими перевагами є:

- **Гнучкість у роботі з даними**. MongoDB дозволяє зберігати інформацію у форматі JSON із довільною структурою. Це забезпечує високу адаптивність під час розробки, оскільки не потрібно заздалегідь задавати жорстку схему бази даних.
- **Висока продуктивність**. Завдяки оптимізованим процесам, таким як індексація, кешування та розподілені запити, MongoDB демонструє відмінну швидкодію при роботі з великими обсягами даних.
- **Зручність використання.** База даних надає простий та інтуїтивний API, що спрощує виконання операцій із даними за допомогою її власної мови запитів.
- **Масштабованість**. MongoDB підтримує горизонтальне масштабування, яке дозволяє розподіляти дані між кількома серверами. Це забезпечує ефективну обробку значних обсягів інформації шляхом додавання нових серверів до кластеру, що робить MongoDB ідеальним рішенням для високонавантажених додатків.
- Простота **інтеграції** з Node.js та Express.js.

Також слід звернути увагу на Mongoose — об'єктно-документний моделювальник (ODM) для роботи з MongoDB у середовищі Node.js. [2] Mongoose забезпечує зручну взаємодію з базою даних через JavaScript-об'єкти. Його основне призначення — спростити процес створення додатків, які використовують MongoDB, за рахунок введення структурованих моделей даних, валідації, а також зручного API для запитів та інших функцій.

2.2. Express JS

Основна роль Express полягає в розробці веб-серверів і API для взаємодії з клієнтською стороною. Він спрощує процес розробки, дозволяючи зосередитися на реалізації маршрутизації та логіки додатку, забезпечуючи зручну роботу з HTTP-запитами і файлами. [3]

Основні можливості Express.js:

- **Розробка API**: Express дозволяє створювати маршрути для обробки HTTP-запитів (GET, POST, PUT, DELETE), що забезпечує ефективний обмін даними між клієнтом і сервером.
- **Взаємодія з базою даних**: За допомогою Express можна легко інтегруватися з MongoDB, використовуючи офіційний драйвер або бібліотеки, такі як Mongoose. Це полегшує створення маршрутів і контролерів для виконання CRUD-операцій (створення, читання, оновлення та видалення даних).
- **Обробка статичних файлів**: Фреймворк дозволяє зручно працювати зі статичними файлами, такими як HTML, CSS, зображення та інші медіа. Це особливо корисно при розгортанні фронтенду, створеного на React, який може функціонувати окремо від серверної частини.
- **Інтеграція** з іншими пакетами: Express має багату екосистему додаткових модулів. Наприклад, можна використовувати Passport.js для аутентифікації, Express-validator для валідації даних, Redis або Memcached для кешування. Це дозволяє легко розширювати функціональність додатку.
- Загалом, Express.js — це потужний інструмент для швидкої розробки серверної частини веб-додатків у стеку MERN. Він сприяє зосередженню на логіці додатка, забезпечуючи ефективну обробку запитів і формування відповідей для клієнтів.

2.3. React JS

Весь інтерфейс розроблений за допомогою бібліотеки React — одного з найпопулярніших інструментів для створення веб-додатків. Створений компанією Facebook, React має велику екосистему, яка включає численні документації, плагіни, пакети та активну спільноту.

Головна концепція React — компонентна архітектура. Кожен елемент інтерфейсу представлений як окремий компонент, який можна повторно використовувати. Компоненти можуть мати власний стан і оновлювати відображення в залежності від його змін, що дозволяє динамічно керувати інтерфейсом.

Для оптимізації оновлень React використовує віртуальну DOM-модель, яка дозволяє уникати зайвих операцій із реальною DOM. Це значно підвищує продуктивність і знижує навантаження на додаток.

Основу синтаксису React становить JSX — синтаксис, який поєднує HTML-подібний код із JavaScript. JSX спрощує створення компонентів, дозволяючи зосередити розмітку, обробники подій і логіку в одному місці. Він також підтримує умовний ітераційний рендеринг, дозволяючи використовувати такі оператори, як if, switch, for або map.

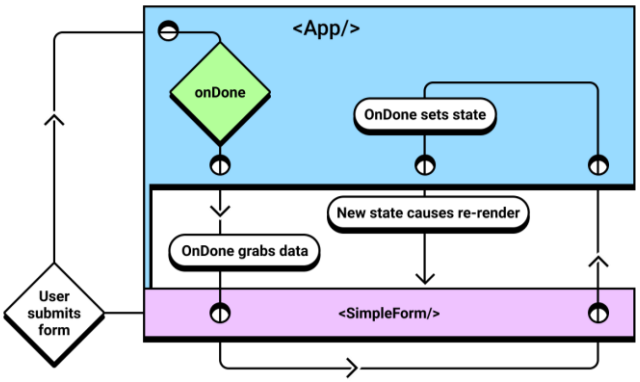

Рис. 2.1: Однонаправлений потік даних у React JS [4]

Основні завдання React у стеку MERN:

- Створення компонентів інтерфейсу: сторінок, форм, кнопок тощо.
- Реалізація навігації: за допомогою бібліотек, таких як react-router-dom.
- Управління станом: відстеження та оновлення стану компонентів.
- Взаємодія з сервером: за допомогою AJAX-запитів або RESTful API для обміну даними без перезавантаження сторінки.

React також містить потужні інструменти, такі як хуки і методи життєвого циклу, які спрощують управління станом та відстеження змін.

Загалом, React відіг рає ключову роль у стеку MERN, забезпечуючи ефективну розробку інтерфейсу та інтеграцію з серверною частиною додатків.

2.4. Node JS

Node.js — це серверне середовище виконання JavaScript, побудоване на основі двигуна V8 від браузера Chrome. Воно дозволяє виконувати JavaScript-код на серверній стороні, що раніше було можливим лише в клієнтському середовищі браузера. [5]

Основні переваги Node.js:

- Швидкодія
    - Асинхронна модель вводу/виводу забезпечує оптимальне використання серверних ресурсів, дозволяючи швидко обробляти запити.
- Масштабованість
    - Завдяки неблокуючій архітектурі Node.js може ефективно працювати з тисячами одночасних з'єднань.
    - Підтримує горизонтальне масштабування, що дозволяє розподіляти навантаження між кількома серверами.

- Єдина мова програмування
    - Використання JavaScript як на серверній, так і на клієнтській стороні значно спрощує розробку, дозволяє ділитися кодом між компонентами і скорочує час створення додатків.
- Розвинута екосистема
    - Широка спільнота розробників сприяє створенню численних модулів та пакетів, які можна легко інтегрувати через Node Package Manager (NPM). Це підвищує ефективність розробки та скорочує час реалізації функціоналу.
- Розширюваність
    - Просте управління залежностями та додавання сторонніх бібліотек для розширення функцій додатку.

У рамках стеку MERN Node.js виконує роль серверної складової. Його головна функція — створення серверної логіки та API для зв'язку клієнта з базою даних MongoDB. Разом із фреймворком Express Node.js дозволяє:

- Обробляти HTTP-запити клієнтів.
- Взаємодіяти з базою даних.
- Реалізовувати бізнес-логіку.

Node.js забезпечує побудову потужних і масштабованих серверних додатків, підтримуючи швидкий зв'язок між клієнтом та сервером.

Висновки до розділу 2

Стек MERN є сучасним і потужним інструментом для створення веб-додатків, оскільки він охоплює як клієнтську, так і серверну частину розробки. Використання MongoDB, Express, React і Node.js забезпечує високу продуктивність, гнучкість та масштабованість додатку. Компонентна архітектура React спрощує розробку інтерфейсів, а Express та Node.js забезпечують ефективну взаємодію з сервером. Завдяки синхронній роботі всіх компонентів стеку, MERN дозволяє створювати динамічні, інтерактивні веб-додатки з високою продуктивністю та зручністю в управлінні даними.

Розділ 3. ОПИС РОБОТИ РЕАЛІЗОВАНОГО ПРОЕКТУ

3.1. Загальна файлова структура проекту

Стек MERN надає гнучкі можливості по організації файлової структури при розробці. Відповідно, кількість директорій та файлів в проекті залежить від його складності, масштабів проекту, кількості використання бібліотек та плагінів.

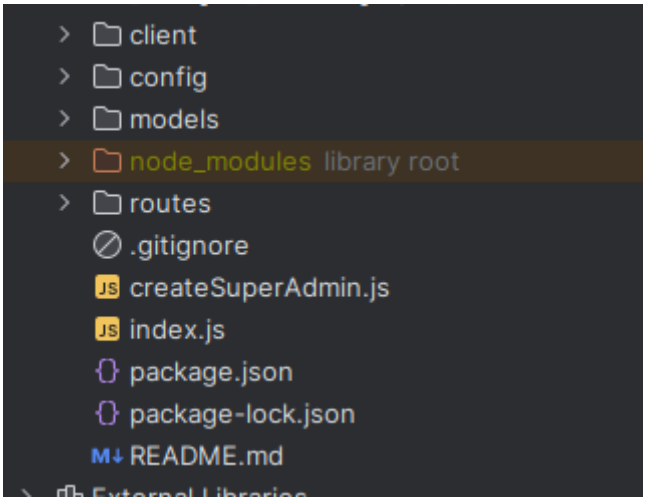

Рис. 3.1: Структура проекту для веб-додатку

Структура проекту складається з наступних обов’язкових елементів:

- директорія client - повністю містить фронтенд-частину веб додатку. Вона має свій package.json файл - файл, в якому встановлюються залежності, версії пакетів, скрипти, що використовуються виключно для клієнтської частини проекту та запускається незалежно від серверної частини.
- директорія config - містить файли конфігів - їх кількість не обмежена, але це надає зручність при розробці, коли нам слід використовувати різні порти, токени авторизації та інші дані.
- директорія models - містить моделі кожної сутності, що використовуються в проекті. Це користувач, кошик товарів, шина (головна одиниця товару), бренд-виробник. Вони визначають структуру

колекції документів MongoDB, включаючи поля, типи даних, валідацію,

значення за замовчуванням. Надає можливість виконувати операції CRUD (створення, читання, оновлення та видалення) з базою даних MongoDB завдяки зручним методам.

- Node Modules - директорія, що містить встановленні файли усіх зовнішніх пакетів та бібліотек, що встановлюються при ініціалізації проекту командою ***npm install (npm i)***. Назви та версії пакетів, необхідні для встановлення, зберігаються у JSON файлі - package.json. В ньому ж описуються всі команди, що можна запустити завдяки npm.
- Router - директорія, в якій розташовуються посилання для CRUD операцій, відповідно для кожної моделі - окремий файл. Всі ці файли імпортуються в головний серверний файл - app.js. Для роботи з кожною моделлю створюється окремий файл, це полегшує роботу з кодом, спрощує тестування та надає гнучкість в розробці.
- Файл .gitignore - містить дані про файли та директорії, які повинні ігноруватись при оновлені git-репозиторія. Зазвичай це директорія node modules, скомпільовані версії скриптів.
- Файл app.js - головний серверний файл:

Імпорт усіх необхідних пакетів та налаштувань для роботи

```
const config : Config | {...}  = require('config');
const mongoose :   = require('mongoose');
const PORT : value | number  = config.get('port') || 5000;
const app : any | Express  = express();
const path : PlatformPath | path  = require('path');

const cors : function(any): function(any, a... | {...}  = require('cors');
```

Підключення роутів та проміжних фукнцій

```
app.use(cors());
app.use(express.json( options: { extended: true }));
app.use('/api/auth', require('./routes/auth.routes'));
app.use('/api/cart', require('./routes/cart.router'))
app.use('/api/tire-crud', require('./routes/tire-crud.router'))
app.use('/api/brand-crud', require('./routes/brand-crud.router'))
app.use('/api/user-crud', require('./routes/user-crud.router'))
app.use(express.static(path.join(__dirname, './client/build')));
new *
app.get('*', (req, res) => {
   res.json( body: "STATUS 200");
});
```

Асинхронна функція для запуску додатку, де відбувається з'єднання з MongoDB

```
async function start() {
  try {
    await mongoose.connect(config.get('mongoUri'), options: {
      useNewUrlParser: true,
      useUnifiedTopology: true,
      useCreateIndex: true,
    });
    app.listen( port: 5000, callback: () => console.log(`App has been started on port ${PORT} `));
  } catch (e) {
    console.log('Server error', e.message);
    process.exit( code: 1);
  }
}
start();
```

Першочергово, імпортуються усі пакети, що використовуються при запуску серверної частини: express, config, path, mongoose. Також отримується порт для роботи додатку згідно обраної наразі конфігурації.

Далі відбувається ініціалізації Express, підключення усіх роутів, що розташовані в директорії routes. Наступний програмний код задає шляхи статичних файлів - це обов'язкова процедура при розгортанні веб-додатку, коли використовуються оптимізовані та стиснуті файли для деплою, що займають менше місця на сервері та швидше опрацьовуються.

Наступний етап це ініціалізація та виклик функції для запуску серверу, де встановлюється підключення до кластеру MongoDB згідно конфігураційного файлі та параметри підключення. Після чого сервер запускається та очікує запитів до вказаного порту. При помилці спрацьовує

конструкція catch де ми отримуємо опис помилки, що виникла на етапі запуску або під час використання серверу.

3.2. Файлова структура фронент частини

Типова структура веб-додатку, реалізованому на React: директорія public, в якій містяться статичні файли та медіа, директорія build - там зберігаються стиснуті файли, готові до деплою, директорія src - ключова, саме в ній розроблений весь функціонал додатку.

Також присутні файли package.json, package-lock.json, директорія node_modules - вони виконують ті самі функції, що і в корені проекту.

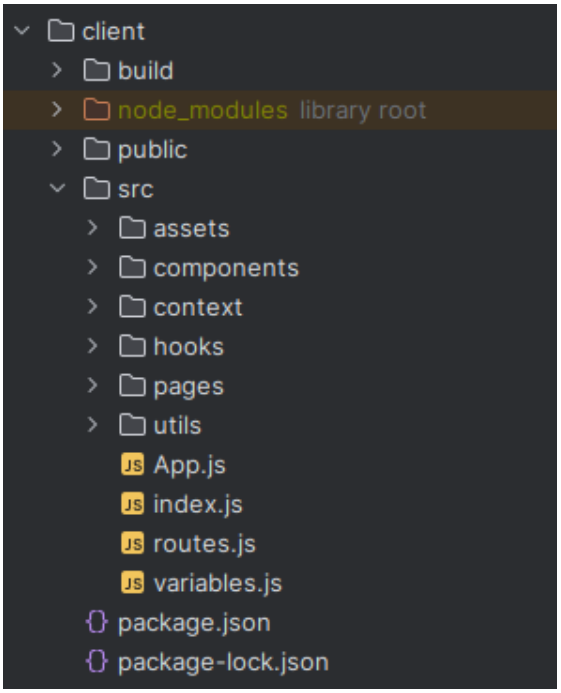

Рис. 3.2: Файлова структура фронент частини

Детально зупинимось на директоріях та файлах в директорії src, після чого, в наступному розділі, детально перейдем до опису їх застосування.

- Assets - туту зберігаються всі файли стилів, шрифтів, зображення та інше.
- Components - звична для React додатку директорія. Оскільки React - це бібліотека, орієнтовна на компоненти, там зберігаються компоненти, що використовуються у веб-додатку. Найкращим прикладом таких є

шапка та підвал сайту, навігація. Необхідність цієї директорії, як і ряду інших, залежить від масштабу проекту.

- Context - директорія, в якій зберігаються контексти - унікальні механізми, які дозволяють отримати доступ до змінних чи функцій на будь-якому рівні дерева компонентів веб-додатку, не передаючи їх всередину на кожному рівні. В цьому проекті був використаний контекст Auth - він передає дані користувача, функції для авторизації та виходу з акаунту - вони часто використовуються на різних рівнях додатку, тому механізм контексу значно оптимізує процес розробки та читабельність коду. [6]
- Hooks - містить власно написані хуки - функції, які дозволяють використовувати стан та інші можливості React в компонентах без класів. [7]
- Pages - директорія з сторінками сайту, базовими сторінками, контент яких будується за допомогою використання створених компонент.
- Utils - містить окремо винесені функції, які використовують в проекті неодноразово.
- Файл variables.js містить статичні змінні, що використовуються в проекті неодноразово. Це організовано для можливості імпорту замість дублювання коду.
- Файли App.js, index.js, routes.js - ключові файли роботи веб-додатку. Їх зміст та функціонал буде детальніше описано в наступному розділі.

3.3. Опис моделей та CRUD-роутів, що використовуються на серверній стороні

В цьому підрозділі буде детально розглянуто приклади організації моделей та роутів для CRUD - операцій. Як вже було згадано раніше, в проекті використовується 4 сутності. Зупинимось конкретніше на одній з них - на моделі Tire - шина (вона є основною одиницею товару на сайті): *Опис моделі Tire*

```
const { Schema, model } = require('mongoose');

const schema = new Schema({
  id: { type: String, require: true},
  article: {type: String, required: true, unique: true },
  brand: { type: String, require: true },
  diameter: {type: String, required: true},
  width: {type: String, required: true},
  height: {type: String, required: true},
  construction: {type: String, required: true},
  speedIndex: {type: String, required: true},
  countAvailable: {type: String},
  season: {type: String},
  image: {type: String},
  year: {type: String},
  price: {type: String}
});

module.exports = model('Tire', schema);
```

Моделі необхідні для опису структури документу, що використовується в базі даних MongoDB. Вона може мати безліч полів, проте кожне з ним обов'язково повинно мати тип даних. Додаткові атрибути це required - вказує на обов'язковість його визначення при записі в БД, unique - вказує на унікальність значення поля в БД. Також може мати інші параметри та методи: get, set, alias, select, validate тощо.

Приклад CRUD-роутів розглянемо для взаємодії з цією ж моделлю - моделлю Tire. Назви файли відповідають змісту - tire-crud та мають закінчення router.js, тобто *tire-crud.router.js*. Такий патерн назв файлів використовується для читабельності коду.

Підключення роутеру з бібліотеки express та імпорт моделі Tire

```
const { Router } = require('express');
const router = Router();
const Tire = require("../models/Tire");
```

Роут для отримання усіх екземплярів моделі Tire

```
router.get('/get-items', async (req, res) => {
  try{
    const userId = req.query.userId;

    const items = await Tire.find();

    return res.status(200).json({
      success: true,
      count: items.length,
      data: items,
    });
  } catch(err) {
    console.log(err);
    res.status(500).json({ error: 'Помилка севреру' });
  }
```

Роут для отримання екземплярів Tire, враховуючи пагінацію та сортування

```
router.get( path: '/get-items-sorting',  handlers: async (req, res) => {
    try{
        const sortBy = req.query.sortBy, sortOrder = req.query.sortOrder,
            limit = Number(req.query.limit), skip = Number(req.query.skip),
            search = req.query.search;
        sortObj[sortBy] = sortOrder === 'desc' ? -1 : 1;
        const _search = JSON.parse(search);
        const itemsWithoutPagination = await Tire.find(_search),
            itemsWithPagination = await Tire.find(_search).skip(skip)
                .limit(limit).sort(sortObj);
        return res.status( code: 200).json( body: {
            success: true,
            count: itemsWithoutPagination.length,
            data: itemsWithPagination,
        });
    } catch(err) {
        res.status( code: 500).json( body: { error: 'Помилка серверу' });
    }
});
```

Роут для отримання конкретного екземплярів Tire

```
router.get( path: '/get/:id',  handlers: async (req, res) => {
    try{
        const result = await Tire.findById(req.params.id);

        return res.status( code: 200).json( body: {
            success: true,
            data: result
        });
    } catch(err) {
        res.status( code: 500).json( body: { error: 'Помилка серверу' });
    }
});
```

Роут для оновлення конкретного екземплярів Tire

```
router.put( path: '/update/:id',  handlers: async (req, res) => {
    try{
        const update = await Tire.findByIdAndUpdate(req.params.id,  update: {
            $set: req.body
        });
        res.status( code: 200).json( body: { message: 'Товар було оновлено' });
    } catch(err) {
        res.status( code: 500).json( body: { error: 'Помилка серверу' });
    }
});
```

Роут для видалення конкретного екземплярів Tire та імпорт роутера

```
reznikkvova *
router.delete( path: '/delete/:id',  handlers: async (req, res) => {
    try{
        const item = await Tire.findByIdAndRemove(req.params.id);
        return res.status( code: 201).json( body: { message: 'Товар було видалено' });
    } catch(err) {
        res.status( code: 500).json( body: { error: 'Помилка серверу' });
    }
})

module.exports = router;
```

У проекті, використання Express значно спрощує роботу з запитами. Розглянемо приклад створення нового товару (шини) за допомогою моделі Tire. Для цього використовується метод POST, який передає дані через тіло

запиту. Значення полів форми з адмін-панелі деструктуризуються, а для поля article генерується унікальне значення за заданою схемою.

Процес обробки запиту реалізований через конструкцію try-catch:

- Спочатку за допомогою методу findOne перевіряється унікальність поля article. Якщо таке значення вже існує, сервер повертає код помилки *400* і повідомлення "Товар вже існує!", яке відображається в адмін-панелі.
- Якщо article унікальний, створюється новий товар і викликається метод save. У разі успішного збереження сервер повертає код *200* і повідомлення про успішну операцію.
- У разі виникнення помилки на сервері повертається код *500* з повідомленням "Помилка при створенні товару".

Інші CRUD-операції (отримання, оновлення, видалення) працюють за аналогічним принципом із використанням відповідних методів моделі:

- find — отримання всіх елементів моделі.
- skip та limit — реалізація пагінації.
- sort — сортування даних.
- findById — пошук за ідентифікатором.
- findByIdAndUpdate — оновлення даних за ID.
- findByIdAndRemove — видалення за ID.

Усі ці запити побудовані за шаблоном try-catch, де основна відмінність полягає у типі запиту, параметрах і даних, що повертаються. API Mongoose надає широкі можливості для роботи з базою даних, дозволяючи легко реалізовувати основні та додаткові функції.

3.4. Опис ключового фронтенд файлу, організації внутрішніх роутів та авторизації для фронтенд частини

Будь-який React-додаток починається з точки входу (entry-point), яка зазвичай називається index.js. Цей файл відповідає за ініціалізацію додатку і монтування React-компонентів у DOM, він виконується першим при завантаженні сторінки.

Імпорт необхідних для роботи бібліотек

```
import React from 'react';
import ReactDOM from 'react-dom';
import { BrowserRouter } from 'react-router-dom';
import App from './App';
```

Метод для монтування React додатку в HTML

```
ReactDOM.render(
  <React.StrictMode>
    <BrowserRouter>
        <App />
    </BrowserRouter>
  </React.StrictMode>,
  document.getElementById( elementId: 'root'),
);
```

У цьому коді відбувається імпорт React, додаткових бібліотек і компонентів, після чого викликається метод ReactDOM.render(). Цей метод монтує основний компонент React у HTML-елемент із id="root". Використання саме цього ідентифікатора стало стандартом, хоча його можна змінювати за потреби.

Щоб виявляти можливі помилки у додатку, застосовується обгортка React.StrictMode. Для навігації між сторінками підключається бібліотека react-router-dom, а компонент App виступає основною частиною додатку.

Усі наступні функції чи компоненти також використовуватимуть стандартні імпорти бібліотек, однак для уникнення повторень вони не будуть зазначатися при описі.

Переходимо до опису функції App:

Підключення всіх необхідних бібліотек, контекстів та файлів із стилями

```
import React from 'react';
import { useRoutes } from './routes';
import './assets/css/style.min.css';
import './assets/css/admin.css';
import { useAuth } from './hooks/auth.hook';
import { AuthContext } from './context/AuthContext';
```

Опис функції App

```
function App() {
    // Ініціалізація змінних
  const { token, login, logout, userId, isAdmin } = useAuth();
  const isAuthenticated = !!token;
  const routes = useRoutes(isAuthenticated, Boolean(isAdmin), userId);
  return (
      // Ініціалізація провайдеру авторизації
    <AuthContext.Provider
      value={{
        token,
        login,
        logout,
        userId,
        isAuthenticated,
        isAdmin: Boolean(isAdmin)
      }}>
         {/*Ініціалізація роутів, імпортованих з іншого файлу*/}
      {routes}
    </AuthContext.Provider>
  );
}
```

Спочатку, використовуючи хук useAuth (описаний далі), отримуємо дані користувача: токен, ідентифікатор (id), функції для входу/виходу з акаунту та статус адміністратора. Токен, що зберігається у сесії браузера, використовується для автоматичного входу при повторному відвідуванні сайту. [8] Ініціалізується компонент маршрутизації, який згодом використовується в основному компоненті. Створюється AuthContext, що забезпечує доступ до ключових даних авторизації (токен, ID, функції входу/виходу, статус авторизації та адміністратора) у будь-якому компоненті додатку.

В деяких випадках уникають створення проміжного компонента App і реалізують усю логіку безпосередньо у index.js. Однак це може знизити зрозумілість і читабельність коду, особливо у великих проектах.

Переходимо до організації роутів, що містить в компоненті UseRoutes, що також знаходиться в кореневій директорії фронтенд-частини. Це ключовий файл в організації навігації всередині додатку та останній з технічних файлів, надалі проект використовує лише компонентний підхід. *Підключення всіх необхідних бібліотек та компонентів*

```
import React, {useEffect, useState} from 'react';
import { Switch, Redirect, Route } from 'react-router';
import Header from './components/Header/Header';
import Footer from './components/Footer/Footer';
```

Ініціалізація змінних

```
const [itemsInCart, setItemInCart] = useState( initialState: 0);
const [params, setParams] = useState( initialState: {});
const [searchFromHome, setSearchFromHome] = useState( initialState: false);
```

Ініціалізація методів для роботи з роутером

```
const handleSetParams = (par) => {
    setParams(par);
}
const handleSetSearchFromHome = (bool) => {
    setSearchFromHome(bool);
}
const handleUserLogout = () => {
    setItemInCart( value: 0);
}
useEffect( effect: () => {
    if(userId !== null) {
        handleRequest();
    }
}, deps: [userId])
```

Приклад використання роуту: визначення посилання, умови для відображення компонентів:

```
<Route exact path='/account'>
    {!isAuthenticated ?
        <Redirect to="/login" />
        :
        <Account handleRequest={handleRequest}
                 handleUserLogout={handleUserLogout}/> }
</Route>
```

У цьому проекті активно використовуються два основних React-хуки: useState та useEffect.

- useState відстежує стан компонента і дозволяє змінювати його через спеціальну функцію. Стан ініціалізується значенням за замовчуванням.
- useEffect виконує побічні ефекти, таких як запити до сервера або оновлення стану. Хук запускається при рендері компонента або зміні заданих залежностей.

Через useState створюються стани для кількості товарів у кошику, параметрів пошуку та його статусу. Це дозволяє передавати параметри між компонентами, зокрема з головної сторінки. Хоча для цього можна було б використати Redux, для такої простої логіки він є зайвим.

Дві функції з префіксом handle змінюють стани, гарантуючи імутабельність даних. Функція GET-запиту, створена за допомогою axios, отримує дані про кошик користувача на основі його userId. [9] Отримана кількість товарів відображається у шапці сайту, а функція викликається при наявності userId.

Далі реалізується навігація: компоненти Header і Footer виводяться на кожній сторінці, а маршрути створюються через Switch і Route. Для захищених маршрутів (наприклад, /account) перевіряється авторизація користувача. У разі її відсутності виконується переадресація за допомогою Redirect із бібліотеки react-router.

3.5. Опис та приклад застосування компонентного підходу

Як вже згадувалося раніше, React є компонентно-орієнтованим фреймворком, тому ключові елементи інтерфейсу користувача були винесені

в окремі компоненти. [10] Це дозволяє уникнути дублювання коду, підвищити читабельність та значно прискорити процес розробки.

Серед основних елементів інтерфейсу, які реалізовані у вигляді компонентів, можна виділити наступні:

- Хлібні крихти (breadcrumbs): Вони є чудовим прикладом використання компонентного підходу, оскільки цей елемент присутній на кожній сторінці сайту. При його використанні на відповідній сторінці в компонент передаються параметри, які безпосередньо використовують.

Приклад використання компоненту

```
<BreadCrumbs crumbs={[{route: '/about-us',
  label: 'Про нас'}]}/>
```

Приклад створеного компоненту

```
export default function BreadCrumbs({crumbs}) {
  return (
    <section className="breadcrumbs">
      <div className="container">
        <div className="breadcrumbs__body">
          <Link to='/'>Головна</Link>
          {!!crumbs.length && crumbs.map(item =>
                <Link to={item.route} className='breadcrumbs__item'>
                  <img src={breadCrumbsPng} alt="breadcrumbs" />
                  <h2>{item.label}</h2>
                </Link>
              )}
        </div>
      </div>
    </section>
  );
}
```

Вміст компоненту формується динамічно, базуючись на масиві crumbs, що передається до нього. Сам компонент виглядає так:

- Список товарів та кошик:

  Компонент для відображення товарів реалізовано таким чином, щоб можна було використовувати один і той самий шаблон для відображення різних даних. Для багаторазового рендеру

використовується метод масиву map(), до якого передаються дані через властивості компоненту. [11] Наприклад: *Відображення усіх товару, використовуючи один і той самий компонент*

```
{!loading && items && items.data &&
 items.data.map((obj) =>
    <ItemBlock token={token} inCart={itemsCart.some(item =>
item.productId == obj.id)} key={obj.id} {...obj}
handleAddItemToCart={handleAddItemToCart} />
  )}
```

Масив items формується після отримання даних через GET-запит при завантаженні сторінки.

- слайдери для наповнення контенту сторінки;
- фільтри для пошуку;
- елементи для вибору типу сортування;
- шапка і підвал сайту.

Кожна сторінка веб-додатку має свій власний файл у директорії pages. За своєю структурою сторінки також є компонентами, які містять організацію роутів та підключають описані вище компоненти, такі як хлібні крихти чи списки товарів. На цих сторінках визначено більшість функцій та станів, оскільки вони є батьківськими для вкладених компонентів. Через подібну структуру сторінки мають схожий функціонал і структуру, тому всі реалізовані елементи використовують вже описані хуки та методи.

Висновки до розділу 3

У цьому розділі детально розглянуто організацію файлової структури проекту, зокрема фронтенд і бекенд частин. На серверній стороні описано реалізацію моделей даних і CRUD-роутів для взаємодії з базою даних, з використанням зручного та надійного API бібліотеки Mongoose. Для фронтенд частини досліджено ключовий файл додатку, механізми авторизації користувачів і організацію внутрішніх роутів за допомогою React Router.

Окрему увагу приділено компонентному підходу, який забезпечує повторне використання коду, гнучкість у розробці та полегшує підтримку проекту. Розглянуто приклади створення компонентів для таких елементів інтерфейсу, як хлібні крихти, списки товарів, кошик, слайдери, фільтри пошуку та інші. Таким чином, організація коду дозволяє ефективно підтримувати та масштабувати проект, забезпечуючи чітку структуру та прозорість логіки.

Розділ 4. УСУНЕННЯ БАГІВ ПОЧАТКОВОЇ ВЕРСІЇ

Початкова версія додатку, представлена у попередній частині роботи - була значно менше функціональна, робота з нею супроводжувалась великою кількістю багів та непередбачуваних помилок.

На основі попередньої версії проекту, студенткою спеціальності “122 - Комп’ютерні науки”, Арделян Веронікою було в ручному режимі протестовано та описано усі баги та проблеми. На основі цих баг-репортів, було оптимізовано роботу веб-додатку та виправлено усі проблеми.

Проблеми, описані при тестуванні, можна поділити на наступні групи:

- Валідація та заповнення невaділних даних, результати взаємодії веб-додатку з невалідними даними.
- Помилки при взаємодії з товарами (додавання однакових товарів в кошик тощо).
- Сумісність та адаптивність веб-додатку для різних пристроїв та браузерів.
- Проблеми продуктивності та робота при повільній швидкості мережі.

Валідація та заповнення невaділних даних, результати взаємодії веб-додатку з невалідними даними.

Попередня версія веб-додатку не містила в собі код, для обробки невалідних значень. Це означає, що за умови введення невалідних даних, користувачу не отримав жодних повідомлень про це, що значно погіршувало досвід роботи з веб-додатком. В усі функції та методи, що обробляють дані, було додано умови для перевірки валідації та виведення сповіщення, у разі проблеми. Приклад перевірки на валідність даних на сервері, та виведення сповіщення для користувача про невалідні дані:

Валідатор помилок на сервері

```
const errors = validationResult(req);

if (!errors.isEmpty()) {
 return res.status(400).json({
  errors: errors.array(),
  message: 'Неправильний формат даних',
 });
}
```

При зміні статусу помилки, користувач отримує повідомлення про неї

```
useEffect(() => {
 if(error !== null) {
  NotificationManager.error(error);
 }
 clearError();
}, [error, message, clearError]);
```

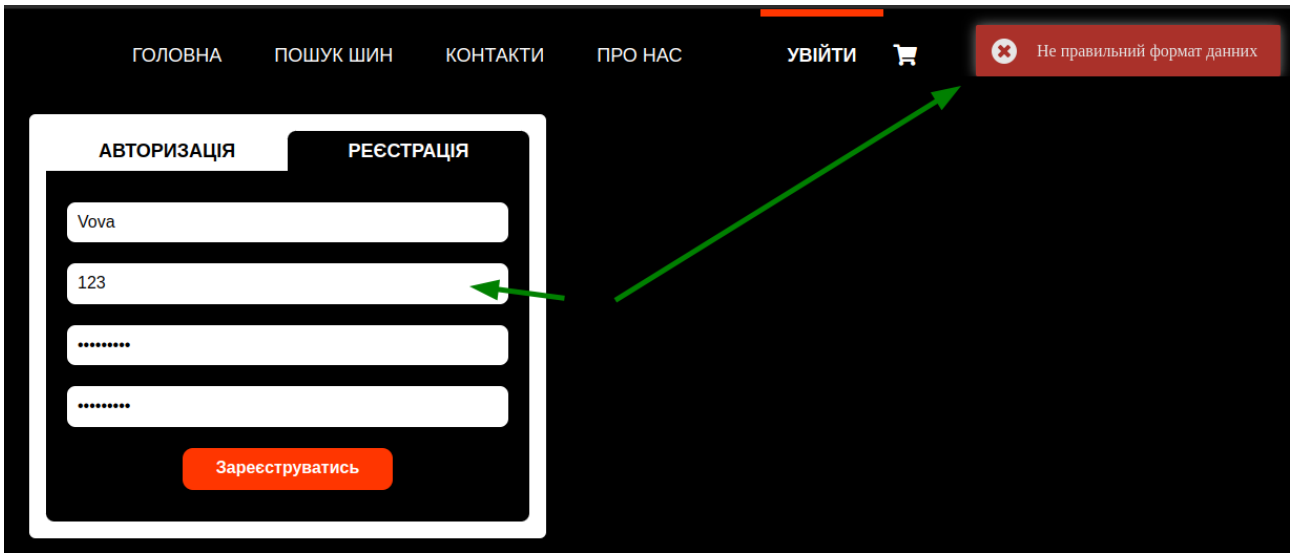

Рис. 4.1. Помилка при валідації поля email

Помилки при взаємодії з товарами

На сторінці пошуку товарів, виникали помилки при взаємодії з кошиком: функціонал не передбачав обробку сценарію, коли товар, що вже міститься в кошику, повторно додається в кошик. Це серйозна проблема, оскільки такий сценарій приводив до критичних помилок, після яких робота додатку зупинялась. Проблема була виправлена методом перевірки статусу товару у

кошику та відображення різних статусів кнопки “додати в кошик”.

Відображення усіх товару, використовуючи один і той самий компонент

```
{!loading && items && items.data &&
 items.data.map((obj) =>
    <ItemBlock token={token} inCart={itemsCart.some(item =>
item.productId == obj.id)} key={obj.id} {...obj}
handleAddItemToCart={handleAddItemToCart} />
  )}
```

Відображення повідомлення, що товар вже в кошику

```
{itemInCart ?
    <div className="item-list__item--button cursor in_cart">
      <span>В кошику</span>
    </div> :
    ''}
```

Сумісність та адаптивність веб-додатку

Адаптивність веб-додатку була реалізована в попередній версії, проте більше глибоке тестування показало проблему взаємодії при використанні планшетів та телефонів з деякими розширеннями. Враховуючи це, для усіх сторінок веб-додатку було розроблено медіа запити, що дозволяють змінювати позиціонування та стилізацію елементів, відповідно до певних розширень екрану.

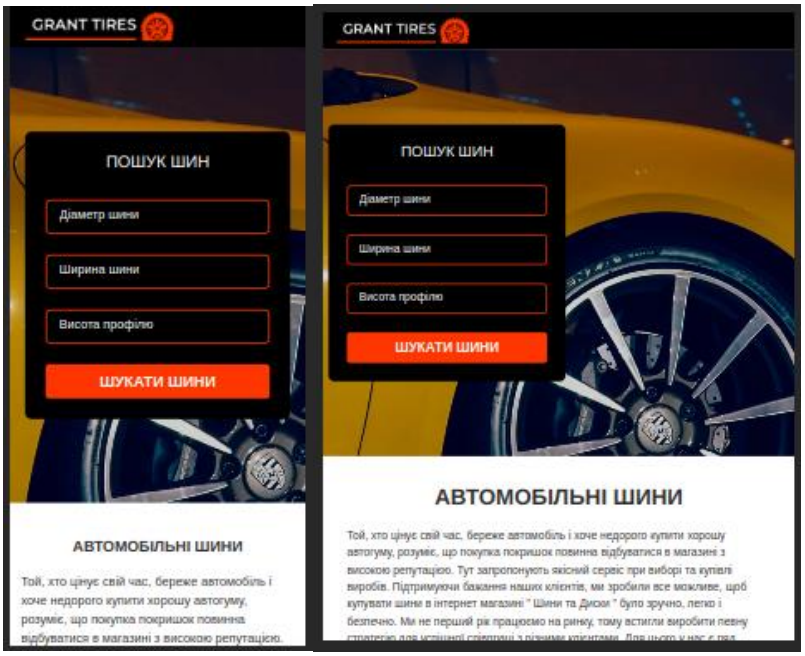

Рис. 4.2, 4.3 Відображення головної сторінки на телефоні та планшеті

Проблеми продуктивності та робота при повільній швидкості мережі.

Використання бібліотеки React надає нам динамічну взаємодію з веб-додатком - постійні оновлення даних у реальному часі. Це означає, що при будь-яких взаємодіях користувач не мігрує між сторінками, а весь час працює в межах однієї сторінки. Проблема, вказана в баг-репорті, полягає в тому, що під час будь-якого процесу завантаження даних, користувач не знав, що цей процес відбувається. Особливо це створює проблеми при повільній швидкості інтернет з’єднання, коли користувач виконав якусь операція, наприклад, пошук, і не розуміє, що відбувається процес завантаження. Для вирішення цієї проблеми, на усіх сторінках, де може виникати така помилка, було додано прелоадер. Він активується щоразу, коли починається запит на бекенд, та деактивнується, коли запит рахується виконаним.

Зміна статусу завантаження при початку та кінці обробки запиту на сервер

```
const handleSetSearchParams = (params) => {
 setLoading(true);
 setSearchParams(params);
}

Axios.get('/api/tire-crud/get-items-sorting', {
 params: {
  sortBy: selectedSort.type,
  sortOrder: selectedSort.order,
  skip: page * itemsPerPage,
  limit: itemsPerPage,
  search: JSON.stringify(filteredObj)
 }
}).then((response) => {
 setItems(response.data);
 setItemsCount(response.data.count);
 setLoading(false);
});
```

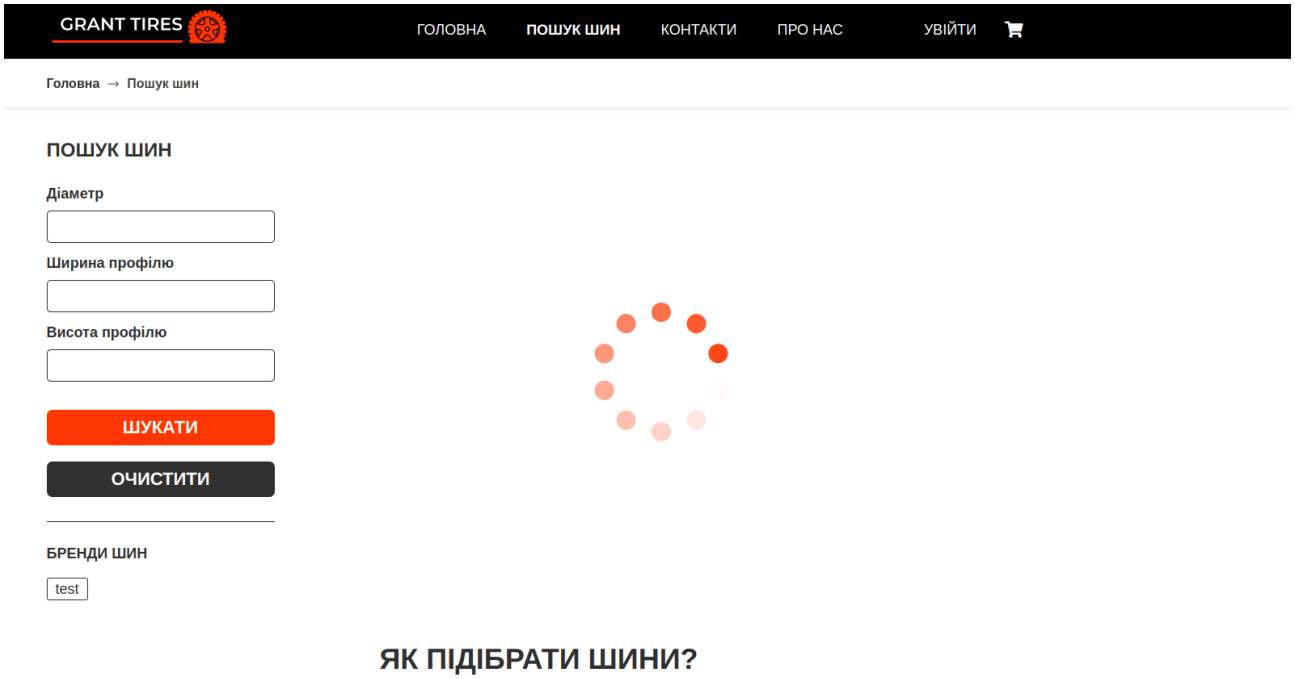

Рис. 4.4. Процес завантаження товарів

Висновки до розділу 4

У процесі тестування початкової версії веб-додатку для продажу автомобільних шин було виявлено низку проблем, що суттєво впливали на функціональність, продуктивність та зручність використання. Початкова версія мала обмежений функціонал і супроводжувалася багатьма помилками, які потребували детального аналізу та усунення.

Завдяки детальному тестуванню та аналізу усіх проблем, веб-додаток було доопрацьовано та виправлено усі помилки. Це повинно значно покращити взаємодію з веб-додатком, мінімізувати кількість помилок та виключити непередбачувані сценарії при роботі з веб-додатком.

Розділ 5. ОПИС ФУНКЦІОНАЛУ ПРОГРАМИ

Оскільки веб-додаток можна розділити на 2 частини: інтерфейс клієнту та інтерфейс адміністратора, розглянемо їх окремо один від одного.

5.1. Опис інтерфейсу клієнта

Клієнтська частина веб-додатку побудована з урахуванням сучасних вимог до зручності користування та адаптивності. Вона включає в себе шапку (header), підвал (footer), а також шість сторінок, які змінюються в залежності від URL-адреси. Завдяки використанню технології Flexbox, весь інтерфейс оптимізовано для різних типів пристроїв — від настільних ПК до смартфонів.

Шапка сайту виконує роль головного елементу навігації та включає логотип і меню. Активна сторінка додатково позначається графічно, підкресленням відповідного пункту меню. У мобільній версії реалізовано бургер-меню, що забезпечує зручність користування на невеликих екранах.

Рис. 5.1. Шапка сайту

Футер сайту містить повторювані елементи, такі як логотип та посилання на соціальні мережі, що додає завершеності дизайну і підтримує загальний стиль сайту.

Рис. 5.2. Футер сайту

Головна сторінка наповнена як інформативними блоками, так і функціональними елементами. Зокрема, на ній розміщено форму для швидкого пошуку шин. Ця форма дозволяє користувачам вводити основні параметри, такі як діаметр, висота та ширина шин, що полегшує процес навігації до необхідного розділу.

Рис. 5.3. Форма для швидкого пошуку на головній сторінці

Така структура сприяє зручності використання додатку, забезпечуючи одночасно візуальну привабливість та функціональність.

На сторінці пошуку товарів, яка є ключовим елементом взаємодії з користувачем, реалізовано кілька функціональних компонентів. Вона включає:

- Фільтр для пошуку товарів — дозволяє користувачам задавати параметри пошуку, які автоматично оновлюють список товарів.
- Меню сортування — надає можливість сортувати товари за ціною чи роком виготовлення, у порядку зростання або спаду.
- Список товарів — відображає результати пошуку або весь асортимент залежно від вибраних параметрів.
- Меню пагінації — використовується, якщо кількість товарів перевищує встановлений ліміт для відображення на одній сторінці.

При зміні параметрів пошуку оновлюється відповідний список товарів, кількість сторінок у пагінації та відображувана кількість товарів.

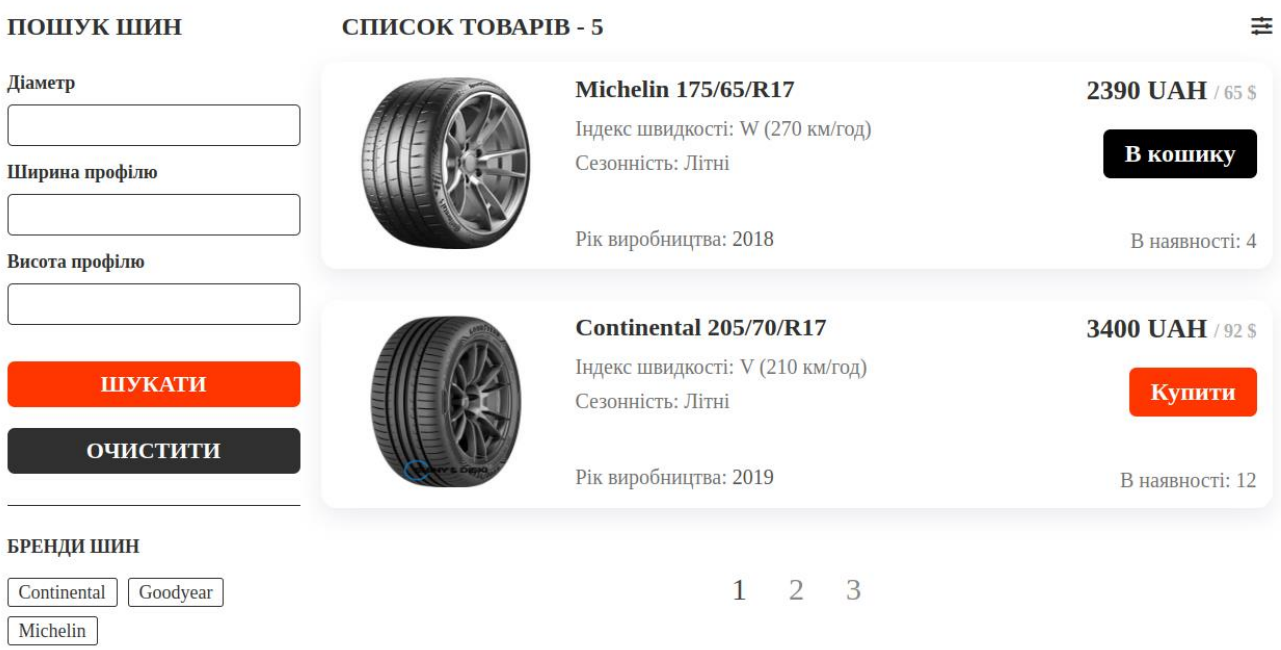

Рис. 5.4. Основні компоненти сторінки пошуку товарів

Функціональність кнопки "Додати в кошик" залежить від стану товару:

- Якщо товар уже є в кошику, кнопка стає недоступною.
- Якщо товару немає в наявності (кількість встановлена як 0 в адмін-панелі), кнопка також недоступна.

Параметри брендів, що використовуються у фільтрах, додаються через адмін-панель, деталі якої будуть описані у відповідному розділі.

Цей підхід забезпечує зручний інтерфейс для користувача, даючи можливість ефективно знаходити необхідні товари, сортувати їх та здійснювати інші ключові дії.

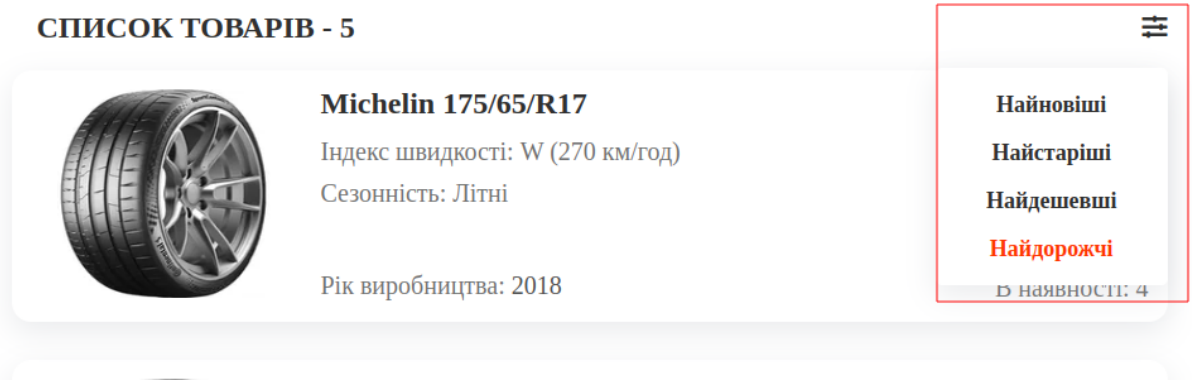

Рис. 5.5. Сортування товарів

У системі реалізовано ряд функціональних сторінок для забезпечення повноцінного користувацького досвіду:

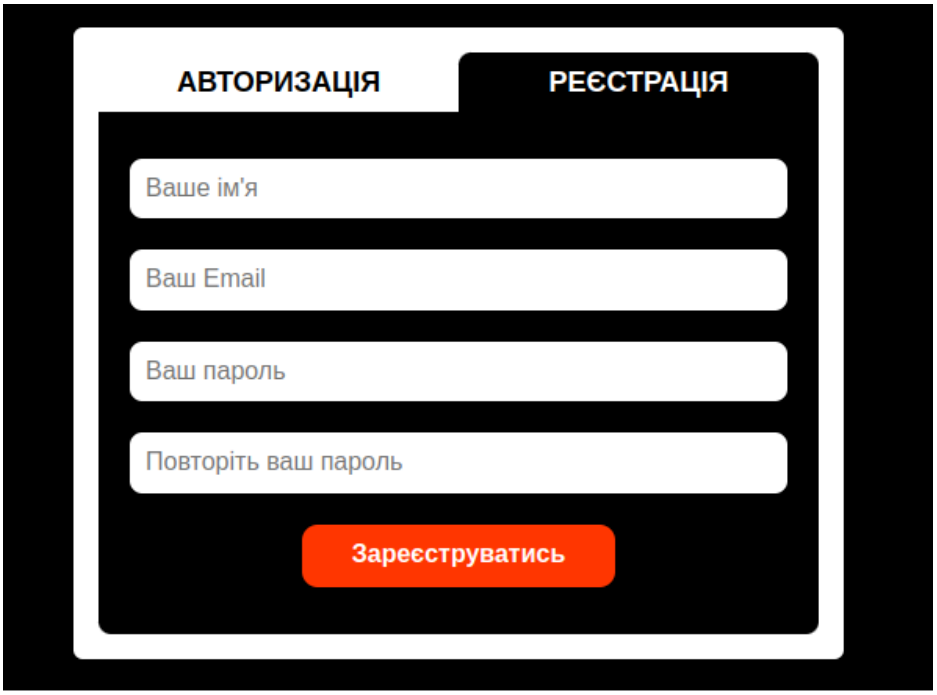

Рис. 5.6. Сторінка авторизації та реєстрації

Для реєстрації потрібно ім’я, унікальный email та пароль. Якщо email уже використовується, система повідомить про помилку. У разі успішної реєстрації - авторизація здійснюється автоматично. Після входу пункт меню "Увійти" змінюється на "Кабінет", де можна редагувати дані або вийти з акаунту.

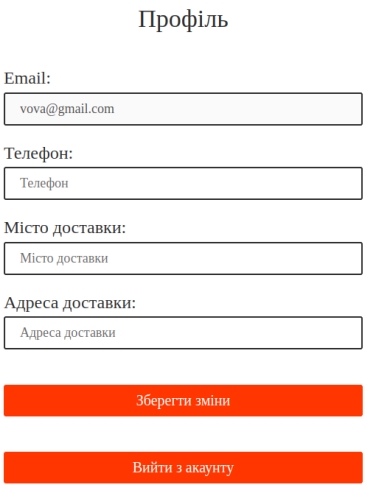

Рис. 5.7. Сторінка редагування даних користувача

Сторінка "Про нас" - інформативна сторінка, яка надає користувачам інформацію про компанію.

Сторінка "Контакти" включає контактні дані компанії та форму для відгуків. Форма дозволяє вводити ім'я, телефон, тему, email та текст відгуку. Відгуки автоматично надсилаються адміністратору в Telegram через інтеграцію з ботом, що забезпечує оперативність отримання зворотного зв’язку.

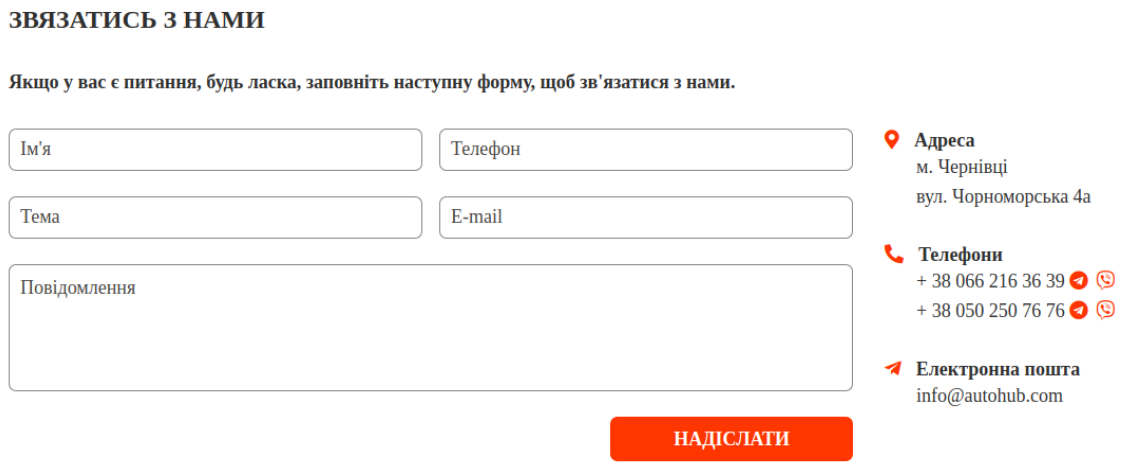

Рис. 5.8. Сторінка редагування даних користувача

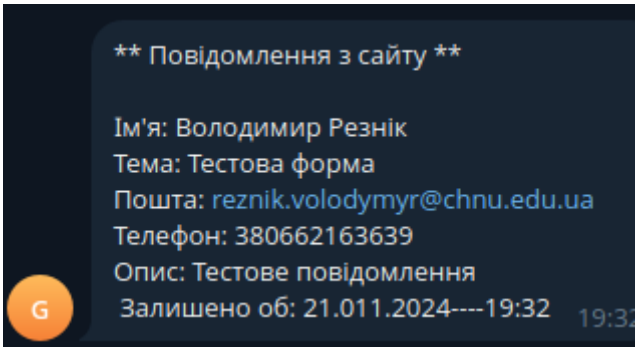

Рис. 5.9. Сповіщення про заповнення форми у телеграм

Сторінка кошика - найбільш інтерактивна сторінка на сайті. Відображає список товарів, доданих із пошуку. Її основний функціонал включає в себе:

- Очищення кошика.
- Видалення окремих товарів.
- Зміна кількості товарів.
- Оформлення замовлення.

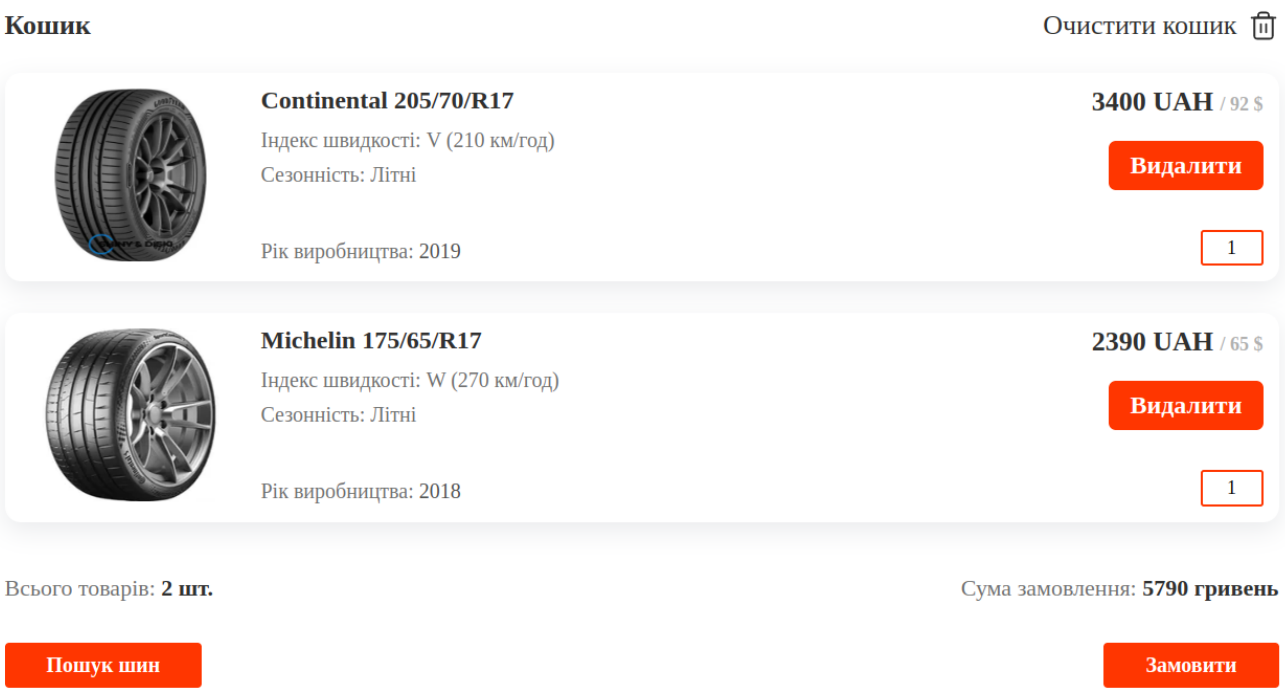

Рис. 5.10. Кошик з товарами товарів

Дані оновлюються в реальному часі, включаючи загальну кількість товарів та підсумкову вартість. Автоматично отримується актуальний курс іноземної валюти, для зручності відображення ціни.

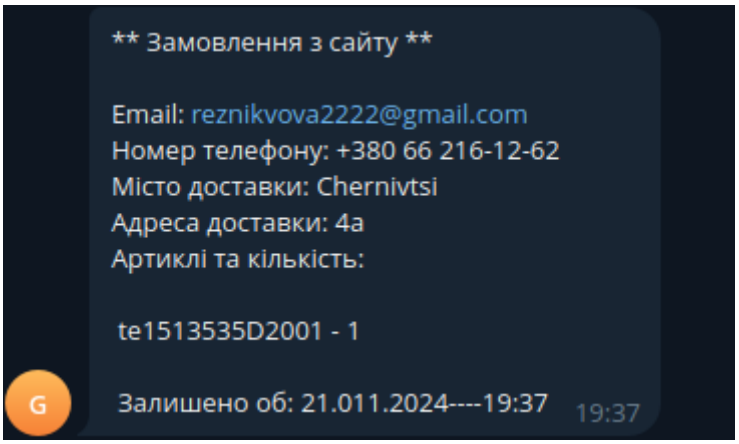

Рис. 5.11. Сповіщення про замовлення товару у телеграмі

Ці сторінки забезпечують інтуїтивну навігацію та надають користувачам необхідні інструменти для взаємодії із сервісом. Інтеграція з Telegram, а також реальний час оновлення стану корзини значно покращують зручність і ефективність використання.

5.2. Опис інтерфейсу адміністратора

Доступ до панелі адміністратора має лише спеціальний користувач, параметр ‘isAdmin’ має значення true. Такий користувач, після натискання на логотип в шапці сайту, або після прямого переходу по посиланню */admin* потрапляє в адмін-панель. Адмін панель забезпечує наступні функції:

- Головне меню: містить інструменти для взаємодії зі списком товарів та іншими елементами. Доступний гнучкий пошук товарів то усім параметрам для зручності взаємодії.

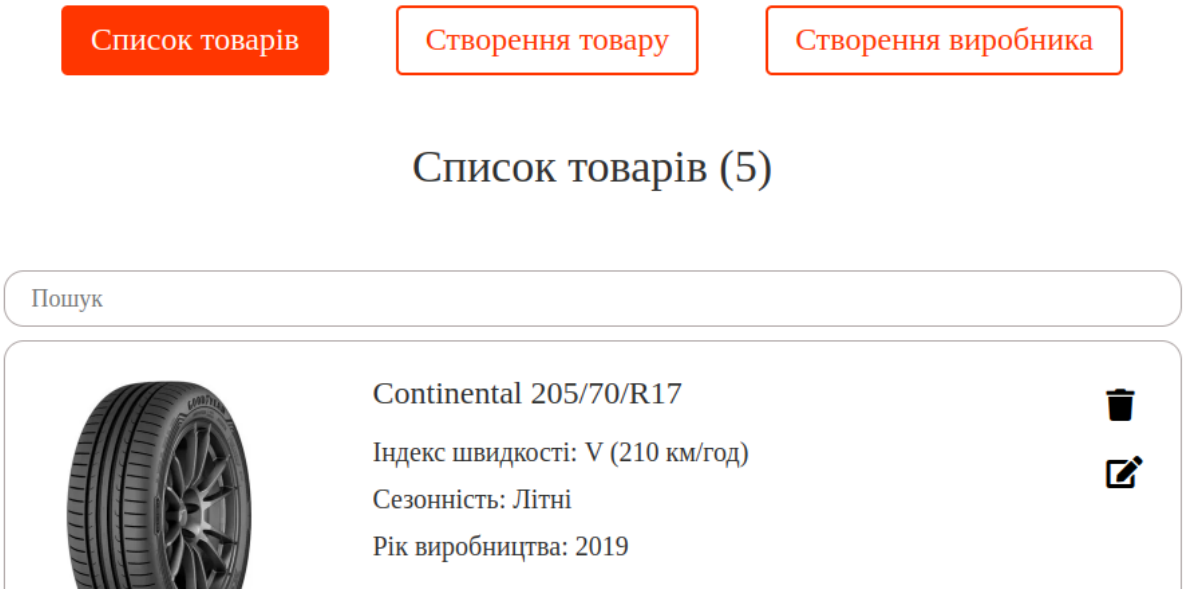

Рис. 5.12. Основне меню адмін-панелі

- Список товарів: реалізовано пошук по всіх полях. Картки товарів мають кнопки для редагування та видалення.
- Холдери створення та редагування товарів:
    - Поля для введення даних мають зручні випадаючі списки.
    - При створенні нового товару всі поля порожні, а при редагуванні — заповнюються існуючими даними з бази.

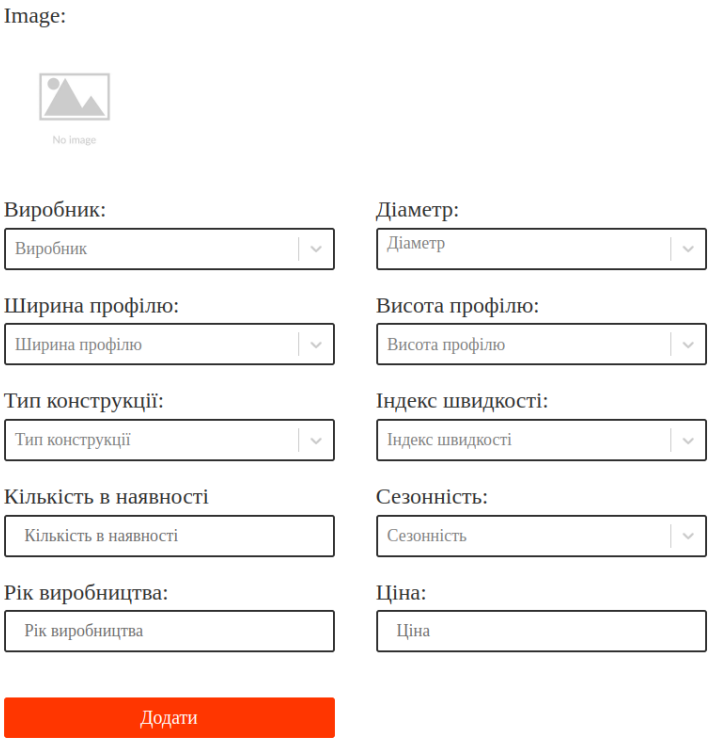

Рис. 5.13. Створення нового товару

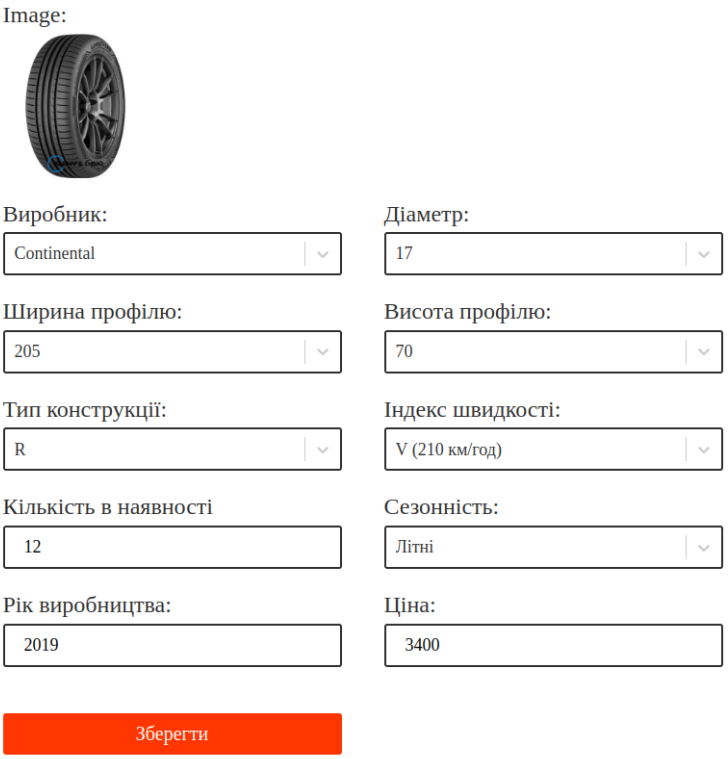

Рис. 5.14. Редагування товару

- Редаговані списки:
    - Діаметр, висота профілю, сезонність та інші статичні параметри зберігаються у файлі *variables.js*.
    - Виробники шин додаються через спеціальний пункт меню, що полегшує пошук і організацію товарів.

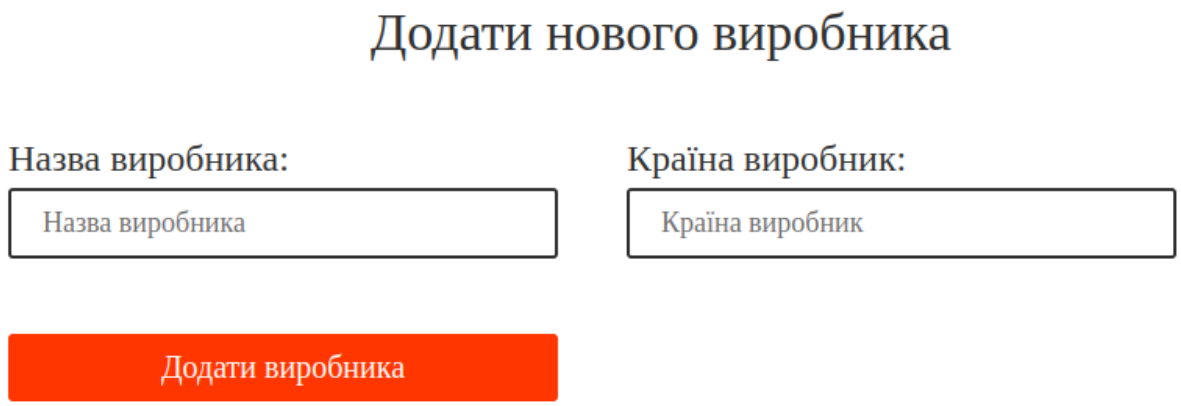

Рис. 5.15. Створення нового елементу для випадаючих списків (виробник)

У перспективі розвитку адмін-панелі планується:

- Реалізація можливості перегляду та управління замовленнями.
- Додавання інструментів для редагування контенту інформаційних сторінок.
- Створення блогу з функціоналом керування постами безпосередньо через панель адміністратора.

Ця функціональність забезпечує простоту роботи адміністратора та дозволяє розширювати можливості сайту в майбутньому.

ВИСНОВОК

Розроблений веб-додаток створений для ефективного онлайн-продажу автомобільних шин. Його структура передбачає інтеграцію адміністративної панелі для управління товарами та клієнтську частину для взаємодії користувачів із каталогом товарів.

Під час роботи над проектом був реалізований повноцінний додаток із фронтенд та бекенд частинами. Це дозволило мені значно поглибити теоретичні знання, а також вдосконалити навички розробки серверної частини. Стек MERN (MongoDB, Express, React, Node.js) обраний як оптимальний інструмент, що дозволяє поєднати потужний API для роботи з базою даних із сучасним, зручним інтерфейсом. У якості альтернативи можна було б використати стек MEAN (з Angular замість React), але обраний підхід краще відповідає поставленим завданням та вимогам.

Веб-додаток планується використовувати в реальних умовах. У контексті моєї професійної діяльності, пов’язаної з продажем шин та наданням шиномонтажних послуг, цей додаток стане основою для розширення бізнесу. Надалі передбачається розробка сторінки блогу, що буде заповнюватись із панелі адміністратора, яка сприятиме SEO-оптимізації та підвищить видимість сайту в пошукових системах. Також у планах — інтеграція аналітичних інструментів для моніторингу відвідуваності та поведінки користувачів.

Наступний етап включає розгортання сайту на власному хостингу, вибір доменного імені та налаштування параметрів для SEO-оптимізації. Ці кроки дозволять покращити позиції сайту в пошуковій видачі, що сприятиме зростанню трафіку та підвищенню конверсії.

Переваги розробленого додатку:

Власна розробка, замість використання готових рішень

- Повна свобода в реалізації функціоналу, необмежені можливості для масштабування та вдосконалення.
- Незалежність від сторонніх платформ і сервісів.

Інтеграція адміністративної панелі

- Простий і зручний інтерфейс для управління товарами: створення, редагування, видалення.
- Доступ до налаштування фільтрів і пошуку в клієнтській частині, що полегшує роботу адміністраторів.

Гнучкість у дизайні

- Сайт адаптований для всіх видів пристроїв: від ПК до мобільних телефонів, завдяки використанню flexbox і компонентного підходу.
- Зручна навігація та інтуїтивний інтерфейс.

Використання стеку MERN

- Легкість у роботі з MongoDB через зручний API.
- React забезпечує швидкодію і зручність розробки компонентів.
- Node.js дозволяє масштабувати серверну частину під високі навантаження.

Можливість подальшого розвитку

- Плани на впровадження блогу, аналітики, покращення SEO.
- Підключення нових модулів і функцій для інтерактивності.

Інтеграція з Telegram

- Швидке отримання відгуків користувачів через бот у Telegram, що полегшує зворотний зв’язок.

Недоліки розробленого додатку:

Витрати часу на розробку

- Створення власного програмного рішення займає значно більше часу порівняно з використанням готових платформ для онлайн-продажів (наприклад, Shopify чи OpenCart).

Відсутність тестування в умовах високого навантаження

- Додаток поки що не протестовано на великій кількості користувачів, що користуються ним одночасно. Це може вплинути на проблеми з продуктивністю.

Обмежений функціонал адміністративної панелі

- Наразі панель не включає роботу із замовленнями та аналітикою, що є важливим для повноцінного управління бізнесом.

Потреба в технічному обслуговуванні

- Самописне рішення вимагає постійної підтримки, оновлень і виправлення багів, що створює додаткове навантаження на розробника.

Обмежена SEO-оптимізація на старті

- Необхідність у ручному налаштуванні параметрів SEO та впровадженні інструментів для поліпшення видимості в пошукових системах.

Початкові фінансові витрати

- Потреба в хостингу, домені та додаткових ресурсах для запуску й підтримки сайту може бути значною для стартового етапу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. BusinessDasher. 46+ Online Shopping Statistics: The Facts & Trends [Електронний ресурс]. – Режим доступу: https://www.businessdasher.com/online-shopping-statistics/
2. Mongoose [Електронний ресурс]. – Режим доступу: https://mongoosejs.com/docs/index.html
3. Express 4.21.1 [Електронний ресурс]. – Режим доступу: https://expressjs.com/
4. Unidirectional data flow in React? [Електронний ресурс]. – Режим доступу: https://www.educative.io/answers/what-is-unidirectional-data-flow-in-react
5. Node JS [Електронний ресурс]. – Режим доступу: https://nodejs.org/uk
6. React Docs. Context. [Електронний ресурс]. – Режим доступу: https://legacy.reactjs.org/docs/context.html
7. Built-in React Hooks. [Електронний ресурс]. – Режим доступу: https://react.dev/reference/react/hooks
8. MDN. Window.sessionStorage [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/ru/docs/Web/API/Window/sessionStorage
9. AXIOS [Електронний ресурс]. – Режим доступу: https://axios-http.com/uk/docs/intro
10. Компоненти і пропси [Електронний ресурс]. – Режим доступу: https://uk.legacy.reactjs.org/docs/components-and-props.html
11. Lists and Keys [Електронний ресурс]. – Режим доступу: https://legacy.reactjs.org/docs/lists-and-keys.html

ДОДАТОК

1. Головний серверний файл **index.js**:

```
const express = require('express');
const config = require('config');
const mongoose = require('mongoose');
const PORT = config.get('port') || 5000;
const app = express();
const path = require('path');

const cors = require('cors');


app.use(cors());
app.use(express.json({ extended: true }));
app.use('/api/auth', require('./routes/auth.routes'));
app.use('/api/cart', require('./routes/cart.router'))
app.use('/api/tire-crud', require('./routes/tire-crud.router'))
app.use('/api/brand-crud', require('./routes/brand-crud.router'))
app.use('/api/user-crud', require('./routes/user-crud.router'))
app.use(express.static(path.join(__dirname, './client/build')));
app.get('*', (req, res) => {
  res.json("STATUS 200");
});


async function start() {
  try {
    await mongoose.connect(config.get('mongoUri'), {
      useNewUrlParser: true,
      useUnifiedTopology: true,
      useCreateIndex: true,
    });
    app.listen(5000, () => console.log(`App has been started on port ${PORT}
`));
  } catch (e) {
    console.log('Server error', e.message);
    process.exit(1);
  }
}
start();
```

2. Файл **package.json** в директорії проекту

```
{
  "name": "reznik.v",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "proxy": "http://127.0.0.1:5000/",
  "scripts": {
    "start": "node index.js",
    "server": "nodemon index.js --ignore client",
    "client": "PORT=4000 npm run start --prefix client",
    "dev": "concurrently \"npm run server\" \"npm run client \"",
    "register:super-admin": "node createSuperAdmin.js --
email=%npm_config_email% --password=%npm_config_password%"
  },
  "keywords": [
    "react",
    "express"
  ],
  "author": "Reznik Volodymyr <reznikvova2222@gmail.com>",
  "license": "ISC",
  "dependencies": {
    "axios": "^1.4.0",
    "bcryptjs": "^2.4.3",
    "config": "^3.3.6",
    "cors": "^2.8.5",
    "express": "^4.17.1",
    "express-validator": "^6.11.1",
    "jsonwebtoken": "^8.5.1",
    "lodash": "^4.17.21",
    "mongodb": "^3.6.9",
    "mongoose": "^5.12.11",
    "querystring": "^0.2.1",
    "react-notifications": "^1.7.4",
    "react-select": "^5.7.3",
    "react-toastify": "^8.0.0",
    "semantic-ui-css": "^2.4.1",
    "semantic-ui-react": "^2.0.3",
    "url": "^0.11.0",
    "uuid": "^9.0.0"
  },
  "devDependencies": {
    "concurrently": "^6.2.0",
    "nodemon": "^2.0.7"
  }
}
```

3. Опис моделі User:

```
const { Schema, model } = require('mongoose');
const schema = new Schema({
  email: { type: String, require: true, unique: true },
  name: {type: String},
  password: { type: String, require: true },
  isAdmin: {type: String, required: true},
  phone: {type: String},
  deliveryCity: {type: String},
  deliveryAddress: {type: String},
});

module.exports = model('User', schema);
```

4. Опис моделі Tire:

```
const { Schema, model } = require('mongoose');
const schema = new Schema({
    id: { type: String, require: true},
    article: {type: String, required: true, unique: true },
    brand: { type: String, require: true },
    diameter: {type: String, required: true},
    width: {type: String, required: true},
    height: {type: String, required: true},
    construction: {type: String, required: true},
    speedIndex: {type: String, required: true},
    countAvailable: {type: String},
    season: {type: String},
    image: {type: String},
    year: {type: String},
    price: {type: String}
});

module.exports = model('Tire', schema);
```

5. Опис моделі Cart:

```
const { Schema, model } = require('mongoose');
const schema = new Schema({
   userId: {
      type: Schema.Types.ObjectId,
      ref: "User"
   },
   products: [
      {
         productId: Number,
         quantity: Number,
         price: Number
      }
   ]
});

module.exports = model('Cart', schema);
```

6. Опис моделі Brand:

```
const { Schema, model } = require('mongoose');
const schema = new Schema({
   id: { type: String, require: true },
   name: { type: String, require: true, unique: true },
   country: { type: String, require: true }
});

module.exports = model('Brand', schema);
```

7. Опис CRUD-роутеру Tire, **tire-crud.router.js**:

```
const { Router } = require('express');
const router = Router();
const Tire = require("../models/Tire");

router.post('/create', async (req, res) => {
   const {id, brand, diameter, width, height, construction, speedIndex,
countAvailable, season, image, year, price} = req.body;
   const article = brand.slice(0, 2) + diameter + width + height + construction
+ year;

   const item = new Tire({id, brand, diameter, width, height, construction,
speedIndex, countAvailable, season, image, year, price, article});

   try {

      if (await Tire.findOne({ article })) {
```

```
            return res.status(400).json({ message: 'Товар вже
зареєстрований' });
        }
        await item.save();
        return res.status(201).json({ message: 'Товар було успішно
створено' });
    } catch (e) {
        console.log(e);
    }

})
router.get('/get-items', async (req, res) => {
    try{
        const userId = req.query.userId;

        const items = await Tire.find();

        return res.status(200).json({
            success: true,
            count: items.length,
            data: items,
        });
    } catch(err) {
        console.log(err);
        res.status(500).json({ error: 'Помилка севреру' });
    }
});

router.get('/get-items-sorting', async (req, res) => {
    try{
        const sortBy = req.query.sortBy, sortOrder = req.query.sortOrder,
            limit = Number(req.query.limit), skip = Number(req.query.skip),
            search = req.query.search;
        sortObj[sortBy] = sortOrder === 'desc' ? -1 : 1;
        const _search = JSON.parse(search);
        const itemsWithoutPagination = await Tire.find(_search),
            itemsWithPagination = await Tire.find(_search).skip(skip)
                .limit(limit).sort(sortObj);
        return res.status(200).json({
            success: true,
            count: itemsWithoutPagination.length,
            data: itemsWithPagination,
        });
    } catch(err) {
        res.status(500).json({ error: 'Помилка серверу' });
    }
});

router.get('/get/:id', async (req, res) => {
    try{
        const result = await Tire.findById(req.params.id);
```

```
        return res.status(200).json({
            success: true,
            data: result
        });
    } catch(err) {
        res.status(500).json({ error: 'Помилка серверу' });
    }
});

router.put('/update/:id', async (req, res) => {
    try{
        const update = await Tire.findByIdAndUpdate(req.params.id, {
            $set: req.body
        });
        res.status(200).json({ message: 'Товар було оновлено' });
    } catch(err) {
        res.status(500).json({ error: 'Помилка серверу' });
    }
});

router.delete('/delete/:id', async (req, res) => {
    try{
        const item = await Tire.findByIdAndRemove(req.params.id);
        return res.status(201).json({ message: 'Товар було видалено' });
    } catch(err) {
        res.status(500).json({ error: 'Помилка серверу' });
    }
})

module.exports = router;
```

8. Опис CRUD-роутеру User, **user-crud.router.js**:

```
const { Router } = require('express');
const router = Router();
const User = require("../models/User");
const Tire = require("../models/Tire");

const bcrypt = require('bcryptjs');
const { check, validationResult } = require('express-validator');
```

```
router.put('/update/:id', async (req, res) => {
    try {
        const update = await User.findByIdAndUpdate(req.params.id, {
            $set: req.body
        });

        res.status(200).json({ message: 'Дані було успішно оновлено' });
    } catch (e) {
        console.log(e);
    }
})

router.put('/update-password/:id', [
    check('current', 'Ви не ввели поточний пароль').notEmpty(),
    check('new', 'Новий пароль повинен містити 6
символів').isLength({ min: 6 })
], async (req, res) => {
    try {
        const errors = validationResult(req);
        if (!errors.isEmpty()) {
            return res.status(400).json({
                errors: errors.array(),
                message: 'Некоректно введені дані'
            });
        }

        const { current, new: newPassword } = req.body;
        const user = await User.findById(req.params.id);

        if (!user) {
            return res.status(404).json({ message: 'Користувача не знайдено' });
        }

        // Проверка текущего пароля
        const isMatch = await bcrypt.compare(current, user.password);
        if (!isMatch) {
            return res.status(400).json({ message: 'Поточний пароль
невірний' });
        }

        // Хэшируем новый пароль
        const hashedPassword = await bcrypt.hash(newPassword, 12);
        user.password = hashedPassword;

        // Сохраняем изменения
        await user.save();

        res.status(200).json({ message: 'Пароль було успішно змінено' });
    } catch (e) {
        console.error(e);
        res.status(500).json({ message: 'Помилка сервера' });
```

```
    }
});

router.get('/get-user', async (req, res) => {
    try{
        const userId = req.query.userId;

        const user = await User.findOne({_id: userId});

        return res.status(200).json({
            success: true,
            user: user
        });

    } catch(err) {
        res.status(500).json({ error: 'Помилка серверу' });
    }
});


module.exports = router;
```

9. Опис роутеру для авторизації та реєстрації, **auth.routers.js**:

```
const { Router } = require('express');
const bcrypt = require('bcryptjs');
const jwt = require('jsonwebtoken');
const { check, validationResult } = require('express-validator');
const User = require('../models/User');
const router = Router();
const config = require('config');


// /api/auth/register
router.post(
  '/register',
  [
    check('email', 'Incorrect email').isEmail(),
    check('password', 'Password should contains minimum 6
symbols').isLength({ min: 6 }),
  ],
  async (req, res) => {
    try {
      const errors = validationResult(req);

      if (!errors.isEmpty()) {
        return res.status(400).json({
          errors: errors.array(),
          message: 'Не правильний формат данних',
        });
```

```
    }
    const { email, password, name } = req.body;
    const candidate = await User.findOne({ email });

    if (candidate) {
      return res.status(400).json({ message: 'Користувач вже
зареєстрований' });
    }

    const hashedPassword = await bcrypt.hash(password, 12);
    const user = new User({ email, password: hashedPassword, name:
name, isAdmin: false, phone: '', deliveryCity: '', deliveryAddress: '' });
    await user.save();

    res.status(201).json({ message: 'Користувача створено!' });
  } catch (e) {
    res.status(500).json({ message: 'Помилка серверу' });
  }
 },
);
// /api/auth/login
router.post(
 '/login',
 [
  check('email', 'Не правильний формат
пошти').normalizeEmail().isEmail(),
  check('password', 'Password doesnt exist').exists(),
 ],
 async (req, res) => {
  try {
    const errors = validationResult(req);

    if (!errors.isEmpty()) {
      return res.status(400).json({
        errors: errors.array(),
        message: 'Данні не валідні',
      });
    }
    const { email, password } = req.body;
    const user = await User.findOne({ email });
    if (!user) {
      return res.status(400).json({ message: 'Користувач не існує' });
    }

    const isMatch = await bcrypt.compare(password, user.password);
    if (!isMatch) {
      return res.status(400).json({ message: 'Пароль не правильний!' });
    }
    const token = jwt.sign({ userId: user.id }, config.get('jwtSecret'),
{ expiresIn: '1h' });
```

```
      res.json({ token, userId: user.id, isAdmin: user.isAdmin === 'true',
message: 'Авторизація успішна' });
    } catch (e) {
      res.status(500).json({ message: 'Помилка серверу' });
    }
  },
);


module.exports = router;
```

10.Головний файл фронтенду **index.js:**

```
import React from 'react';
import ReactDOM from 'react-dom';
import { BrowserRouter } from 'react-router-dom';
import App from './App';

ReactDOM.render(
  <React.StrictMode>
    <BrowserRouter>
      <App />
    </BrowserRouter>
  </React.StrictMode>,
  document.getElementById('root'),
);
```

11.Файл **App.js:**

```
import React from 'react';
import { useRoutes } from './routes';
import './assets/css/style.min.css';
import './assets/css/admin.css';
import { useAuth } from './hooks/auth.hook';
import { AuthContext } from './context/AuthContext';

function App() {
    // Ініціалізація змінних
  const { token, login, logout, userId, isAdmin } = useAuth();
  const isAuthenticated = !!token;
  const routes = useRoutes(isAuthenticated, Boolean(isAdmin), userId);
  return (
     // Ініціалізація провайдеру авторизації
    <AuthContext.Provider
      value={{
        token,
        login,
        logout,
        userId,
```

```
        isAuthenticated,
        isAdmin: Boolean(isAdmin)
      }}>
        {/*Ініціалізація роутів, імпортованих з іншого файлу*/}
      {routes}
    </AuthContext.Provider>
  );
}

export default App;
```

12. Файл **routes.js:**

```
import React, {useEffect, useState} from 'react';
import { Switch, Redirect, Route } from 'react-router';
import Header from './components/Header/Header';
import Footer from './components/Footer/Footer';
import Home from './pages/Home/Home';
import Shop from './pages/Shop';
import About from './pages/About';
import Contact from './pages/Contact';
import Cart from './pages/Cart';
import AuthPage from './pages/Auth';
import Account from "./pages/Account";
import AdminPanel from "./pages/Admin/AdminPanel";
import AdminUpdateTire from "./pages/Admin/AdminUpdateTire";
import Axios from "axios";

export const useRoutes = (isAuthenticated, isAdmin, userId) => {

    const [itemsInCart, setItemInCart] = useState(0);
    const [params, setParams] = useState({});
    const [searchFromHome, setSearchFromHome] = useState(false);

    const handleSetParams = (par) => {
        setParams(par);
    }
    const handleSetSearchFromHome = (bool) => {
        setSearchFromHome(bool);
    }
    const handleUserLogout = () => {
        setItemInCart(0);
    }
    useEffect(() => {
        if(userId !== null) {
            handleRequest();
        }
    }, [userId])

    const handleRequest = () => {
```

```
    Axios.get('/api/cart/get-items-count', {
      params: {
        userId: userId
      }
    }).then((response) => {
            setItemInCart(response.data.count);
    });
  }

  return (
    <Switch>
      <>
        <div className="loading">
        </div>
       <Header isAdmin={isAdmin} itemsInCart={itemsInCart}/>
       <div className="content">
        <Route exact path="/">
          <Home handleSetParams={handleSetParams}
handleSetSearchFromHome={handleSetSearchFromHome}/>
        </Route>
        <Route exact path="/tires">
          <Shop handleRequest={handleRequest} params={params}
handleSetParams={handleSetParams}
searchFromHome={searchFromHome}
handleSetSearchFromHome={handleSetSearchFromHome}/>
        </Route>
        <Route exact path="/about-us">
          <About />
        </Route>
        <Route exact path="/contacts">
          <Contact />
        </Route>
        <Route exact path="/cart">
           {!isAuthenticated ? <Redirect to="/login" /> : <Cart
handleRequest={handleRequest}/> }
        </Route>

          <Route exact path='/account'>
             {!isAuthenticated ?
                <Redirect to="/login" />
                :
                <Account handleRequest={handleRequest}
                      handleUserLogout={handleUserLogout}/> }
          </Route>


          <Route exact path='/login'>
             {isAuthenticated ? <Redirect to="/account" /> : <AuthPage
handleRequest={handleRequest}/> }
          </Route>
        <Route exact path='/admin'>
```

```
          {/*{isAdmin ? <AdminPanel/> : <Redirect to="/"/> }*/}
           <AdminPanel/>
          </Route>
          <Route exact path='/admin/edit/:id'>
           {/*{isAdmin ? <AdminPanel/> : <Redirect to="/"/> }*/}
            <AdminUpdateTire/>
          </Route>
        </div>
        <Footer />
       </>
     </Switch>
   );
};
```

13.Головна сторінка ../pages/**Home.jsx:**

```
import React, { useEffect, useState } from 'react';
import { NewItems, BrandsSlider } from '../../components/index';
import {useHistory} from "react-router-dom";



export default function Home({handleSetParams,
handleSetSearchFromHome}) {
 const history = useHistory();

 const [form, setForm] = useState({
  diameter: '',
  height: '',
  width: '',
 })

 const changeHandler = (event) => {
  setForm({ ...form, [event.target.name]: event.target.value });
 };


 const onSearch = (e) => {
  handleSetParams(form);
  handleSetSearchFromHome(true);

  history.push('/tires');
 };
 return (
  <main>
   <section
    className="search"
    style={{
     backgroundImage: `url(${process.env.PUBLIC_URL + '/background-
car.png'})`,
```

```
    backgroundPosition: 'center',
    backgroudSize: '100% 100%',
  }}>
  <div className="container">
    <div className="search__body">
      <form  className="search__form">
        <h3 className="search__form--title">Пошук шин</h3>
        <input
          className="search__form--input"
          type="text"
          id="brand"
          placeholder="Діаметр шини"
          onChange={changeHandler}
          value={form.diameter}
          name={'diameter'}
        />
        <input
          className="search__form--input"
          type="text"
          id="model"
          placeholder="Ширина шини"
          onChange={changeHandler}
          value={form.width}
          name={'width'}
        />
        <input
          className="search__form--input"
          type="text"
          id="item"
          placeholder="Висота профілю"
          onChange={changeHandler}
          value={form.height}
          name={'height'}
        />
        <div className="search__form--button" onClick={() => onSearch()}>
          Шукати шини
        </div>
      </form>
    </div>
  </div>
</section>

<section className="information">
  <div className="container">
    <div className="information__body">
      <div className="information__title">
        <h2>Автомобільні шини</h2>
      </div>
      <div className="information__text">
        <p>
          Той, хто цінує свій час, береже автомобіль і хоче недорого
```

купити хорошу автогуму, розуміє, що покупка покришок повинна відбуватися в магазині з високою репутацією. Тут запропонують якісний сервіс при виборі та купівлі виробів. Підтримуючи бажання наших клієнтів, ми зробили все можливе, щоб купувати шини в інтернет магазині " Шини та Диски " було зручно, легко і безпечно.

Ми не перший рік працюємо на ринку, тому встигли виробити певну стратегію для успішної співпраці з різними клієнтами. Для цього у нас є ряд переваг, які відрізняють наш інтернет-магазин від більшості подібних ресурсів

```
            </p>
          </div>
        </div>
      </div>
    </section>

    <BrandsSlider />

    <section className="blog">
      <div className="container">
        <div className="blog__body">
          <div className="blog__title">
            <h2>блог</h2>
          </div>
          <div className="blog__items">
            <div className="blog__item">
              <h3 className="blog__item--title">Як злити бензин з бензобаку
авто?</h3>
              <p className="blog__item--info">
                Багато автолюбителів пам'ятають ситуацію, коли після
суворого морозу взимку колеса
                на машині не рухалися. Такі речі не завжди помічали. Водій
міг ...
              </p>
              <a
                rel="noreferrer"
                href="https://amastercar.ru/blog/slivaem_benzin_s_avto.html"
                className="blog__item--link"
                target="_blank">
                Читати повністю
              </a>
            </div>

            <div className="blog__item">
              <h3 className="blog__item--title">Погано гріє пічка в авто -
причини</h3>
              <p className="blog__item--info">
                Всі причини, за якими не гріє пічка в машині, діляться на дві
групи: збої в роботі
                системи опалення, і неполадки охолоджуючої системи. ...
              </p>
              <a
```

```
            rel="noreferrer"
            href="https://unit-car.com/diagnostika-i-remont/204-ploho-greet-pechka-v-mashine.html"
            className="blog__item--link"
            target="_blank">
            Читати повністю
           </a>
          </div>

          <div className="blog__item">
           <h3 className="blog__item--title">Утеплюємо акумулятор автомобіля до зими</h3>
           <p className="blog__item--info">
            Холод сприяє прискореної розрядці акумулятора в машині. Тому взимку зранку можна
            побачити, як люди безуспішно намагаються завести авто, всю ніч ...
           </p>
           <a
            rel="noreferrer"
            href="https://amastercar.ru/blog/termo-akb.html"
            className="blog__item--link"
            target="_blank">
            Читати повністю
           </a>
          </div>
         </div>
        </div>
       </div>
      </section>

      {/*<NewItems newItems={items} />*/}

      <section className="information">
       <div className="container">
        <div className="information__body">
         <div className="information__title">
          <h2>Безпека в мережі</h2>
         </div>
         <div className="information__text">
          <p>
           Раніше про інтернет-магазинах в Україні, де можна замовляти онлайн запчастини для
           авто, мало хто чув. Такі сайти не користувалися популярністю, тому що люди ще не
           звикли до інтернету і до його можливостей та гарантій. І не дивно, адже ще недавно
           безпеку залишалася головним питанням при покупках за допомогою мережі. але тепер,
           коли системи безпеки різних сайтів вдосконалилися, такі проблеми залишилися позаду.
```

```
        Звичайно, це не означає, що варто забути про обережність,
але ймовірність
        наштовхнутися на ошуканців куди менше. Адже, якщо
користувач став жертвою нечесного
        продавця, він завжди може залишити відгук про те, у кого
проводилася покупка, або ж
        поскаржитися адміністрації сайту. Не факт, що гроші
повернутися до користувача, але
        це дозволить застерегти інших користувачів від подібної
помилки
          </p>
         </div>
        </div>
       </div>
      </section>
      <section className="service">
       <div className="container">
        <div className="service__body">
         <div className="service__items">
          <div className="service__item">
           <div className="service__item--logo">
            1
           </div>
           <div className="service__item--info">
            <h3 className="service__item--title">Доставка по Україні</h3>
            <p className="service__item--text">Нова Пошта, Justin, Meest
Express</p>
           </div>
          </div>

          <div className="service__item">
           <div className="service__item--logo">
            2
           </div>
           <div className="service__item--info">
            <h3 className="service__item--title">Гарантия
повернення</h3>
            <p className="service__item--text">14 днів після
замовлення</p>
           </div>
          </div>
         </div>

         <div className="service__items">
          <div className="service__item">
           <div className="service__item--logo">
            3
           </div>
           <div className="service__item--info">
            <h3 className="service__item--title">Підбір шин</h3>
            <p className="service__item--text">Для будь-якої моделі
```

```
авто</p>
            </div>
           </div>

           <div className="service__item">
            <div className="service__item--logo">
             4
            </div>
            <div className="service__item--info">
             <h3 className="service__item--title">Подарунки клієнтам</h3>
             <p className="service__item--text">Призи та знижки для
постійних клієнтів.</p>
            </div>
           </div>
          </div>
         </div>
        </div>
       </section>
     </main>
  );
}
```

14. Сторінка для пошуку товарі ../pages/**Shop.jsx**:

```
import React, { useEffect, useCallback, useState } from 'react';
import { BreadCrumbs, SortPopup, Filter } from '../components';
import ItemBlock from './../components/itemBlock/index';
import Axios from "axios";
import {useAuth} from "../hooks/auth.hook";
import {useHistory} from "react-router-dom";
import loadingSpinner from '../assets/img/loading-spinner.svg'
import {NotificationContainer, NotificationManager} from 'react-
notifications';

export default function Shop({handleRequest, params, handleSetParams,
searchFromHome, handleSetSearchFromHome}) {

  const history = useHistory();
  const itemsPerPage = 2;

  const sortItems = [
    {id: '1', name: 'Найновіші', type: 'year', order: 'desc' },
    {id: '2', name: 'Найстаріші', type: 'year', order: 'asc' },
    {id: '3', name: 'Найдешевші', type: 'price', order: 'desc' },
    {id: '4', name: 'Найдорожчі', type: 'price', order: 'asc' }
  ];

  const [visibleFilter, setVisibleFilter] = useState(false);
  const [items, setItems] = useState([]);
  const [selectedSort, setSelectedSort] = useState(sortItems[3]);
```

```
  const [loading, setLoading] = useState(true);


  const [page, setPage] = useState(0);
  const [itemsCount, setItemsCount] = useState(0);

  const [searchParams, setSearchParams] = useState(params);
  const [itemsCart, setItemsCart] = useState([])

  const { token, userId } = useAuth();

  const handleSetSearchParams = (params) => {
    setLoading(true);
    setSearchParams(params);
  }

  const handleCloseVisibleFilter = (e) => {
    if(e.target.className === 'filter__body-overlay') {
      setVisibleFilter(false);
    }
  }

  useEffect(() => {
    if(loading) {
      const filteredObj = Object.entries(searchParams).reduce((acc, [key,
value]) => {
        if (value !== '') {
          acc[key] = value;
        }
        return acc;
      }, {});


      Axios.get('/api/tire-crud/get-items-sorting', {
        params: {
          sortBy: selectedSort.type,
          sortOrder: selectedSort.order,
          skip: page * itemsPerPage,
          limit: itemsPerPage,
          search: JSON.stringify(filteredObj)
        }
      }).then((response) => {
        setItems(response.data);
        setItemsCount(response.data.count);
        setTimeout(() => {
          setLoading(false);
        }, 300)
      });
      if(searchFromHome) {
        handleSetSearchFromHome(false);
        handleSetParams({});
```

```
    }

  }
}, [selectedSort, page, searchParams]);

useEffect(() => {
  Axios.get('/api/tire-crud/get-items', ).then((response) => {
    setItemsCount(response.data.count);
  });

}, [])

useEffect(() => {
  if(userId !== null) {
    Axios.get('/api/cart/get-items', {
      params: {
        userId: userId
      }
    }).then((response) => {
      setItemsCart(response.data.cart.products);
    });
  }
}, [userId])


const toggleVisibleFilter = () => {
  setVisibleFilter(!visibleFilter);
};
const handleAddItemToCart = (productId, quantity = 1, price) => {
  if(!!token) {
    Axios.post('/api/cart/add',{
      userId: userId,
      productId: productId,
      quantity: quantity,
      price: price
    }).then((response) => {
      handleRequest();
    });
  } else {
    NotificationManager.warning('Додавання товарів в корзину
доступно лише після авторизації.')
  }
};

const handleSelectSort = (item) => {
  setSelectedSort(item);
  setLoading(true);
}

const handleChangePage = (page) => {
  setPage(page);
```

```
    setLoading(true);
  }

  return (
    <main>
      <NotificationContainer/>
      {visibleFilter ? <div className="filter__body-overlay" onClick={(e) =>
handleCloseVisibleFilter(e)}></div> : ''}
      <BreadCrumbs crumbs={[{ route: '/tires', label: 'Пошук шин'}]}/>
      <section className="items-selling">
        <div className="container">
          <div className="items-selling__body">
            <Filter visibleFilter={visibleFilter}
handleSetSearchParams={handleSetSearchParams}
searchParams={searchParams}/>
            {!loading ? <div className="item-list__body">
              <div className="item-list__filter">
                <div className="item-list__filter--show">
                  <p>Список товарів - {itemsCount}</p>
                </div>
                <div
                  className="item-list__filter--filter--mobile filtermobile"
                  onClick={toggleVisibleFilter}>
                  <p>Фільтр</p>
                </div>
                <SortPopup
                  items={sortItems}
                  selectedSort={selectedSort}
                  handleSelectSort={handleSelectSort}
                />
              </div>

              <div className="item-list__wrapper fd-col">
                {!loading && items && items.data &&
                  // eslint-disable-next-line
                  items.data.map((obj) =>
                      <ItemBlock token={token} inCart={itemsCart.some(item =>
item.productId == obj.id)} key={obj.id} {...obj}
handleAddItemToCart={handleAddItemToCart} />
                  )}
              </div>

              <div className="item-list__pagination">
                {Math.ceil(itemsCount/itemsPerPage) > 1 &&
Array(Math.ceil(itemsCount/itemsPerPage)).fill(1).map((el, i) =>
                    <div className={page === i ? 'active--pagination' : ''} onClick={()
=> handleChangePage(i)}>{i+1}</div>
                )}
              </div>
            </div> :
              <img src={loadingSpinner} alt="loading" className='search-page-
```

```
loading-spinner' />}
          </div>
        </div>
      </section>
      <section className="information items-seo">
        <div className="container">
          <div className="information__body">
            <div className="information__title">
              <h2>Як підібрати шини?</h2>
            </div>
            <div className="information__text">
              <p>
                1. Підбір шин відносно існуючої. Параметри шини
підбираються відповідно до діаметру, висоти і ширини профілю вже
встановлених шин. Цей варіант дозволить Вам не переплутати розмір
та купити необхідну шину.
                <br />
                2. Підбір відносно паметрів диску. Шина підбирається
виключно під параметри диску, такі як: Ширина, діаметр диску. При
цьому необхідно враховувати висоту профіля, щоб колесо не терлось
із аркою.
              </p>
            </div>
          </div>
        </div>
      </section>
    </main>
  );
}
```

15. Хук **auth.hook.js:**

```
import { useState, useCallback, useEffect } from 'react';

const storageName = 'userData';

export const useAuth = () => {
  const [token, setToken] = useState(null);
  const [userId, setUserId] = useState(null);
  const [isAdmin, setIsAdmin] = useState(false);

  const login = useCallback((jwtToken, id, admin) => {
    setToken(jwtToken);
    setUserId(id);
    setIsAdmin(admin);

    localStorage.setItem(storageName, JSON.stringify({ userId: id, token:
jwtToken, isAdmin: admin }));
  }, []);

  const logout = useCallback(() => {
```

```
    setToken(null);
    setUserId(null);
    setIsAdmin(null);

    localStorage.removeItem(storageName);
  }, []);

  useEffect(() => {
    const data = JSON.parse(localStorage.getItem(storageName));
    if (data && data.token) {
      login(data.token, data.userId, Boolean(data.isAdmin));
    }
  }, [login]);

  return { login, logout, token, userId, isAdmin };
};
```

16. Хук **http.hook.js:**

```
import { useState, useCallback } from 'react';

export const useHttp = () => {
  const [loading, setLoading] = useState(false);
  const [error, setError] = useState(null);
  const request = useCallback(async (url, method = 'GET', body = null,
headers = {}) => {
    setLoading(true);
    try {
      if (body) {
        body = JSON.stringify(body);
        headers['Content-Type'] = 'application/json';
      }
      const response = await fetch(url, { method, body, headers });
      const data = await response.json();
      if (!response.ok) {
        throw new Error(data.message || 'Something goes wrong :(');
      }

      setLoading(false);
      return data;
    } catch (e) {
      setLoading(false);
      setError(e.message);
      throw e;
    }
  }, []);

  const clearError = useCallback(() => setError(null), []);
  return { loading, request, error, clearError };
};
```