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

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

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Розробка веб-застосунку для онлайн продаж одягу із використанням Node.js 14 та React 17

Виконав: студент IV курсу, групи СН-41
спеціальності 122 Комп’ютерні науки
(шифр і назва спеціальності)

_____ Тененський М. В.
(підпис) (прізвище та ініціали)

Керівник _____ Никитюк В. В.
(підпис) (прізвище та ініціали)

Нормоконтроль _____ Шимчук Г. В.
(підпис) (прізвище та ініціали)

Завідувач кафедри _____ Боднарчук І.О.
(підпис) (прізвище та ініціали)

Рецензент _____ Золотий Р.З.
(підпис) (прізвище та ініціали)

Тернопіль
2022

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

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра комп'ютерних наук
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Боднарчук І.О.
(підпис) (прізвище та ініціали)
«__» _____ 2022 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Бакалавр
(назва освітнього ступеня)

за спеціальністю 122 Комп'ютерні науки
(шифр і назва спеціальності)

Студенту Тененському Максиму Васильовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка веб-застосунку для онлайн продаж одягу із використанням Node.js 14 та React 17

Керівник роботи Никитюк Вячеслава Вячеславович, к.т.н., доцент кафедри КН
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від « 16 » березня 2022 року № 4/7-161

2. Термін подання студентом завершеної роботи 21 червня 2022р.

3. Вихідні дані до роботи Літературні та інтернет джерела інформації про технології розробки веб-застосунків для онлайн продаж одягу із використанням Node.js 14 та React 17.

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ. Розділ 1. Постановка задачі та формування вимог до веб-застосунку. Розділ 2. Реалізація та тестування веб-застосунку для онлайн продаж одягу. Розділ 3. Безпека життєдіяльності, основи охорони праці. Висновки. Перелік джерел. Додатки.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
1. Титульна сторінка. 2. Мета, об'єкт та предмет дослідження. 3. Актуальність обраної теми. 4. Аналіз предметної області. 5. Актори веб-застосунку. 6. Огляд використовуваних технологій.7.Структура веб-застосунку. 8. Колекції в базі даних MongoDB. 9. Зв'язки колекцій в БД. 10. Створення моделей. 11. Бібліотеки MobX та React. 12. Axios запити та Styled Components. 13. Вигляд користувацького інтерфейсу. 14.Висновки.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці	Гурик О. Я., доцент кафедри МТ		

7. Дата видачі завдання 24 січня 2022 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи	24.01.2022	*Виконано*
2.	Підбір джерел про розробку сучасних веб-застосунків	04.01.2022-30.01.2022	*Виконано*
	для онлайн продаж одягу із використанням Node.js 14		
	та React 17		
3.	Переклад та опрацювання джерел про розробку	31.01.2022-06.02.2022	*Виконано*
	сучасних веб-застосунків для онлайн продаж одягу із		
	використанням Node.js 14 та React 17		
4.	Виконання дослідження щодо веб-застосунків для	07.02.2022-13.02.2022	*Виконано*
	онлайн продаж одягу. Власне розроблення веб-		
	застосунку із використанням Node.js 14 та React 17		
5.	Оформлення розділу «Постановка задачі та формува-	14.02.2022-06.03.2022	*Виконано*
	ння вимог до веб-застосунку»		
6.	Оформлення розділу «Реалізація та тестування веб-	07.03.2022-03.04.2022	*Виконано*
	Застосунку для онлайн продаж одягу»		
7.	Виконання завдання до підрозділу «Безпека життєдіяльності»	04.04.2022-17.04.2022	*Виконано*
8.	Виконання завдання до підрозділу «Основи охорони	18.04.2022-01.05.2022	*Виконано*
	праці»		
9.	Оформлення кваліфікаційної роботи	02.05.2022-15.05.2022	*Виконано*
10.	Нормоконтроль	16.05.2022-22.05.2022	*Виконано*
11.	Перевірка на плагіат	25.05.2022	*Виконано*
12.	Попередній захист кваліфікаційної роботи	07.06.2022	*Виконано*
13.	Захист кваліфікаційної роботи	21.06.2022	

Студент _____ Тененський М. В.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Никитюк В. В.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Розробка веб-застосунку для онлайн продаж одягу із використанням Node.js 14 та React 17 // Кваліфікаційна робота освітнього рівня «Бакалавр» // Тененський Максим Васильович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра комп'ютерних наук, група СН-41 // Тернопіль, 2022 // С. – 54 , рис. – 32, табл. – 3, кресл. – 0, додат. – 4, бібліогр. – 30.

Ключові слова*:* бази даних, веб-застосунок, компоненти, програмування, MongoDB, Node.js, React, Express.

Кваліфікаційна робота присвячена розробці сучасного та зручного у користуванні веб-застосунку для онлайн продаж одягу із використанням серверної платформи Node.js та бібліотеки React.

Мета роботи полягає в збільшенні клієнтської бази та популяризації продаж в мережі інтернет серед її користувачів за допомогою розроблюваного веб-застосунку.

В *першому розділі* кваліфікаційної роботи проведено докладний аналіз обраної предметної області, а також сформовано перелік вимог до розроблюваного веб-застосунку для онлайн продаж одягу, розроблено діаграми використань, аргументовано вибір середовища та технологій розробки.

В *другому розділі* кваліфікаційної роботи описано проектування БД для веб-застосунку, проведено моделювання робочої архітектури, описано розробку як серверної, так і клієнтської частин застосунку, вказано основні особливості даного процесу, проведено тестування розробленого функціоналу.

Об'єктом дослідження є сучасний веб-застосунок для онлайн продаж одягу та нереляційна БД MongoDB.

Предметом дослідження є засоби і методи розробки веб-застосунків для онлайн продаж одягу із використанням Node.js 14 та React 17.

ANNOTATION

Development of a web application for online clothing sales using Node.js 14 and React 17 // Qualification work of the education level “Bachelor” // Tenenskyi Maksym // Ternopil Ivan Pului National Technical University, Faculty of Computer Information Systems and Software Engineering, Computer Science Department, group CH-41 // Ternopil, 2022 // P. – 54, pic. – 32, tables – 3, drawings – 0, add. – 4, ref. – 30.

Keywords: data base, web application, components, programming, MongoDB, Node.js, React, Express.

Qualifying work is devoted to the development of a modern and easy in use web application for online clothing sales using server platform Node.js and library React.

The purpose of the work is to increase the customer`s base and promote online sales among internet users using developed web application.

In the first section of the qualification work a detailed analysis of the selected subject area was conducted, as well as a list of requirements for the development web application for online clothing sales, developed use case diagrams, argued the choice of development environment and used technologies.

The second section of the qualification works describes the design of the database of web application, modeling the working architecture, development of both server and client parts of the application, indicates the main specifics of this process. Also, in this section the developed functionality was tested.

Objects of research are modern web application for online clothing sales and non-relational data base MongoDB.

Subjects of research are tools and methods of developing web applications for online clothing sales using Node.js 14 and React 17.

ПЕРЕЛІК СКОРОЧЕНЬ І ТЕРМІНІВ

СУБД – Система управління базою даних.

БД – База даних.

ІТ – Інформаційні технології.

ПЗ – програмне забезпечення.

ТЗ – технічне завдання.

Фреймворк – ПЗ, яке полегшує розробку шляхом поєднання в собі різних функціональних можливостей.

React – відкрита JS-бібліотека, призначена для розробки користувацьких інтерфейсів.

HTML – (англ. Hypertext Markup Language) – мова розмітки для створення веб-сайтів і веб-застосунків [1].

CSS (англ. Cascading Style Sheets) – спеціальна мова стилів, що використовується для опису зовнішнього вигляду сторінок [2].

JS – скриптова мова програмування JavaScript.

SQL (англ. Structed Query Language) – декларативна мова програмування для реляційних БД.

HTTP (англ. HyperText Transfer Protocol) – протокол прикладного рівня передачі даних.

CRUD (англ. Create read update delete) – основні операції управління даними в БД.

Axios – JS-бібліотека для виконання HTTP запитів в Node.js.

API (англ. Application Programming Interface) – певний набір структур, класів чи процедур, що використовуються для взаємодії одного компоненту програми з іншими.

ЗМІСТ

ВСТУП

Актуальність теми. Внаслідок того, що в наш час відбувається стрімкий розвиток IT, все більше найрізноманітніших галузей людської діяльності, так чи інакше, переплітається із використанням тієї чи іншої бази даних та веб-застосунків. Окрім того, через епідеміологічну ситуацію в світі, а саме поширення вірусу COVID-19 та впровадження карантинних заходів, кількість онлайн покупок щодня невпинно продовжує свій ріст, а традиційний процес придбання нових речей одягу в магазинах потроху відходить у минуле. Тому, спираючись на вищеописані аргументи, можна стверджувати, що розробка веб-застосунку для онлайн продаж із використанням серверної платформи Node.js 14 та бібліотеки React 17 є досить актуальним напрямом дослідження.

Мета і задачі дослідження. Метою цієї кваліфікаційної роботи освітнього рівня «Бакалавр» є суттєве збільшення клієнтської бази та популяризації продаж одягу в мережі інтернет серед її користувачів за допомогою розроблюваного веб-застосунку із використанням Node.js 14 та React 17. Для успішного досягнення поставлених задач необхідно:

– Проаналізувати обрану предметну область.

– Розробити моделі для зберігання даних та спроектувати БД.

– Провести проектування робочої архітектури веб-застосунку.

– Розробити веб-застосунок відповідно до поставлених вимог.

– Провести повне тестування усіх функціональних можливостей веб-застосунку.

Практичне значення одержаних результатів. Розроблений в ході виконання поставлених задач веб-застосунок для онлайн продаж одягу впровадить ефективну взаємодію із клієнтами та буде зручним в роботі. Також він буде забезпечувати супровід усіх процесів купівлі людиною різноманітних товарів в мережі інтернет, що, в свою чергу, збільшить клієнтську базу та популяризує онлайн торгівлю.

РОЗДІЛ 1. ПОСТАНОВКА ЗАДАЧІ ТА ФОРМУВАННЯ ВИМОГ ДО ВЕБ-ЗАСТОСУНКУ

1.1 Аналіз предметної області

Одним із завдань цієї кваліфікаційної роботи є розробка сучасного веб-застосунку для онлайн продаж одягу. Фактично це повинен бути інтернет-магазин, який забезпечить ефективний підхід в процесі здійснення онлайн продаж. Такі веб-застосунки дозволяють їх користувачам здійснювати вибір та замовлення потрібного одягу, техніки, предметів вжитку тощо, а також дозволяють провести автоматизацію управління бізнесом.

Важливим є як забезпечення оновлення наявного асортименту одягу, так і впровадження типового функціоналу інтернет-магазину: реєстрація, додавання товарів в кошик, здійснення покупки через платіжні системи, перегляд каталогу товарів, збереження інформації про здійснені покупки та вподобаний товар тощо. Усе це не буде можливим без використання бази даних, HTTP запитів та клієнтських скриптів.

До значних переваг веб-застосунків даного типу можна беззаперечно віднести як порівняно не високу вартість речей, так і, зазвичай, широкий асортимент товару, а також надзвичайну доступність. Пов’язано це із тим, що скористатись такими веб-застосунками можна з будь-якого пристрою підключеного до мережі інтернет. Ось чому клієнтська база онлайн магазинів дуже валика, відповідно до чого останні намагають конкурувати шляхом зниження цін та підвищення якості свого асортименту.

Також, порівнюючи веб-застосунками для онлайн продаж із звичайними крамницями, можна виділити ряд типових недоліків. До них відносяться неможливість об’єктивно оцінити реальний зовнішній вигляд та якість матеріалу товару, а також наявна необхідність очікувати доставку замовлення певний проміжок часу чи можливі загрози. Останнє стосується того випадку, коли користувач надає дані своєї банківської картки для оплати замовлення в непопулярних чи ненадійних інтернет магазинах тощо.

1.2 Формування вимог до веб-застосунку для онлайн продаж одягу

Для коректної розробки та функціонування будь-якого веб-застосунку варто завжди робити докладний опис основний його функціональних можливостей, відповідно до яких і буде сформовано ТЗ. Зазвичай, в ТЗ описують чотири комплексні вимоги [2]:

– Загальні архітектурні вимоги до застосунку.

– Вимоги до функціоналу панелі адміністратора.

– Вимоги до функціональних можливостей застосунку.

– Вимоги до використовуваного інтерфейсу на клієнтській стороні.

Спершу розглянемо загальні архітектурні вимоги [2]:

– Веб-застосунок повинен працювати в усіх, без винятків, сучасних браузерах та використовувати HTTP протокол для надання доступу;

– В якості бази даних повинна використовуватись нереляційна документо-орієнтована БД – MongoDB;

– Серверна частина веб-застосунку повинна бути розроблена на базі програмної платформи Node.js;

– Клієнтська сторона повинна бути розроблена із використанням бібліотеки React;

– Усі програмні файли, пов’язані із панеллю адміністратора, повинні зберігатись в окремому від клієнтської частити каталозі веб-серверу.

Перелік вимог до функціональних можливостей веб-застосунку для онлайн продаж одягу виглядає наступним чином [3]:

– Обов’язкова наявність можливості фільтрувати список одягу за кольором та розміром.

– Необхідно реалізувати функціонал, який дозволить переглядати кожну сторінку товару окремо.

– Додавання товару в корзину. Автоматичне визначення загальної суми покупки в корзині з урахуванням можливої знижки.

– Можливість змінювати кількість товару до покупки в корзині.

– Повинен бути функціонал додавання одягу в список вподобань.

– Система має розмежовувати функціонал для не- та зареєстрованих користувачів.

Розглянемо перелік вимог до панелі адміністратора [3]:

– Адміністратор має мати змогу додавати інформацію про новий товар в БД та видаляти вже існуючий запис.

– Необхідно впровадити функціонал редагування інформації про вже існуючі товари.

– Варто дозволити адміністратору надавати іншим довіреним користувачам адмін-права.

– Необхідно, використовуючи різноманітні діаграми, виводити інформацію про місячний дохід, кількість нових користувачів магазину тощо.

Вимоги до використовуваного інтерфейсу на клієнтській стороні [3]:

– Користування функціоналом веб-застосунку має бути зручним.

– Кольорова схема інтерфейсу не повинна містити занадто яскраві кольори.

Відповідно до наведених вище вимог розуміємо, що розроблюваний веб-застосунок для онлайн продаж одягу повинен відповідати найсучаснішим патернам розробки та проектування, покривати широкий спектр типового для таких типів застосунків функціоналу та мати можливість до подальшого розширення і масштабування.

1.3 Пошук актантів та варіантів використання

Суть пошуку актантів та варіантів використань полягає в зображенні усіх можливих шляхів застосування веб-застосунку. Доцільно буде виділити три актори для розроблюваного веб-застосунку для онлайн продаж одягу (див. рис. 1.1):

– Незареєстрований користувач.

– Зареєстрований користувач.

– Адміністратор.

Рисунок 1.1 – Актори розроблюваного веб-застосунку

Спираючись на список вимог до веб-застосунку, виділимо основні варіанти використання та наведемо їх детальний опис в таблицях 1.1-1.3.

Таблиця 1.1 – Варіанти використання для актору «Незареєстрований користувач»

Актор	**Найменування ВВ**	**Формулювання**
Незареєстрований користувач	Перегляд списку одягу. Фільтрація списку	Користувач може переглядати увесь наявний список товарів, фільтрувати його за різними параметрами.
	Заповнення та надсилання реєстраційної форми	Дозволяє користувачу зареєструватись в системі та отримати відповідне сповіщення на пошту
	Перегляд сторінки товару	Користувач має змогу перейти на сторінку із детальним описом одягу, що зацікавив його

Як видно із таблиці 1.1, незареєстрований користувач має досить вузький спектр можливостей, але це і не дивно, адже увесь основний функціонал потребує авторизації. Таким чином, варіанти використання для актора «Зареєстрований користувач» (див. табл. 1.2) розроблено на основі можливостей актора «Незареєстрований користувач» із додаванням нових [5].

Таблиця 1.2 – Додаткові варіанти використання для актору «Зареєстрований користувач»

Актор	Найменування ВВ	Формулювання
Зареєстрований користувач	Відновлення і редагування особистих даних	Дозволяє користувачу змінювати своє ім’я, відновлювати пароль тощо.
	Корзина	Зареєстрований користувач має змогу додавати різноманітні товари в корзину, змінювати їх кількість в корзині.
	Список вподобань	Надає можливість користувачу зберігати товар, який сподобався, в так званому «списку вподобань»
	Авторизація	Впровадження системи перевірки існування користувача в БД та його прав

Розглянемо варіанти використання для актора «Адміністратор», поданих в таблиці 1.3. Варто зазначити, що адміністратором може бути тільки зареєстрований в системі користувач, якому було надано відповідні права через адміністративну панель або інтерфейс взаємодії із БД MongoDB.

Таблиця 1.3 – Варіанти використання для актору «Адміністратор»

Актор	Найменування ВВ	Формулювання
Адміністратор	Авторизація	Перевірка наявності адмін-прав перед входом в адміністративну панель
	Статистика	Перегляд графіків із даними продаж, кількості нових користувачів тощо.
	Управління даними	Надає можливість адміністратору виконувати типові CRUD-операції над об’єктами даних в БД веб-застосунку

В загальному розроблюваний веб-застосунок для онлайн продаж одягу має слідувати правилам, описаним в таблицях 1.1-1.3. На основі інформації, поданої в вищеописаних таблицях, доцільно розробити діаграми варіантів використань для усіх визначених акторів веб-застосунку для онлайн продаж одягу із використанням Node.js 14 та React 17: «Незареєстрований користувач» (див. рис. 1.2), «Зареєстрований користувач» (див. рис. 1.3) та «Адміністратор» (див. рис. 1.4).

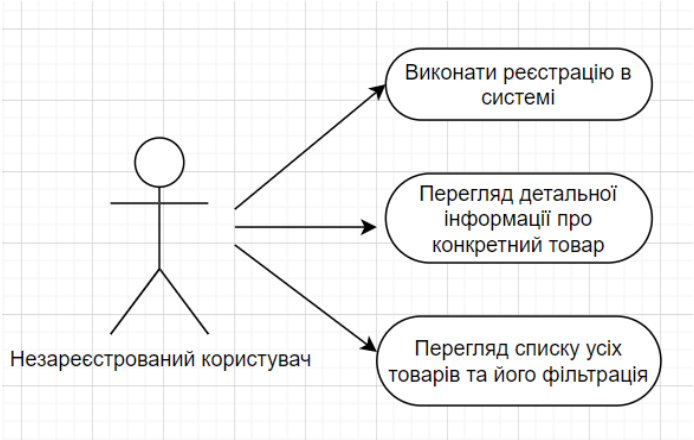


Рисунок 1.2 – Діаграма варіантів використань для актора «Незареєстрований користувач»

Як вже було сказано, незареєстрований користувач має доступ до значно меншої кількості функціональних можливостей в веб-застосунку, ніж інші два актори, діаграми використань яких подано на рисунках 1.3 та 1.4. Наприклад, такий користувач не зможе здійснити покупку або додати річ, що йому сподобалась, в список вподобань.

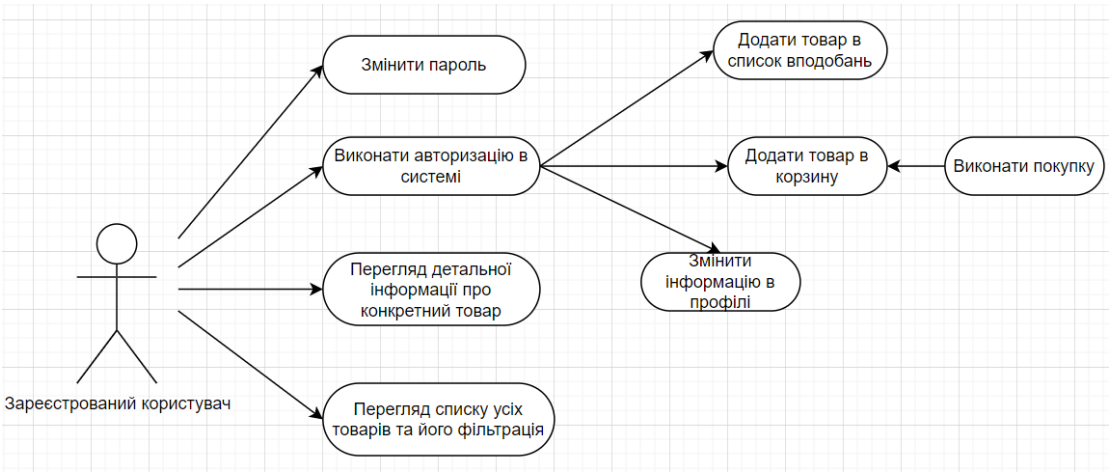


Рисунок 1.3 – Діаграма варіантів використань для актора «Зареєстрований користувач»

Як видно із дігаграми варіантів використань актору «Зареєстрований користувач», йому доступний широкий спекстр функціональних можливостей, пов’язаних із покупкою одягу, зміною особистої інформації та здійснення платежів.

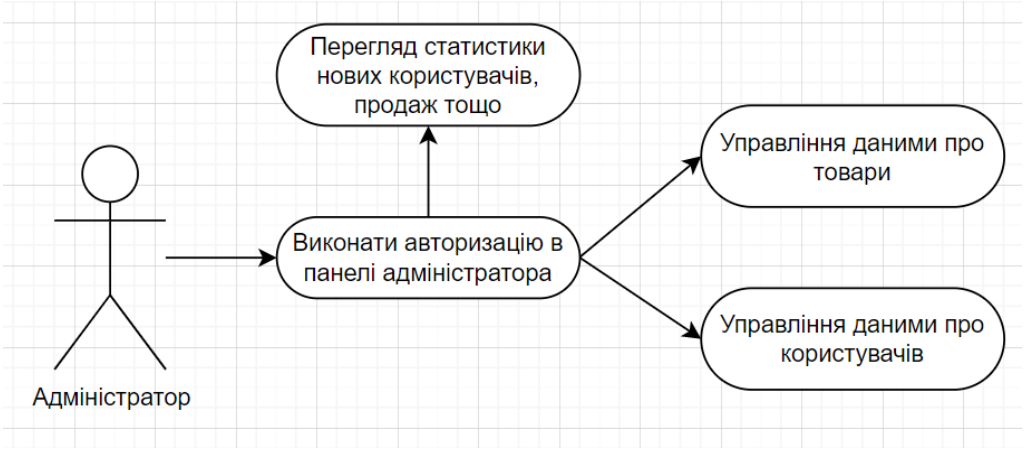


Рисунок 1.4 – Діаграма варіантів використань для актора «Адміністратор»

Відповідно до наведеної вище діаграми, адміністратору доступний типовий управлінський функціонал – перегляд товарів та усіх зареєстрованих користувачів, виконання над цими даними CRUD-операцій, а також додавання нового адміністратора тощо.

1.4 Вибір середовища розробки

Для розробки веб-застосунку для онлайн продаж із використанням серверної платформи Node.js 14 було застосовано наступне: мову розмітки HTML 5 в поєднанні з бібліотекою React 17, мову каскадних стилів CSS 3, скриптову мову програмування JS, власне платформу для серверної частини - Node.js із фреймворком Express [6]. Таким чином можна добитись відносно швидко розробки та розгортання, а в подальшому і масштабування повнофункціонального веб-застосунку. Окрім того, описаний стек технологій дозволяє використовувати велику кількість додаткових бібліотек, які впроваджують нетиповий функціонал, що значно покращить досвід використання даного веб-застосунку кінцевими користувачами

Відповідно до вищепереліченої сукупності засобів, було обрано два середовища розробки. Перший текстовий редактор носить назву Visual Studio Code. Він, на відміну від аналогів, впроваджує ефективну підсвітку синтаксису, підтримує швидкий рефакторинг та відладку коду, надає зручний Git інструментарій тощо.

Його нативні можливості можна розширити надзвичайно великою кількістю додаткових плагінів, які привносять свій унікальний та необхідний для швидкої, зручної та ефективної розробки функціонал і фактично перетворюють Visual Studi Code на повноцінну IDE. Проте дану особливість можна зарахувати і до недоліків, адже більшість із пакетів розширення не є офіційними та можуть працювати не коректно.

Окрім Visual Studio Code, доцільно використати і Web Storm. Ця IDE має необхідний для розробника будь-якого веб-застосунку інструмент – “дебаг”. За допомогою нього можна швидко відловлювати помилки, тестувати написаний код, змінюючи робочі значення змінних тощо. Так, аналогічні середовища розробки або не мають схожого функціоналу, або він значно обмежений у використання. Окрім того, Web Storm має унікальну особливість – він здатен пропонувати розробнику рішення щодо оптимізації та покращення вже написаного коду.

Також в даній IDE присутні усі вищезгадані переваги Visual Studio Code. Відмінним є те, що більшість із них доступні до використання без інсталяції додаткових плагінів, а, наприклад, робота із Git організована значно краще Так, у Web Stormі нема необхідності вручну прописувати команди для керування історією версій. Окрім того, наявний інструмент індексації файлів забезпечує значно швидкий пошук по коду, ніж в аналогічних IDE. Проте бувають певні недоліки, коли на малопотужних пристроях через індексацію середовище може працювати досить повільно, адже вона вимагає значних ресурсів оперативної пам’яті [7].

1.5 Обґрунтування використовуваного стеку технологій для розробки веб-застосунку для онлайн продаж одягу

Для розробки веб-застосунку відповідно до поставлених вимог, окрім типових мов програмування, було використано стек технологій MERN – база даних MongoDB, серверний фреймворк Express, бібліотека для клієнтської сторони React та серверна платформа Node.js [8].

Для початку визначимо, чому було обрано саме нереляційну базу даних *MongoDB*. В цій БД нема звиклих реляційних таблиць, запитів SQL та багатьох інших, притаманним реляційним базам даних речей. MongoDB забезпечує високу швидкість запису та читання даних (за умови правильного індексування), відповідно до чого горизонтальне масштабування такої бази даних не викликатиме жодних проблем. Завдяки тому, що використовується документо-орієнтована модель зберігання даних, кодування і групування даних відбувається на порядок легше, ніж в реляційних СУБД. А завдяки присутній логіці доступу типу «ключ-значення», паралельно із високою продуктивністю, забезпечується підтримка великої кількості функціональних можливостей.

Окрім того, MongoDB, як і інші NoSQL бази даних, не потребує попередньо визначених схем і здатна зберігати будь-які типи даних в одному документі, що, в свою чергу, дає можливість створювати будь-яку кількість полів у документі, тим самим полегшуючи в майбутньому масштабування та розробку веб-застосунку для онлайн продаж одягу. Також СУБД MongoDB може представляти не лише одну базу даних на одному сервері, але й декілька БД на різних фізичних серверах одночасно. Вони можуть обмінюватись між собою даними, зберігаючи при цьому цілісність інформації [9].

Прототипно-орієнтовану мову програмування *JavaScript* було обрано через її легкість використання та динамічну типізацію. Окрім того, вона часто використовуються в небраузерних середовищах в поєднанні із фреймворком Electron. Також варто виділити те, що *JavaScript* має часткову підтримку функціональної та імперативної парадигм, що робить її досить потужним інструментом програмування.

Для роботи із базою даних MongoDB доцільно використовувати офіційну JS бібліотеку *Mongoose*. Завдяки ній можна наперед визначати схеми даних із строгою типізацією, проводити їх тонке налаштування. Власне на основі даних схеми і створюються моделі, які в подальшому синхронізуються із документами, що зберігаються в БД MongoDB за допомогою сконфігурованих раніше зв’язків між ними. Також розробник, за допомогою *Mongoose,* отримує доступ до найрізноманітніших функцій на кшталт перевірки даних, їх збереження, читання,

видалення, оновлення тощо [9]. Окрім того, дана бібліотека дозволяє створювати агрегатні запити до БД, що, при коректному застосуванні, значно оптимізує роботу серверної частини веб-застосунку.

Відповідно до зазначених вимог, найкращим вибором в якості серверної платформ є *Node.js*. Вона ґрунтується на виконанні власне *JavaScript* коду та впровадженні неблокуючої моделі вводу-виводу подій. Ось чому в основі *Node.js* лежить поєднання асинхронного підходу із подієвим, що, в свою чергу, не тільки значно спрощує процес розробки і подальшого масштабування веб-застосунків, але й дозволяє їм адекватно реагувати на дії користувачів. Ще однією її перевагою є відкритість, із чого і випливає існування широкого вибору багатофункціональних бібліотек і готових програмних модулів.

Невід'ємною частиною використання серверної платформи *Node.js* є застосування мінімалістичного та гнучкого фреймворку під назвою *Express*. Він надає розробникам широкий вибір технологій та функцій, зокрема, серед найбільш вживаного, виділяють службові методи HTTP запитів та їх обробників [6]. Окрім того, на відміну від аналогів, *Express* не заважає застосовувати нативні можливості *Node.js*, а лише ефективно доповнює та розширює їх новим функціоналом.

Для розробки клієнтської частини веб-застосунку для онлайн продаж одягу було обрану відому бібліотеку із відкрити кодом – *React*. Вона досить популярна серед розробників та широко використовуються для створення користувацьких інтерфейсів. Так, за допомогою даної бібліотеки можна не тільки полегшити процес створення, але й без зайвих складнощів реалізувати та підтримувати високу швидкість роботи клієнтської сторони веб-застосунків [10]. Варто відмінити, що в даній кваліфікаційній роботі *React* використовувався в поєднанні із іншою бібліотекою MobX, призначення якої полягає в керуванні глобальним станом веб-застосунків та впровадження ряду інструментів для спрощення передачі даних через контекст компонентів тощо.

Для стилізації React компонентів варто використати формальну мову стилів CSS 3. Також вона застосовується для опису зовнішнього вигляду сторінок сайтів та веб-застосунків [11]. Зазвичай, CSS 3 використовується

розробниками у поєднанні із так званими препроцесорами на кшталт Sass або SCSS або бібліотеками розширеної стилізації, які привносять нові можливості до застосування мови стилів, не змінюючи при цьому початковий функціонал.

Вищезгадана бібліотека *MobX*, в поєднанні із *React*, відповідатиме за управління глобальним станом розроблюваного веб-застосунку для онлайн продаж одягу. Так, завдяки її використанню, можна добитись постійної відповідності глобального внутрішнього стану застосунку із відображенням інформації на клієнтському інтерфейсі. Окрім того, на відміну від прямого аналогу – бібліотеки *React Redux, MobX* не тільки легша у використанні, але й дозволяє проводити розробку веб-застосунків значно швидше, забезпечуючи при цьому потенціал до подальшого їх масштабування.

Окрім власноруч створених *React* компонентів, існує можливість використовувати вже готові із сучасного та прогресивного front-end фреймворк *Material* UI. В своїй логіці він спирається на так званий *Material Design* підхід, в основі якого лежить зручний для користувача стиль графічного дизайну програмного забезпечення та інтерфейсів веб-застосунків, розробленого корпорацією Google. Відповідно до цього, *Material UI* в, певній мірі, можна назвати більш кращим аналогом до не менш відомого фреймворку *Bootstrap*.

1.6 Формування структури веб-застосунку для онлайн продаж одягу

Відповідно до проведеного аналізу предметної області та описаним варіантам використання можна зробити висновок, що для розробки даного веб-застосунку найкраще підійде структура на базі трирівневої архітектури, поданої на рисунку 1.5. Умовно він складається із трьох рівнів: рівень клієнта (frontend), структурно-логічний рівень (backend) та рівень даних (база даних) [12].

Рівень клієнта містить сторінки із React компонентами, MobX кодом для управління глобальним станом застосунку і JS кодом для оброки подій, що відбувається на клієнтському інтерфейсі шляхом взаємодії користувача веб-застосунку для онлайн продаж одягу із ним.

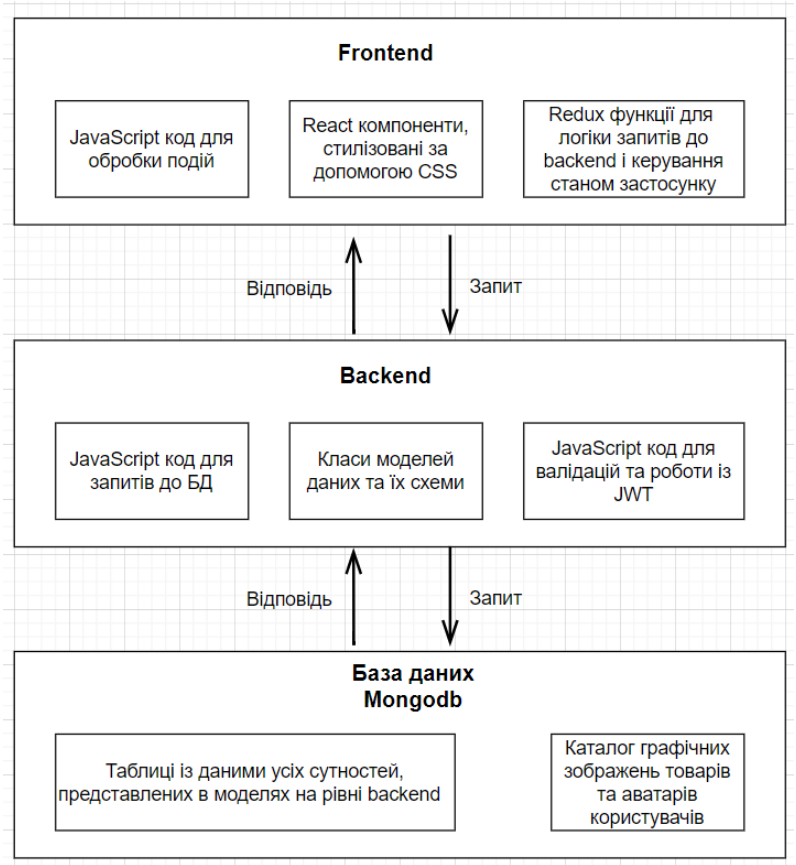


Рисунок 1.5 – Структура веб-застосунку для онлайн продаж одягу

Структурно-логічний рівень представлений основним робочим ядром – це більшість функціональних засобів, службові модулі, бібліотеки шифрування даних, надсилання електронних листів і запитів до MongoDB тощо [12]. Також на цьому рівні присутні скрипти та код, які безпосередньо беруть участь у взаємодії серверу із клієнтом: надсилання запитів, отримання відповідей і так далі.

Рівень даних представлений відомою нереляційною БД MongoDB, яка зберігає дані колекцій веб-застосунку та впроваджує типовий функціонал, що дозволяє ефективно та зручно управляти ними [14].

1.7 Висновок до першого розділу

В першому розділі даної кваліфікаційної роботи освітнього рівня «Бакалавр» було проаналізовано обрану предметну область та аргументовано її актуальність, а також сформовано перелік вимог до веб-застосунку для онлайн

продаж із використанням серверної платформи Node.js 14 та бібліотеки для розробки клієнтської сторони React 17. Так, застосунок має забезпечувати типовий функціонал на кшталт розмежування функціоналу для різних актантів, можливість фільтрувати товари, додавати їх в корзину чи список вподобань, здійснювати оплату тощо [14].

Виконавши пошук актантів розуміємо, що їх буде три: зареєстрований, незареєстрований користувачі та адміністратор. До кожного із них сформовано повноцінні діаграми варіантів використань. Відповідно до вищеописаної інформації, було обрано та аргументовано середовища такі середовища розробки, як Web Storm та Visual Studio Code, а також описано основні технології для них. Окрім того зображено структуру веб-застосунку для онлайн продаж одягу на базі трирівневої архітектури та детально описано кожний її рівень, що дозволяє в повній мірі сконцентруватись на розробці.

РОЗДІЛ 2. РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ВЕБ-ЗАСТОСУНКУ ДЛЯ ОНЛАЙН ПРОДАЖ ОДЯГУ

2.1 Проектування та створення БД для розроблюваного веб-застосунку

Як вже було в першому розділі даної кваліфікаційної роботи, в розроблюваному веб-застосунку для онлайн продаж одягу із використанням Node.js 14 та React 17 буде одна із найвідоміших нереляційних баз даних – MongoDB. Застосовуватись вона буде на базі глобального хмарного сервісу Atlas. Перевагою його використання є те, що даний сервіс дозволяє зробити більшість важливих налаштувань БД в автоматичному режимі, що, в свою чергу, дозволяє більше сфокусуватись на розробці вибраного веб-застосунку. Також Atlas дозволяє розвернути СУБД на таких сервісах, як AWS, GCP, AZURE тощо [9]. Структуру кластера MongoDB подано на рисунку 2.1.

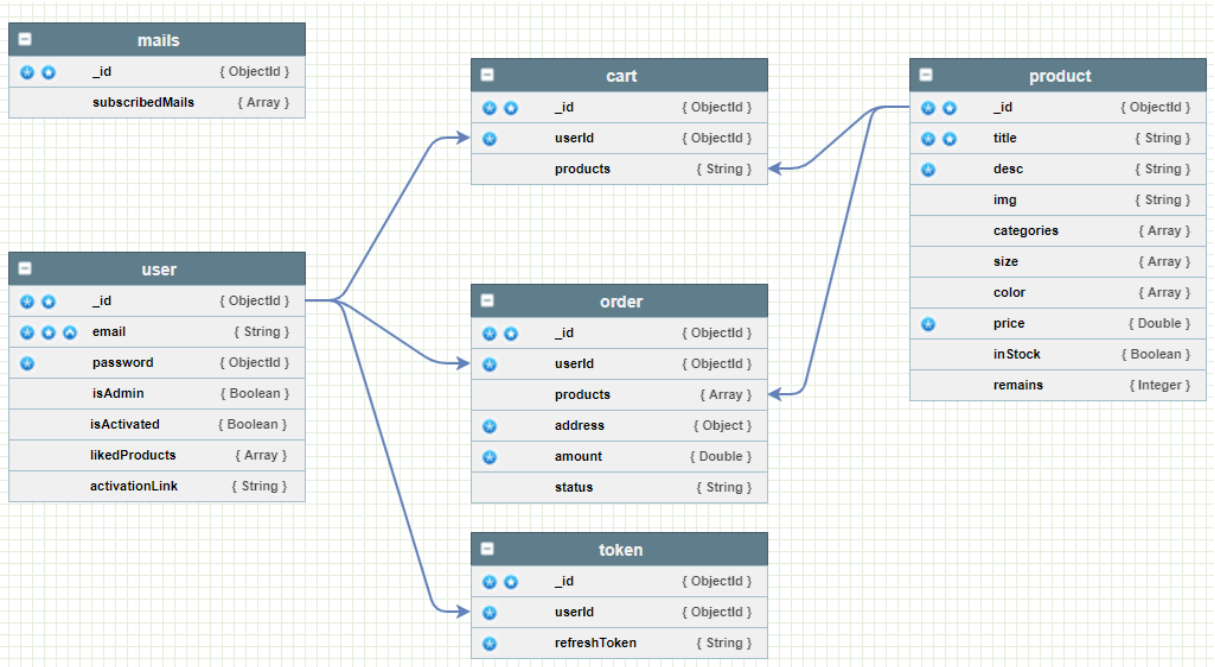


Рисунок 2.1 – Структура БД, подана у вигляді діаграми зв’язків

Відповідно до вищенаведеного рисунку розуміємо, що в кластері MongoDB до даного веб-застосунку для онлайн продаж одягу буде знаходитись

6 колекцій: “carts”, “mails”, “orders”, “products”, “tokens”, “users”. За допомогою цих колекцій можна ефективно та безпечно опрацьовувати дані на серверній частині застосунку, використовуючи спеціальні методи завантажених бібліотек.

Окремо варто звернути увагу на модель “token” (див. рис. 2.2). В ній будуть зберігатись refreshToken, згенеровані за допомогою бібліотеки jwt – “json web token”. Такий підхід дозволяє впроваджувати надійну та стійку до вразливостей системи авторизації користувачів. Детальніша інформація про неї знаходиться в пункті 2.2.2.

Рисунок 2.2 – Структура об’єктів колекції “token”

Якщо розглядати інші колекції, наведені на рисунку 2.1, можна виділити наступну важливу інформацію:

– В колекції “user” міститиметься докладна інформація про зареєстрованих користувачів – ім’я, пошта, захешований пароль, список вподобаних товарів із посиланнями на останні тощо.

– В колекції “cart” зберігаються об’єкти – корзини кожного користувача із посиланнями як на об’єкти колекції “product”, так і колекції “user”. Окрім того, вона міститиме загальну ціну доданих до корзини товарів.

– Колекція “product” зберігатиме об’єкти асортименту магазину – власне одяг, взуття, аксесуари тощо. Так, об’єкт даної колекції матиме такі поля, як назву, опис, посилання на фото, список доступних кольорів та розмірів, кількість залишків на складі тощо.

– Колекція “mails” зберігатиме список електронних скриньок користувачів, які підписались на новинну розсилку.

– Колекція “order” відповідає за зберігання усіх замовлень кожного із користувачів інтернет-магазину. За зв’язок відповідає поле “userId”, яке містить унікальний ідентифікатор користувача, що використовується в якості посилання на колекцію “user”.

Визначившись із структурою бази даних веб-застосунку для онлайн продаж одягу із використанням Node.js 14 та React 17, переходимо до власне її створення. Для цього необхідно скористуватись офіційним сайтом MongoDB, на якому можна створити новий проект, провести налаштування кластеру, отримувати аналітику щодо його використання веб-застосунком тощо. На першому етапі створення вказуються паролі користувача БД – фактично це її розробник. Також, відповідно до специфікації хмарного сервісу Atlas, зазначаються дозволені ip-адреси доступу та вибирається спосіб з’єднання веб-застосунку із базою даних (див. рис. 2.3).

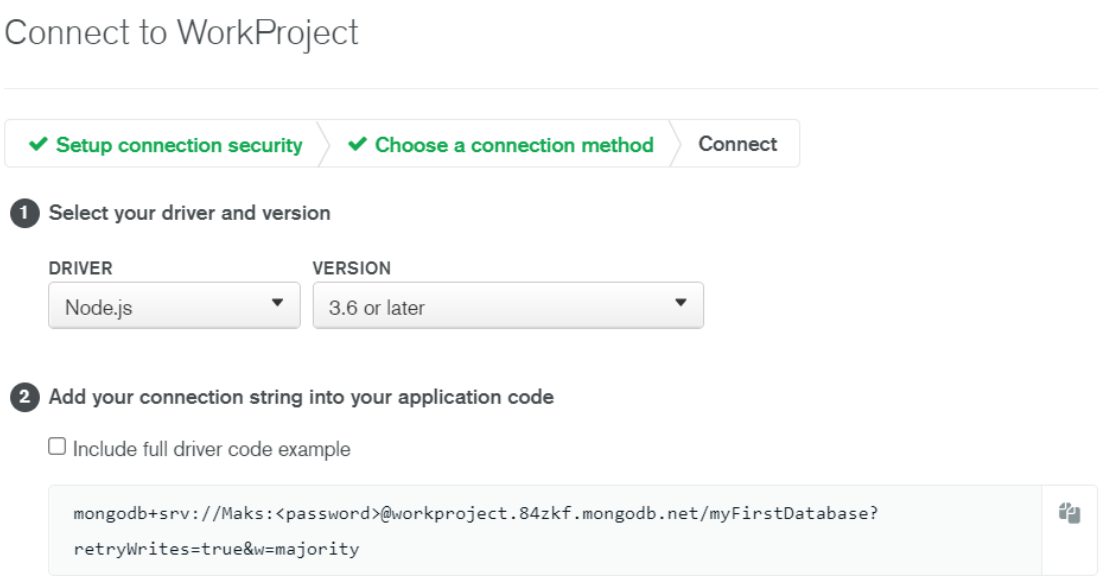


Рисунок 2.3 – Вибір способу під’єднання БД до серверної частини проекту

Відповідно до вищенаведеного рисунку, посилання на підключення містить так звані “заглушки” – це текст, написаний за наступним шаблоном: <…>. Такі значення треба замінити на реально існуючі. Окрім того, замість назви по замовчуванню “myFirstDatabase” можна задати будь-яку іншу.

2.2 Розробка веб-застосунку для онлайн продаж одягу із використанням Node.js 14 та React 17

Опісля проектування та створення бази даних на офіційному сайті MongoDB, можна переходити до власне розробки як серверної, так і клієнтської частин веб-застосунку. Розробка серверу буде проходити із використанням відомої платформи Node.js версії 14, фреймворку Express та декількох необхідних бібліотек, на кшталт Mongoose, BCrypt, JWT, Nodemon тощо [16].

2.2.1 Підключення веб-застосунку до бази даних MongoDB. Створення моделей

Підключення будь-якого сучасного веб-застосунку для онлайн продаж одягу відбувається на серверній частині, а саме у вхідному файлі проекту під назвою “index.js”. Зв’язок із БД встановлюється за допомогою програмного коду, наведеного на рисунку 2.4, де process.env.MONGO_URL – посилання на з’єднання із MongoBD.

Рисунок 2.4 – Програмний JS код для підключення до MongoDB

Як видно із рисунку 2.4, за встановлення з’єднання відповідає бібліотека Mongoose. Також вона дозволяє гнучко налаштовувати поля, задавати їх строгу типізацію, визначати, чи є дане поле унікальним та обов’язковим до заповнення тощо.

Після того, як під’єднання було успішно встановлено, настає етап розробки моделей для зберігання даних. Їх структура повинна відповідати зазначеній на рисунку 2.1. Програмний JavaScript код усіх моделей даних буде подано в додатку А. Для прикладу розглянемо структуру та код моделі користувача на

рисунку в 2.5, реалізованої із використанням вищеописаної бібліотеки Mongoose. Дана модель визначає структуру об’єкта типу “user”, описує усі його поля тощо.

```
1  const mongoose = require("mongoose");
2
3  const UserSchema = new mongoose.Schema(
4    {
5      username: { type: String, required: true },
6      email: { type: String, required: true, unique: true },
7      password: { type: String, required: true },
8      isAdmin: { type: Boolean, default: false },
9      isActivated: { type: Boolean, default: false },
10     likedProducts: {type: Array, default: []},
11     activationLink: { type: String}
12   },
13   { timestamps: true }
14 );
15
16 module.exports = mongoose.model("User", UserSchema);
```

Рисунок 2.6 – Програмний JS код моделі “User”

Так, відповідно до коду, наведеного на рисунку 2.6, розуміємо, що користувач матиме усі стандарті для такого типу об’єктів поля. Окрім того, модель містить поле activationLink (див. 11 лінійку коду), яке використовується під час активації новоствореного аккаунту. Опція конфігурації “timestamps” означає, в базі даних MongoDB також зберігатиметься ревізія і час створення та останньої модифікації даного запису [15].

Окремо варто розглянути модель “cart”, реалізація якої подана на рисунку 2.7. Головною її особливістю, вартої уваги, є те, що декілька із полів містять посилання на інші реалізовані схеми. Таке поле-посилання на іншу модель задається за допомогою типу ObjectId (саме такий тип має будь-який ідентифікатор в MongoDB) та атрибутом ref із іменем схеми, на яку посилається дане поле.

Такий підхід відповідає найсучаснішим патернам проектування нереляційних баз даних та значно спрощує роботу із ними. Проводячи аналогію із реляційними SQL БД на кшталт MySQL, PostgreSQL тощо, можна стверджувати, що посилання на іншу модель – аналог до так званого зовнішнього

ключа. Відповідно до цього, використовуючи метод populate із вказанням полів-посилань, ми можемо отримати об’єднані дані декількох колекцій за один запит до БД. Окрім того, на рисунку 2.7 наведено приклад того, що Mongoose дозволяє задати значення по замовчуванню.

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

const CartSchema = new Schema(
    {
        userId: {type: Schema.Types.ObjectId, ref: "User", required: true},
        products: [
            {
                productId: {
                    type: Schema.Types.ObjectId, ref: "Product",
                },
                quantity: {
                    type: Number,
                    default: 1,
                },
                size: {
                    type: String,
                    required: true
                },
                color: {
                    type: String,
                    required: true
                }
            },
        ],
    },
    {timestamps: true}
);

module.exports = model("Cart", CartSchema);
```

Рисунок 2.7 – Програмний JS код колекції “Cart”

Не варто забувати і про індексацію даних. Вона допомагає більш ефективно та швидко отримувати дані із БД під час запитів до неї. Для того, щоб індексувати будь яке поле, необхідно лише в його описі додати атрибут index із значенням true.

2.2.2 Обробка роутів веб-застосунку для онлайн продаж одягу

Перш за все варто дати визначення поняттю роут, або, інакше кажучи, маршрут. Фактично це посилання в адресному рядку, яке притаманне будь-якому веб-застосунку чи сайту без виключень. Так, відповідно до обраної модель

застосунку, запити (також часто в документаціях вони позначаються англіцизмом “реквести”) надсилаються із front-end частити, створеної із використанням React 17, до серверу, де їх обробку здійснюватиме Node.js 14 із використанням усіх необхідних додаткових бібліотек, завантажених за допомогою пакетного менеджера NPM [16].

На прикладі авторизаційних роутів розглянемо, як власне проходить процес від запитів з клієнтської сторони до їх обробки на серверній частині веб-застосунку для онлайн продаж одягу (див. рис. 2.8). Програмний JavaScript код усіх інших роутів подано в додатку Б.

```
const router = require("express").Router();
const authController = require("../controllers/auth-controller");
const {body} = require("express-validator");

router.post("/register",
    body("email").isEmail(),
    body("password").isLength({min: 6, max: 32}),
    body("username").isLength({min: 3}),
    authController.register)
router.post("/login",
    body("email").isEmail(),
    body("password").isLength({min: 6, max: 32}),
    authController.login)
router.post("/logout", authController.logout)
router.get("/activate/:link", authController.activate)
router.get("/refresh", authController.refresh)
router.post("/resendActivationMail", authController.resendActivation)
router.post("/reset",
    body("email").isEmail(),
    authController.reset)
router.get("/reset/:id/:token", authController.checkResetLink)
router.post("/update-password",
    body("password").isLength({min: 6, max: 32}),
    authController.updatePassword)

module.exports = router;
```

Рисунок 2.8 – Програмний JS код усіх авторизаційних роутів

Із рисунку 2.8 видно, що в розроблюваному веб-застосунку для онлайн продаж одягу використовується декілька типів запитів – post та get. Фактично їх існує значно більше, але в сучасних підходах до розробки веб-застосунків прийнято використовувати лише цих два: post запити у випадку зміни будь-якої інформації в БД, get – для їх отримання. Самі роутери оголошуються за допомогою фреймворку Express. Для їх активації необхідно у вхідному файлі

веб-застосунку виконати їх реєстрацію за допомогою методу use із вказанням шляху даного роута [17]. Повний код даного файлу подано в додатку В.

Окрім того, використовуючи бібліотеку express-validator в якості middleware, можна валідувати тіло запитів за величезною кількістю найрізноманітніших параметрів [18]. Наприклад, код body(“email”).isEmail() перевіряє, щоб значення поля email відповідало усім патернам запису електронної пошти.

Варто зазначити і те, що, зазвичай, в сучасних веб-застосунків на back-end`і використовується mvc підхід – поділ коду на три логічні компоненти: модель, представлення та контролер (див. рис. 2.9).

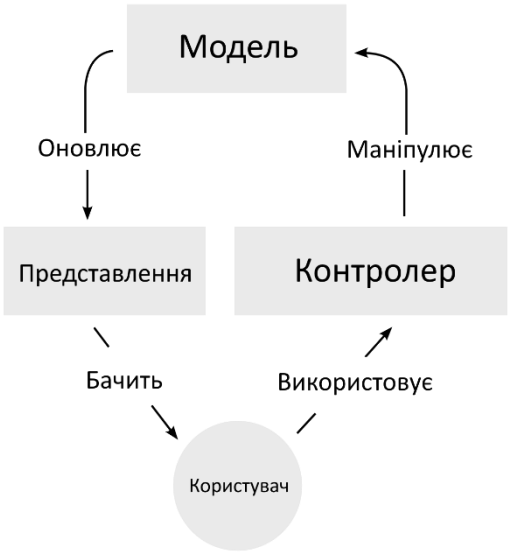


Рисунок 2.9 – Схематичне зображення використання MVC моделі

Отож, користувач має змогу зареєструватись, увійти в систему, активувати свій аккаунт, відновити пароль тощо. Все це працює із використанням jwt access та refresh токенів, що гарантує високу безпеку зберігання та обробки персональних даних [19].

Якщо розглядати програмний код реєстраційного контролера, то варто розуміти, що спочатку відбувається валідація вхідних даних, опісля викликається метод authService.register, який виконує основні операції (див. рис. 2.10). Далі в cookie додається створений refreshToken із часом життя 30 днів та повертається відповідь на клієнт.

```
async register(email, username, parole, isAdmin) {
        const candidate = await UserModel.findOne({email})
        if(candidate) {
            throw ApiError.BadRequest("User with given email already exists");
        }

        const hashedPassword = await bcrypt.hash(parole, 3);
        const activationLink = uuid.v4();
        const newUser = new UserModel({
            username,
            email,
            isAdmin,
            password: hashedPassword,
            activationLink
        });

        const {_doc: {password, ...others}} = await newUser.save();
        await mailService.sendActivationMail(email, `${process.env.API_URL}/api/auth/activate/${activationLink}`);

        const cart = new CartModel({
            userId: others._id,
            products: []
        })
        const {_doc} = await cart.save();

        const tokens = tokenService.generateTokens({...others});
        await tokenService.saveToken(others._id, tokens.refreshToken);

        return { ...tokens, user: {...others}, cart: _doc };
    }
```

Рисунок 2.10 – Програмний JS код методу register сервісу authRegister

Так, із наведеного вище коду бачимо, що спочатку виконується перевірка існування користувача із введеним email`ом. Для цього викликається метод findOne() в моделі користувача, який приймає в себе об’єкт з параметрами-фільтрами (якщо таких не задати, то отримаємо перший запис в колекції). Якщо такого запису не існує, продовжується процес реєстрації.

Оскільки, відповідно до вимог безпеки, категорично не можна зберігати в БД пароль у відкритому вигляді, він хешується. Головна відмінність даного методу від звичного шифрування полягає в тому, що розхешувати пароль назад не можливо. Єдиний варіант перевірити достовірність такого пароля – захешувати новий та порівняти їх між собою. Саме такий інструментарії впроваджує бібліотека bcrypt, яка використовується в багатьох веб-застосунках.

Наступним кроком в MongoBD зберігаються вхідні дані шляхом виклику методу save() в об’єкта моделі “user” та відбувається генерація access і refresh токенів. Сам токен – закодована стрічка, що складається із 3 частин: заголовок, дані, сигнатура. В заголовку вказується тип токена, в даних міститься публічна інформація про користувача. Сигнатура містить дані, закодовані із використанням ключа, який знає тільки сервер нашого веб-застосунку.

Усі токени мають обмежений проміжок життя. Власне саме тому їх існує два типи, адже refreshToken (зазвичай живе 30 днів) перезаписує accessToken (зазвичай живе лише 30 хвилин), колі він вмирає. Токен доступу зберігається в localStorage браузера та використовується під час приватних запитів на сервер, а refresh – в базі даних MongoDB (щось на кшталт сесії) та в httpOnly cookie. За допомогою інтерцепторів на клієнті ми перехоплюємо кожний запит на сервер і, якщо access токен вже мертвий, перезаписуємо його, використовуючи refresh токен. Генеруються вони за допомогою методо sing() із вказанням параметрів та додаткових опцій, який імпортується із бібліотеки jsonwebtoken.

Повертаючись до коду, наведеного на рисунку 2.10, бачимо, що також відбувається надсилання інформаційного листа на вказану новим користувачем пошту. Даний функціонал не є нативним для Node.js будь-якої версії та реалізовується із використанням додаткової бібліотеки Nodemailer. Спершу виконується конфігурацію транспортера (див. рис. 2.11), опісля чого можна реалізовувати надсилання електронних листів за допомогою методу sendMail() який приймає в якості параметрів вміст повідомлення, його тему, адреси адресата та адресанта.

В об’єкті auth вказуються пароль та власне пошта, звідки надсилається інформаційний лист; такі поля, як host та port можна знайти у налаштування скриньки, до якої прив’язана використовувана пошта.

Так, для Gmail ці дані матимуть значення smtp.gmail.com та 587 відповідно. Це дозволяє значно покращити досвід взаємодії користувача із розроблюваним веб-застосунком для онлайн продаж одягу.

```
this.transporter = nodemailer.createTransport({
            host: process.env.SMTP_HOST,
            port: process.env.SMTP_PORT,
            secure: false,
            auth: {
                user: process.env.SMTP_USER,
                pass: process.env.SMTP_PASSWORD
            }
        })
```

Рисунок 2.11 – Конфігураційний код транспортера для надсилання листів

Аналогічно функціонують й інші роути в веб-застосунку для онлайн-продаж одягу. Деякі з ниє є публічними, тобто для їх виклику не потрібно бути авторизованим, інші ж потребують наявності живого access токена або навіть прав адміністратора. Як вже було зазначено, докладний JS код подано в додатку Б.

2.2.3 Реалізація клієнтської сторони та панелі адміністратора із використанням бібліотеки React 17

Головна перевага React над аналогічними бібліотеками чи навіть фреймворками – простота використання та можливість легкого налаштування під будь-які вимоги і задачі веб-застосунку [20]. Завдяки цьому розробка клієнтської сторони та панелі адміністратора фактично не відрізняється.

Аналогічно до back-end`у, front-end теж має вхідний файл – App.jsx, повний код якого подано в додатку Г. Саме в цьому файлі налаштовується роутинг клієнтської сторони. Для цього використовується бібліотека react-router-dom. Вона дозволяє швидко та легко задати шлях для кожної із сторінок, застосувати умовний рендеринг щодо них, використовувати свої кастомні хуки для переходу між сторінками тощо. Так, реєстрація декількох роутів клієнтської сторони веб-застосунку для онлайн продаж одягу подано на рисунку 2.12. Із особливостей варто відмітити, що існує можливість отримання додаткових даних із шляху тієї чи іншої сторінки, які потім можна застосувати в запиті до серверу.

```
<Route path="/login">{store.isAuth ? <Redirect to="/" /> : <Login />}</Route>
<Route path="/register">
  {store.isAuth ? <Redirect to="/" /> : <Register />}
</Route>
<Route path="/reset">
  {store.isAuth ? <Redirect to="/" /> : <Reset />}
</Route>
<Route path="/new-password/:id/:token">
  {store.isAuth? <Redirect to="/" /> : <NewPassword /> }
</Route>
```

Рисунок 2.12 – Реєстрація роутів на клієнтській стороні

Окрім того, в розроблюваному веб-застосунку використовувалась автономна бібліотека керування станом під назвою MobX. Це прямий аналог до усім відомого React Redux, який забезпечує консистентність внутрішнього стану застосунку та впроваджує зручні інструменти для його зміни і модифікації.

Таке сховище зберігається в localStorage браузера, тому навіть після перезавантаження сторінки користувачем дані не втрачаються і, в свою чергу, відпадає потреба робити зайві запити на сервер застосунку [21]. Код MobX сховища подано на рисунку 2.13.

```
user = {};
cart = {products: []};
isAuth = false;
isLoading = false;
errorText = "";

constructor() {
    makeAutoObservable(this)
    makePersistable(this, {name: "SampleStore", properties:["user", "isAuth", "cart"], storage: window.localStorage})
}
setAuth(text) {
    this.isAuth = text;
}
setCart(cart) {
    this.cart = cart;
}
setErrorText(value) {
    this.errorText = value;
}
setUser(user) {
    this.user = user;
}
setLoading(value) {
    this.isLoading = value;
}
```

Рисунок 2.13 – Налаштування, модифікатори та приватні поля MobX сховища

Для того, щоб використовувати дане сховище в будь-якому місці, необхідно помістити його в контекст. Він дозволяє передавити глобальні дані між компонентами без явного передавання пропсів в DOM дереві.

Такий підхід відповідає найсучаснішим патернам, що використовуються при проектуванні front-end`у та значно полегшує розробку, масштабування і подальшу підтримку веб-застосунку. На рисунку 2.14 зображено створення контексту із MobX сховищем, а також його отримання в компоненті.

```
1  import Store from "./store/store";
2
3  const store = new Store()
4  export const Context = React.createContext({store})
5
6  ReactDOM.render(
7      <Context.Provider value={{store}}>
8          <App />
9      </Context.Provider>,
10     document.getElementById("root")
11 );
12
13 // -----------------
14
15 // Отримання сховища
16 const {store} = useContext(Context)
```

Рисунок 2.14 – Створення контексту із сховищем глобальних даних та приклад його отримання в будь-якому React компоненті

Як видно із 16 стрічки на рисунку 2.14, для отримання контексту необхідно скористуватися хуком useContext. React хуки – це відносно нове API, яке дозволяє створювати функціональні компоненти із внутрішнім станом та життєвими циклами без використання класів. Таким чином усі сучасні підходи до розробки веб-застосунків передбачають використання виключно функціональних компонентів в поєднанні із хуками [10].

Загальну ж спрощену схему функціонування MobX, можна описати наступним чином: компонент, загорнутий в метод observer, слідкує за станом сховища та викликає методи, що мутують його; сховище застосовує ред'юсери та оновлює власний стан; компоненти, що слідкують за сховищем, отримують оновлені дані та змінюють свою поведінку чи вигляд [22]. Також було використано головну відмінність MobX від React Redux – мутація стану сховища.

Розглянемо на прикладі компоненту сторінки профілю користувача використання хуків useState та useEffect (див. рис. 2.15). Перша функція використовується для зберігання локального стану в компоненті. Як аргумент приймає початкове значення, а повертає два елементи – власне поточне значення стану та функцію для його оновлення. Друга функція дозволяє керувати супутніми сайд-ефектами в компоненті [10]. Так, це можуть бути запити на сервер, використання таймерів тощо. Фактично це заміна старим методам життєвого циклу класового компонента.

```
const { store } = useContext(Context);
const user = { ...store.user }
const [tab, setTab] = React.useState("1");
const [wishList, setWishList] = React.useState([])
const [orders, setOrders] = React.useState([])
const [open, setOpen] = React.useState(false)
const [openUpdate, setOpenUpdate] = React.useState(false)
const [openSnack, setOpenSnack] = React.useState(false)
const [username, setUsername] = React.useState(user.username);

useEffect(() => {
    (async () => {
        const wishedProducts = await store.wishList([...user.likedProducts])
        setWishList(wishedProducts);

        const orderList = await store.orderList(user._id);
        setOrders(orderList);
    })()
}, [user.likedProducts])
```

Рисунок 2.15 – Використання React хуків в компоненті Account.jsx

Із вищенаведеного рисунка 2.15 бачимо, що сайд-ефектами компонента є виклик функцій із глобального MobX сховища. Запити до серверу відбуваються із використанням відкритої JS бібліотеки axios. Для того, щоб надіслати приватний запит, необхідно додати в його хедери поле авторизації, що міститиме раніше створений сервером accessToken.

Такий запит буде провалідовано відповідним middleware на back-end`i. Оскільки токен має обмежений час життя, існує ймовірність того, що під час запиту він не буде валідним. В такому випадку веб-застосунок, використовуючи перехоплювач (англ. interceptor), спробує згенерувати новий accessToken на основі збереженого в БД refreshToken'у (див. рис. 2.16) та в разі успіху повторить запит на сервер [23].

Таким чином перехоплювачі дозволяють відловлювати будь-які запити клієнту та виконувати необхідні над ними операції, наприклад змінювати дані чи навіть викликати інші команди.

```
import axios from "axios";

export const API_URL = "http://localhost:5000/api"

const $api = axios.create({
    withCredentials: true,
    baseURL: API_URL
})

$api.interceptors.request.use((config) => {
    config.headers.Authorization = `Bearer ${localStorage.getItem(`token`)}`
    return config;
})

$api.interceptors.response.use((config) => {
    return config;
},  async (error) => {
    const originalRequest = error.config;

    if (error.response.status === 401 && error.config && !error.config._isRetry) {
        try {
            const response = await axios.get(`${API_URL}/auth/refresh`, {withCredentials: true});
            localStorage.setItem('token', response.data.accessToken);

            return $api.request(originalRequest);
        } catch (error) {
            console.error("User is not authorized!")
        }
    }
    throw error
})

export default $api;
```

Рисунок 2.16 – Налаштування та модифікація API запитів на основі axios

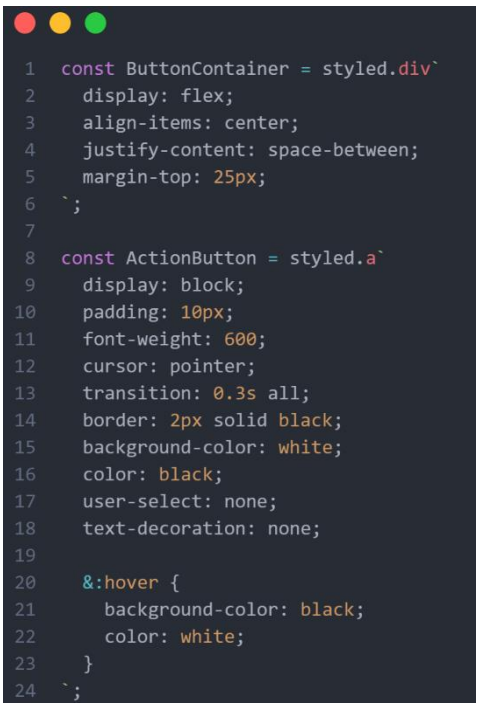
Для того, щоб скористатись налаштованим axios, необхідно викликати один із його HTTP методів – post чи get, вказати роут запиту та дані для передачі на сервер. Параметри можна передавати як в тілі запиту, так в об’єкті з параметрами чи безпосередньо в його url (див. рис. 2.17). Фактично це залежить як від контролера, який прийматиме цей запит, так і від його типу.

```
static async addToCart(cartId, productId, quantity, color, size) {
    return await $api.post("/cart/add", {cartId, productId, quantity, color, size})
}

static async orderList(userId) {
    return await $api.get(`/order/find/${userId}`)
}

static async getWishlist(productIds) {
    return await $api.get("/product/wishlist", {params: productIds})
}
```

Рисунок 2.17 – Реалізація HTTP запитів із використанням налаштованого axios

Створення React компонентів реалізується багатьма способами, один із яких – використання бібліотеки “styled components”, яка дозволяє оголошувати власні компоненти на основі HTML тегів із інкапсульованими CSS стилями [9].

Даний підхід є дуже поширеним в розробці клієнтської сторони сучасних веб-застосунків. Приклад створення компонентів за допомогою вищеописаної бібліотеки подано на рисунку 2.18.



```
const ButtonContainer = styled.div`
  display: flex;
  align-items: center;
  justify-content: space-between;
  margin-top: 25px;
`;

const ActionButton = styled.a`
  display: block;
  padding: 10px;
  font-weight: 600;
  cursor: pointer;
  transition: 0.3s all;
  border: 2px solid black;
  background-color: white;
  color: black;
  user-select: none;
  text-decoration: none;

  &:hover {
    background-color: black;
    color: white;
  }
`;
```

Рисунок 2.18 – Створення styled компонентів

Так, використання бібліотеки “styled components” дійсно є зручним та ефективним інструментом в розробці front-end`у, адже її використання також функціональні можливості CSS препроцесорів – зміна стилів відповідно до взаємодії користувача із компонентом, застосування псевдо-класів та псевдо-елементів тощо [24].

Так, відповідно до наведеного коду бачимо, що компонент *ActivationButton* змінюватиме свій фоновий колір при наведенні користувачем вказівника миші на нього. Це дозволяє зробити веб-застосунок більш зручним та приємним у використанні для кінцевих споживачів.

Окрім власноруч створених, на клієнтській стороні та панелі адміністратора застосовуються готові компоненти з бібліотеки “Material UI”. Основною її перевагою над аналогами є широкий вибір елементів верстки, значний їх функціонал та можливість легко модифікувати поведінку того чи іншого компоненту.

2.3 Тестування розробленого веб-застосунку для онлайн продаж одягу із використанням Node.js 14 та React 17

Завершивши розробку веб-застосунку, варто перевірити його на працездатність. Дане тестування відбуватиметься в декілька етапів – власне тестування в якості користувача застосунку та перевірка даних колекцій в MongoDB.

2.3.1 Перевірка реєстрації, логіну, відновлення паролю та зміни імені

Для початку необхідно створити нового користувача. Для цього необхідно заповнити форму реєстрації. Якщо на даному етапі ввести вже існуючий в базі даних email, користувачу буде повідомлено про помилку, що такий аккаунт вже існує.

У випадку успішної реєстрації в MongoDB з’являється новий запис з даними користувача (див. рис. 2.19), а також зберігаються відповідні jwt токени. Окрім того, на вказану пошту приходить інформаційний лист-вітання із посиланням для активації акаунту. Вона потрібна для можливості оформити замовлення та сплатити його.

Рисунко 2.19 – Запис про новго користувача в БД

Одразу після реєстрації, як і в більшості сучасних веб-застосунках, автоматично виконується вхід в систему. Протестуємо випадок, коли користувач забув пароль та хоче його відновити. Для цього він повинен перейти на відповідну сторінку і вказати свою пошту. Опісля на неї приходить лист із

посиланням на форму відновлення паролю, приклад якого наведено на рисунку 2.20.

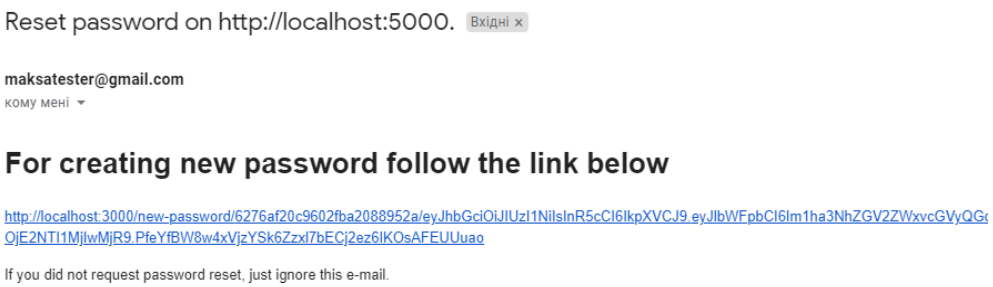


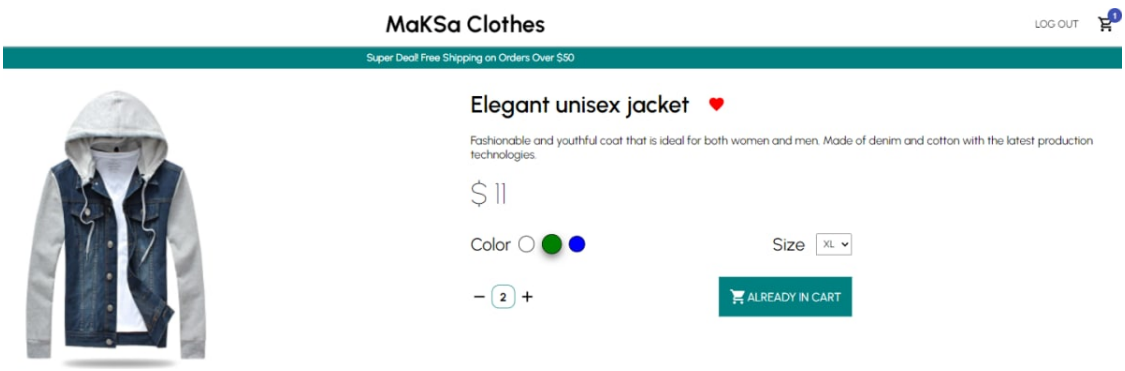
Рисунок 2.20 – Інформаційний лист з посиланням на форму для відновлення паролю аккаунта

Дане посилання має обмежений термін дії, таким чином гарантується безпека та надійність функціоналу по відновленню пароля. Якщо зайти в особистий кабінет, користувач може змінити своє ім’я. Форма для оновлення даної формації подано на рисунку 2.21.

Рисунок 2.21 – Форма оновлення імені користувача

2.3.2 Перевірка основного функціоналу зареєстрованого користувача веб-застосунку для онлайн продаж одягу.

Перш за все, користувач має змогу додавати товари в корзину із різними параметрами: колір, розмір та кількість. Також на сторінці товару він може додати його в список вподобань. Вигляд сторінки товару, доданого в корзину, зображено на рисунку 2.22.



Риснок 2.22 – Шаблон сторінки товару із параметрами вибору

Додаємо ще декілька інших товарів в корзину та спробуємо їх придбати. Процес оплати замовлення забезпечується популярним сервісом для онлайн покупок під назвою Stripe. Його API, відповідно до офіційної документації, надає широкий функціонал для модифікації поведінки форми та надзвичайно точну і важливу аналітику [25]. Для оплати необхідно ввести адресу, свої персональні дані та дані банківської картки (див. рис. 2.23).

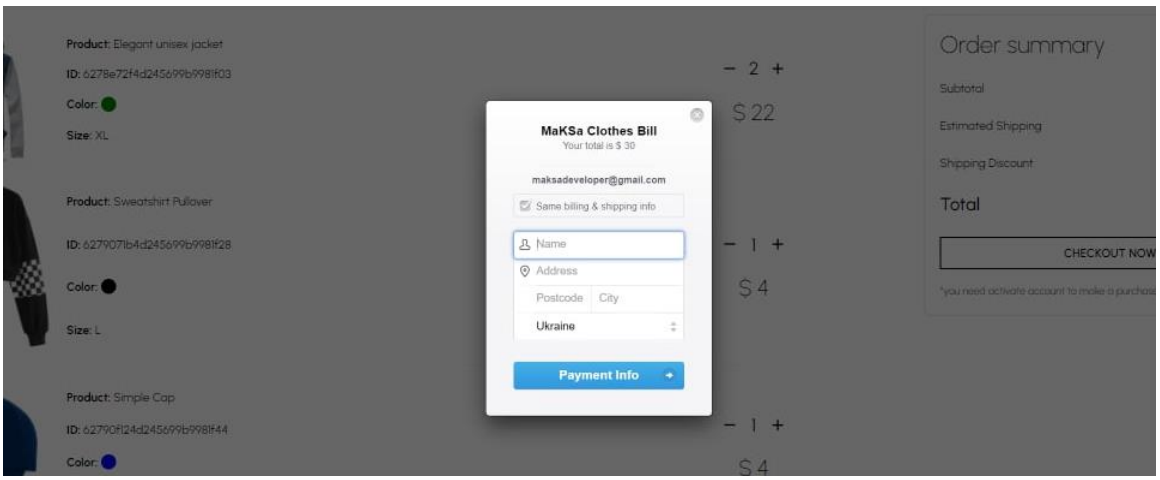


Рисунок 2.23 – Форма оплати замовлення на фоні замовлення, зображеного в корзині користувача

Окрім того, присутнє схематичне зображення усіх здійснених замовлень на сторінці користувача (див. рис. 2.24). Це дозволяє не тільки переглянути їх в будь-який момент, але й порівняти ціни на товари, уточнити замовлений розмір чи колір одягу тощо.

Рисунок 2.24 – Відображення замовлень користувача

Якщо ж говорити за адміністратора, то для управління веб-застосунком він повинен увійти у відповідній для цього формі. Так, для прикладу, спробуємо створити новий продукт, вказавши усі необхідні поля. Після перевіряємо запис в базі даних (див. рис. 2.24)

Рисунок 2.24 – Запис в БД про новий, створений адміністратором продукт

Відповідно до вищенаведеного рисунку бачимо, що запис про нову футболку було успішно створено та збережено в БД. Окрім панелі адміністратора, вносити чи змінювати інформацію про товари, користувачів, замолевння; отримувати статистику продаж тощо можна через інтерфейс на офіційному сайті MongoDB, або із використанням завчасно налаштованих REST-клієнтів по типу Insomnia чи Postman.

2.4 Перспективи модернізації і масштабування веб-застосунку

Як і будь який сучасний інтернет магазин, розроблюваний веб-застосунок для онлайн продаж одягу повинен мати можливість до подальшого розширення функціоналу та масштабування, в тому числі і бази даних MongoDB.

Якщо говорити про перспективи модернізації, то, в ході подальшої підтримки, можна додати відображення декількох фото для одного товару, ввести підтримку загальнопоширеної повноцінної системи знижок із використанням спеціальних купонів тощо. Окрім того, варто проаналізувати готовий код та, при необхідності, провести його оптимізацію. Критичними місцями є запити до MongoDB, структура та спосіб зберігання даних, запити клієнтського інтерфейсу на сервер чи навпаки, важкі і ресурсозатратні алгоритми оброки отриманої із БД інформації, на які потрібно звернути особливу увагу.

Щодо бази даних MongoDB, то сама її концепція забезпечує високу швидкість роботи. Все, що варто зробити – налаштувати коректним чином індексацію та, по можливості, оптимізувати запити до БД таким чином, щоб зменшити їх кількість. Окрім того, оскільки MongoDB є нереляційною, то більшість навантаження можна перенести з неї на серверну частину веб-застосунку.

Варто зазначити те, що універсального підходу до модернізації та масштабування веб-застосунків не існує. Все залежить від сукупності багатьох факторів: це використовувані в ході розробки інструменти і технології, асортимент магазину, його клієнтська база, бажання замовника чи розробника, вимоги, подані в ТЗ тощо.

2.5 Висновок до другого розділу

В другому розділі даної кваліфікаційної роботи освітнього рівня «Бакалавр» було виконано проектування необхідної моделі БД, описано основні етапи та ключові моменти розробки як користувацького інтерфейсу, так і серверної частини веб-застосунку для онлайн продаж одягу із використанням

Node.js 14 та React 17. Сама ж розробка відбувалась із використанням описаних в першому розділі даної кваліфікаційної роботи технологій, дотримуючись сучасних підходів до написання коду.

Окрім того, було відображено структуру певних компонентів користувацького інтерфейсу та моделей даних, створених за допомогою бібліотеки Mongoose. Також було аргументовано використання того чи іншого підходу для вирішення поставлених в ході розробки веб-застосунку задач. Як результат – реалізовано повноцінний веб-застосунок для онлайн продаж одягу з багатими функціональними можливостями.

РОЗДІЛ 3. БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

3.1 Долікарська допомога при ураженні електричним струмом

Перш за все варто розуміти, що удар струмом – це критична загроза здоров'ю та навіть життю людини, тому виклик лікаря в таких випадках є обов'язковим. Окрім того, будь-який прояв зволікання при допомозі чи не якісне її надання можуть заподіяти більших травм чи навіть викликати смерть потерпілого.

Спочатку необхідно проаналізувати поточну ситуацію та зрозуміти, чи потерпілий все ще перебуває під дією електричного струму. Якщо це так, то потрібно, при можливості, якомога швидше спробувати ізолювати джерело струму, вимкнути його, перерізати провідник струму за допомогою інструмента з ізольованими або електро-непровідними ручками, відштовхнути його від потерпілого за допомогою електро-непровідного підручного засобу.

Окрім того, варто спробувати відтягнути потерпілого від джерела електричного струму, уникаючи при цьому контакту із оголеними частинами тіла або навколишніми, особливо металевими предметами. При цьому необхідно ізолювати себе – за наявності одягнути діелектричні рукавиці, обмотати долоні сухою тканиною. Вищеописані дії будуть справедливими у випадках, коли на потерпілого діє струм напругою до 1000 В [26].

Якщо ж відомо, що потерпілий уражений струмом напругою від 1000 В і більше, то безпечно допомогти йому можна тільки при використанні спеціального діелектричного взуття і рукавиць, при цьому, за можливості, відкидаючи джерело струму спеціальною ізольованою штангою. Також можна спробувати заземлити провідники, дія яких спричинила до електричного ураження.

В обидвох випадках, якщо джерелом є провід, що торкається землі чи звисає з ЛЕП, не варто забувати про значну небезпеку так званої крокової напруги [26]. Ось чому, після звільнення потерпілого від дії джерела електричного струму, без наявних засобів захисту на кшталт вищезгаданого

діелектричного взуття, життєво необхідно в зоні розтікання струму (вона становить приблизно 8 метрів) пересуватись таким чином, щоб не відривати однієї ноги від іншої (див. рис. 3.1).

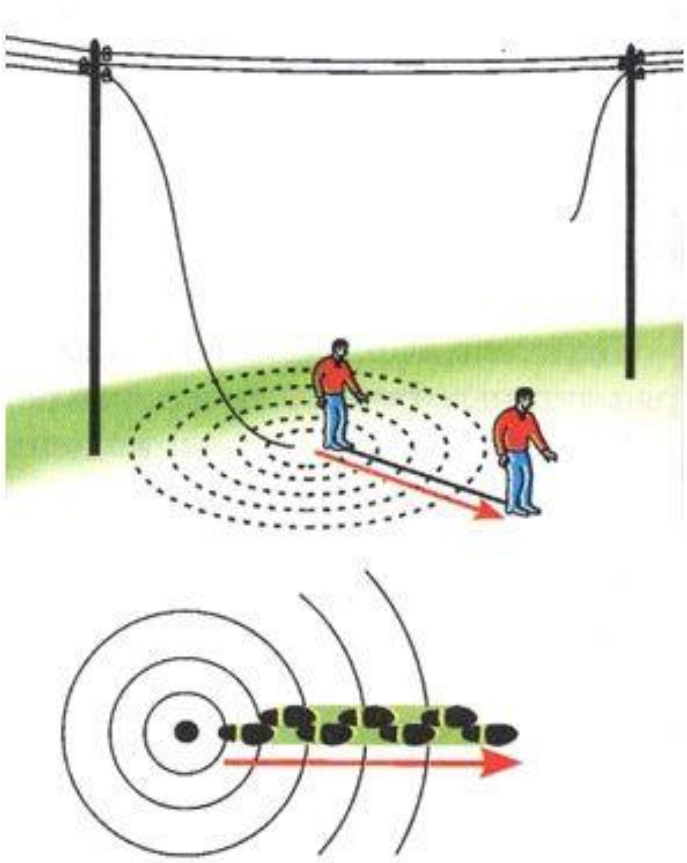


Рисунок 3.1 – Пересування в 8-ми метровій зоні дії крокової напруги

Після того, як дію провідника електричного струму на потерпілого було припинено (або вона була відсутня з самого початку), необхідно провести діагностування стану людини. Загалом, в наслідок дії електричного струму на організм, прийнято виділяти три стани [27]:

– перший стан: потерпілий знаходиться в свідомості. В цьому випадку необхідно надати йому зручного положення та забезпечувати спокій до прибуття медичного працівника [27]. Окрім того, не можна дозволяти людині рухатись чи продовжувати будь-яку фізичну діяльність, адже це може погіршити його стан;

– другий стан: потерпілий знаходиться без свідомості, але продовжує дихати. Аналогічно до першого, необхідно покласти людину в стабільне положення, розстебнути його одяг, що заважає диханню та, по можливості,

забезпечити надходження свіжого повітря [27]. Окрім того, можна спробувати привести потерпілого до тями, давши йому понюхати нашатирний спирт або бризнувши на його обличчя водою;

– третій стан: потерпілий не дихає зовсім, або його дихання переривчасте, судомне, поверхневе. Це найважчий стан із всіх трьох. При ньому необхідно невідкладно почати проводити заходи серцево-легеневої реанімації у вигляді зовнішнього масажу серця та штучного дихання (рис. 3.2). Якщо цього не зробити, то до приїзду бригади швидкої допомоги неодмінно настане смерть потерпілого [28]. Перед проведенням штучного дихання необхідно надати потерпілому правильного положення, перевірити наявність сторонніх тіл чи рвотних або кров'яних мас в ротовій порожнині та, за необхідності, очистити її.

Рисунок 3.2 – Схематичне зображення виконання серцево-легеневої реанімації однією людиною

Справедливим до усіх трьох станів буде наступне: переміщати потерпілого можна тільки тоді, коли існує загроза його життю чи життю людини, яка надає першу допомогу. Також, при роботі з веб-застосунками користувач повинен правильно поводитися з ПК, адже це потенційне джерело напруги та небезпеки. Окрім того, удар електричним струмом здатен призвести до смерті. Ось чому надання негайної долікарської допомоги є важливим фактором задля збереження здоров'я та життя. Для превентивного запобігання уражень електричним струмом при роботі з ПК слід встановити додаткові захисні пристрої, які гарантуватимуть ізольованість струмопровідних частин.

3.2 Загальні вимоги безпеки з охорони праці для користувачів ПК

Перелік загальних вимог для користувачів ПК є досить об’ємними. Так, перш за все, до самостійної роботи із ЕОМ чи ПК повинні допускатись тільки ті особи, які ознайомлені із правилами безпеки, пройшли відповідний навчальний інструктаж з питань охорони праці, а також пожежної безпеки при роботі з ПК. Окрім того, за необхідності дані персони повинні мати довідку про медичний огляд та дозвіл бути допущеними до комп’ютера за станом здоров’я [29].

Сам користувач повинен дотримуватися ряду заздалегідь визначених та затверджених документально правил: не можна допускати до наданого ПК сторонніх осіб; забороняється встановлювати на обладнання неліцензійне ПЗ; категорично забороняється виконувати дії, що суперечать загальним вимогам з охорони праці.

Якщо ж говорити про вимоги безпеки з охорони праці для користувачі ПК безпосередньо перед початком роботи, то останні повинні переконатись у відсутності пошкоджень обладнання та проводки, а у разі їх виявлення повідомити відповідальну особу і ні в якому разі не приступати до роботи. Окрім того, робоче місце повинно розміщуватись на відстані не менше 1 метра від стіни. Кут нахилу монітора ПК при цьому повинен становити від 10° до 25°, а відстань від очей користувача до монітору – в діапазоні від 500 мм до 600 мм. Окрім того, до поля зору користувача не повинні попадати прямі чи віддзеркалені сонячні промені [29].

Відповідно до цього робоче місце варто розмістити таким чином, щоб світло від вікна падало з лівого чи правого боків. Правильне облаштування робочого місця подано на рисунку 3.3. Якщо поруч наявне будь-яке джерело тепла, то, відповідно до вимог безпеки з охорони праці, ПК повинен розташовуватись від нього на відстані не менше, ніж 1 метр.

При цьому освітленість робочого місця користувача ПК має становити не менше 400 лк на висоті 80 см від підлоги та не більше 200 лк у площині екрану [30]. Приклад коректної посадки користувача ПК так його робочого місця відповідно до загальних вимог безпеки з охорони праці подано на рисунку 3.3.

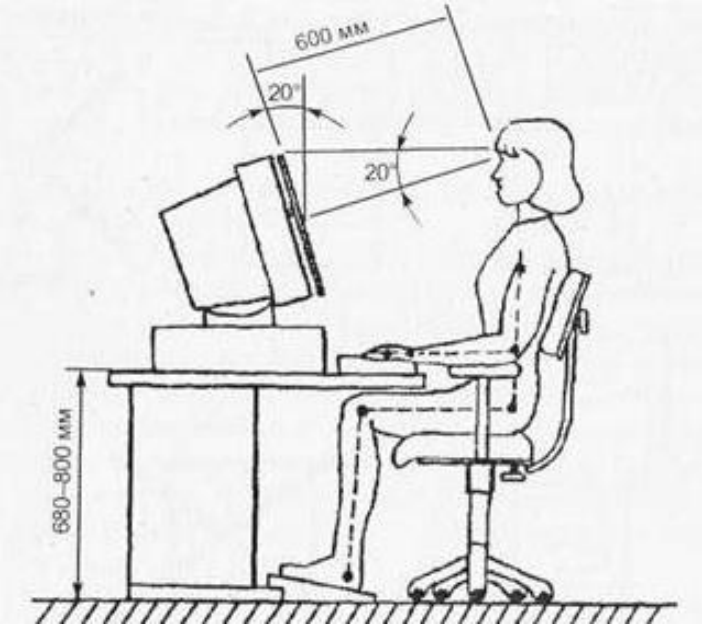


Рисунок 3.3 – Зображені схематично вимоги до облаштування та організації робочого місця користувача ЕОМ чи ПК

Окрім того, відповідно до вимог з безпеки охорони праці для користувачів ПК, останні мають знати алгоритм дій в критичних чи аварійних ситуаціях [30]. Так, при виявленні стороннього звуку чи гореного запаху, необхідно негайно припинити роботу із ПК, вимкнути все обладнання, знеструмити його та повідомити про це відповідальну особу.

При роботі із веб-застосунками та ПК загалом користувач повинен дотримуватись усіх вищезазначених правил та рекомендацій, що сприятиме комфортній та достатньо продуктивній роботі.

3.3 Висновок до третього розділу

В третьому розділі кваліфікаційної роботи описано алгоритм надання першої долікарської допомоги постраждалій від електричного струму людині. Так, спершу варто оцінити обстановку та ізолювати чи позбутись джерела електричного струму, і тільки після цього, відповідно до одного із трьох наведених та детально описаних станів постраждалого, надавати долікарську

допомогу із дотриманням усіх зазначених вимог та порад правил безпеки життєдіяльності людини.

Окрім того, описано загальні вимоги безпеки з охорони праці для користувачів ПК. Дотримання цих правил є досить важливим, адже нехтування ними може призвести не тільки до виходу з ладу комп’ютерного обладнання чи окремо взятої ЕОМ, але, так чи інакше, нашкодити здоров’ю власне користувача ПК та оточуючим його людям [30].

ВИСНОВКИ

В результаті виконання даної кваліфікаційної роботи освітнього рівня «Бакалавр», відповідно до найсучасніших патернів та вимог, було спроектовано та реалізовано веб-застосунок для онлайн продаж одягу із використанням Node.js 14 та React 17. Він проваджує ефективну взаємодію із клієнтами та є зручним в роботі, забезпечуючи супровід усіх процесів купівлі людиною речей в мережі інтернет.

Для розробки використовувались такі засоби, як нереляційна БД MongoDB, серверна платформа Node.js в поєднанні із фреймворком Express, бібліотеки React, MobX, Nodemailer, Material Ui, JsonWebToken тощо. Можна стверджувати, що розроблений веб-застосунок збільшить клієнтську базу та популяризуватиме онлайн продажі.

В першому розділі кваліфікаційної роботи освітнього рівня «Бакалавр»:

– Проведено загальний аналіз обраної предметної області.

– Розглянуто та сформовано перелік вимог до розроблюваного веб-застосунку для онлайн продаж одягу.

– Вибрано середовище розробки веб-застосунку для онлайн продаж одягу із використанням Node.js 14 та React 17.

– Проаналізовано та сформовано діаграми варіантів використань, виконано пошук актантів.

В другому розділі кваліфікаційної роботи:

– Розроблено та спроектовано структуру нереляційної бази даних веб-застосунку для онлайн продаж одягу.

– Відображено структуру моделей та роутів, аргументовано використання того чи іншого підходу до їх реалізації.

– Спроектовано серверну та клієнтську частини веб-застосунку.

– Протестовано загальні функціональні можливості зареєстрованого користувача та адміністратора.

У розділі «Безпека життєдіяльності, основи хорони праці» описано алгоритми надання долікарської допомоги при ураженні електричним струмом,

висвітлено основні небезпеки, пов’язані із струмом та наведено ряд порад до надання допомоги потерпілому відповідно до його стану. Також в даному розділі було описано та наведено загальні вимоги безпеки з охорони праці для користувачів ПК відповідно до найновіших стандартів.

ПЕРЕЛІК ДЖЕРЕЛ

1. Кириченко А. В. Основы современного WEB-дизайна / А. В. Кириченко, А. А. Хрусталев. – 2-е изд. – Санкт-Петербург: Наука и техника, 2021. – 352 с.

2. Що таке CSS [Електронний ресурс]. Режим доступу до ресурсу: https://css.in.ua/article/shcho-take-html_10.

3. Essential Ecommerce Website Requirements [Електронний ресурс]. Режим доступу до ресурсу: https://www.purchasecommerce.com/blog/10-essential-ecommerce-requirement-best-in-class-feature.

4. Functional Requirements for eCommerce Websites [Електронний ресурс]. Режим доступу до ресурсу: https://dinarys.com/blog/functional-requirements-for-ecommerce-site.

5. What is Use Case Diagram? [Електронний ресурс]. Режим доступу до ресурсу:https://www.visual-paradigm.com/guide/uml-unifiedmodelinglanguage/what-is-use-case-diagram/.

6. Brown E. Web Development with Node and Express: Leveraging the JavaScript Stack / E. Brown. – 2nd edit. – Cambridge: O`Reilly Media, 2018. – 325 p.

7. Meet Websotrm [Електронний ресурс]. Режим доступу до ресурсу: https://www.jetbrains.com/help/webstorm/meet-webstorm.html.

8. Херрон Д. Node.js. Разработка серверных веб-приложений на JavaScript / Д. Херрон. – Москва: ДМК Пресс, 2014. – 144 с.

9. Введение в MongoDB. Что такое MongoDB [Електронний ресурс]. Режим доступу до ресурсу: https://metanit.com/nosql/mongodb/1.1.php.

10. Посібник: знайомство з React [Електронний ресурс]. Режим доступу до ресурсу: https://uk.reactjs.org/tutorial/tutorial.html.

11. Meyer E. A. CSS: The Definitive Guide: Visual Presentation for the Web / E. A. Meyer, E. Weyl. – 4th Edition. USA: O'Reilly Media, 2017. – 1090 p.

12. Head First. Патерни проєктування / Е. Фрімен, Е. Робсон, Б. Бейтс, К. Сьерра ; пер. А. Якубовська. – Харків: Фабула, 2020. – 672 с.

13. MongoDB Documentation. [Електронний ресурс]. Режим доступу до ресурсу: https://docs.mongodb.com/#welcome-to-the-mongodb-documentation.

14. Каскіаро М. Шаблони проектування Node.js / М. Каскіаро. – Харків: ДМК Прес, 2017. – 396 с.

15. Mongoose API Docs [Електронний ресурс]. Режим доступу до ресурсу: https://mongoosejs.com/docs/api.html.

16. Ramon J. Everything you need to know about Node.js [Електронний ресурс]. Режим доступу до ресурсу: https://dev.to/jorge_rockr/everything-you-need-to-know-about-node-js-lnc#theproblemwithcpuintensivetasks.

17. Довідкова документація про API Node.js [Електронний ресурс]. Режим доступу до ресурсу: https://nodejs.org/uk/docs/.

18. Использование промежуточных обработчиков [Електронний ресурс]. Режим доступу до ресурсу: https://expressjs.com/ru/guide/using-middleware.html.

19. Node.js в действии / М. Кантелон, М. Хартер, Т. Головайчук, Н. Райлих. – 2-е изд. – Санкт-Петербург: Питер, 2018. – 432 с.

20. Banks A. Learning React: Functional Web Development with React and Redux / A. Banks, E. Porcello. – 1st edit. – USA: O`Reilly Media, 2018. – 320 p.

21. React + MobX: в чём смысл? [Електронний ресурс]. Режим доступу до ресурсу: https://habr.com/ru/post/471048/.

22. MobX Documentation [Електронний ресурс]. Режим доступу до ресурсу: https://mobx.js.org/README.html.

23. Simpson K. You Don`t Know JS: Up & Going / K. Simpson. – 1st edit. – USA: O`Reilly Media, 2020. – 88 p.

24. Understanding client-side JavaScript frameworks [Електронний ресурс]. Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks.

25. Stripe API Documentation [Електронний ресурс]. Режим доступу до ресурсу: https://stripe.com/docs/api.

26. Миченко І. М. Забезпечення життєдіяльності людини в навколишньому середовищі / І. М. Миченко. – Кіровоград, 1998. – 292 с.

27. Джигирей В. С. Безпека життєдіяльності : навч. посіб. – 3-тє вид. – Львів: Афіша, 2000. – 256 с.

28. Кобилянський О. В. Охорона праці в робітничій професії: навч посіб. / О. В. Кобилянський, В. В. Присяжнюк, В. В. Богачук. – Вінниця: ВНТУ, 2009. – 144 с.

29. Бедрій Я. І. Основи охорони праці користувачів персональних комп'ютерів / Я. І. Бедрій. – Тернопіль: Навчальна книга – Богдан, 2014. – 144 с.

30. Гандзюк М. П. Основи охорони праці : навч. посіб. / М. П. Гандзюк, Є. П. Желібо, О. М. Халімовський. – Київ: Каравелла, 2004. – 407 с.

ДОДАТКИ

Додаток А

Програмний JavaScript код усіх моделей веб-застосунку

```
const {Schema, model} = require("mongoose");
const CartSchema = new Schema(
    {
        userId: {type: Schema.Types.ObjectId, ref: "User",
required: true},
        products: [
            {
                productId: {
                    type: Schema.Types.ObjectId, ref: "Product",
                },
                quantity: {
                    type: Number,
                    default: 1,
                },
                size: {
                    type: String,
                    required: true
                },
                color: {type: String, required: true}
            },
        ],
    },
    {timestamps: true}
);
module.exports = model("Cart", CartSchema);

const mongoose = require("mongoose");
const MailsSchema = new mongoose.Schema(
    {
        subscribedMails: {type: Array, default: []},
    },
);
module.exports = mongoose.model("Mails", MailsSchema);

const {Schema, model} = require("mongoose");
const OrderSchema = new Schema(
    {
        userId: {type: Schema.Types.ObjectId, ref: "User",
required: true},
        products: [
            {
                productId: {
                    type: Schema.Types.ObjectId, ref: "Product"
                },
                quantity: {
                    type: Number,
                    default: 1,
                },
                size: {
                    type: String,
```

```
                    required: true
                },
                color: {
                    type: String,
                    required: true
                }
            },
        ],
        amount: {type: Number, required: true},
        address: {type: Object, required: true},
        status: {type: String, default: "pending"},
    },
    {timestamps: true}
);
module.exports = model("Order", OrderSchema);
const mongoose = require("mongoose");
const ProductSchema = new mongoose.Schema(
  {
    title: { type: String, required: true, unique: true },
    desc: { type: String, required: true },
    img: { type: String, required: true },
    categories: { type: Array },
    size: { type: Array },
    color: { type: Array },
    price: { type: Number, required: true },
    inStock: { type: Boolean, default: false },
    remains: {type: Number, default: 0}
  },
  { timestamps: true }
);
module.exports = mongoose.model("Product", ProductSchema);
const {Schema, model} = require("mongoose");
const TokenSchema = new Schema({
    user: {type: Schema.Types.ObjectId,  ref: "User"},
    refreshToken: {type: String, required: true}
});
module.exports = model("Token", TokenSchema);
const mongoose = require("mongoose");
const UserSchema = new mongoose.Schema(
  {
    username: { type: String, required: true },
    email: { type: String, required: true, unique: true },
    password: { type: String, required: true },
    isAdmin: { type: Boolean, default: false },
    isActivated: { type: Boolean, default: false },
    likedProducts: {type: Array, default: []},
    activationLink: { type: String}
  },
  { timestamps: true }
);
module.exports = mongoose.model("User", UserSchema);
```

Додаток Б

Програмний JavaScript код усіх роутів веб-застосунку

```
router.post("/register",
    body("email").isEmail(),
    body("password").isLength({min: 6, max: 32}),
    body("username").isLength({min: 3}),
    authController.register)
router.post("/login",
    body("email").isEmail(),
    body("password").isLength({min: 6, max: 32}),
    authController.login)
router.post("/logout", authController.logout)
router.get("/activate/:link", authController.activate)
router.get("/refresh", authController.refresh)
router.post("/resendActivationMail",
authController.resendActivation)
router.post("/reset",
    body("email").isEmail(),
    authController.reset)
router.get("/reset/:id/:token", authController.checkResetLink)
router.post("/update-password",
    body("password").isLength({min: 6, max: 32}),
    authController.updatePassword)

router.get("/find/:userId", verifyToken, cartController.find);
router.get("/list", verifyTokenAndAdmin, cartController.list);
router.post("/delete", verifyToken, cartController.delete);
router.post("/add", verifyToken, cartController.add);
router.post("/create", verifyToken, cartController.create);
router.post("/manage", verifyToken, cartController.manage);

router.post("/subscribe",
    body("email").isEmail(),
    mailsController.subscribe);

router.get("/income", verifyTokenAndAdmin,
orderController.income);
router.get("/list", verifyTokenAndAdmin, orderController.list);
router.get("/find/:userId", verifyToken, orderController.find);
router.post("/delete", verifyToken, orderController.delete);
router.post("/update", verifyToken, orderController.update);
router.post("/create", verifyToken, orderController.create);

router.get("/list", productController.list);
router.get("/find/:id", productController.find);
router.post("/delete", verifyTokenAndAdmin,
productController.delete);
router.post("/create", verifyTokenAndAdmin,
productController.create);
router.post("/update", verifyTokenAndAdmin,
productController.create);
router.post("/like", verifyToken, productController.like);
```

```
router.get("/wishlist", verifyToken, productController.wishlist)

router.post("/payment", (req, res) => {
    stripe.charges.create(
        {
            source: req.body.tokenId,
            amount: req.body.amount,
            currency: "usd",
        },
        (stripeErr, stripeRes) => {
            if (stripeErr) {
                res.status(500).json(stripeErr);
            } else {
                res.status(200).json(stripeRes);
            }
        }
    );
});

router.get("/list", verifyTokenAndAdmin, userController.list);
router.get("/stats", verifyTokenAndAdmin, userController.stats);
router.get("/find/:id", verifyTokenAndAdmin, userController.find);
router.post("/delete", verifyToken, userController.delete);
router.post("/update",
    body("username").isLength({min: 3}),
    body("email").isEmail(),
    verifyToken, userController.update);
router.post("/manageAdmin", verifyTokenAndAdmin,
userController.admin);
router.post("/clearWishList", verifyToken,
userController.clearWishList)
```

Додаток В

Програмний JavaScript код до вхідного серверного файлу “index.js”

```
const express = require("express");
const app = express();
const mongoose = require("mongoose");
const dotenv = require("dotenv");
dotenv.config();
const errorMiddleware = require("./middlewares/error-middleware")
const authRoute = require("./router/auth");
const userRoute = require("./router/user");
const mailsRoute = require("./router/mails");
const stripeRoute = require("./router/stripe");
const productRoute = require("./router/product");
const orderRoute = require("./router/order");
const cartRoute = require("./router/cart");
const cors = require("cors");
const cookieParser = require("cookie-parser")

mongoose
  .connect(process.env.MONGO_URL)
  .then(() => console.log("DB Connected Successfully!"))
  .catch((err) => {
    console.log(err);
  });

app.use(cors({
    credentials: true,
    origin: process.env.CLIENT_URL
}));
app.use(cookieParser())
app.use(express.json());

app.use("/api/auth", authRoute);
app.use("/api/user", userRoute);
app.use("/api/mails", mailsRoute);
app.use("/api/product", productRoute);
app.use("/api/cart", cartRoute);
app.use("/api/order", orderRoute);
app.use("/api/checkout", stripeRoute);

app.use(errorMiddleware);

const PORT = process.env.PORT || 5000

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

Додаток Г

Програмний JavaScript код файлу клієнтської сторони “App.jsx”

```
import Product from "./pages/Product";
import Home from "./pages/Home";
import ProductList from "./pages/ProductList";
import Register from "./pages/Register";
import Login from "./pages/Login";
import Cart from "./pages/Cart";

import {
  BrowserRouter as Router,
  Switch,
  Route,
  Redirect,
} from "react-router-dom";
import Success from "./pages/Success";
import {useContext, useEffect} from "react";
import {Context} from "./index";
import {observer} from "mobx-react-lite";
import Reset from "./pages/Reset";
import NewPassword from "./pages/NewPassword";
import Account from "./pages/Account";

const App = () => {
  const {store} = useContext(Context);
  useEffect(() => {
    if(localStorage.getItem("token")) {
      store.checkAuth()
    }
    document.body.style.paddingTop = "60px"

    return () => {
      document.body.style.paddingTop = "0px"
    };
  }, [])
  return (
    <Router>
      <Switch>
        <Route exact path="/">
          <Home />
        </Route>
        <Route path="/products/:category">
            <ProductList />
        </Route>
        <Route path="/product/:id">
          <Product />
        </Route>
        <Route path="/cart">
          {!store.isAuth ? <Redirect to="/" /> : <Cart />}
        </Route>
        <Route path="/account">
          {!store.isAuth ? <Redirect to="/" /> : <Account />}
        </Route>
```

```
        <Route path="/success">
          <Success />
        </Route>
        <Route path="/login">{store.isAuth ? <Redirect to="/" /> :
<Login />}</Route>
        <Route path="/register">
          {store.isAuth ? <Redirect to="/" /> : <Register />}
        </Route>
        <Route path="/reset">
          {store.isAuth ? <Redirect to="/" /> : <Reset />}
        </Route>
        <Route path="/new-password/:id/:token">
          {store.isAuth? <Redirect to="/" /> : <NewPassword /> }
        </Route>
      </Switch>
    </Router>
  );
};
export default observer(App);
```