МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

Секція інформаційно-комунікаційних технологій

ВИПУСКНА РОБОТА

на тему:

«Інтернет-магазин з використанням React.js»

Завідувач
випускаючої кафедри **Довбиш А. С.**

Керівник роботи **Проценко О.Б.**

Студента гр. ІН-73 **Авхутський А.І.**

СУМИ 2021

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

Кафедра комп'ютерних наук

ЗАТВЕРДЖУЮ
Зав. кафедрою

_____ А.С. Довбиш
«__» _____ 2021 р.

ЗАВДАННЯ
до випускної роботи

Студента четвертого курсу, групи ІН-73 спеціальності «Комп'ютерні науки» денної форми навчання Авхутського Антона Ігоровича.

Тема: «Інтернет-магазин з використанням React.js»

Затверджена наказом по СумДУ

№_____ від _____2021 р.

Зміст пояснювальної записки: 1) аналіз проблеми та постановка задачі; 2) вибір метода розв'язання задачі; 3) розробка інформаційного і програмного забезпечення системи

Дата видачі завдання «____»_____2021 р.

Керівник випускної роботи _____Проценко О.Б.

Завдання прийняв до виконання _____Авхутський А.І.

РЕФЕРАТ

Записка: 51 стор., 26 рис., 1 табл., 3 додатки, 10 джерел.

Об’єкт дослідження – сфера інтернет-торгівлі в охоронній діяльності

Мета роботи – розробити інтернет-магазин з продажу охоронних інструментів з використанням актуальних інструментів для створення веб-ресурсів

Методи дослідження – експертний аналіз інтернет-магазинів аналогів, експертний аналіз інструментів для розробки інтернет-магазину

Результати – розроблений інтернет-магазину бренду «MONO» за допомогою стеку MERN (MongoDB, Express.js, React.js, Express.js) який дозволяє користувачам легко купувати охоронні системи.

ІНТЕРНЕТ-МАГАЗИН, MERN, MONGODB, REACTJS, EXPRESSJS, NODEJS, АСИНХРОННЕ ПРОГРАМУВАННЯ

ЗМІСТ

ВСТУП

Безпека приватної власності – понад усе. Роками людство працює над тим, щоб захистити свою власність від недоброзичливих громадян. З’являються більш розвинені охоронні системи, які дозволяють у реальному часі відслідковувати стан майна, яким вони володіють. Попит на такі системи залишається незмінно високим.

Разом з цим, з’являється питання, як правильно їх продавати. Тут на допомогу приходить всесвітня мережа інтернет, яка дозволяє створити власний інтернет-магазин. В ньому потенційний покупець може переглянути наявний асортимент, порівняти ціни, характеристики, у декілька кроків замовити потрібний товар, послугу або проконсультуватися з менеджером. Це якщо говорити про еталонний інтернет-магазин.

В більшості випадків, не дивлячись на ріст охоронних та інтернет-технологій, сайти магазинів частіше за всього повільні, не відповідають трендам UI/UX або їх взагалі не існує.

На основі аналізу ринку аналогів було сформульовано мету дипломної роботи: розробка інтернет-магазину продажу охоронних систем з використанням JavaScript.

Для досягнення мети необхідно вирішити наступний перелік задач:

– проаналізувати предметну область;
– розробити структуру інтернет-магазину;
– розробити інтернет-магазин;
– додати товари до інтернет-магазину;
– провести тестування продукту.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Актуальність проблеми

Перш за все, коли ми говоримо про інтернет-магазин, ми повинні зрозуміти що це – інструмент, який повинен принести прибуток на продажі товарів, які в наявності. Саме тому важливо не лише знати, які зараз тренди торгівлі, а також при розробці розуміти, які наразі є тренди в дизайні (UI/UX), які інструменти краще використовувати під час розробки.

Перед тим, як створити інтернет-магазин, потрібно проаналізувати:

- Актуальність існуючого інструментарію для розробки: мови програмування, фреймворки;
- Працездатність інструментарію, який ми плануємо використовувати;
- Тенденції торгівлі, тренди UI/UX.

Замовник повинен спільно з розробником проаналізувати потрібні аспекти, спираючись на user experience, адже основна ціль – це не існування майданчику для торгівлі, а прибуток з реалізації товарів, що на ньому розташовані.

Роль розробника в цьому процесі – це бути в курсі веб-технологій, які вплинуть на зручність та працездатність інтернет-магазину. Окрім цього, розробник повинен в загальних аспектах розуміти специфіку бізнесу, для якого виконує замовлення, адже це допоможе правильно проаналізувати потрібні інструменти та тренди UI/UX.

Сфера інтернет-магазинів переходить до мінімалістичності, адже все що користувачу потрібно – переглянути наявний асортимент, додати потрібні товари/послуги у кошик, ввести деталі для отримання замовлення та отримати його у короткий строк.

Окрім цього, інтернет-магазин дозволяє зекономити кошти, тому що для його розробки треба замовити розробника, який виконає роботу, а для підтримки потрібно оплачувати хостинг та послуги адміністратора сайту. В залежності від масштабів продаж, наймаються онлайн-консультанти, які контролюють процес

замовлень: передача їх до поштових сервісів та доставка до покупця, але навіть це можна замовити на аутсорсі.

Приміщення потрібні лише для складів, на яких зберігатиметься товар з інтернет-магазину.

Тому інтернет-магазин це не лише зручність цільової аудиторії виробника, а також економія його коштів завдяки автоматизації багатьох процесів.

1.2 Аналіз існуючих аналогів

Перед розробкою інтернет-магазину продажу охоронних систем, потрібно проаналізувати аналоги. Варто згадати, що цільова аудиторія доволі широка, адже охорони потребують всі: від домівок до великих компаній, від автомобілів до вантажівок тощо. Перш за все, проблема полягає в тому, що більшість інтернет-магазинів не відповідають трендам UI/UX. Недолік інтернет-магазині – це максимальна економія, що впливає на кількість продаж через повільну роботу або незручність у використанні, а відповідно на прибуток компанії, яка займається реалізацією товару.

Першим аналогом інтернет-магазину з продажу охоронних систем є Ohrana.ua. Ohrana.ua – це магазин з продажі різних охоронних систем від різноманітних виробників [1].

Виконавши аналіз відгуків від користувачів магазину, було отримано наступні дані:

– Висока швидкість виконання замовлення;
– Гарне ставлення співробітників інтернет-магазину до клієнтів;

Проаналізувавши сайт самостійно, було отримано наступні дані:

- UI/UX сайту не відповідає останнім трендам, через що є ризик втрати певної кількості замовлень;
- Сайт є першим у пошуковій видачі при Google-запиті «охранные системы Украина»;

- Велика кількість неструктурованої інформації на головній сторінці, через що потенційний клієнт може розгубитися та не зробити замовлення;

На рисунку 1.1 представлена початкова сторінка Ohrana.ua.

Рисунок 1.1 – Початкова сторінка Ohrana.ua

Платформа передбачає реєстрацію користувача. Проте, немає можливості авторизації за допомогою соціальних мереж, що значно пришвидшило би реєстрацію/авторизацію. Реєстраційну форму можна переглянути на рисунку 1.2.

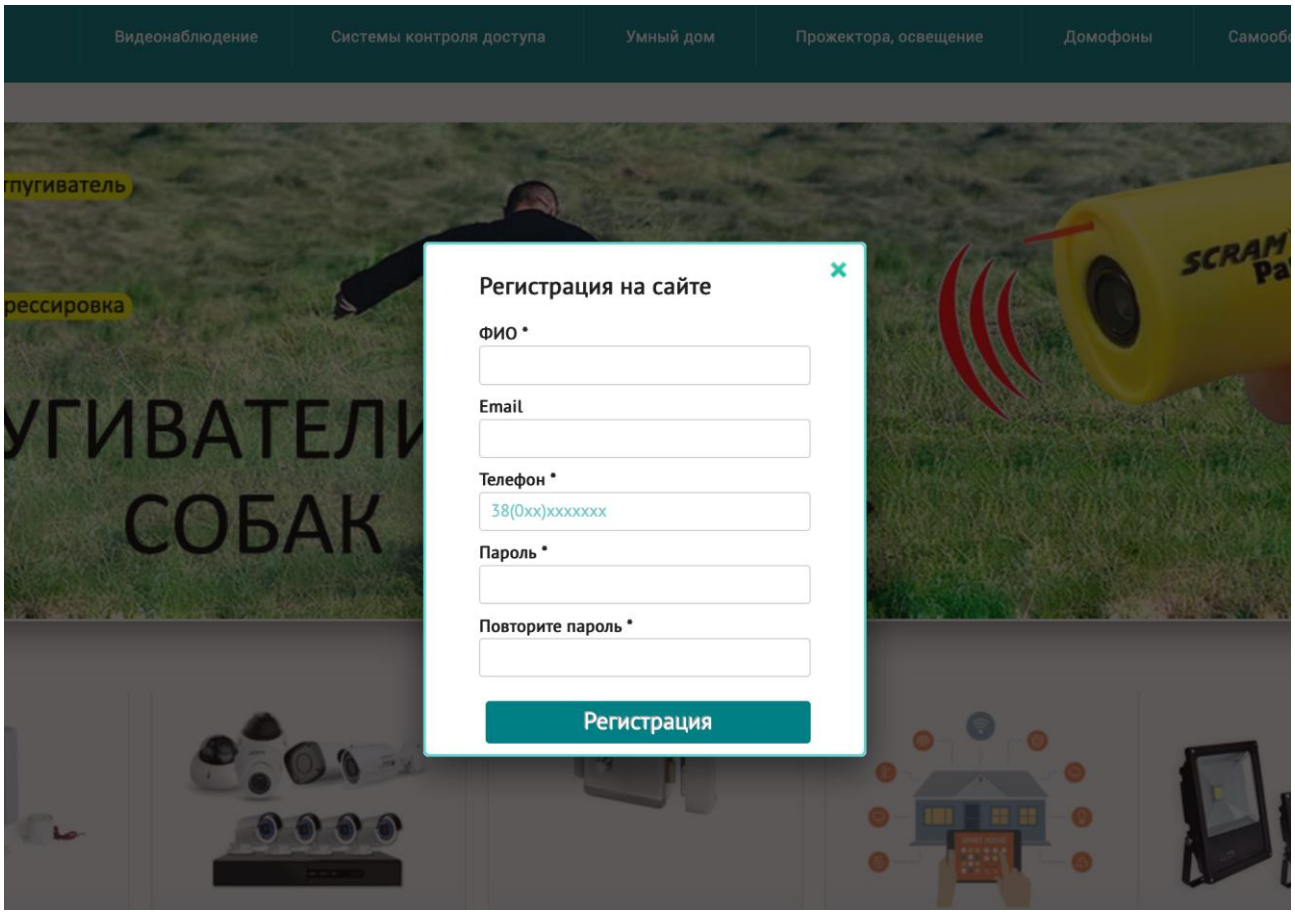

Рисунок 1.2 – Реєстраційна форма Ohrana.ua

Наступний аналог розглянутих інтернет-магазинів– «Охоронні системи», інтернет-магазин охоронних систем [2]. Авторизуватися на даному сайті можна за допомогою Facebook або «Вконтакте», проте друга соціальна мережа є забороненою в Україні. Форма реєстрації не відрізняється нічим від попереднього аналогу.

Рисунок 1.3 – Інтернет-магазин «Охоронні системи»

Дане зображення демонструє, який асортимент пропонується користувачу при першому відвідуванню сторінки. Недоліки даного інтернет-магазину наступні:

– Невідповідність трендам UI/UX, що візуально відштовхує від себе потенційнії покупців;

– Повільне завантаження сайту.

Ще одним розглянутим аналогом є інтернет-магазин Secur [3]. Магазин відповідає актуальним трендам UI/UX, що позитивно впливає на кількість покупців. Головна сторінка надає нам доступ до каталогу, контактів та іншої інформації про магазин, має структуровану панель навігації.

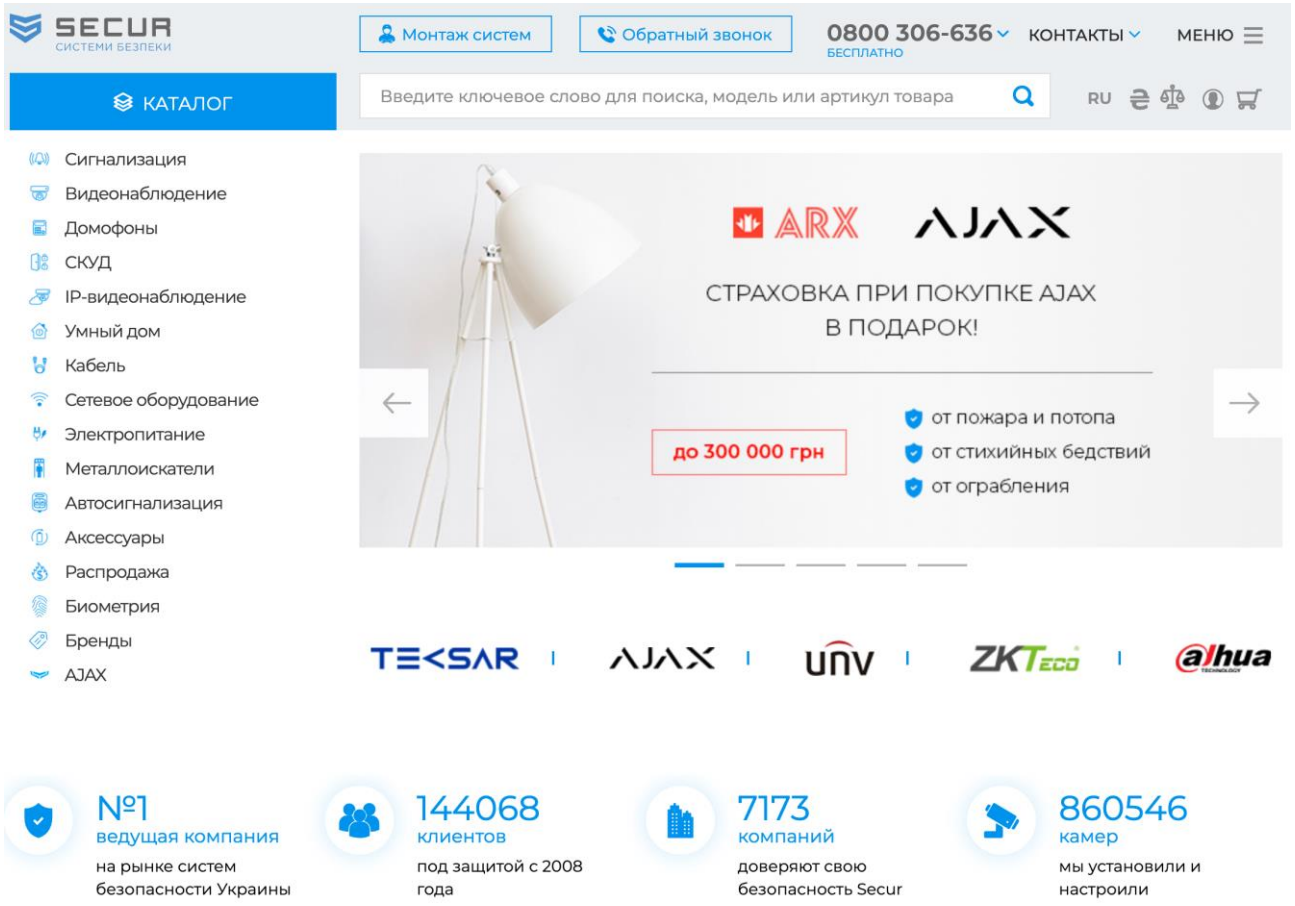

Рисунок 1.4 – Інтернет-магазин Secur

Згідно аналізу аналогів, була сформована порівняльна таблиця розглянутих прикладів та майбутнього інтернет-магазину. Результати аналізу представлено у таблиці 1.1.

Таблиця 1.1 – Порівняльна характеристика аналогів

Назва критерію	Назва ресурсу			
	Ohrana.ua	Охоронні системи	Secur	MONO
1. Використання ресурсу без реєстрації	+	+	+	+
2. Зручність навігації	-	-	+	+
3. Адаптивність сайту під мобільну версію	-	+	+	+

1.3 Постановка задачі

Метою випускної роботи є розробка інтернет-магазину з продажу охоронних систем «MONO» з використанням Javascript-стеку MERN. Продукт дозволить задовольнити потреби цільової аудиторії, яка потребує охорону їх власності.

Інтернет-магазин матиме в собі такі функції:

- Перегляд наявного асортименту товарів охоронної індустрії;
- Перегляд характеристик обраних товарів;
- Перегляд відгуків про наявні товари в інтернет-магазині;
- Можливість додавання товарів у кошик;
- Можливість реєстрації та авторизації;
- Підключення платіжної системи;

Інтернет-магазин «MONO» буде розроблений за останніми трендами UI/UX, що закріпить магазин на ринку охоронної індустрії серед аналогічних сервісів. Також інтерфейс буде адаптивним для різних типів пристроїв, адже на сьогоднішній день більшість користувачів віддають перевагу сайтам, в яких є можливість зручно оформити замовлення з телефону або планшету.

Для реалізації дипломного проекту необхідно реалізувати наступне:

- провести аналіз технологій розробки інтернет-магазину;
- розробити інтерфейс інтернет-магазину;
- спроектувати базу даних;
- підключення бази даних до інтернет-магазину;
- тестування коректної роботи інтернет-магазину.

2 ІНСТРУМЕНТИ ДЛЯ ВИРІШЕННЯ ПОСТАВЛЕНИХ ЗАДАЧ

Процес розробки інтернет-магазинів включає наступні етапи:

- розробка макету сайту;
- верстка шаблону;
- програмування інтерактивних елементів;
- наповнення сайту товаром;

У зв’язку з вище зазначеними етапами розробки та необхідність наявності вище згаданих елементів, було прийнята рішення про використання наступних технологій:

- MongoDB;
- Express.js;
- React.js;
- Node.js;

Дана робота є яскравим прикладом використання MERN [5] – стек, який передбачає використання мови програмування Javascript на кожному з рівнів – front-end (React.js), back-end (MongoDB, Express.js, Node.js).

MongoDB – документо-орієнтована система керування базами даних, яка не потребує опису схеми таблиць та оперує даними у форматі ключ/значення. [6]

MongoDB розроблена на C++, що забезпечує високу продуктивність та швидкість при роботі з нею.

Express.js – програмний каркас, який допомагає при розробці серверної частини веб-застосунків для Node.js. [7]

Express надає чимало готових абстракцій, які полегшують розробку сервера та серверної логіки – наприклад, обробка надісланих форм, робота з куками, Cors-origin resource sharing (CORS). [8]

React.js – відкритий фреймворк для розробки front-end частини сайту, що створений на основі мови програмування Javascript та покликаний вирішувати проблеми часткового оновлення вмісту веб-сторінки. [9]

Node.js – платформа, що надає можливість виконувати JavaScript-скрипти на сервері та відправляти користувачеві результат їхнього виконання. [10]

Платформа Node.js дозволяє розробляти високопродуктивні мережеві застосунки, написаних мовою програмування JavaScript. Окрім цього, також використовується для створення клієнтських та серверних програм.

В платформі використовується V8 – рушій, розроблений компанією Google, а також імплементовані принципи асинхронного програмування, що дозволяє уникнути потокового голодування та забезпечити високу продуктивність веб-застосунку.

2.1 Проектування інтернет-магазину

Для того, щоб перейти до програмної реалізації інтернет-магазину, потрібно спроектувати його наповнення, а також його зв’язок із сервером та базою даних у базових процесах.

Спочатку ми розробимо структуру інтернет-магазину для користувача. Він повинен мати у собі такі сторінки:

- Головна сторінка;
- Сторінка продукту;
- Кошик;
- Форма реєстрації/авторизації;
- Форма оформлення замовлення;
- Платіжна форма, на якій користувач може оплатити замовлення;

Результат представлено на рис.2.1.

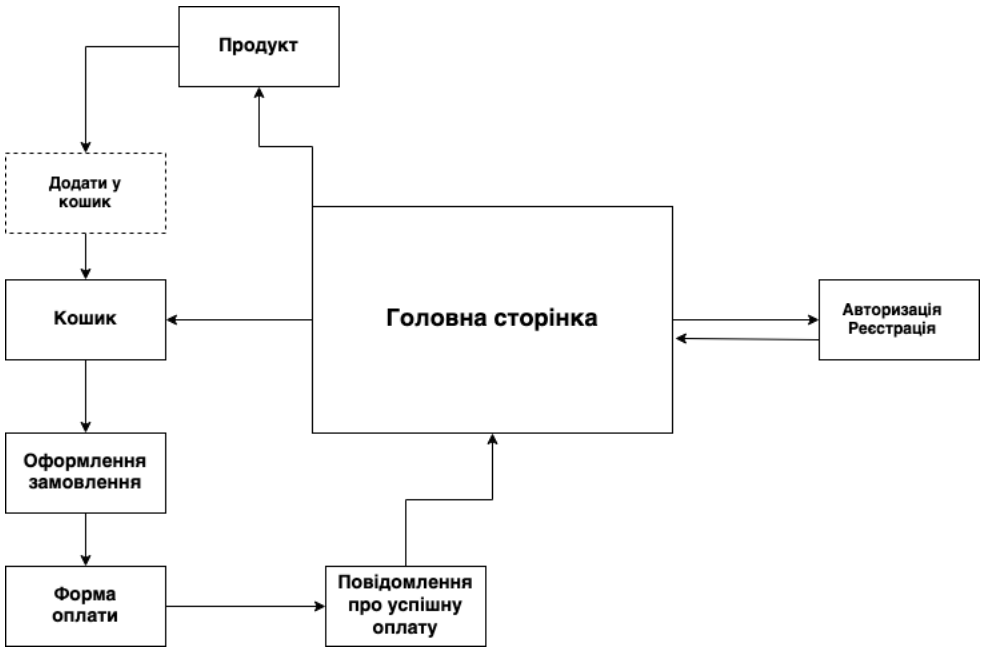

Рис.2.1. Базова структура сайту

Тепер спроектуємо, як працюватиме наш сайт із сервером та базою даних. Для цього ми розглянемо такі процеси:

1. Поява товарів на сторінках сайту та деталей про них
2. Додавання у кошик користувачем
3. Оформлення замовлення
4. Оплата замовлення

На рис.2.2 можна переглянути деталізацію процесів 1 та 2.

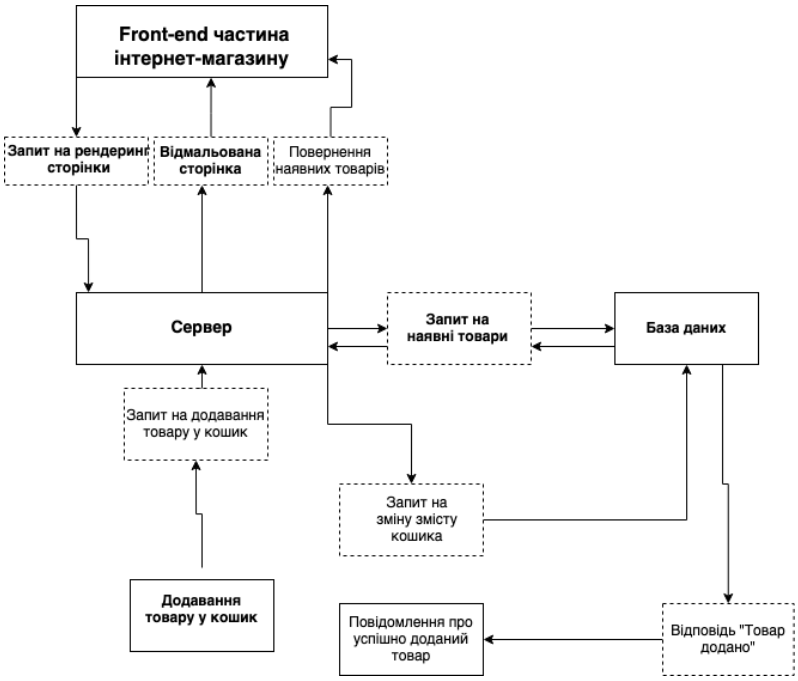

Рис.2.2 Деталізація процесів «Поява товарів на сторінках сайту та деталей про них» та «Додавання у кошик користувачем»

На рис.2.3 представлено деталізацію процесів «Оформлення замовлення» та «Оплата замовлення».

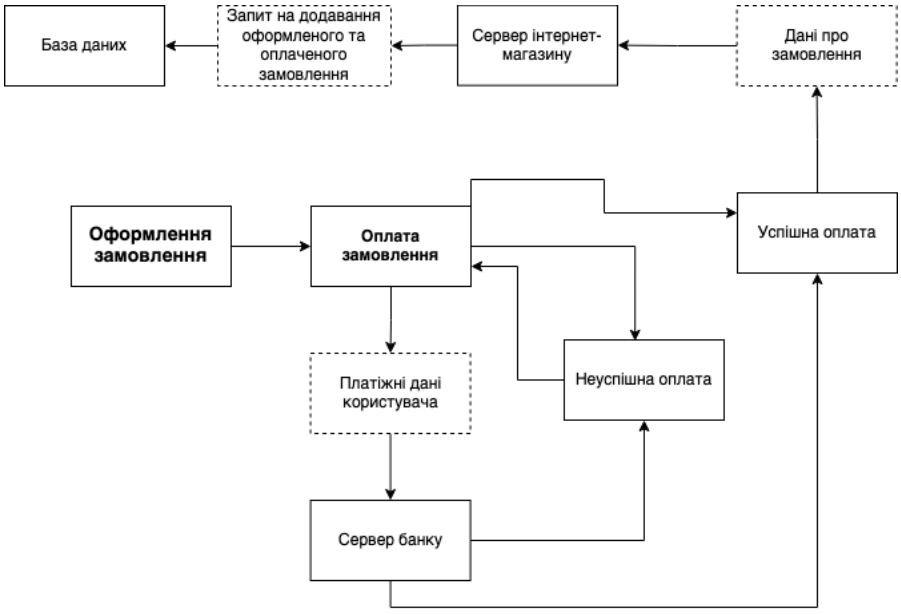

Рис.2.3 Деталізація процесів «Оформлення замовлення» та «Оплата замовлення»

Дані деталізації процесів показують, що наш сайт базуватиметься на класичній клієнт-сервісній архітектурі RESTful, коли клієнт подає запит на

сервер, в цей же час сервер оброблює та надає відповідь клієнту, який займається відображенням даних. Окрім цього, все це працюватиме асинхронно, тобто клієнт не чекатиме поки завантажиться усе, а спочатку проведе рендеринг головної сторінки, а вже потім заповнить її товарами.

Згідно розробленої та деталізованої структури сайту, ми можемо переходити до реалізації нашого інтернет-магазину.

3 РЕАЛІЗАЦІЯ ІНТЕРНЕТ-МАГАЗИНУ З ПРОДАЖУ ОХОРОННИХ СИСТЕМ «MONO»

3.1 Реалізація дизайну інтернет-магазину

На основі розробленої структури та деталізації процесів, ми можемо переходити безпосередньо до розроблення дизайну інтернет-магазину.

Інтернет-магазин повинен відповідати трендам UI/UX, тобто:

- Привертати увагу користувача своїм зовнішнім виглядом;
- Мати інтуїтивний інтерфейс, який підійде для різних цільових груп та не залишить негативний досвід роботи з ним у користувача;
- Не навантажувати око користувача, адже навантаження може відвернути увагу покупця від інтернет-магазину, а відповідно це втрата продажі.

На рис.3.1 представлено головну сторінку інтернет-магазину.

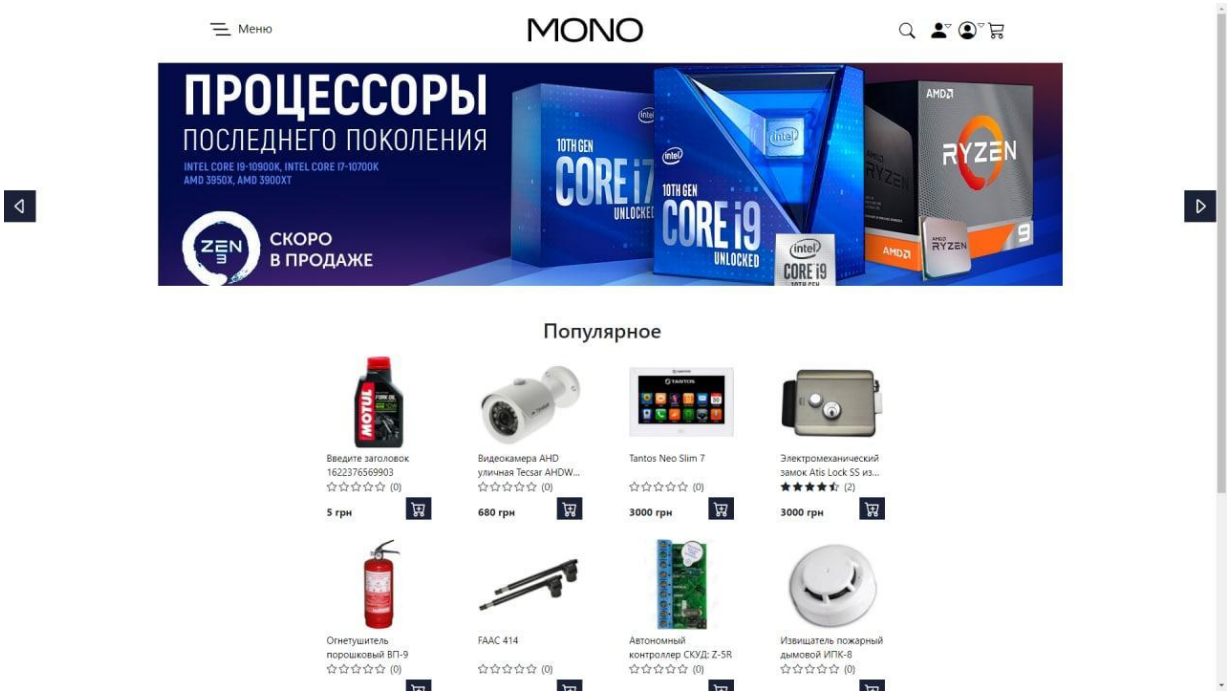

Рисунок 3.1 – Головна сторінка інтернет-магазину

Як бачимо, було розроблено сучасний дизайн з використанням засобів React.js та Sass. Для того, щоб продовжити роботу над розробкою інтернет-магазину, потрібно спроектувати базу даних, яка буде однією зі складових фундаменту сервісу.

3.2 Реалізація бази даних

Після того, як було визначено та спроектовано структуру інтернет-магазину, для повноцінної та коректної роботи інтернет-магазину необхідно реалізувати базу даних та підключити до інтернет-магазину. База даних інтернет-магазину «MONO» складається із таких таблиць: userSchema, orderSchema, shippingAddress, orderItems, paymentResult, productSchema, reviewSchema. На рисунку 3.6 зображена структура бази даних інтернет-магазину, а саме те що пов’язане із замовленням товару.

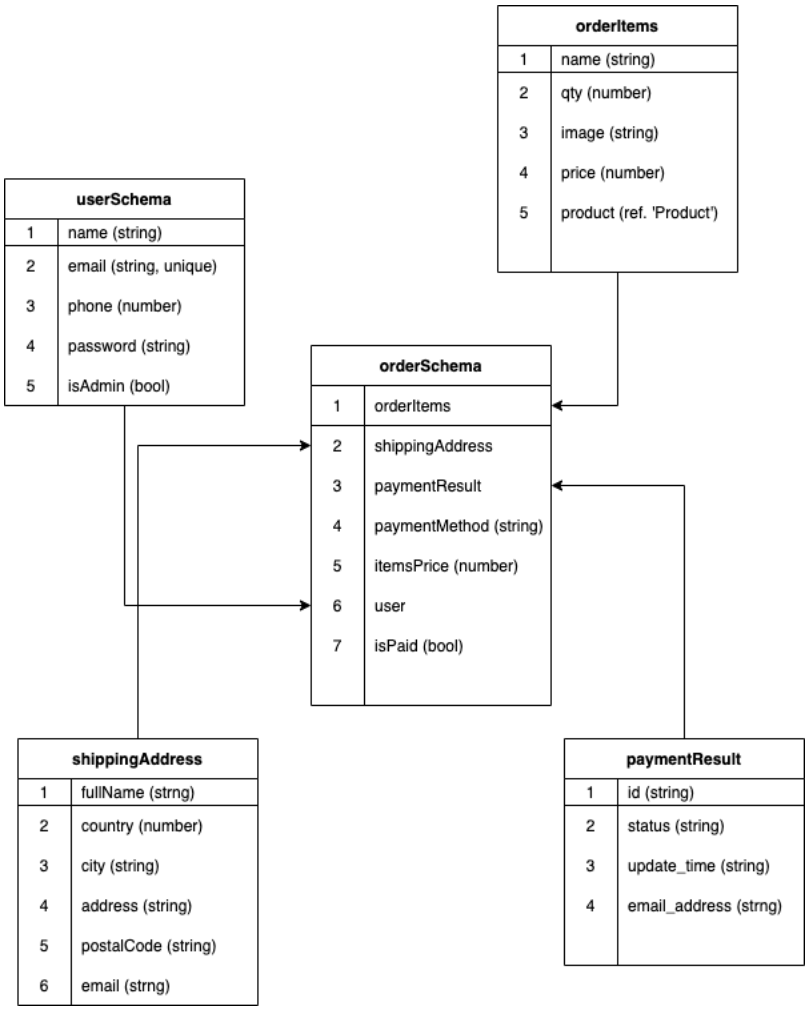

Рисунок 3.6 – Схема таблиць, пов’язаних з користувачами та замовленнями інтернет-магазину «MONO»

На рисунку 3.7 зображено діаграму таблиць бази даних, що пов’язані з відгуками

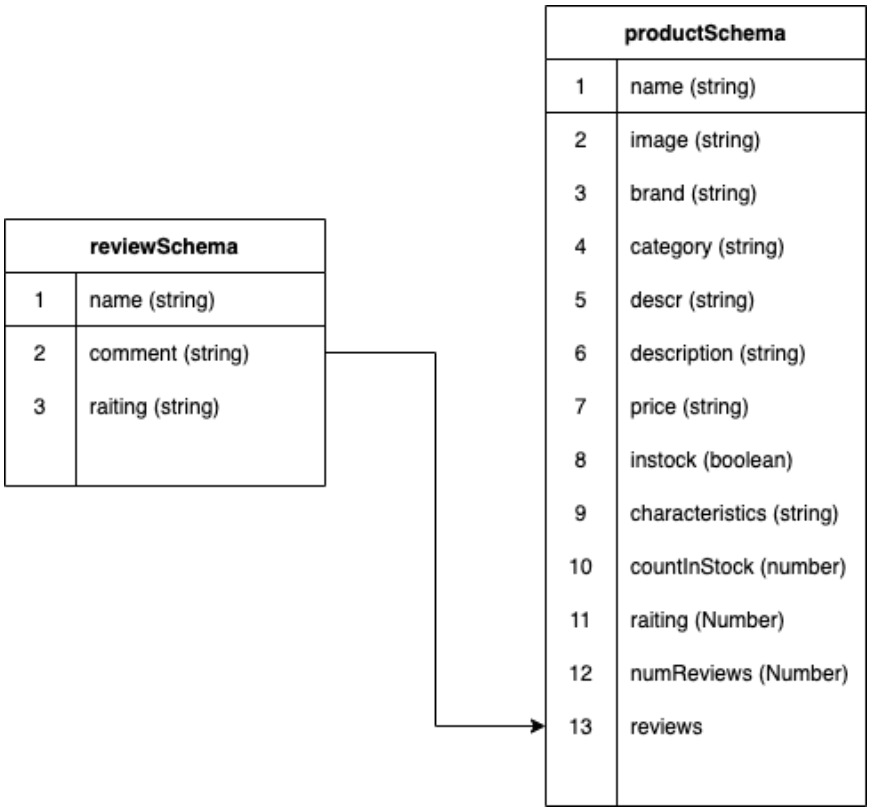

Рисунок 3.7 – Схема таблиць, пов'язаних з відгуками та товарами інтернет-магазину «MONO»

Створення та підключення до бази даних MongoDB відбувається за допомогою засобів Node.js/Express.js. На рис.3.8, рис.3.9 та рис.3.9 – вихідний код кожної з таблиць.

backend > models > JS user.js > ...

```
1  import mongoose from 'mongoose';  483.5K (gzipped: 119.3K)
2
3  const userSchema = new mongoose.Schema({
4      name: {type: String, required: true},
5      email: {type: String, required: true, unique: true},
6      phone: {type: Number, required: true},
7      password: {type: String, required: true},
8      isAdmin: {type: Boolean, default: false, required: true}
9      },
10     {
11     timestamps: true
12     }
13 );
14
15 const User = mongoose.model("User", userSchema);
16
17 export default User;
18
```

Рисунок 3.8 – Вихідний код таблиці userSchema

```
backend > models > JS order.js > ...
1   import mongoose from 'mongoose';  483.5K (gzipped: 119.3K)
2
3   const orderSchema = new mongoose.Schema({
4       orderItems: [{
5           name: {type: String, required: true},
6           qty: {type: Number, required: true},
7           image: {type: String, required: true},
8           price: {type: Number, required: true},
9           product: {type: mongoose.Schema.Types.ObjectId,
10          ref: 'Product', required: true}
11      }],
12      shippingAddress: {
13          fullName: {type: String, required: true},
14          country: {type: Number, required: true},
15          city: {type: String, required: true},
16          address: {type: String, required: true},
17          postalCode: {type: String, required: true},
18          email: {type: String}
19      },
20      paymentResult: {id: String, status: String, update_time: String,
        email_address: String},
21      paymentMethod: {type: String, required: true},
22      itemsPrice: {type: Number, required: true},
23      user: {type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true}
        ,
24      isPaid: {type: Boolean, default: false},
25      paidAt: {type: Date},
26  },
27      {
28          timestamps: true,
29      }
30  );
31
32  const Order = mongoose.model('Order', orderSchema);
33
34  export default Order;
```

Рисунок 3.9 – Вихідний код таблиці orderSchema

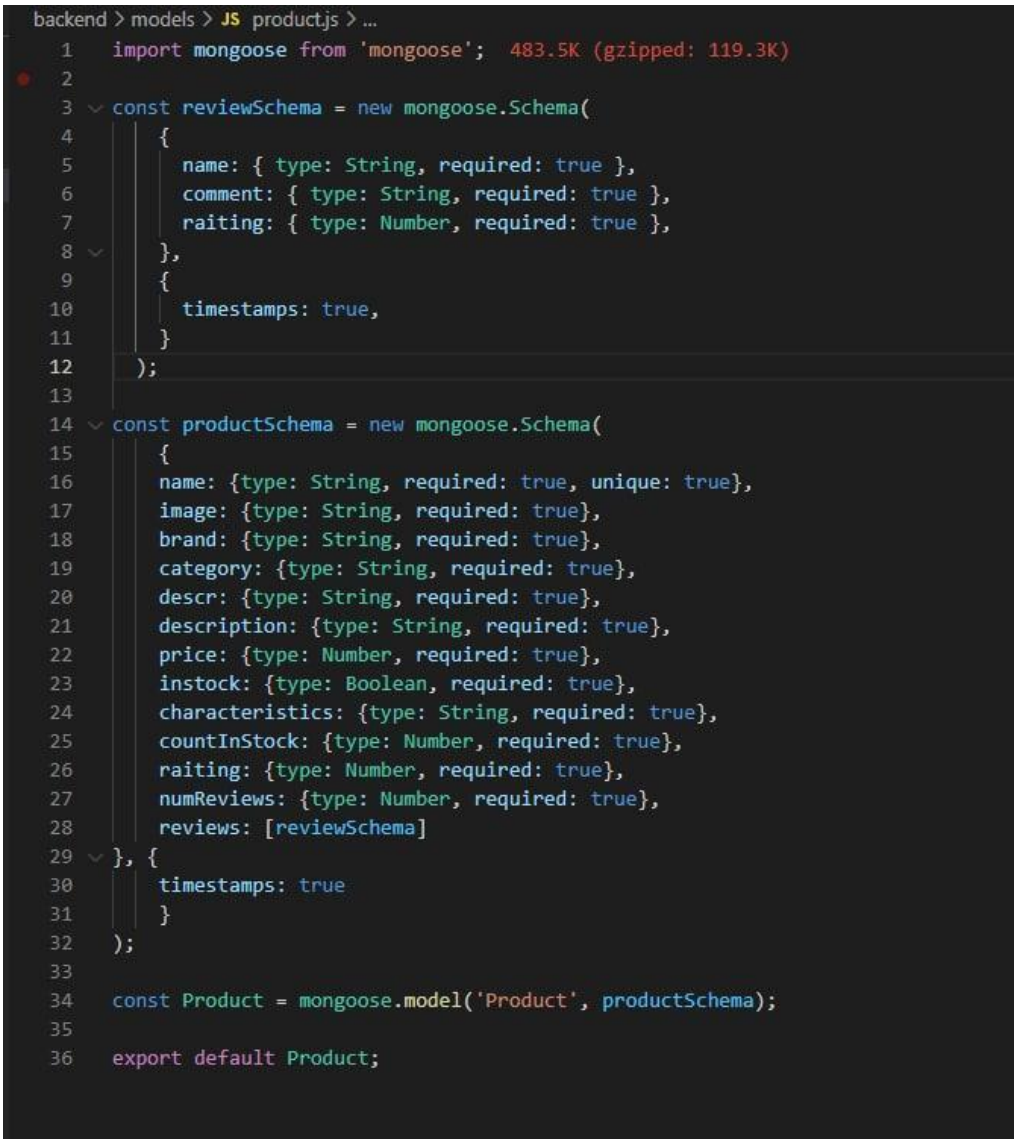

Рисунок 3.10 – Вихідний код таблиць reviewSchema та productSchema

3.3 Розробка повного інтерфейсу інтернет-магазину

Так як вже було створено дизайн головної-сторінки інтернет магазину, при розробці наступних сторінок ми спиратимемося на неї, а саме:

- Мінімалістичність;
- Відсутність складних елементів;
- Білі та чорні тони;
- Логічне розташування блоків

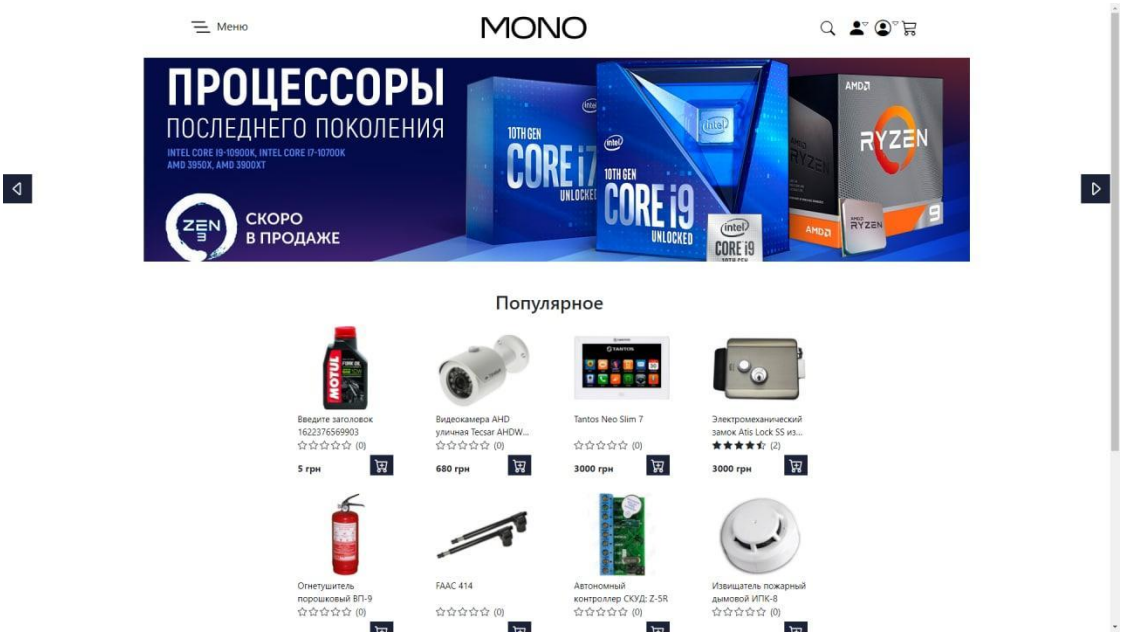

Рисунок 3.9 – Головна сторінка інтернет-магазину

Завдяки React.js, було реалізовано взаємозв'язок між сторінками згідно спроектованої структури, а також зв'язок між frontend-частиною та backend.

Панель навігації сайту має в собі наступні кнопки, які перенаправляють на сторінки: «Каталог», «Пошук», «Особистий кабінет», «Кошик» (рис. 3.10).

Рисунок 3.10 – Навігація інтернет-магазину у шапці

«Кошик» зберігає у собі товари, що були обрані користувачем в інтернет-магазині. Ця сторінка показує загальну вартість замовлення, а також допомагає вказати, скільки товару йому потрібно. Після цього, він натискає на кнопку «Оформити замовлення», яка перенаправляє його на форму для замовлення (рисунок 3.12).

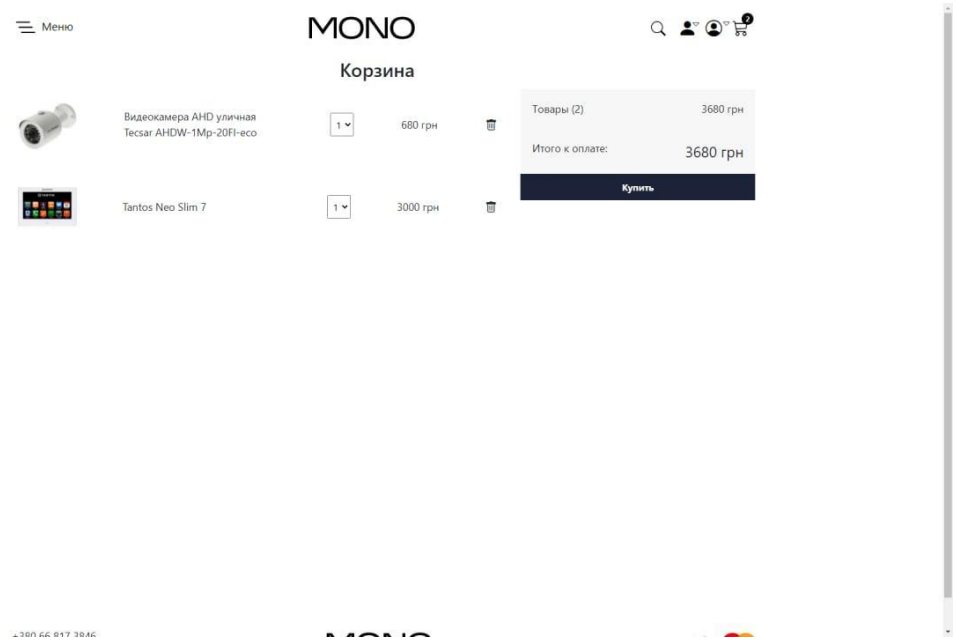

Рисунок 3.11 – сторінка «Кошик» з обраними товарами

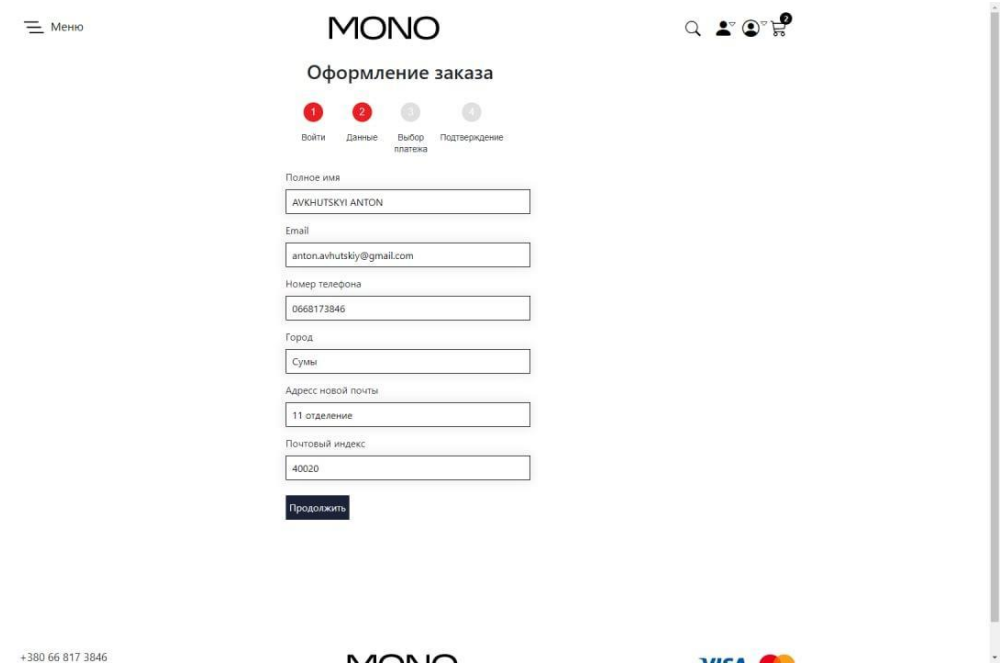

Рисунок 3.12 – Меню оформлення замовлення

Після заповнення потрібних даних, користувач обирає як хоче оплатити товар (рис.3.13).

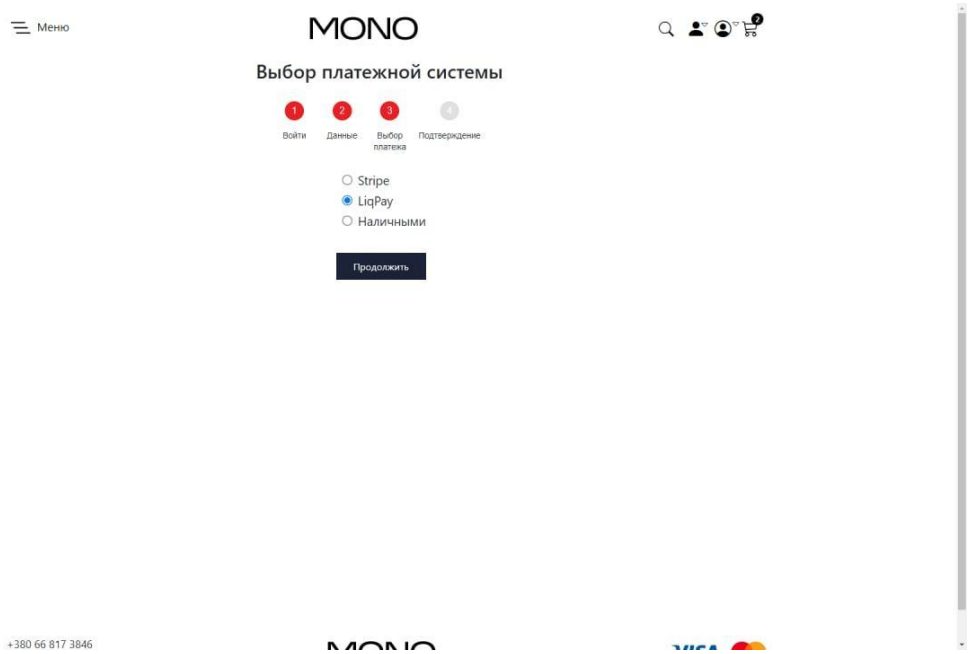

Рисунок 3.13 – Сторінка вибору платіжного методу

Якщо користувач обирає оплату карткою (Stripe, LiqPay), він буде перенаправлений на сторінку оплати (рис.3.14).

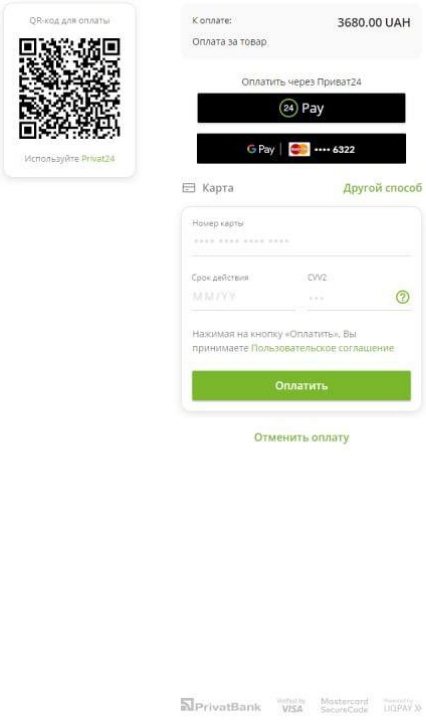

Рисунок 3.14 – Сторінка оплати замовлення

Сторінка товару є мінімалістичною та має усі потрібні блоки – коротка характеристика товару, фото, ціна (рис.3.15). За потреби, користувач може переглянути більш детальну характеристику товару (рис.3.16), а також відгуки від покупців (рис.3.17).

Рисунок 3.15 – Приклад сторінки з товаром у інтернет-магазині

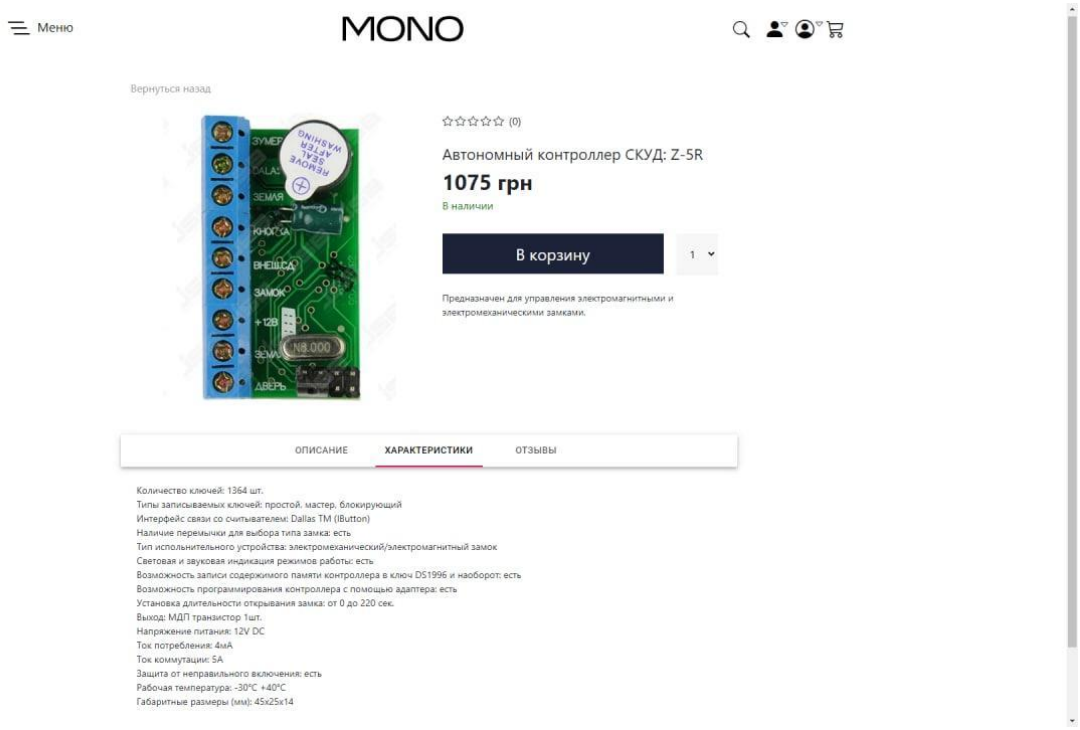

Рисунок 3.16 – Відображення детальної характеристики товару у інтернет-магазині

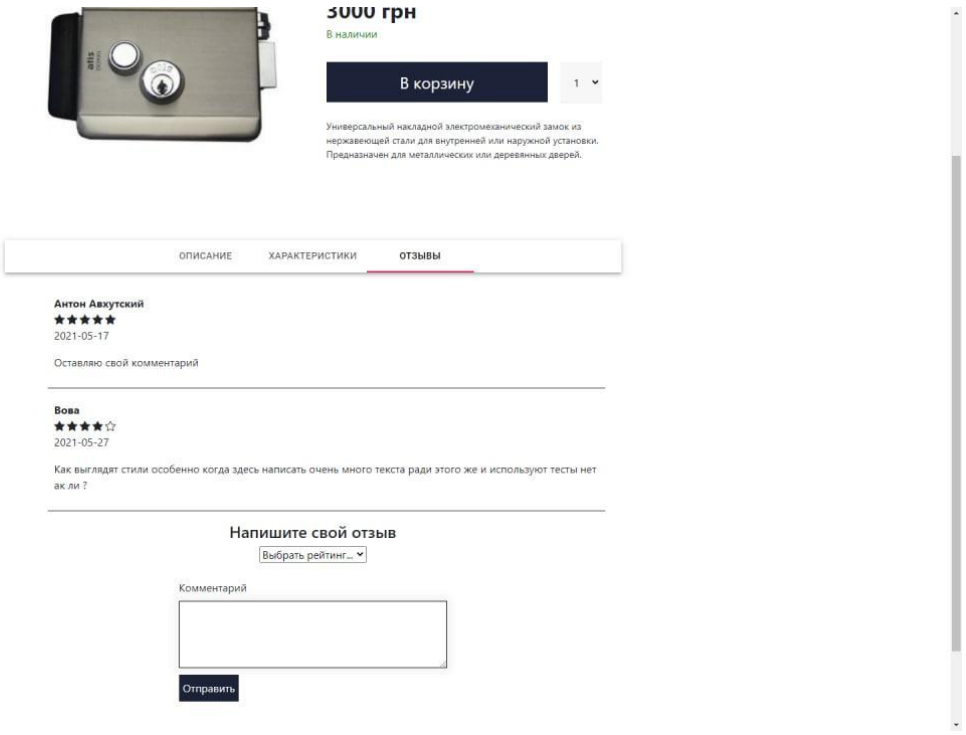

Рисунок 3.17 – Відображення відгуків від покупців товару у інтернет-магазині

Інтернет-магазин має інтуїтивну адміністраторську панель, в якій адміністратори можуть відредагувати товар (рис.3.18), переглянути список користувачів в інтернет-магазині (рис.3.19) та побачити історію замовлень (рис.3.20). Така адміністраторська панель дозволяє будь-кому швидко розібратися у керуванні інтернет-магазином – як досвідченому розробнику, так і звичайному менеджеру.

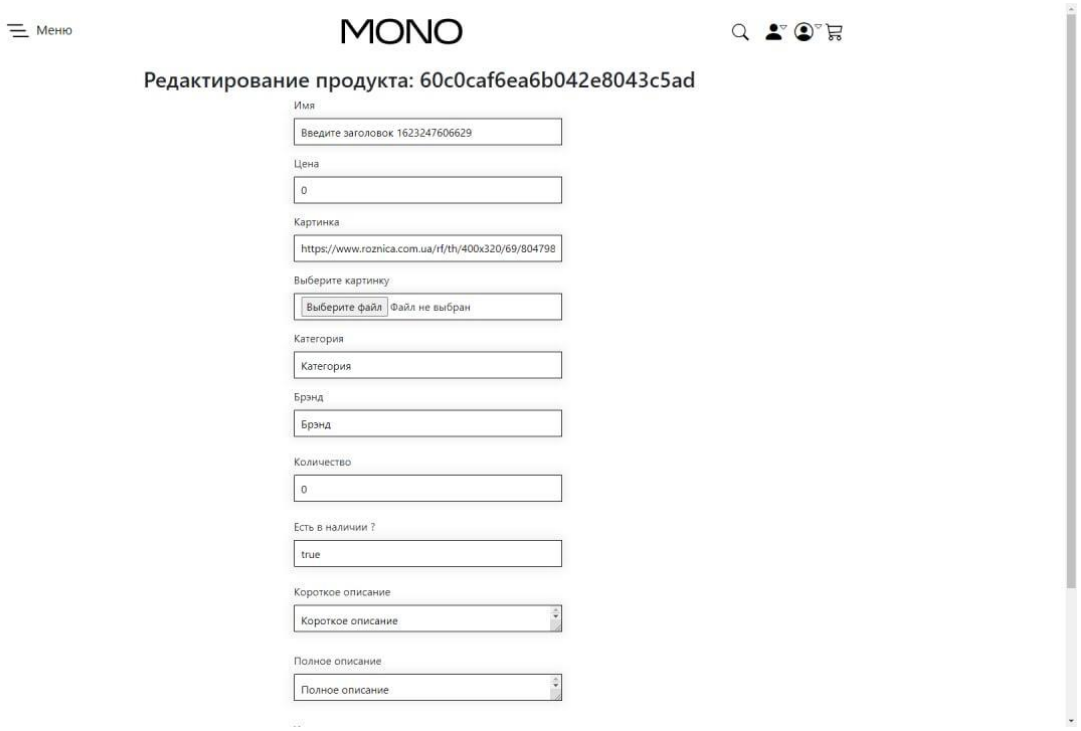

Рисунок 3.18 – Редагування товару в інтернет-магазині

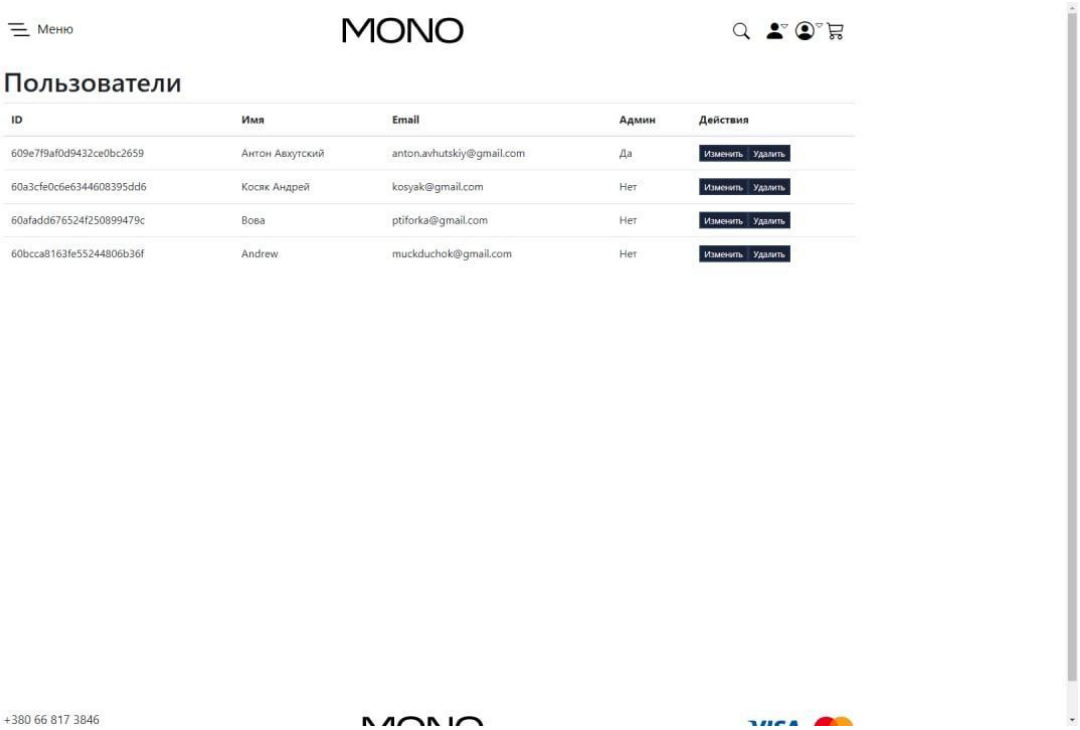

Рисунок 3.19 – Список користувачів в інтернет-магазині

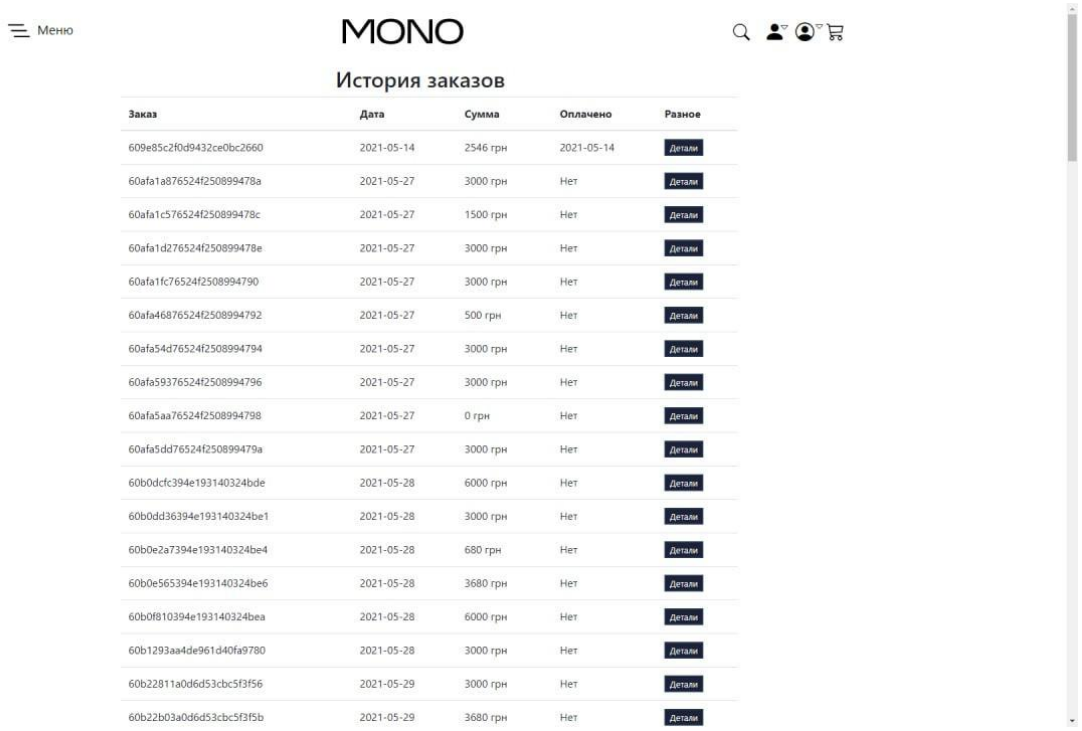

Рисунок 3.20 – Історія замовлень в інтернет-магазині

Для того, щоб користувач міг будь де та в будь-який час замовити товар, інтернет-магазин було адаптовано для мобільних пристроїв за допомогою засобів React.js (рис.3.21). Це є перевагою серед аналогів, адже адаптивність сайту для мобільних пристроїв на сьогоднішній день стоїть на першому місці.

Рисунок 3.21 – мобільна версія інтернет-магазину

ВИСНОВКИ

Під час виконання випускної роботи «Інтернет-магазин з використанням React.js» був розроблений сучасний інтернет-магазин з продажу охоронних систем.

Розглянуто аналоги, які допомогли у проектуванні інтернет-магазину з врахуванням переваг та недоліків конкурентів.

Спроектовано структуру сайта та його зв’язок з back-end частиною.

Після того, як був реалізований інтернет-магазин, було проведено його тестування. Результати даного етапу описані нижче:

- Інтернет-магазин є адаптивним для різноманітних типів пристроїв;
- Сервіс привертає увагу потенційних покупців, що впливає на купівлю продукції, представленої на сайті;
- Інтернет-магазин має інтуїтивний дизайн, який допомагає користувачу якомога швидше розібратися в його роботі та у декілька простих кроків оформити замовлення;

СПИСОК ЛІТЕРАТУРИ

1. Інтернет магазин ohrana.ua [Електронний ресурс] – режим доступу: https://ohrana.ua
2. Інтернет-магазин «Охоронні системи» [Електронний ресурс] – режим доступу: https://oc.com.ua/
3. Інтернет-магазин «Secur» [Електронний ресурс] – режим доступу: https://secur.ua
4. Що таке MERN-стек, та як з ним працювати? – Habr [Електронний ресурс] – режим доступу: https://habr.com/ru/company/piter/blog/458096/
5. MongoDB – Official website [Електронний ресурс] – режим доступу: https://www.mongodb.com/
6. Express.js – Official website [Електронний ресурс] – режим доступу: https://expressjs.com/
7. Початок роботи з Express – METANIT.COM [Електронний ресурс] – режим доступу: https://metanit.com/web/nodejs/4.1.php
8. React.js – Official Website [Електронний ресурс] – режим доступу: https://reactjs.org/
9. Node.js – Official Website [Електронний ресурс] – режим доступу: https://nodejs.org/en/
10. Структура сайта: схемы, советы по разработке, примеры [Електронний ресурс] – режим доступу: https://bit.ly/2Yw3ONW

ДОДАТОК А

ОСНОВНІ ФАЙЛИ ПРОГРАМНОГО КОДУ

Backend-частина

Data.js

```
import bcrypt from 'bcryptjs';

const data = {
    users: [
        {
            name: 'Anton',
            email: 'anton.avhutskiy@gmail.com',
            password: bcrypt.hashSync('1234', 8),
            isAdmin: true
        },
        {
            name: 'Vova',
            email: 'vova@gmail.com',
            password: bcrypt.hashSync('12345', 8),
            isAdmin: false
        },


    ],
    products: [
        {
            name: 'Блок вызова БВД-313Т',
            category: 'Домофоны',
            image:
'https://alba.sumy.ua/upload/resize_cache/iblock/9f5/800_600_140cd750bba9870f18a
ada2478b24840a/9f59c2969bcf5bd85e127c16fb92a633.jpg',
            price: 800,
            brand: 'Дом',
            raiting: 4.5,
            instock: true,
            countInStock: 5,
            numReviews: 10,
            descr: 'Здесь будет короткое описание данного товара'
        },

        {
            name: 'Извищатель пожарный дымовой ИПК-8',
            category: 'Домофоны',
            image:
'https://alba.sumy.ua/upload/iblock/443/44397559a7f98fe892a978163903720f.jpg',
            price: 135,
            brand: 'Дом',
```

```
            raiting: 4.5,
            instock: true,
            countInStock: 5,
            numReviews: 10,
            descr: 'Здесь будет короткое описание данного товара'
        },

        {
            name: 'Автономный контроллер СКУД: Z-5R',
            category: 'Домофоны',
            image:
'https://alba.sumy.ua/upload/iblock/342/3426497bb9b5aa7eab67dbecba47ee0c.jpg',
            price: 1075,
            brand: 'Дом',
            raiting: 4.5,
            instock: true,
            countInStock: 5,
            numReviews: 10,
            descr: 'Здесь будет короткое описание данного товара'
        },

        {
            name: 'FAAC 414',
            category: 'Домофоны',
            image:
'https://alba.sumy.ua/upload/iblock/666/666e88317ddb81ee4c0211f02c2789ec.jpg',
            price: 1500,
            brand: 'Дом',
            raiting: 4.5,
            instock: true,
            countInStock: 4,
            numReviews: 10,
            descr: 'Здесь будет короткое описание данного товара'
        },

        {
            name: 'Огнетушитель порошковый ВП-9',
            category: 'Домофоны',
            image:
'https://alba.sumy.ua/upload/iblock/572/5723bef4484bbca658d0b6a11a09f4b9.jpg',
            price: 500,
            brand: 'Дом',
            raiting: 4.5,
            instock: true,
            countInStock: 2,
            numReviews: 10,
            descr: 'Здесь будет короткое описание данного товара'
        },

        {
            name: 'Электромеханический замок Atis Lock SS из нержавеющей стали',
```

```
            category: 'Домофоны',
            image:
'https://alba.sumy.ua/upload/resize_cache/iblock/f0b/310_310_140cd750bba9870f18a
ada2478b24840a/f0b45966b312d9aa9c02956661c44b1b.jpg',
            price: 350,
            brand: 'Дом',
            raiting: 4.5,
            instock: false,
            countInStock: 3,
            numReviews: 10,
            descr: 'Здесь будет короткое описание данного товара'
        },
    ],
};

export default data;
```

Server.js

```
import express from 'express';
import mongoose from 'mongoose';
import dotenv from 'dotenv';
import path from 'path';
import userRouter from './routers/user.js';
import productRouter from './routers/product.js';
import orderRouter from './routers/order.js';
import uploadRouter from './routers/upload.js';

dotenv.config();

const app = express();

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

mongoose.connect(process.env.MONGODB_URL || 'mongodb://localhost/avhuta-store',
{
    useNewUrlParser: true,
    useUnifiedTopology: true,
    useCreateIndex: true
})
const port = process.env.PORT || 5000;

app.use('/api/uploads', uploadRouter);
app.use('/api/users', userRouter);
app.use('/api/products', productRouter);
app.use('/api/orders', orderRouter);
app.get('/api/config/stripe', (req, res) => {
    res.send(process.env.STRIPE_SECRET_KEY || 'sb');
});
```

```
const __dirname = path.resolve();
app.use('/uploads', express.static(path.join(__dirname, '/uploads')));

app.get('/', (req, res) => {
    res.send('Server is running');
});

app.use((err, req, res, next) => {
    res.status(500).send({message: err.message});
})

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

Utils.js

```
import jwt from "jsonwebtoken"

export const generateToken = (user) => {
    return jwt.sign({
        _id: user._id,
        name: user.name,
        email: user.email,
        isAdmin: user.isAdmin
    }, process.env.JWT_SECRET || 'somethingsecret', {
        expiresIn: '30d',
    });
};

export const isAuth = (req, res, next) => {
    const authorization = req.headers.authorization;
    if (authorization) {
        const token = authorization.slice(7, authorization.length);
        jwt.verify(token, process.env.JWT_SECRET || 'somethingsecret', (err,
decode) => {
            if (err) {
                res.status(401).send({message: 'Неправильный токен'})
            } else {
                req.user = decode;
                next();
            }
        });
    } else {
        res.status(401).send({message: 'Нет токена'});
    }
};

export const isAdmin = (req, res, next) => {
    if (req.user && req.user.isAdmin) {
```

```
        next();
    } else {
        res.status(401).send({message: 'Неправильный токен админа'});
    }
};
```

Routers/order.js

```
import express from 'express';
import expressAsyncHandler from 'express-async-handler';
import Order from '../models/order.js';
import { isAuth, isAdmin } from '../utils.js';
import Stripe from 'stripe';
import dotenv from 'dotenv';

dotenv.config();

const stripe = new Stripe(process.env.STRIPE_SECRET_KEY);

const orderRouter = express.Router();

orderRouter.get('/', isAuth, isAdmin, expressAsyncHandler (async (req,res) => {
    const orders = await Order.find({}).populate('user', 'name');
    res.send(orders);
}));

orderRouter.get('/my', isAuth, expressAsyncHandler( async (req,res) => {
    const orders = await Order.find({user: req.user._id});
    res.send(orders);
}));

orderRouter.post('/', isAuth, expressAsyncHandler (async (req, res) => {
    if (req.body.orderItems.length === 0) {
        res.status(400).send({message: "Корзина пустая"});
    } else {
        const order = new Order({
            orderItems: req.body.orderItems,
            shippingAddress: req.body.shippingAddress,
            paymentMethod: req.body.paymentMethod,
            itemsPrice: req.body.itemsPrice,
            totalPrice: req.body.totalPrice,
            user: req.user._id
        });
        const createdOrder = await order.save();
        res.status(201).send({message: 'Новый заказ создан', order:
createdOrder});
    }
}));

orderRouter.get('/:id', isAuth, expressAsyncHandler (async (req, res) => {
```

```
    const order = await Order.findById(req.params.id);
    if (order) {
        res.send(order);
    } else {
        res.status(404).send({message: 'Заказ не найден'});
    }
}));

orderRouter.post('/create-checkout-session', async (req, res) => {
    const session = await stripe.checkout.sessions.create({
        payment_method_types: ['card'],
        mode: 'payment',
        success_url: `http://localhost:3000/order/${req.body.url}?success=true`,
        cancel_url: `http://localhost:3000/order/${req.body.url}`,
        line_items: [{
            amount: req.body.amount,
            currency: 'uah',
            name: 'Покупка',
            quantity: 1
        }]
    });
    res.json({
        id: session.id
    });
});


orderRouter.put('/:id/pay', isAuth, expressAsyncHandler( async (req, res) => {
    const order = await Order.findById(req.params.id);
    if (order) {
        order.isPaid = true;
        order.paidAt = Date.now();
        order.paymentResult = {
            id: req.body.id,
            status: req.body.status,
            update_time: req.body.update_time,
            email_address: req.body.email_address
        };
        const updatedOrder = await order.save();
        res.send({message: 'Заказ оплачен', order: updatedOrder});
    } else {
        res.status(404).send({message: 'Заказ не найден'});
    }
}));

orderRouter.delete('/:id', isAuth, isAdmin, expressAsyncHandler( async (req,res)
=> {
    const order = await Order.findById(req.params.id);
    if (order) {
        const deleteOrder = await order.remove();
        res.send({message: 'Заказ удален', order: deleteOrder});
```

```
    } else {
        res.status(404).send({message: 'Заказ не найден'});
    }
}));

export default orderRouter;
```

Routers/product.js

```
import express from 'express';
import expressAsyncHandler from 'express-async-handler';
import data from '../data.js';
import Product from '../models/product.js';
import { isAdmin, isAuth } from '../utils.js';

const productRouter = express.Router();

productRouter.get('/', expressAsyncHandler( async (req, res) => {
    const products = await Product.find({});
    res.send(products);
}));

productRouter.get('/seed', expressAsyncHandler( async(req, res) => {
    //await Product.remove({});
    const createdProducts = await Product.insertMany(data.products);
    res.send({createdProducts});
}));

productRouter.get('/:id', expressAsyncHandler( async (req, res) => {
    const product = await Product.findById(req.params.id);

    if (product) {
        res.send(product);
    } else {
        res.status(404).send({message: 'Продукт не найден'});
    }
}));

productRouter.post('/', isAuth, isAdmin, expressAsyncHandler (async (req, res)
=> {
    const product = new Product({
      name: 'sample name ' + Date.now(),
      image: 'https://www.roznica.com.ua/rf/th/400x320/69/804798.jpg',
      price: 0,
      category: 'sample category',
      brand: 'sample brand',
      countInStock: 0,
      instock: true,
      raiting: 3,
      numReviews: 0,
```

```
      descr: 'sample description',
    });
    const createdProduct = await product.save();
    res.send({message: 'Продукт создан', product: createdProduct});
}));

productRouter.put('/:id', isAuth, isAdmin, expressAsyncHandler (async (req, res)
=> {
    const productId = req.params.id;
    const product = await Product.findById(productId);
    if (product) {
        product.name = req.body.name;
        product.price = req.body.price;
        product.image = req.body.image;
        product.category = req.body.category;
        product.brand = req.body.brand;
        product.countInStock = req.body.countInStock;
        product.instock = req.body.instock;
        product.descr = req.body.descr;
        const updatedProduct = await product.save();

        res.send({message: 'Товар обновлен', product: updatedProduct});
    } else {
        res.status(404).send({message: "Товара не найдено"});
    }
}));

productRouter.delete('/:id', isAuth, isAdmin, expressAsyncHandler (async (req,
res) => {
    const product = await Product.findById(req.params.id);
    if (product) {
        const deleteProduct = await product.remove();
        res.send({message: 'Товар удален', product: deleteProduct});
    } else {
        res.status(404).send({message: 'Товар не найден'});
    }
}));

export default productRouter;
```

Routers/upload.js

```
import multer from 'multer';
import express from 'express';
import { isAuth } from '../utils.js';

const uploadRouter = express.Router();

const storage = multer.diskStorage({
    destination(req, file, cb) {
```

```
        cb(null, 'uploads/');
    },
    filename(req, file, cb) {
        cb(null, `${Date.now()}.jpg`);
        cb(null, `${Date.now()}.png`);
    },
});

const upload = multer({storage});

uploadRouter.post('/', isAuth, upload.single('image'), (req, res) => {
    res.send(`/${req.file.path}`);
});

export default uploadRouter;
```

Routers/user.js

```
import express from 'express';
import data from '../data.js';
import User from '../models/user.js';
import expressAsyncHandler from 'express-async-handler';
import bcrypt from 'bcryptjs';
import { generateToken, isAdmin, isAuth } from '../utils.js';

const userRouter = express.Router();

userRouter.get(
    '/seed',
    expressAsyncHandler( async (req, res) => {
    // await User.remove({});
    const createdUsers = await User.insertMany(data.users);
    res.send({createdUsers});
}));

userRouter.post('/signin', expressAsyncHandler (async(req, res) => {
    const user = await User.findOne({email: req.body.email});

    if(user) {
        if(bcrypt.compareSync(req.body.password, user.password)) {
            res.send({
                _id: user._id,
                name: user.name,
                email: user.email,
                isAdmin: user.isAdmin,
                token: generateToken(user)
            });
            return;
        }
    }
```

```
    res.status(401).send({message: 'Нерпавильно емейл или пароль'});
}));

userRouter.post('/register', expressAsyncHandler(async(req, res) => {
    const user = await User({
        name: req.body.name,
        email: req.body.email,
        phone: req.body.phone,
        password: bcrypt.hashSync(req.body.password, 8)});

    const createdUser = await user.save();
    res.send({
        _id: createdUser._id,
        name: createdUser.name,
        email: createdUser.email,
        phone: createdUser.phone,
        isAdmin: createdUser.isAdmin,
        token: generateToken(createdUser)
    });
}));

userRouter.get('/:id', expressAsyncHandler( async (req,res) => {
    const user = await User.findById(req.params.id);

    if (user) {
        res.send(user);
    } else {
        res.status(404).send({message: 'Пользователь не найден'})
    }
}));

userRouter.put('/profile', isAuth, expressAsyncHandler (async (req, res) => {
    const user = await User.findById(req.user._id);
    if (user){
        user.name = req.body.name || user.name;
        user.email = req.body.email || user.email;
        user.phone = req.body.phone || user.phone;
        if (req.body.password) {
            user.password = bcrypt.hashSync(req.body.password, 8);
        }
    const updatedUser = await user.save();
    res.send({
        _id: updatedUser._id,
        name: updatedUser.name,
        email: updatedUser.email,
        phone: updatedUser.phone,
        isAdmin: updatedUser.isAdmin,
        token: generateToken(updatedUser),
    });
    }
}));
```

```
userRouter.get('/', isAuth, isAdmin, expressAsyncHandler (async (req,res) => {
    const users = await User.find({});
    res.send(users);
}));

userRouter.delete('/:id', isAuth, isAdmin, expressAsyncHandler (async (req,res)
=> {
    const user = await User.findById(req.params.id);
    if (user) {
        if (user === 'admin@gmail.com') {
            res.status(400).send({message: 'Нельзя удалить админа'});
            return;
        }
        const deleteUser = await user.remove();
        res.send({message: 'Пользователь удален', user: deleteUser});
    } else {
        send.status(404).send({message: 'Пользователь не найден'});
    }
}));

userRouter.put('/:id', isAuth, isAdmin, expressAsyncHandler (async (req,res) =>
{
    const user = await User.findById(req.params.id);
    if (user) {
        user.name = req.body.name || user.name;
        user.email = req.body.email || user.email;
        user.phone = req.body.phone || user.phone;
        user.isAdmin = req.body.isAdmin || user.isAdmin;

        const updatedUser = await user.save();
        res.send({message: 'Пользователь обновлен', user: updatedUser});
    } else {
        res.status(404).send({message: 'Пользователь не найден'});
    }
}));

export default userRouter;
```

models/order.js

```
import mongoose from 'mongoose';

const orderSchema = new mongoose.Schema({
    orderItems: [{
        name: {type: String, required: true},
        qty: {type: Number, required: true},
        image: {type: String, required: true},
        price: {type: Number, required: true},
        product: {type: mongoose.Schema.Types.ObjectId,
        ref: 'Product', required: true}
    }],
```

```
    shippingAddress: {
        fullName: {type: String, required: true},
        country: {type: String, required: true},
        city: {type: String, required: true},
        address: {type: String, required: true},
        postalCode: {type: String, required: true},
    },
    paymentResult: {id: String, status: String, update_time: String,
email_address: String},
    paymentMethod: {type: String, required: true},
    itemsPrice: {type: Number, required: true},
    user: {type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true},
    isPaid: {type: Boolean, default: false},
    paidAt: {type: Date},
},
    {
        timestamps: true,
    }
);

const Order = mongoose.model('Order', orderSchema);

export default Order;
```

models/product.js

```
import mongoose from 'mongoose';

const productSchema = new mongoose.Schema(
    {
    name: {type: String, required: true, unique: false},
    image: {type: String, required: true},
    brand: {type: String, required: true},
    category: {type: String, required: true},
    descr: {type: String, required: true},
    price: {type: Number, required: true},
    instock: {type: Boolean, required: true},
    countInStock: {type: Number, required: true},
    raiting: {type: Number, required: true},
    numReviews: {type: Number, required: true}
}, {
    timestamps: true
    }
);

const Product = mongoose.model('Product', productSchema);

export default Product;
```

models/user.js

```
import mongoose from 'mongoose';

const userSchema = new mongoose.Schema({
    name: {type: String, required: true},
    email: {type: String, required: true, unique: true},
    phone: {type: Number, required: true},
    password: {type: String, required: true},
    isAdmin: {type: Boolean, default: false, required: true}
    },
    {
    timestamps: true
    }
);

const User = mongoose.model("User", userSchema);

export default User;
```

Frontend-частина

Index.js

```
import React from 'react';
import {Provider} from 'react-redux';
import ReactDOM from 'react-dom';
import App from './App';
import store from './store';

  ReactDOM.render(
    <Provider store={store}>
      <React.StrictMode>
        <App />
      </React.StrictMode>
    </Provider>
,
  document.getElementById('root')
);
```

App.js

```
import React from 'react';
import HomeScreen from './screens/HomeScreen';
import ProductScreen from './screens/ProductScreen';
import CartScreen from './screens/CartScreen';
import { signout } from './actions/user.js';
import { BrowserRouter, Link, Route } from 'react-router-dom';
```

```
import './app.sass';
import { useDispatch, useSelector } from 'react-redux';
import SigninScreen from './screens/SigninScreen';
import RegisterScreen from './screens/RegisterScreen';
import ShippingScreen from './screens/ShippingScreen';
import PaymentScreen from './screens/PaymentScreen';
import PlaceOrderScreen from './screens/PlaceOrderScreen';
import OrderScreen from './screens/OrderScreen';
import OrderHistoryScreen from './screens/OrderHistoryScreen';
import ProfileScreen from './screens/ProfileScreen';
import PrivateRouter from './components/AdminRouter';
import AdminRouter from './components/AdminRouter';
import ProductListScreen from './screens/ProductListScreen';
import ProductEditScreen from './screens/ProductEditScreen';
import OrderListScreen from './screens/OrderListScreen';
import UserListScreen from './screens/UserListScreen';
import UserEditScreen from './screens/UserEditScreen';

function App() {

  const cart = useSelector(state => state.cart);
  const { cartItems } = cart;
  const userSignin = useSelector((state) => state.userSignin);
  const { userInfo } = userSignin;
  const dispatch = useDispatch();
  const signoutDo = () => {
    dispatch(signout());
  }

  return (
    <BrowserRouter>
    <div className="main">
    <header className="header container">
      <div className="header__content">

        <div className="header__area-burger">
          <button type="button" className="button-burger">
            <span className="icon-burger"><i className="bi bi-list-
nested"></i></span>
            <span className="icon-text">Меню</span>
          </button>
        </div>

        <Link to="/"className="header__area-logo">
          <span className="header__logo">
              <img src="https://static-
sl.insales.ru/files/1/2933/14871413/original/Group_8.svg" alt="logo"></img>
          </span>
        </Link>
```

```
        <div className="header__area-controls">

          <div className="control-search">
            <form action="/search" method="get" className="header__search-form">
              <input type="text" autoComplete="off" className="form-control"
placeholder="Поиск"></input>
            </form>
            <button type="button" className="search-btn">
              <span className="icon-search _show"><i className="bi bi-
search"></i></span>
              <span className="icon-close _hide"><i className="bi bi-x-
circle(hide)"></i></span>
            </button>

            {
              userInfo ? (
                <div className="dropdown">
                  <Link to="/#" className="control-profile">
                  <span type="button" className="icon-user" >
                    <i className="bi bi-person-fill"></i>
                    <i className="bi bi-caret-down"></i>
                    </span>
                  </Link>
                  <ul className="dropdown-content">
                    <li>
                    <Link to="/profile">Профиль</Link>
                    </li>
                    <li>
                    <Link to="/orderhistory">История</Link>
                    </li>
                    <li>
                    <Link to="#signout" onClick={signoutDo}>Выйти</Link>
                    </li>
                  </ul>

                </div>

              ) :
              (
                <Link to="/signin" className="control-profile">
                <span className="icon-user"><i className="bi bi-
person"></i></span>
                </Link>
              )}

              {userInfo && userInfo.isAdmin && (
                <div className="dropdown">
                  <Link to="#admin">
                    <span className="icon-admin">
                    <i className="bi bi-person-circle"></i> {' '}
                    <i className="bi bi-caret-down caret-admin"></i>
```

```
                    </span>
                    <ul className="dropdown-content">
                      <li>
                        <Link to="/users">Пользователи</Link>
                      </li>
                      <li>
                        <Link to="/products">Продукты</Link>
                      </li>
                      <li>
                        <Link to="/table">Таблица</Link>
                      </li>
                      <li>
                        <Link to="/orderlist">Заказы</Link>
                      </li>
                    </ul>
                  </Link>
                </div>
              )}

            <Link to="/cart" className="control-cart">
              <span className="icon-cart">
                <i className="bi bi-cart3"></i>
                {cartItems.length > 0 && (
                  <span className="control-bage">{cartItems.length}</span>
                )}
                </span>
            </Link>
          </div>
        </div>
      </div>
    </header>

    <PrivateRouter path="/products"
component={ProductListScreen}></PrivateRouter>
    <PrivateRouter path="/profile" component={ProfileScreen}></PrivateRouter>
    <Route path="/orderhistory" component={OrderHistoryScreen}></Route>
    <Route path="/order/:id" component={OrderScreen}></Route>
    <Route path="/placeorder" component={PlaceOrderScreen}></Route>
    <Route path="/payment" component={PaymentScreen}></Route>
    <Route path="/shipping" component={ShippingScreen}></Route>
    <Route path="/register" component={RegisterScreen}></Route>
    <Route path="/signin" component={SigninScreen}></Route>
    <Route path="/cart/:id?" component={CartScreen}></Route>
    <Route path="/product/:id" component={ProductScreen} exact></Route>
    <AdminRouter path="/product/:id/edit" component={ProductEditScreen}
exact></AdminRouter>
    <AdminRouter path="/orderlist" component={OrderListScreen}></AdminRouter>
    <AdminRouter path="/users" component={UserListScreen}></AdminRouter>
    <AdminRouter path="/user/:id/edit" component={UserEditScreen}></AdminRouter>
    <Route path="/" component={HomeScreen} exact ></Route>
```

```
    <footer className="footer container">
      <div className="footer__spans">
        <span className="footer__spans-phone">+380 66 817 3846</span>
        <span className="footer__spans-city">г. Сумы</span>
      </div>
      <div className="footer__logo">
        <img className="footer__logo-img" src="https://static-
sl.insales.ru/files/1/2933/14871413/original/Group_8.svg" alt="logo"></img>
      </div>
      <div className="footer__cards">
        <img className="footer__cards-img1" src="https://static-
sl.insales.ru/files/1/1305/14550297/original/Visa.svg" alt="logo"></img>
        <img className="footer__cards-img2" src="https://static-
sl.insales.ru/files/1/1311/14550303/original/Group.svg" alt="logo"></img>
      </div>
  </footer>

</div>

</BrowserRouter>
  );
}

export default App;
```

Store.js

```
import { applyMiddleware, combineReducers, compose, createStore } from 'redux';
import thunk from 'redux-thunk';
import { cartReducer } from './reducers/cart';
import { orderCreateReducer, orderDeleteReducer, orderDetailsReducer,
orderListReducer, orderMyListReducer, orderPayReducer } from './reducers/order';
import {productCreateReducer, productDeleteReducer, productDetailsReducer,
productListReducer, productUpdateReducer} from './reducers/products';
import { userDeleteReducer, userDetailsReducer, userEditReducer,
userListReducer, userRegisterReducer, userSigninReducer,
userUpdateProfileReducer } from './reducers/user';

const initialState = {
    cart: {
        cartItems: localStorage.getItem('cartItems') ?
JSON.parse(localStorage.getItem('cartItems')) : [],

        shippingAddress: localStorage.getItem('shippingAddress') ?
JSON.parse(localStorage.getItem('shippingAddress')) : {},

        paymentMethod: 'PayPal'
    },
    userSignin: {
        userInfo: localStorage.getItem("userInfo") ?
JSON.parse(localStorage.getItem("userInfo")) : null
    }
```

```
};

const reducer = combineReducers({
    productList: productListReducer,
    productDetails: productDetailsReducer,
    cart: cartReducer,
    userSignin: userSigninReducer,
    userRegister: userRegisterReducer,
    userList: userListReducer,
    userDelete: userDeleteReducer,
    userEdit: userEditReducer,
    orderCreate: orderCreateReducer,
    orderDetails: orderDetailsReducer,
    orderPay: orderPayReducer,
    orderMyList: orderMyListReducer,
    orderList: orderListReducer,
    orderDelete: orderDeleteReducer,
    userDetails: userDetailsReducer,
    userUpdateProfile: userUpdateProfileReducer,
    productCreate: productCreateReducer,
    productUpdate: productUpdateReducer,
    productDelete: productDeleteReducer
});

const composeEnhancer = window.__REDUX_DEVTOOLS_EXTENSION_COMPOSE__ || compose;

const store = createStore(
    reducer,
    initialState,
    composeEnhancer(applyMiddleware(thunk)));

export default store;
```

Переглянути повний код можна на репозиторії за посиланням: https://github.com/muckduchok/avhuta-store/