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

Факультет інформаційних систем та програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Розробка веб-маркетплейсу аксесуарів для ПК на базі Node.js та MySQL

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

		Гураль В. І.
	(підпис)	(прізвище та ініціали)
Керівник		Цуприк Г. Б.
	(підпис)	(прізвище та ініціали)
Нормоконтроль		Стоянов Ю.М.
	(підпис)	(прізвище та ініціали)
Завідувач кафедри		Петрик М.Р.
	(підпис)	(прізвище та ініціали)
Рецензент		Жаровський Р.О.
	(підпис)	(прізвище та ініціали)

Тернопіль 2025

АНОТАЦІЯ

Кваліфікаційна робота бакалавра. Тернопільський національний технічний університет імені Івана Пулюя, кафедра програмної інженерії, спеціальність 121 «Інженерія програмного забезпечення». ТНТУ, 2025. Сторінок 82, рисунків 25, додатків 4, презентація.

Ключові слова: веб-маркетплейс, електронна комерція, Node.js, MySQL, аксесуари для ПК, веб-розробка.

У валіфікаційній робота бакалавра розроблено сучасний веб-маркетплейсс для продажу аксесуарів для персональних комп'ютерів з використанням Node.js та MySQL.

Об'єктом дослідження є сучасний веб-маркетплейс для торгівлі комп'ютерними аксесуарами та реляційна база даних MySQL.

Метою цієї роботи є створення ефективної веб-платформи для торгівлі комп'ютерними аксесуарами, яка забезпечує зручний інтерфейс для покупців та продавців.

Методи дослідження включають використання Node.js як основного серверного середовища для розробки веб-додатків, Express.js як веб-фреймворку для створення REST API, MySQL для зберігання даних, EJS для серверного рендерингу, React для клієнтської частини та Passport.js для реалізації системи аутентифікації користувачів.

Розроблена система відповідає поставленим вимогам і демонструє практичну цінність для впровадження у сфері електронної комерції. Її архітектура дозволяє легко масштабувати функціонал і адаптувати платформу до потреб різних категорій користувачів.

Результати дослідження можуть бути використані для створення подібних веб-маркетплейсів у сфері електронної комерції. У перспективі доцільно додати адміністративну панель, систему оформлення замовлень та інші інструменти для покращення роботи платформи.

ABSTRACT

Bachelor's qualification work. Ternopil National Technical University named after Ivan Puluj, Department of Software Engineering, specialty 121 "Software Engineering". TNTU, 2025. 82 pages, 25 figures, 4 appendices, presentation.

Keywords: web marketplace, e-commerce, Node.js, MySQL, PC accessories, web development.

In this bachelor's thesis, a modern web marketplace for selling personal computer accessories has been developed using Node.js and MySQL.

The object of research is a modern web marketplace for trading computer accessories and the MySQL relational database.

The purpose of this work is to create an effective web platform for trading computer accessories that provides a convenient interface for buyers and sellers.

Research methods include the use of Node.js as the main server environment for web application development, Express.js as a web framework for creating REST API, MySQL for data storage, EJS for server-side rendering, React for the client-side, and Passport.js for implementing user authentication system.

The developed system meets the set requirements and demonstrates practical value for implementation in the field of e-commerce. Its architecture allows for easy scaling of functionality and adaptation of the platform to the needs of different user categories.

The research results can be used to create similar web marketplaces in the field of e-commerce. In the future, it is advisable to add an administrative panel, order processing system, and other tools to improve the platform's operation.

ЗМІСТ

АНОТАЦІЯ 4
ABSTRACT 5
ЗМІСТ 6
ПЕРЕЛІК СКОРОЧЕНЬ 8
ВСТУП 9
1 РОЗРОБКА ПРОГРАМНОЇ СИСТЕМИ 11
1.1 Аналіз вимог до програмної системи 11
1.1.1 Функціональні вимоги: 11
1.1.2 Нефункціональні вимоги 11
1.1.3 Технічні вимоги 12
1.1.4 Аналіз предметної області 13
1.1.5 Постановка задачі 14
1.1.6 Пошук актантів та варіантів використання 15
1.1.7 Опис ключових варіантів використання 16
1.1.8 Моделювання словника системи 18
1.2 Проектування програмної системи 19
1.2.1 Вибір процесу розробки 20
1.2.2 Побудова схеми бази даних 22
1.2.3 Побудова UML-діаграми класів 24
1.2.4 Моделювання архітектури системи 27
1.2.5 Маршрутизація 29
1.3 Конструювання програмної системи 30
1.3.1 Вибір мови та середовища розробки 32
1.3.2 Вибір СУБД та опис її фізичної моделі 33
1.3.3 Реалізація основних класів та методів 35
1.3.4 Розробка серверної частини 37
1.4. Використання програмної системи 40
1.4.1 Розгортання програмної системи та системні вимоги 40

1.4.2 Опис типових схем використання системи 42
1.4.3 Верифікація програмної системи 47
2 ТЕСТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ 48
2.1 План тестування 48
2.2 Тестування веб сайту Techno Market 50
3 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ ТА ОСНОВИ ОХОРОНИ ПРАЦІ 52
3.1 Ризик як кількісна оцінка небезпек 52
3.2 Правила техніки безпеки при експлуатації обладнання 54
ВИСНОВКИ 56
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 57
ДОДАТКИ 60
Додаток А Структура Проєкту 61
Додаток Б Ілюстрації варіантів використання системи 77
Додаток В Публікація у науковій конференції 78
Додаток Г Диск з роботою 82

ПЕРЕЛІК СКОРОЧЕНЬ

1. Сесія (Session) - тимчасовий стан взаємодії користувача з системою, що зберігає інформацію про авторизованого користувача та його поточні дії.
2. Аутентифікація (Authentication) - процес перевірки особистості користувача при вході в систему за допомогою електронної пошти та пароля.
3. Авторизація (Authorization) - процес перевірки прав доступу користувача до певних ресурсів системи відповідно до його ролі.
4. Маршрутизація (Routing) - механізм визначення обробника для кожного HTTP-запиту в системі, що забезпечує правильну обробку запитів користувачів.
5. Веб-маркетплейс — веб-платформа, яка забезпечує взаємодію між продавцями та покупцями. Дозволяє продавцям розміщувати товари, а покупцям — переглядати, додавати в кошик.
6. Електронна комерція (E-commerce) — форма торгівлі, що здійснюється за допомогою цифрових технологій, зокрема через інтернет. Включає продаж, купівлю товарів або послуг, оплату та доставку онлайн.
7. Node.js — середовище виконання JavaScript-коду на стороні сервера, яке дозволяє створювати високопродуктивні веб-додатки. Часто використовується для реалізації бекенду сайтів.
8. MySQL — система управління реляційними базами даних з відкритим кодом. Використовується для зберігання, обробки та запиту даних у веб-додатках.
9. Аксесуари для ПК — апаратні компоненти або периферійні пристрої, які підключаються до персонального комп’ютера. До них належать мишки, клавіатури, навушники, зовнішні диски тощо.
10. Веб-розробка — процес створення веб-сайтів і веб-додатків. Охоплює фронтенд (інтерфейс користувача), бекенд (серверна логіка), бази даних та тестування.

ВСТУП

Сучасний ринок комп'ютерних аксесуарів характеризується динамічним розвитком та постійним зростанням попиту. Це зумовлено як розвитком технологій, так і зміною споживчих звичок, коли все більше людей віддають перевагу онлайн-шопінгу. У таких умовах виникає нагальна потреба у створенні спеціалізованих платформ, які б ефективно об'єднували продавців та покупців на ринку комп'ютерних аксесуарів.

Розробка веб-маркетплейсу для аксесуарів ПК є актуальним завданням, оскільки існуючі рішення часто не відповідають специфічним вимогам цього сегменту ринку. Створювана платформа має забезпечити не лише зручний інтерфейс для користувачів, але й надійну технічну інфраструктуру, здатну обробляти великі обсяги даних.

Особливого значення набуває доступність якісних онлайн-сервісів для широкого кола користувачів. Веб-маркетплейси відіграють дедалі важливішу роль у формуванні нових каналів взаємодії між бізнесом та кінцевими споживачами. Вони дозволяють значно спростити процес покупки, забезпечити персоналізований підхід до клієнтів, і відкриває нові можливості для подальшого розвитку електронної комерції в цій галузі.

Ринок комп'ютерних аксесуарів має свою специфіку: велика різноманітність товарів, швидке оновлення асортименту, висока конкуренція та постійні зміни в технологіях. Це створює додаткові вимоги до функціоналу маркетплейсу, включаючи необхідність швидкого оновлення каталогу, ефективної системи пошуку та фільтрації. Економічна доцільність проекту підтверджується постійним зростанням ринку комп'ютерних аксесуарів та збільшенням частки онлайн-продажів у загальному обсязі торгівлі.

Для реалізації проекту було обрано сучасний стек технологій, що включає Node.js як серверну платформу та MySQL як систему управління базами даних. Такий вибір обумовлений низкою переваг: висока продуктивність, масштабованість, надійність та відкрита архітектура. Node.js дозволяє створювати

швидкі та ефективні веб-додатки завдяки своїй подієво-орієнтованій архітектурі, а MySQL забезпечує надійне зберігання та швидкий доступ до структурованих даних.

Проектування системи почалося з детального аналізу вимог та розробки архітектури. Було враховано необхідність забезпечення високої продуктивності, безпеки даних та масштабованості системи. Особливу увагу приділено розробці модулів управління товарами, системі пошуку та фільтрації, а також механізмам безпечної автентифікації користувачів.

Процес розробки включав реалізацію ключових функцій маркетплейсу: реєстрацію та автентифікацію користувачів, управління каталогом товарів, функціонал кошика покупок. Для забезпечення якості програмного забезпечення було впроваджено комплексну систему тестування та валідації даних.

Важливим аспектом розробки стало забезпечення безпеки системи. Були реалізовані механізми захисту від поширених веб-вразливостей, шифрування конфіденційних даних та безпечного зберігання паролів. Особливу увагу приділено захисту персональних даних користувачів та валідації вхідних даних.

Для впровадження системи планується використання надійного хостингу, що дозволить забезпечити стабільну роботу платформи. Процес розгортання включає налаштування серверного середовища, конфігурацію бази даних та оптимізацію продуктивності системи.

Розробка веб-маркетплейсу аксесуарів для ПК є комплексним завданням, що поєднує в собі аспекти веб-розробки, безпеки даних та електронної комерції. Економічна доцільність проекту обумовлена постійним зростанням ринку комп'ютерних аксесуарів та збільшенням частки онлайн-продажів у загальному обсязі торгівлі. Створена платформа дозволить виробникам та постачальникам ефективно досягати цільової аудиторії, зменшити операційні витрати та оптимізувати процеси продажу, а для покупців забезпечить зручний доступ до широкого асортименту товарів, можливість порівняння цін та технічних характеристик.

1 РОЗРОБКА ПРОГРАМНОЇ СИСТЕМИ

1.1 Аналіз вимог до програмної системи

1.1.1 Функціональні вимоги:

Управління користувачами:

Система повинна охоплювати весь процес взаємодії з користувачами. Кожен може створити персональний обліковий запис шляхом реєстрації, вказавши базову інформацію: ім’я, email та пароль. Після завершення реєстрації користувачу відкривається доступ до кошику, де він може переглядати обрані товари.

Каталог товарів:

Платформа забезпечує зручний пошук товарів за назвою або обраною категорією. Користувач може скористатися рядком пошуку для введення ключових слів або фільтрами для вибору потрібної категорії, що дозволяє швидко знаходити товари, які відповідають критеріям пошуку. Такий підхід покращує навігацію по каталогу та підвищує зручність взаємодії з платформою.

Корзина покупок:

У систему інтегровано модуль кошика покупок, який дозволяє зберігати обрані товари перед завершенням покупки. Користувач має можливість додавати до кошика товари, змінювати їх кількість, видаляти непотрібні позиції, а також переглядати загальну вартість замовлення. Дані кошика зберігаються між сесіями, тому навіть після виходу з акаунту вибрані товари залишаються доступними.

1.1.2 Нефункціональні вимоги

Захист даних є одним із ключових аспектів роботи системи. Паролі користувачів зберігаються у вигляді хешів, що гарантує їхню безпеку від несанкціонованого доступу. Реалізовано механізми протидії CSRF-атакам та іншим

типовим загрозам веб-додатків. Уся вхідна інформація проходить перевірку, щоб уникнути ін’єкційних атак та інших потенційно небезпечних дій.

Продуктивність:

Система спроєктована з урахуванням високої продуктивності та раціонального використання ресурсів. Зображення товарів обробляються таким чином, щоб забезпечити їхнє швидке завантаження. Для зниження навантаження на сервер застосовується механізм кешування. Структура бази даних налаштована для оперативного пошуку та ефективного доступу до інформації.

Масштабованість:

Архітектура системи розроблена з урахуванням можливості подальшого розширення функціоналу. Завдяки модульному підходу можна інтегрувати нові елементи без порушення роботи вже реалізованих функцій. Платформа підтримує різноманітні категорії товарів і може бути налаштована відповідно до специфіки різних бізнес-потреб.

1.1.3 Технічні вимоги

Технологічний стек:

Система реалізована з використанням актуального технологічного стеку. Серверна частина створена на базі Node.js у поєднанні з Express.js, що дозволяє ефективно обробляти запити та взаємодіяти з даними. Для рендерингу інтерфейсу використовується EJS, а сучасний та адаптивний дизайн реалізовано за допомогою фреймворку Bootstrap. Як система керування даними застосовується MongoDB, яка забезпечує гнучке збереження та обробку інформації.

1.1.4 Аналіз предметної області

Сфера комп’ютерних аксесуарів є активно зростаючим напрямом у галузі електронної комерції, що вирізняється значною конкуренцією та постійним оновленням товарного ряду. Основні типи аксесуарів для ПК включають: периферійне обладнання (клавіатури, миші, монітори), аудіотехніку (колонки, гарнітури), засоби для зберігання даних (флешки, зовнішні HDD), мережеві пристрої (мережеві адаптери, маршрутизатори), а також різні зручні доповнення для роботи (сумки, охолоджуючі платформи, підставки для ноутбуків).

Сучасний ринок комп'ютерних аксесуарів характеризується низкою особливостей, які впливають на розробку веб-маркетплейсу:

1. Швидке оновлення асортименту та поява нових продуктів, що вимагає гнучкої системи управління каталогом товарів.
2. Велика різноманітність товарів з різними технічними характеристиками, що необхідно враховувати при розробці системи пошуку та фільтрації.
3. Висока конкуренція між продавцями, що обумовлює необхідність реалізації ефективних механізмів порівняння цін та характеристик товарів.
4. Специфічні вимоги до сумісності аксесуарів з різними моделями комп'ютерів, що вимагає розробки спеціальних інструментів перевірки сумісності.
5. Необхідність забезпечення безпеки транзакцій та захисту персональних даних користувачів.

Проведений аналіз поточних рішень на ринку свідчить про наявність ряду обмежень у більшості маркетплейсів. Механізми пошуку та фільтрації, як правило, не враховують специфічні особливості комп’ютерних аксесуарів, що створює труднощі для користувачів при виборі продукції. Продавці мають обмежені можливості для повного опису технічних характеристик товарів, а відсутність функціоналу перевірки сумісності аксесуарів із конкретними моделями ПК

ускладнює процес прийняття рішень споживачами. До того ж, існуючі платформи часто не забезпечують достатньої деталізації категорій і пропонують обмежені інструменти для порівняння, що знижує зручність та ефективність покупок.

Ці особливості та недоліки поточних рішень підкреслюють потребу у створенні спеціалізованого веб-маркетплейсу, що відповідатиме вимогам ринку комп'ютерних аксесуарів і сприятиме ефективній комунікації між продавцями та споживачами.

1.1.5 Постановка задачі

Враховуючи результати аналізу предметної області та специфіку ринку комп'ютерних аксесуарів, головною метою даної дипломної роботи є створення веб-маркетплейсу, що забезпечить ефективну взаємодію між покупцями та продавцями у цій сфері.

Для досягнення поставленої мети необхідно вирішити наступні завдання:

1. Спроєктувати архітектуру веб-маркетплейсу з використанням технологій Node.js та MySQL, яка гарантуватиме надійну продуктивність і високий рівень безпеки.
2. Створити структуру бази даних для зберігання відомостей про товари, категорії, користувачів, вміст кошика.
3. Реалізувати механізм автентифікації та авторизації з підтримкою різних ролей користувачів (адміністратор, звичайний користувач).
4. Побудувати систему керування каталогом товарів із можливістю додавання характеристик і зображень.
5. Реалізувати функціонал кошика, що дозволяє додавати, видаляти та змінювати кількість товарів.
6. Забезпечити захист системи від CSRF-атак і реалізувати безпечне зберігання персональних даних користувачів.

7. Реалізувати систему для зручного пошуку користувачем товару за назвою.
8. Розробити інтерфейс користувача з адаптивним дизайном для зручного перегляду товарів і здійснення покупок.

У результаті реалізації поставлених завдань буде розроблено повноцінний веб-маркетплейс, який забезпечить зручний та ефективний процес купівлі й продажу комп’ютерних аксесуарів, з урахуванням особливостей даного ринку та потреб як покупців, так і продавців.

1.1.6 Пошук актантів та варіантів використання

Аналіз функціональних вимог до веб-маркетплейсу комп’ютерних аксесуарів дав змогу визначити основних учасників системи та характер їхньої взаємодії. У межах системи виокремлено два головні типи користувачів: Гість (тобто неавторизований відвідувач) та Зареєстрований користувач. Кожен з них має власний набір функцій і можливостей, які визначають спосіб взаємодії з платформою.

Гість, як основний користувач без авторизації, має доступ до базових можливостей системи, зокрема перегляду каталогу товарів та використання функцій пошуку й фільтрації за категоріями. Крім того, він може додавати обрані товари до кошика. Для отримання розширеного функціоналу Гість може пройти реєстрацію або увійти до системи за допомогою наявного облікового запису.

Зареєстрований користувач, на додаток до всіх функцій, доступних Гостю, отримує розширені можливості для повної взаємодії з платформою. Зокрема, він може керувати своїм профілем, змінювати кількість товарів у кошику. Система також зберігає історію покупок і дозволяє редагувати особисту інформацію. Основні сценарії використання системи включають такі випадки:

- Перегляд та пошук товарів, що включає функції фільтрації та пошуку

- Реєстрація та авторизація користувачів
- Управління кошиком покупок
- Використання функції збереження зацікавлених товарів в кошику, які зберігаються в базі даних

Кожен сценарій використання системи супроводжується чітко визначеними прецедентами, які включають основні та альтернативні шляхи виконання дій, що дає змогу повноцінно зрозуміти логіку взаємодії користувача з платформою. Така організація акторів і сценаріїв сприяє зручній навігації на сайті та забезпечує ефективну взаємодію між користувачами й системою.

1.1.7 Опис ключових варіантів використання

Діаграма варіантів використання є ключовим засобом для наочного відображення взаємодії користувачів із системою. Вона дозволяє чітко окреслити основні функціональні вимоги та забезпечити відповідність функціоналу очікуванням кінцевих користувачів. На такій діаграмі зображено три основні складові: актори (тобто користувачі системи), варіанти використання (доступні дії) та зв’язки між ними, які демонструють характер їх взаємодії.

У рамках веб-маркетплейсу, орієнтованого на комп’ютерні аксесуари, визначено два ключові типи користувачів (акторів):

Гість: неавторизований користувач, який має доступ до базового функціоналу

Зареєстрований користувач: авторизований користувач з розширеними можливостями

Діаграма варіантів використання для розуміння функціональності системи представлена на рисунку (див. рис. 1.1).

Рисунок 1.1.7.1 – Діаграма варіантів використання

Опис типових сценаріїв використання:

Перегляд і пошук товарів: неавторизований користувач (Гість) може ознайомлюватися з асортиментом товарів через каталог, використовуючи інструменти фільтрації та пошуку за категоріями. Система виводить доступні позиції разом із детальною інформацією — описом, вартістю та технічними характеристиками. Основні дії включають: введення ключових слів для пошуку, застосування відповідних фільтрів і перегляд результатів.

Реєстрація та вхід у систему: Гість має можливість зареєструватися, ввівши необхідні дані, або авторизуватися за допомогою вже створеного облікового запису. Після успішного входу користувачу відкривається доступ до додаткових функцій платформи. Основні дії включають: заповнення реєстраційної форми, введення логіна й пароля, підтвердження входу в систему.

Керування кошиком: Зареєстрований користувач має можливість додавати обрані товари до кошика, редагувати їх кількість або вилучати зайві позиції. Система в автоматичному режимі оновлює загальну суму замовлення. Основні дії: додавання продукту, зміна кількості одиниць, видалення товарів та перегляд вмісту кошика.

Така структура варіантів використання забезпечує зручну навігацію по сайту, поступове розширення можливостей після реєстрації та логічну послідовність дій при взаємодії з системою.

1.1.8 Моделювання словника системи

У ході розробки веб-маркетплейсу аксесуарів для ПК було сформовано термінологічний словник системи, який охоплює основні поняття та їхні визначення. Цей словник описує ключові сутності та процеси, що реалізуються в межах платформи, і забезпечує спільне розуміння функціональних елементів маркетплейсу всіма учасниками проєкту.

Словник сформовано на основі аналізу предметної області, системних вимог і архітектури. Основна увага приділена взаємозв’язкам між сутностями та їх ролі в системі. Структура словника охоплює технічні та бізнес-аспекти роботи маркетплейсу.

1. Користувач (User) - сутність, що представляє зареєстрованого учасника системи. Може виступати в ролі адміністратора, продавця або покупця. Користувач має унікальний обліковий запис з електронною поштою та паролем.
2. Товар (Product) - сутність, що представляє аксесуар для ПК, доступний для продажу на маркетплейсі. Містить інформацію про назву, опис, ціну та кількість на складі. Кожен товар належить до певної категорії та має власника-продавця.

3. Категорія (Category) - сутність, що визначає класифікацію товарів на маркетплейсі. Категорія містить назву та опис, що допомагає користувачам орієнтуватися серед доступних товарів.
4. Кошик (Cart) - віртуальний контейнер, пов'язаний з обліковим записом користувача, який зберігає товари, відібрані для покупки. Дозволяє користувачу зберігати товари.
5. Продавець (Seller) - роль користувача, що має можливість додавати та керувати власними товарами на маркетплейсі.
6. Покупець (Buyer) - роль користувача, що має можливість переглядати товари, додавати їх до кошика.
7. Сесія (Session) - тимчасовий стан взаємодії користувача з системою, що зберігає інформацію про авторизованого користувача та його поточні дії.
8. Аутентифікація (Authentication) - процес перевірки особистості користувача при вході в систему за допомогою електронної пошти та пароля.
9. Авторизація (Authorization) - процес перевірки прав доступу користувача до певних ресурсів системи відповідно до його ролі.
10. Маршрутизація (Routing) - механізм визначення обробника для кожного HTTP-запиту в системі, що забезпечує правильну обробку запитів користувачів.

Цей словник містить ключові поняття та терміни, що застосовуються в системі веб-маркетплейсу комп’ютерних аксесуарів, і слугує засобом формування спільного розуміння системи серед усіх учасників її розробки та використання.

1.2 Проектування програмної системи

Проектування програмної системи є критично важливим етапом розробки, який визначає архітектуру, структуру та взаємодію компонентів системи. У даному

розділі розглядається детальне проектування інтернет-магазину, який забезпечує повний цикл електронної комерції.

1.2.1 Вибір процесу розробки

У цьому розділі розглянуто процес проєктування та реалізації веб-маркетплейсу, орієнтованого на продаж комп’ютерних аксесуарів. Основною метою стало створення зручного та ефективного інструменту, який би задовольняв потреби онлайн-користувачів. У підпункті подано загальний опис підходів, що були застосовані під час розробки, а також методологій, використаних для досягнення зазначеної мети.

Розробка розпочалася з етапу аналізу та формування вимог. Було здійснено вивчення аналогічних платформ у сфері комп’ютерних аксесуарів, на основі чого сформульовано як функціональні, так і нефункціональні потреби системи. Цей етап став базою для подальших рішень, оскільки дав чітке уявлення про очікування як покупців, так і продавців.

Для організації процесу розробки було обрано гнучкий підхід — методологію Agile з впровадженням Scrum. Така модель дозволяє швидко реагувати на змінні умови ринку, адаптуючи функціональність до актуальних запитів. В умовах постійної зміни асортименту та потреб клієнтів цей підхід виявився найбільш доцільним. Регулярне тестування та збір відгуків сприяли оперативному виявленню проблем і підтримці стабільної якості системи.

На етапі архітектурного проєктування було визначено загальну логіку побудови системи. Основну увагу приділено продуктивності, безпеці та масштабованості, які було закладено в архітектурну основу. Було обрано мікросервісний стиль, що дозволяє окремим компонентам працювати автономно та взаємодіяти через чітко визначені протоколи.

Компонентне проєктування включало розробку користувацького інтерфейсу, API, структури бази даних і модулів автентифікації. Було важливо визначити, як

усі елементи системи обмінюються даними, враховуючи такі протоколи, як HTTP/HTTPS та формати JSON. Окрему увагу приділено захисту даних, використанню шифрування, багатофакторній автентифікації, а також забезпеченню стабільної роботи під високим навантаженням.

База даних є одним із ключових елементів архітектури. Для проєкту було обрано MySQL — перевірене рішення з високим рівнем надійності, здатне масштабуватися й підтримувати транзакції. Як реляційна система, MySQL дозволила ефективно організувати дані, пов’язані з товарами, клієнтами, замовленнями та іншими сутностями. Це забезпечує оперативний доступ до інформації, що є критичним для e-commerce-платформи.

Клієнтська частина (frontend) була реалізована із застосуванням шаблонізатора EJS, який дозволяє розділити логіку й візуальне представлення сторінок. Завдяки цьому спростилася подальша підтримка та розширення інтерфейсу. Серверна частина (backend) базується на Node.js і Express.js, що забезпечує ефективну обробку запитів, автентифікацію користувачів, керування даними й взаємодію з базою.

Інтеграція окремих частин системи здійснювалася поступово, з послідовним впровадженням і тестуванням. Щоб гарантувати стабільність і надійність роботи, застосовувалися як автоматизовані, так і ручні методи перевірки. Особлива увага приділялася перевірці роботи ключових функцій — процесу оформлення замовлення, оплати та управління товарним каталогом.

Важливим аспектом розробки маркетплейсу є забезпечення безпеки та надійності системи. Для цього було реалізовано систему автентифікації та авторизації користувачів з використанням Passport.js, що забезпечує надійний захист даних користувачів та продавців. Особливу увагу приділено захисту платіжної інформації та персональних даних, що відповідає сучасним стандартам безпеки в електронній комерції.

Система логування та моніторингу дозволяє оперативно виявляти та реагувати на потенційні загрози безпеці. Крім того, було реалізовано механізми резервного копіювання даних та відновлення системи у разі виникнення непередбачених ситуацій, що забезпечує високу надійність роботи платформи.

Таким чином, обрана методологія Agile/Scrum у поєднанні з сучасним стеком технологій (Node.js, Express.js, MySQL, EJS) дозволяє ефективно управляти розробкою веб-маркетплейсу аксесуарів для ПК, забезпечуючи гнучкість, якість та відповідність вимогам замовника.

Регулярні ітерації та зворотній зв'язок дозволяють швидко адаптуватися до змін та забезпечувати високу якість продукту. Структурований підхід до розробки в поєднанні з сучасними інструментами дозволяє команді ефективно працювати над створенням інноваційного рішення для електронної комерції в сфері аксесуарів для ПК.

1.2.2 Побудова схеми бази даних

Під час розробки бази даних для веб-маркетплейсу комп’ютерних аксесуарів структура таблиць формувалась відповідно до функціональних вимог системи та її основних задач. Нижче наведено стислий опис кожної таблиці та її ролі в контексті роботи платформи.

Схема products містить в собі наступні поля:

- product_id: Унікальний ідентифікатор товару
- name: Назва товару
- description: Детальний опис товару
- price: Ціна товару
- category_id: Посилання на категорію товару
- seller_id: Посилання на продавця
- stock: Кількість товару на складі
- images: Масив URL-адрес зображень товару
- specifications: Технічні характеристики товару
- rating: Середня оцінка товару
- reviews_count: Кількість відгуків
- status: Статус товару (активний, неактивний)

- created_at: Дата додавання товару
- updated_at: Дата останнього оновлення

Таблиця products призначена для зберігання інформації про товари, що продаються на маркетплейсі. Вона дозволяє відображати детальну інформацію про кожен товар, включаючи технічні характеристики, зображення та відгуки покупців.

Кожен запис у цій таблиці представляє окремий товар на платформі. Поля товару включають базову інформацію (назва, опис, ціна), технічні характеристики, зображення, рейтинг та статус. Кожен товар пов'язаний з категорією через поле category_id та з продавцем через поле seller_id.

Схема categories:

- category_id: Унікальний ідентифікатор категорії
- name: Назва категорії
- description: Опис категорії
- parent_id: Посилання на батьківську категорію
- image: URL зображення категорії
- status: Статус категорії
- created_at: Дата створення категорії
- updated_at: Дата останнього оновлення

Таблиця categories використовується для організації товарів у ієрархічну структуру. Вона дозволяє створювати основні категорії та підкатегорії для зручної навігації по маркетплейсу. Кожен запис у цій таблиці представляє категорію товарів. Поля категорії включають назву, опис, зображення та посилання на батьківську категорію для створення ієрархії. Категорії можуть мати підкатегорії, що дозволяє створювати детальну структуру каталогу товарів.

Схема users:

- user_id: Унікальний ідентифікатор користувача
- email: Електронна пошта користувача
- password: Захешований пароль
- role: Роль користувача (покупець, продавець, адмін)
- first_name: Ім'я користувача
- last_name: Прізвище користувача

- phone: Номер телефону
- address: Адреса доставки
- company_name: Назва компанії (для продавців)
- company_description: Опис компанії (для продавців)
- verification_status: Статус верифікації
- created_at: Дата реєстрації
- status: Статус акаунта

Таблиця users призначена для зберігання даних про всіх учасників платформи — як покупців, так і продавців. Вона забезпечує можливість управління рівнем доступу, визначення ролей та зберігання контактної інформації, необхідної для виконання операцій купівлі-продажу. Кожен запис у таблиці відповідає окремому користувачу. Поля містять особисті та контактні дані, а також додаткову інформацію для продавців, зокрема назву компанії чи її опис. Тип ролі (користувач, продавець або адміністратор) визначає набір функціональних можливостей у межах системи.

Взаємозв'язки між таблицями здійснюються за допомогою зовнішніх ключів (foreign keys). Наприклад, товар посилається на категорію через поле category_id та на продавця через поле seller_id. Це забезпечує цілісність даних та дозволяє ефективно виконувати запити, що включають інформацію з різних таблиць. Така структура бази даних забезпечує ефективну роботу маркетплейсу та зручну навігацію для користувачів.

1.2.3 Побудова UML-діаграми класів

UML-діаграма класів є одним із ключових інструментів моделювання програмного забезпечення, що дозволяє наочно відобразити структуру системи, основні об’єкти, їх атрибути, методи та взаємозв’язки. Використання такої діаграми на етапі проєктування сприяє кращому розумінню архітектури

майбутньої системи, спрощує процес розробки, тестування та подальшої підтримки програмного продукту.

На основі аналізу функціональних вимог до маркетплейсу аксесуарів для ПК була побудована UML-діаграма класів, яка відображає основні сутності системи, їх характеристики та зв’язки між ними. Дана діаграма слугує основою для реалізації об’єктної моделі та визначає логіку взаємодії між різними компонентами програмного забезпечення.

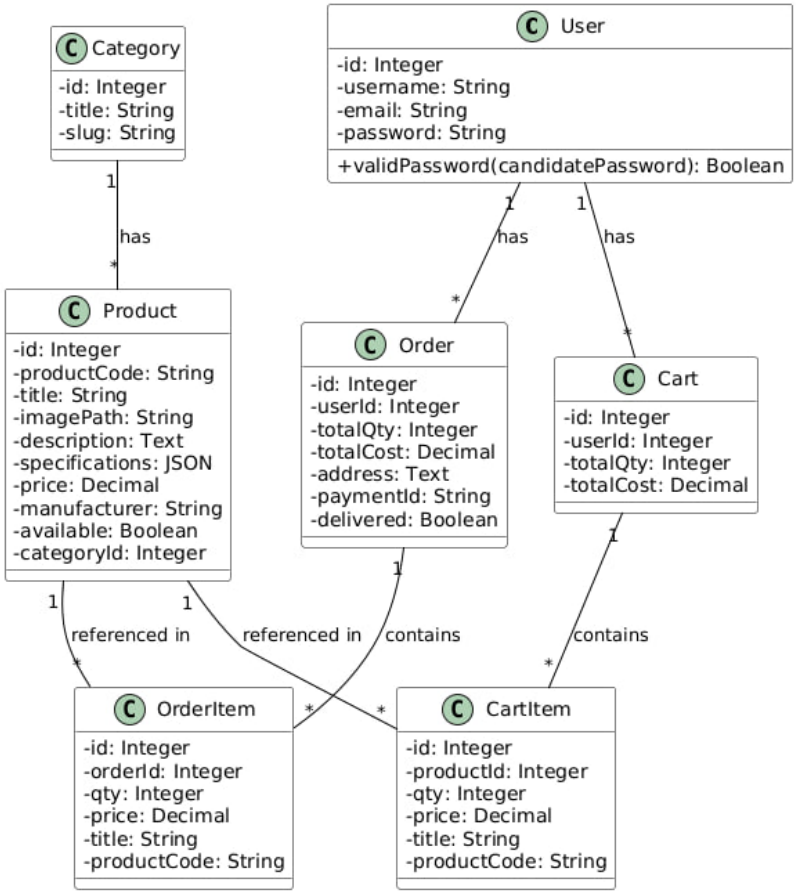

Рисунок 1.2.3.1 – Діаграма класів

Опис класів системи Techno Market:

User (Користувач):

Клас, що представляє користувача системи. Містить основну інформацію для автентифікації та авторизації: унікальне ім'я користувача, електронну пошту та хешований пароль. Має метод validPassword для перевірки правильності пароля. Пов'язаний з кошиком та замовленнями користувача.

Product (Товар):

Клас, що представляє товар в системі. Містить повну інформацію про товар: унікальний код, назву, шлях до зображення, детальний опис, специфікації у форматі JSON, ціну, виробника та статус доступності. Належить до певної категорії та може бути частиною кошику та замовлень.

Category (Категорія):

Клас, що представляє категорію товарів. Містить назву категорії та автоматично генерований slug для URL. Використовується для організації товарів у каталозі. Має зв'язок один-до-багатьох з товарами.

Cart (Кошик):

Клас, що представляє кошик покупок користувача. Містить загальну кількість товарів та загальну вартість. Пов'язаний з користувачем та містить елементи кошика (CartItem). Використовується для тимчасового зберігання товарів перед оформленням замовлення.

CartItem (Елемент кошика):

Клас, що представляє окремий товар у кошику. Містить кількість товару, його ціну, назву та код. Пов'язаний з кошиком та посилається на товар. Використовується для зберігання деталей кожного товару в кошику.

Order (Замовлення):

Клас, що представляє замовлення користувача. Містить загальну інформацію про замовлення: кількість товарів, загальну вартість, адресу доставки, ідентифікатор платежу та статус доставки. Пов'язаний з користувачем та містить елементи замовлення (OrderItem).

OrderItem (Елемент замовлення):

Клас, що представляє окремий товар у замовленні. Містить кількість товару, його ціну, назву та код. Пов'язаний з замовленням та посилається на товар. Використовується для зберігання деталей кожного товару в замовленні.

Використання UML-діаграми класів дозволяє чітко структурувати систему, визначити основні об’єкти та їх взаємодію, що значно спрощує процес розробки, тестування та подальшої підтримки програмного забезпечення. Така візуалізація допомагає швидко зрозуміти логіку побудови системи як розробникам, так і іншим

учасникам проєкту. Крім того, UML-діаграма слугує зручним інструментом для внесення змін та розширення функціоналу в майбутньому.

1.2.4 Моделювання архітектури системи

Архітектура електронного комерційного веб-додатку побудована на основі стеку технологій Node.js та Express.js. Система реалізує архітектуру MVC (Model-View-Controller), що дозволяє ефективно розділити логіку додатку, представлення даних та бізнес-логіку. Такий підхід забезпечує модульність системи та спрощує її підтримку та розширення.

Основний стек технологій:

1. Серверний рівень
   - Express.js як основний веб-фреймворк, що забезпечує швидку обробку HTTP-запитів та маршрутизацію
   - Middleware для обробки запитів, включаючи парсинг JSON, обробку форм та статичних файлів
   - Система маршрутизації, що дозволяє чітко розділити логіку обробки різних типів запитів
   - Управління сесіями через MySQL, що забезпечує надійне зберігання даних сесії
   - Аутентифікація через Passport.js, що надає гнучку систему авторизації користувачів
2. Клієнтський рівень
   - EJS як шаблонізатор, що дозволяє створювати динамічні HTML-сторінки з використанням JavaScript
   - Статичні файли (CSS, JavaScript) для стилізації та інтерактивності інтерфейсу
   - Адаптивний дизайн, що забезпечує коректне відображення на різних пристроях

3. Рівень даних
    - MySQL як основна база даних, що забезпечує надійне зберігання та обробку даних
    - Sequelize як ORM, що спрощує роботу з базою даних та забезпечує безпеку від SQL-ін'єкцій
    - Моделі даних та їх взаємозв'язки, що відображають бізнес-логіку системи

Архітектурні особливості:

Система побудована з урахуванням принципів модульності та розширюваності. Основний додаток (app.js) виконує роль точки входу, налаштовуючи необхідні middleware та маршрути. Особливу увагу приділено безпеці системи через впровадження валідації вхідних даних та безпечного управління сесіями. Система використовує проміжне програмне забезпечення (middleware) для обробки запитів, що дозволяє додавати нову функціональність без зміни основного коду.

Реалізовано багаторівневу систему безпеки, яка включає:

- Аутентифікацію через Passport.js, що забезпечує надійну перевірку особистості користувачів
- Безпечне зберігання сесій в MySQL, що захищає від несанкціонованого доступу
- Валідацію вхідних даних через express-validator, що запобігає введенню некоректних даних
- Захист від XSS та SQL-ін'єкцій через екранування вхідних даних та використання параметризованих запитів
- Безпечне зберігання паролів з використанням bcrypt, що забезпечує їх захист навіть у разі компрометації бази даних

Управління даними:

- Система використовує Sequelize ORM для роботи з базою даних MySQL. Основні моделі даних включають:
- Користувачі (User) - зберігає інформацію про користувачів, включаючи їх облікові дані та налаштування

- Товари (Product) - містить інформацію про товари, включаючи назву, опис, ціну та характеристики
- Категорії (Category) - визначає структуру каталогу товарів
- Кошик (Cart) - зберігає тимчасові дані про вибрані товари

Кожна модель має визначені зв'язки з іншими моделями, що забезпечує цілісність даних та ефективне управління ними. Система підтримує транзакційні операції для забезпечення надійності даних при складних операціях, таких як оновлення залишків товарів.

Система забезпечує зручний інтерфейс для користувачів та ефективне управління всіма аспектами веб-додатку. Особливу увагу приділено зручності використання, що дозволяє користувачам швидко знаходити потрібні товари та додавати їх до кошика.

1.2.5 Маршрутизація

Система використовує Express.js для маршрутизації запитів. Маршрути організовані за функціональними модулями:

Головні сторінки (pages.js) - обробляє запити до головної сторінки, сторінок категорій та сторінок товарів. Реалізує логіку відображення контенту та навігації по сайту.

Користувацькі маршрути (user.js) - обробляє запити, пов'язані з реєстрацією, авторизацією та управлінням профілем користувача. Реалізує логіку аутентифікації та авторизації.

Товари (products.js) - обробляє запити, пов'язані з переглядом, пошуком та фільтрацією товарів. Реалізує логіку роботи з каталогом товарів.

1.3 Конструювання програмної системи

Архітектура веб-маркетплейсу аксесуарів для ПК базується на поділі системи на фронтенд і бекенд, що дозволяє досягти гнучкості, масштабованості та спрощення процесу обслуговування. Інтерфейс користувача реалізований за допомогою бібліотеки React, яка дає змогу створювати інтуїтивно зрозумілий та динамічний інтерфейс. Серверну логіку побудовано на платформі Node.js із використанням Express.js, а для збереження та обробки інформації застосовується реляційна СУБД MySQL.

Фронтенд частина платформи реалізована відповідно до концепції компонентно-орієнтованої архітектури, що є основою роботи React. Інтерфейс складається з набору окремих сторінок і компонентів, кожен з яких відповідає за конкретну частину взаємодії з користувачем. Основні елементи додатку охоплюють такі функціональні блоки:

- Navbar: верхня панель навігації, яка містить логотип, пошук, посилання на основні розділи (каталог, кошик, профіль користувача тощо).
- Sidebar: бічна панель для швидкого доступу до категорій товарів, фільтрів та додаткових функцій.
- ProductList: сторінка зі списком товарів, фільтрами, сортуванням та пагінацією.
- ProductPage: сторінка окремого товару з детальним описом, технічними характеристиками, відгуками та кнопкою "Додати в кошик".
- Cart: сторінка кошика, де відображаються обрані товари, їх кількість, загальна сума.
- UserProfile: сторінка профілю користувача з можливістю перегляду та редагування особистої інформації, зміни паролю, перегляду історії замовлень.
- Login/Register: сторінки для авторизації та реєстрації користувачів.

Для реалізації навігації між різними сторінками застосовується бібліотека react-router-dom, яка забезпечує функціональну багатосторінкову маршрутизацію

без оновлення всієї сторінки. Управління станом у додатку здійснюється за допомогою контекстів, таких як UserContext — для зберігання даних про користувача, і CartContext — для роботи з кошиком. Такий підхід дозволяє ефективно обмінюватися даними між компонентами без дублювання логіки.

Бекенд-сервер виконує ключові функції обробки клієнтських HTTP-запитів, реалізує бізнес-логіку додатку, здійснює автентифікацію й авторизацію користувачів, а також керує такими сутностями, як товари та кошик. Для обміну даними з клієнтською частиною використовується RESTful API, який забезпечує передачу інформації у форматі JSON.

База даних слугує сховищем для всієї важливої інформації, зокрема даних про користувачів, товари, категорії, кошики та інші об’єкти системи. Серверна логіка здійснює взаємодію з базою через SQL-запити, що дозволяє підтримувати цілісність інформації та забезпечує її захищене зберігання.

Обрана архітектура створює сприятливі умови для масштабування функціоналу, підключення зовнішніх сервісів і зручної технічної підтримки. Компонентна модель React сприяє ефективній розробці інтерфейсу, тоді як використання Node.js у поєднанні з Express.js забезпечує продуктивність та гнучкість у реалізації логіки роботи платформи.

У процесі розробки особливу увагу також було приділено модульності та повторному використанню коду, що значно спрощує подальше вдосконалення та підтримку системи. Такий підхід дозволяє гнучко адаптувати платформу під потреби бізнесу, а також швидко реагувати на зворотний зв’язок користувачів, вносячи зміни без ризику порушити роботу інших компонентів системи.

1.3.1 Вибір мови та середовища розробки

Для розробки даного проекту електронної комерції було проведено детальний аналіз сучасних технологій та обрано оптимальний стек, який забезпечує високу продуктивність, безпеку та масштабованість. Як основну мову програмування було обрано Node.js разом з JavaScript. Цей вибір обґрунтований низкою важливих факторів. По-перше, Node.js базується на подійно-орієнтованій архітектурі, що дозволяє ефективно обробляти велику кількість одночасних з'єднань без значного навантаження на сервер. По-друге, платформа має велику екосистему пакетів через npm, що значно прискорює процес розробки та дозволяє використовувати перевірені рішення для різних завдань. Крім того, JavaScript є універсальною мовою, яка використовується як на сервері, так і на клієнті, що спрощує розробку та підтримку коду.

Для розробки веб-інтерфейсу було обрано фреймворк Express.js, який надає потужний інструментарій для створення REST API та обробки HTTP-запитів. Express.js відомий своєю простотою використання та гнучкістю налаштування. Він надає зручну систему маршрутизації, що дозволяє легко організовувати структуру додатку. Особливу увагу заслуговує система middleware, яка дозволяє розширювати функціональність додатку шляхом додавання проміжних обробників для запитів. Це особливо важливо для реалізації таких функцій, як аутентифікація, логування, обробка помилок та валідація даних.

Як систему управління базами даних було обрано MySQL. Ця реляційна база даних забезпечує надійне зберігання та обробку даних, має високу продуктивність та підтримує транзакції, що критично важливо для електронної комерції. MySQL відома своєю стабільністю та широкою підтримкою в індустрії. Вона надає потужні інструменти для оптимізації запитів, індексації даних та управління транзакціями. Крім того, MySQL має хорошу документацію та велику спільноту розробників, що спрощує вирішення проблем, які можуть виникнути під час розробки.

Середовище розробки було обране Visual Studio Code, яке надає зручний інтерфейс для написання коду. Це середовище розробки є безкоштовним, легким у

використанні та має відмінну підтримку JavaScript/Node.js. Воно надає такі корисні функції, як автодоповнення коду, підсвічування синтаксису, вбудований термінал та інтеграція з Git для контролю версій. Крім того, VS Code має багату екосистему розширень, які можуть значно покращити продуктивність розробки.

Для забезпечення безпеки та аутентифікації користувачів у проекті використовується Passport.js. Ця бібліотека надає зручний інструментарій для реалізації різних стратегій авторизації, включаючи локальну аутентифікацію, OAuth, OpenID та інші. Passport.js дозволяє легко інтегрувати різні методи аутентифікації та забезпечує безпечне зберігання сесій користувачів.

Також у проекті реалізовано захист від CSRF-атак для підвищення безпеки додатку. Це досягається за допомогою генерації унікальних токенів для кожного сеансу користувача та їх перевірки при обробці запитів. Такий підхід захищає від несанкціонованих дій з боку зловмисників.

Загалом, обраний стек технологій забезпечує оптимальний баланс між продуктивністю, безпекою та зручністю розробки. Всі використані технології мають хорошу документацію та активну спільноту розробників, що спрощує підтримку та розвиток проекту в майбутньому. Крім того, цей стек технологій дозволяє легко масштабувати додаток при збільшенні навантаження та додавати нові функції без значних змін в архітектурі.

1.3.2 Вибір СУБД та опис її фізичної моделі

Фізична модель бази даних включає кілька ключових таблиць, які забезпечують зберігання та обробку всіх необхідних даних. Основною є таблиця categories, яка зберігає інформацію про категорії товарів. Кожна категорія має унікальний ідентифікатор, назву, опис та зображення. Для швидкого пошуку категорій за назвою створено індекс idx_name.

Таблиця products зберігає інформацію про товари. Кожен товар має зв'язок з категорією через поле category_id, що дозволяє організувати товари по категоріях.

Для товарів зберігається назва, опис, ціна, зображення та кількість на складі. Створено індекси для швидкого пошуку за категорією, назвою та ціною.

Таблиця users містить інформацію про користувачів системи. Кожен користувач має унікальний email, захешований пароль, ім'я та роль (користувач або адміністратор). Для забезпечення унікальності email-адрес створено унікальний індекс.

Таблиця cart_items призначена для зберігання товарів у кошику користувача. Вона містить зв'язки з таблицями users та products, а також кількість товарів. Це дозволяє зберігати стан кошика між сесіями та забезпечувати зручний процес покупки.

Таблиця search_filters зберігає налаштування фільтрів для пошуку товарів. Вона містить інформацію про вибрані категорії, діапазони цін та інші параметри пошуку. Це дозволяє зберігати налаштування пошуку для кожного користувача та покращує користувацький досвід.

Між таблицями встановлені зв'язки через зовнішні ключі, що забезпечує цілісність даних. Наприклад, поле category_id в таблиці products посилається на id в таблиці categories. Це дозволяє легко отримувати повну інформацію про товари та їх категорії.

Для оптимізації пошуку та фільтрації створено кілька індексів. Індекс idx_category дозволяє швидко знаходити всі товари певної категорії, idx_name - шукати товари за назвою, а idx_price - сортувати та фільтрувати товари за ціною.

1.3.3 Реалізація основних класів та методів

1. Клас User (Користувач)

Клас User є центральним компонентом системи, який відповідає за управління користувачами. Він реалізує основні функції реєстрації та авторизації користувачів. При реєстрації система хешує пароль для безпечного зберігання, а при авторизації перевіряє правильність введених даних. Клас також містить методи для управління профілем користувача, включаючи оновлення особистої інформації та зміну пароля.

```
class User {
    constructor(email, password, name) {
        this.email = email;
        this.password = password;
        this.name = name;
        this.role = 'user';
    }

    async register() {
        const hashedPassword = await bcrypt.hash(this.password, 10);
        const user = await User.create({
            email: this.email,
            password: hashedPassword,
            name: this.name,
            role: this.role
        });
        return user;
    }
}
```

Рисунок 1.3.3.1 – Класс User

2. Клас Product (Товар)

Клас Product відповідає за управління товарами в системі. Він дозволяє додавати нові товари, редагувати існуючі та видаляти їх. Кожен товар має такі атрибути, як назва, опис, ціна, категорія та зображення. Клас також містить методи для пошуку товарів за різними критеріями, такими як категорія, ціна чи назва.

```
class Product {
    constructor(name, description, price, categoryId, image) {
        this.name = name;
        this.description = description;
        this.price = price;
        this.categoryId = categoryId;
        this.image = image;
    }

    async create() {
        const product = await Product.create({
            name: this.name,
            description: this.description,
            price: this.price,
            categoryId: this.categoryId,
            image: this.image
        });
        return product;
    }
}
```

Рисунок 1.3.3.2 – Клас Product

3. Клас Cart (Кошик)

Клас Cart відповідає за управління кошиком користувача. Він дозволяє додавати товари до кошика, змінювати їх кількість та видаляти їх. Клас зберігає інформацію про товари в кошику, їх кількість та загальну вартість. Він також містить методи для розрахунку загальної суми кошика та очищення кошика.

```
class Cart {
    constructor() {
        this.items = [];
        this.totalQty = 0;
        this.totalCost = 0;
    }

    add(product, id) {
        let storedItem = this.items[id];
        if (!storedItem) {
            storedItem = this.items[id] = {
                item: product,
                qty: 0,
                price: 0
            };
        }
        storedItem.qty++;
        storedItem.price = product.price * storedItem.qty;
        this.totalQty++;
        this.totalCost += product.price;
    }
}
```

Рисунок 1.3.3.3 – Клас Cart

4. Клас Category (Категорія)

Клас Category управляє категоріями товарів. Він дозволяє створювати нові категорії, редагувати існуючі та видаляти їх. Кожна категорія має назву, опис та зображення. Клас також містить методи для отримання всіх товарів, що належать до певної категорії.

```
class Category {
    constructor(name, description, image) {
        this.name = name;
        this.description = description;
        this.image = image;
    }

    async create() {
        const category = await Category.create({
            name: this.name,
            description: this.description,
            image: this.image
        });
        return category;
    }
}
```

Рисунок 1.3.3.5 – Клас Category

1.3.4 Розробка серверної частини

Основа серверної частини - це Express.js додаток, який налаштовується через конфігураційний файл app.js. Тут відбувається ініціалізація всіх необхідних middleware, налаштування сесій, підключення до бази даних та конфігурація маршрутизації, можна подивитись на Рис.2.2. Особливу увагу приділено безпеці системи через впровадження CSRF-захисту, валідації вхідних даних та безпечного управління сесіями.

```
const express = require("express");
const app = express();
const session = require("express-session");
const MySQLStore = require('express-mysql-session')(session);
const passport = require("passport");


const sessionStore = new MySQLStore({
  host: process.env.DB_HOST || 'localhost',
  port: 3306,
  user: process.env.DB_USER || 'root',
  password: process.env.DB_PASSWORD || '',
  database: process.env.DB_NAME || 'bags_ecommerce'
});

app.use(
  session({
    secret: process.env.SESSION_SECRET,
    resave: false,
    saveUninitialized: false,
    store: sessionStore,
    cookie: { maxAge: 60 * 1000 * 60 * 3 },
  })
);
```

Рисунок 1.3.4.1 – Налаштування сесій (підключення до бази даних)

Моделі даних та взаємодія з базою даних:

Система використовує Sequelize як ORM для роботи з базою даних MySQL. Моделі даних реалізовані з урахуванням всіх необхідних зв'язків та валідацій на Рис.2.3. Модель користувача включає хешування паролів та валідацію електронної пошти.

```
const User = sequelize.define('User', {
  username: {
    type: DataTypes.STRING,
    allowNull: false,
    unique: true
  },
  email: {
    type: DataTypes.STRING,
    allowNull: false,
    unique: true,
    validate: {
      isEmail: true
    }
  },
  password: {
    type: DataTypes.STRING,
    allowNull: false
  }
}, {
  hooks: {
    beforeCreate: (user) => {
      user.password = bcrypt.hashSync(user.password, bcrypt.genSaltSync(5), null);
    }
  }
});
```

Рисунок 1.3.4.2 – Реалізація необхідних зв'язків та валідацій

Маршрутизація та обробка запитів:

Маршрутизація реалізована через модульну структуру, де кожен функціональний модуль має свій роутер на Рис.2.4. Це дозволяє ефективно організувати код та спростити його підтримку. Наприклад, роутер товарів обробляє всі запити, пов'язані з товарами.

```
router.get("/", async (req, res) => {
  try {
    const products = await Product.findAll({
      include: [{
        model: Category,
        as: 'category'
      }]
    });
    res.json(products);
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
});
```

Рисунок 1.3.4.3 – Функціональний модуль (роутер)

Безпека та валідація:

Система безпеки реалізує багаторівневий підхід, включаючи аутентифікацію через Passport.js, хешування паролів, CSRF-захист та валідацію вхідних даних. Валідація реалізована через express-validator.

```
const userSignUpValidationRules = [
  body('username')
    .trim()
    .isLength({ min: 3 })
    .withMessage('Ім\'я користувача повинно містити мінімум 3 символи'),
  body('email')
    .isEmail()
    .withMessage('Введіть коректну електронну пошту'),
  body('password')
    .isLength({ min: 6 })
    .withMessage('Пароль повинен містити мінімум 6 символів')
];
```

Рисунок 1.3.4.4 – Валідація інформації користувача

1.4. Використання програмної системи

Для використання програмної системи електронної комерції необхідно забезпечити базові вимоги до апаратного та програмного забезпечення. Система працює на будь-якій сучасній операційній системі та підтримується всіма популярними веб-браузерами. Для встановлення необхідно мати Node.js та MySQL, які є основними компонентами для роботи системи.

Після встановлення всіх необхідних компонентів, система готова до використання. Користувачі можуть реєструватися, переглядати каталог товарів, додавати товари до кошика. Система забезпечує безпечне зберігання даних та захист від несанкціонованого доступу.

1.4.1 Розгортання програмної системи та системні вимоги

Було обрано MySQL як основну систему управління базами даних. Цей вибір обґрунтований надійністю, стабільністю та ефективною роботою з великими обсягами даних. MySQL забезпечує швидкий пошук товарів, оптимізовану роботу з індексами та ефективне кешування, що критично важливо для електронної комерції.

Фізична модель бази даних включає кілька ключових таблиць. Основною є таблиця categories, яка зберігає інформацію про категорії товарів. Кожна категорія має унікальний ідентифікатор, назву, опис та зображення. Для швидкого пошуку категорій за назвою створено індекс idx_name.

```
CREATE TABLE categories (
    id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(100) NOT NULL,
    description TEXT,
    image VARCHAR(255),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
    INDEX idx_name (name)
);
```

Рис.1.4.1.1 – Структура створення (категорій)

Таблиця products зберігає інформацію про товари. Кожен товар має зв'язок з категорією через поле category_id, що дозволяє організувати товари по категоріях. Для товарів зберігається назва, опис, ціна, зображення та кількість на складі. Створено індекси для швидкого пошуку за категорією, назвою та ціною.

```
CREATE TABLE products (
    id INT PRIMARY KEY AUTO_INCREMENT,
    category_id INT,
    name VARCHAR(255) NOT NULL,
    description TEXT,
    price DECIMAL(10,2) NOT NULL,
    image VARCHAR(255),
    stock INT DEFAULT 0,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
    FOREIGN KEY (category_id) REFERENCES categories(id),
    INDEX idx_category (category_id),
    INDEX idx_name (name),
    INDEX idx_price (price)
);
```

Рис.1.4.1.2 – Структура створення (Товарів)

Заповнення каталогів та товарів відбувається за допомогою скриптів ініціалізації бази даних. Для категорій використовується скрипт seedDB/categories-seed.js, який створює базові категорії товарів. Для товарів використовується скрипт seedDB/products-seed.js, який додає товари до відповідних категорій. Ці скрипти можуть бути запущені командами:

```
node seedDB/categories-seed.js
node seedDB/products-seed.js
```

Рис.1.4.1.3 – Обновлення категорій та товарів

Між таблицями встановлені зв'язки через зовнішні ключі. Наприклад, поле category_id в таблиці products посилається на id в таблиці categories. Це забезпечує цілісність даних та дозволяє легко отримувати повну інформацію про товари та їх категорії.

Для оптимізації пошуку та фільтрації створено кілька індексів. Індекс idx_category дозволяє швидко знаходити всі товари певної категорії, idx_name - шукати товари за назвою, а idx_price - сортувати та фільтрувати товари за ціною.

1.4.2 Опис типових схем використання системи

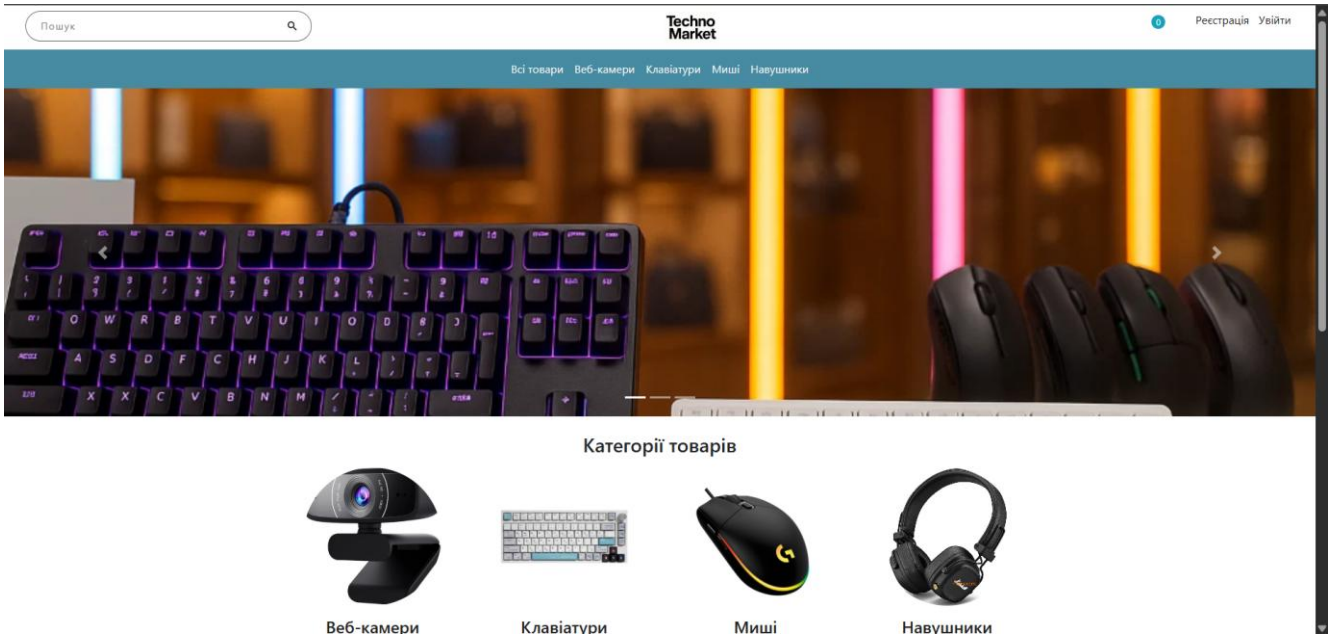

Рис.1.4.2.1 – Голосна сторінка веб-сайту

У даному розділі розглядаються типові сценарії використання веб-маркетплейсу аксесуарів для ПК, які відображають основні бізнес-процеси та взаємодію користувачів з системою.

1. Реєстрація та авторизація:
   - Користувач переходить на головну сторінку маркетплейсу
   - Натискає кнопку "Реєстрація" в верхньому правому куті
   - Заповнює форму реєстрації, вказуючи:
     - Електронну пошту
     - Пароль
     - Підтвердження пароля
   - Після успішної реєстрації виконує вхід в систему
   - При наступних відвідуваннях використовує форму авторизації

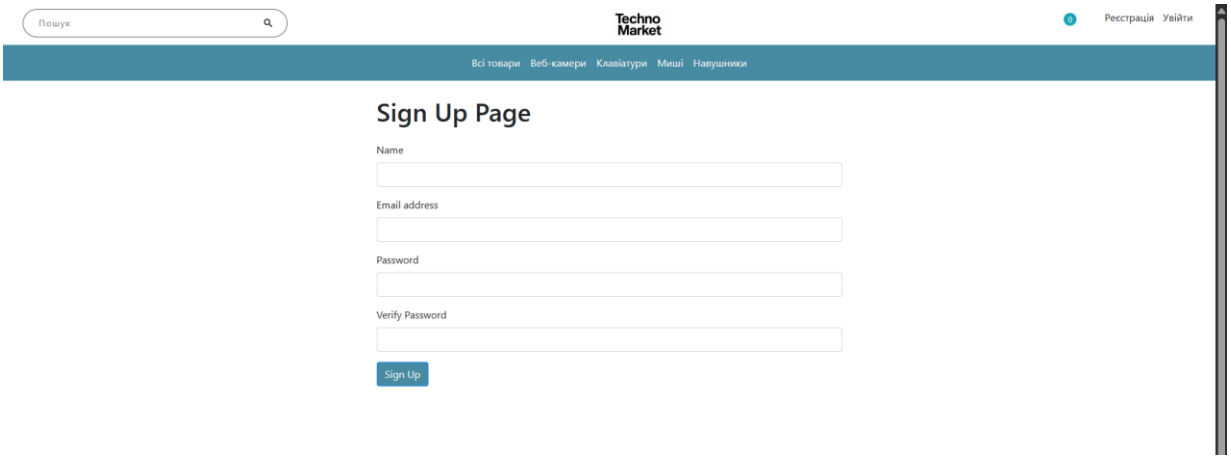

Рис. 1.4.2.2 – Реєстрація та авторизація

2. Перегляд каталогу товарів:
   - Користувач переходить на головну сторінку
   - Бачить категорії товарів у верхній навігаційній панелі
   - Обирає потрібну категорію (наприклад, "Клавіатури", "Миші", "Навушники")
   - Переглядає список товарів у вибраній категорії
   - Може перемикатися між категоріями для пошуку потрібного товару
   - При кліку на товар переходить до його детального опису

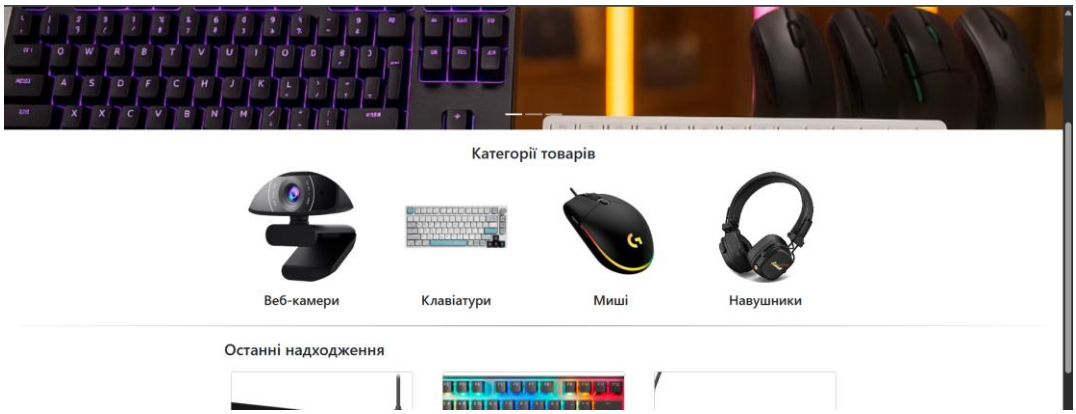

Рис. 1.4.2.3 – Каталог товарів

3. Пошук товарів за назвою:
   - Користувач знаходить поле пошуку в верхній частині сторінки
   - Вводить назву шуканого товару
   - Натискає кнопку пошуку або клавішу «Enter»
   - Система відображає список товарів, що відповідають пошуковому запиту

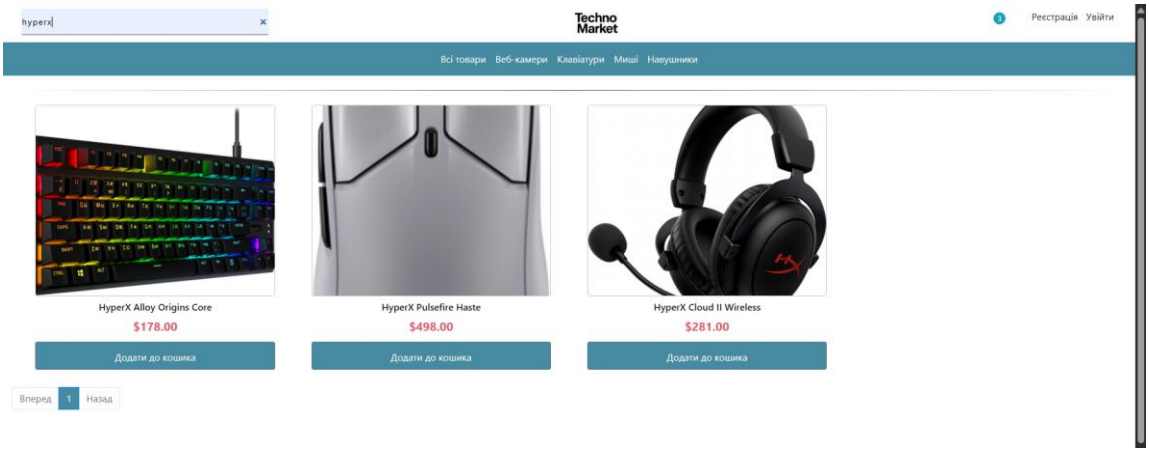

Рис. 1.4.2.4 – Пошук товарів

4. Додавання товарів до кошика:
   - Користувач переглядає детальний опис товару
   - Натискає кнопку "Додати до кошика"
   - Система додає товар до кошика
   - З'являється повідомлення про успішне додавання
   - Може продовжити покупки, додаючи інші товари

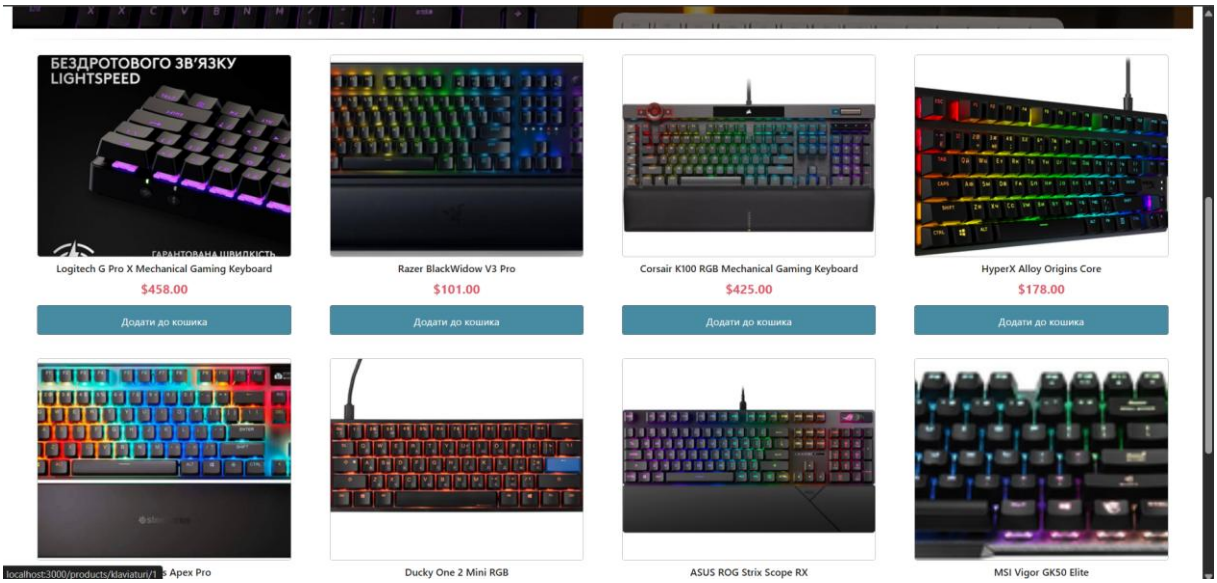

Рис. 1.4.2.5 – Додавання товарів до кошика

5. Перегляд детальної інформації про товар:
   - Користувач відкриває картку товару
   - Бачить детальний опис товару
   - Переглядає технічні характеристики та ціну
   - Обирає кількість товару для покупки

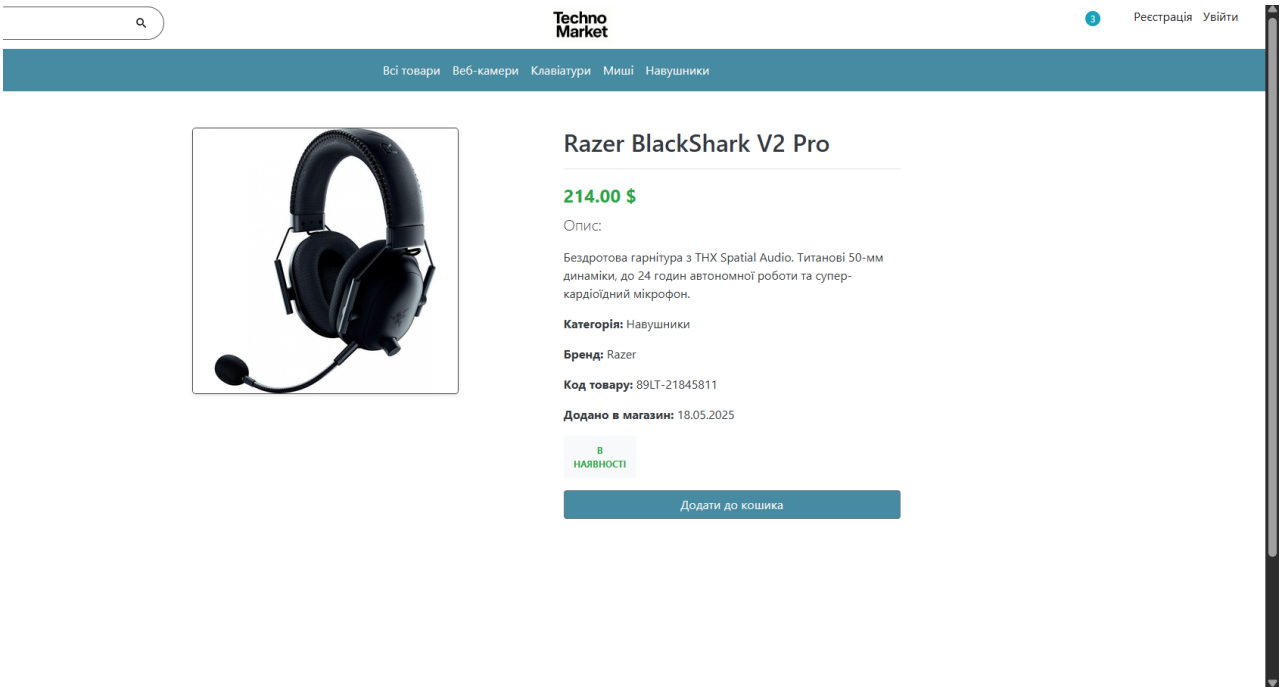

Рис. 1.4.2.6 – Перегляд детальної інформації

6. Управління кошиком:
    - Користувач бачить список доданих товарів
    - Може змінити кількість товарів
    - Може видалити товари з кошика
    - Переглядає загальну суму замовлення
    - Може продовжити покупки

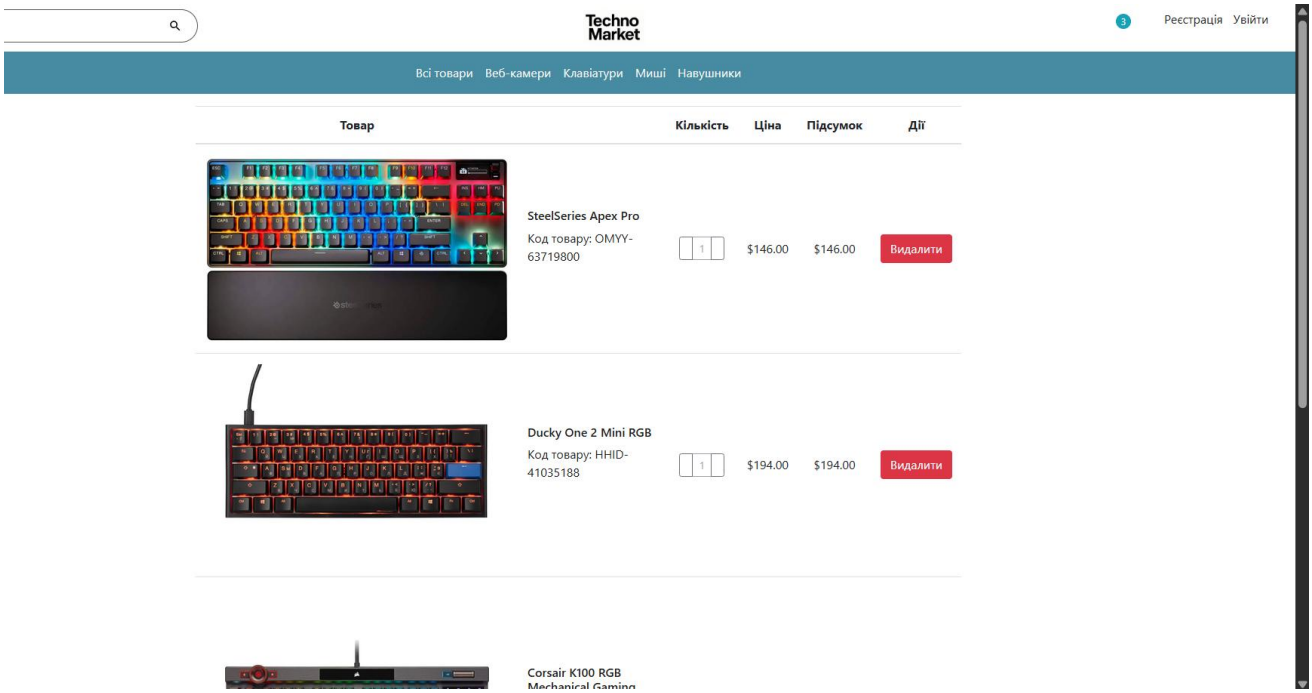

Рис. 1.4.2.7 – Управління кошиком

Розглянуті типові сценарії використання системи демонструють повний цикл взаємодії користувача з веб-маркетплейсом аксесуарів для ПК. Система забезпечує послідовний процес з чітко визначеними етапами та можливостями для користувача. Кожен етап взаємодії має свою логіку та функціональність, що в сукупності створює цілісну систему для ефективної роботи з аксесуарами для ПК.

1.4.3 Верифікація програмної системи

В процесі розробки веб-системи було проведено базову перевірку її функціональності та відповідності основним вимогам. Тестування проводилося локально на персональному комп'ютері з операційною системою Windows 11.

Основна перевірка включала тестування базових функцій системи:

- Коректність відображення веб-інтерфейсу
- Роботу форм введення даних
- Взаємодію з базою даних
- Базову валідацію введених даних

Система була перевірена на наявність критичних помилок та коректність відображення інтерфейсу в різних браузерах. Перевірка показала, що основні функції системи працюють коректно в локальному середовищі.

На даному етапі розробки система не проходила повномасштабного тестування на реальному хостингу, оскільки це виходить за межі поточного етапу розробки. Подальше тестування та впровадження системи на реальний хостинг планується провести після завершення дипломної роботи.

Важливим аспектом верифікації стала перевірка обробки помилкових ситуацій. Система успішно обробляє випадки введення некоректних даних у форми, відсутності з'єднання з базою даних та невалідних запитів. Замість аварійного завершення роботи, система надає зрозумілі повідомлення про помилки в інтерфейсі користувача, що дозволяє швидко виправити ситуацію та продовжити роботу.

В результаті проведеної базової верифікації можна зробити висновок, що система відповідає основним вимогам технічного завдання та готова до подальшого вдосконалення та впровадження.

2 ТЕСТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ

2.1 План тестування

Рис.2.1.1 – Selenium IDE [13]

Тестування програмної системи буде проводитись із використанням розширення Selenium IDE — інструменту для автоматизованого тестування вебінтерфейсів. Selenium IDE є зручним та ефективним рішенням для запису, редагування та виконання тестів без необхідності писати код вручну. Це розширення працює безпосередньо в браузері (наприклад, Google Chrome або Firefox) і дозволяє моделювати дії користувача — натискання кнопок, введення даних, навігацію між сторінками тощо.

Основними перевагами Selenium IDE є простота у використанні, можливість запису тестів у реальному часі, підтримка команд перевірки стану елементів, а також зручне візуальне редагування сценаріїв. Цей інструмент ідеально підходить для швидкого створення функціональних тестів, що забезпечує перевірку працездатності ключових модулів вебсайту в інтерактивному середовищі.

1. Реєстрація та авторизація

Тестування процесу реєстрації та авторизації користувачів включає перевірку створення нового облікового запису, валідацію введених даних та процес входу в систему. При реєстрації перевіряється коректність обробки email-адреси, пароля та інших обов'язкових полів. Система повинна перевіряти унікальність email-адреси та вимоги до складності пароля. При авторизації перевіряється правильність перевірки облікових даних та обробка помилкових спроб входу.

2. Перегляд каталогу товарів

Тестування функціоналу перегляду каталогу включає перевірку відображення категорій товарів, навігації між категоріями та відображення списку товарів. Перевіряється коректність відображення інформації про товари, включаючи назву, опис, ціну та зображення. Також тестується пагінація та фільтрація товарів за різними параметрами, такими як ціна, популярність чи новизна.

3. Пошук товарів

Тестування функції пошуку включає перевірку роботи пошукової системи за різними критеріями. Перевіряється пошук за назвою товару, категорією, ціновим діапазоном та іншими параметрами. Система повинна коректно обробляти пошукові запити, включаючи частикові співпадіння та виправлення помилок у введеному тексті. Також перевіряється відображення результатів пошуку та їх релевантність.

4. Додавання товарів до кошика

Тестування функціоналу кошика включає перевірку процесу додавання товарів, оновлення кількості товарів та видалення товарів з кошика. Перевіряється коректність розрахунку загальної суми замовлення, врахування знижок та акцій. Система повинна коректно обробляти зміну кількості товарів, перевіряти наявність товару на складі та оновлювати загальну суму замовлення в реальному часі.

Кожен з цих функціоналів тестується на різних рівнях:

- Перевірка коректності роботи з базою даних
- Валідація введених даних
- Обробка помилок та граничних випадків
- Перевірка безпеки та захисту даних
- Тестування продуктивності та швидкодії
- Перевірка сумісності з різними браузерами

2.2 Тестування веб сайту Techno Market

Всі тестування проводились в веб середовищі (плагін) Selenium IDE, згідно до плану тестування 2.1, було проведено тестування:

- Авторизація
- Каталог
- Пошук товарів
- Кошик

Тест на (авторизацію) пройдено успішно, було створено від сторони клієнту аккаунт, в самому середовищі Selenium IDE відтворювались дії авторизації такі як введення даних користувача а саме (електрона адреса та пароль) на Рис.2.9.

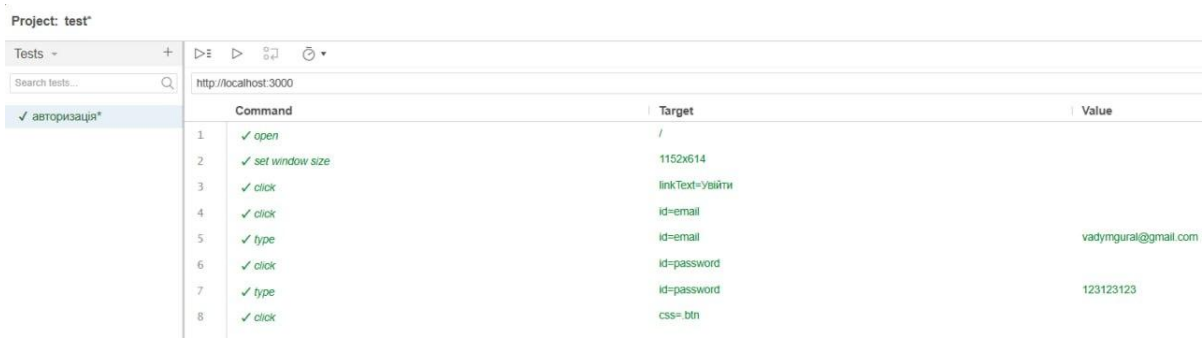

Рис.2.2.1 – Результат тестування авторизації [13]

Тест на (каталог) пройдено успішно, в середовищі Selenium IDE відтворювались події переходу по каталогам товарів на Рис.2.10.

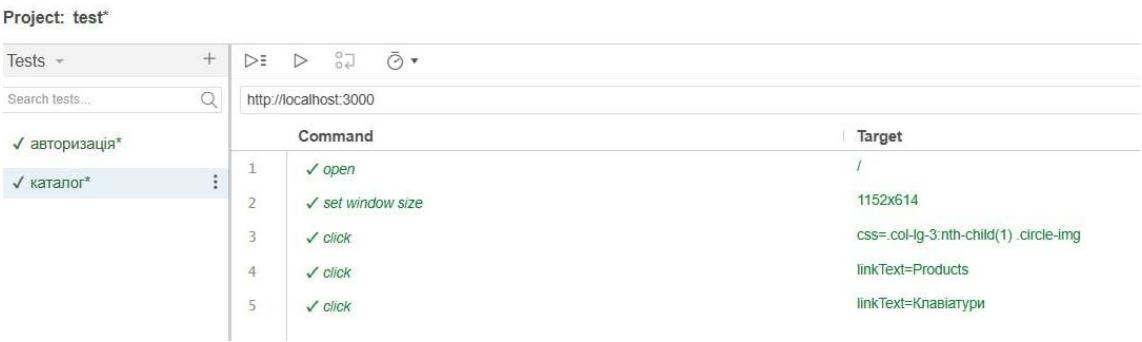

Рис.2.2.2 – Результат тестування каталогу [13]

Тест на (Пошук товарів) пройдено успішно, від сторони клієнту є можливість вводити ім’я або ключові слова товару, проводилось тестування Selenium IDE по іменам товарів такі як Logitech та Razer Рис.2.11.

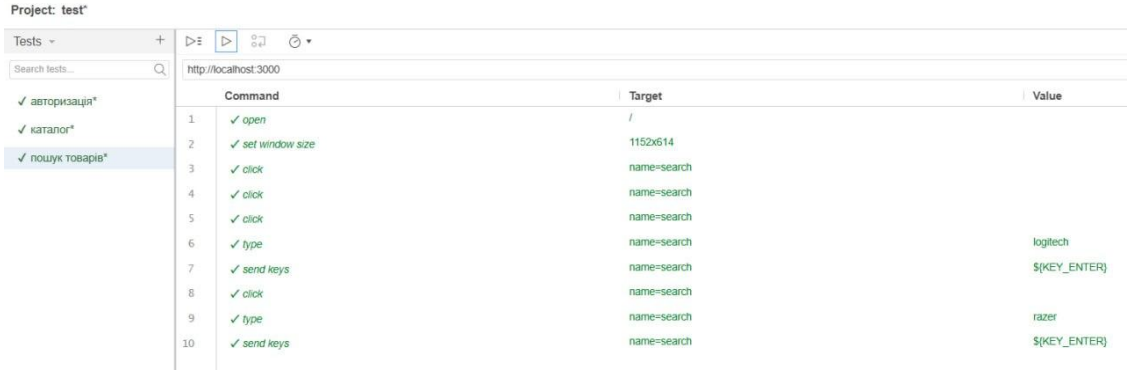

Рис.2.2.3 – Результат тестування пошуку товарів [13]

Тест на (Кошик) пройдено успішно, від сторони клієнта є можливість добавляти товари в кошик різних категорій які доступні на веб-сайті, в середовищі Selenium IDE було відтворені дії додавань товарів в самий кошик, по закінченю тестування видно що все було додано в кошик на Рис.2.12.

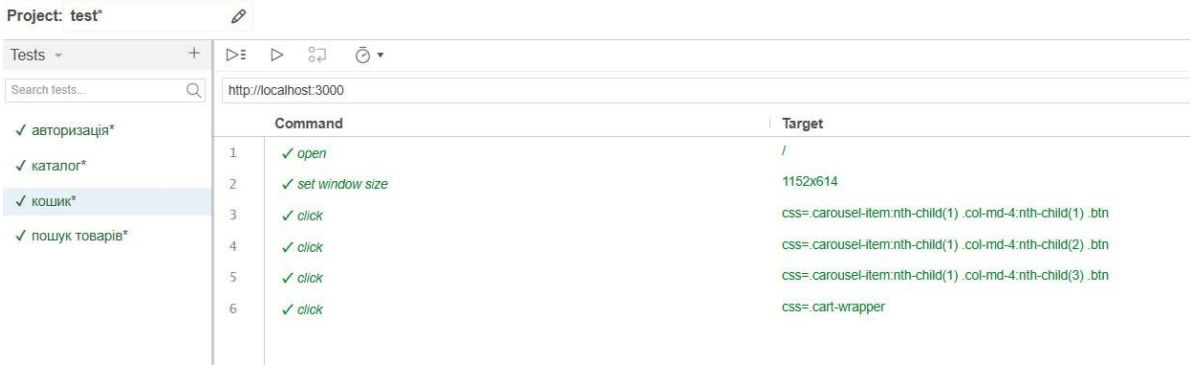

Рис.2.2.4 – Результат тестування кошику [13]

3 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ ТА ОСНОВИ ОХОРОНИ ПРАЦІ

3.1 Ризик як кількісна оцінка небезпек

Небезпека — це сукупність факторів, що здатні створити загрозу життю, здоров'ю або благополуччю людини, об'єктів господарської діяльності та навколишнього середовища. Одним із ключових понять у системі безпеки є ризик, який дозволяє оцінити небезпеку і вжити відповідних заходів безпеки [1, с. 19].

У спеціальній літературі наводяться такі визначення ризику: це міра очікуваної невдачі або неблагополуччя; імовірність людських і матеріальних втрат; діяльність заради досягнення успіху з низькою ймовірністю; усвідомлення небезпеки виникнення події з негативними наслідками; кількісна міра небезпеки, що враховує імовірність наслідків та обсяг втрат [1, с. 28].

Виділяють два основних підходи до трактування ризику: європейський — сприймає ризик як фізичну небезпеку, і американський — як азартний вибір на користь мети попри загрозу. Обидва підходи частково суперечливі, і лише їх поєднання дозволяє розглядати ризик як особливий тип поведінки [1, с. 29].

Оцінка ризику — це процедура, що базується на визначенні ймовірності реалізації небезпеки та оцінці її наслідків. Оцінка може бути:

- кількісною, що спирається на статистичні дані;
- якісною, яка базується на експертних оцінках і суб'єктивних судженнях [1, с. 30].

При якісному підході використовуються описові характеристики: високий, середній, низький рівень ризику, або шкала в балах (67–100 – високий; 34–66 – середній; до 33 – низький) [1, с. 30].

Оцінка ризику у сфері безпеки життєдіяльності може здійснюватися як за кількісним, так і якісним підходом. У межах якісного підходу застосовуються описові характеристики, як-от: високий, середній чи низький рівень ризику. Зустрічається і використання шкали в балах, де значення 67–100 вважаються високим рівнем ризику, 34–66 — середнім, а до 33 — низьким [1, с. 30].

Існує два основні підходи до оцінки ризику: ретроспективний та перспективний. Перший ґрунтується на аналізі попередніх подій і статистичних даних, а другий передбачає прогнозування можливих загроз. Перспективна оцінка є особливо актуальною при реконструкції або модернізації технічних систем, де важливо враховувати потенційні джерела небезпеки ще до їх реалізації [1, с. 30].

Процес оцінки ризику включає такі основні етапи [1, с. 30]:

- Формулювання проблеми — ідентифікація небезпеки, яка може викликати шкідливі наслідки.
- Оцінка експозиції — визначення кількості та типу осіб, що можуть зазнати впливу чинника.
- Аналіз наслідків — опис можливих результатів впливу стресора.
- Характеристика ризику — об'єднання даних, оцінка ймовірності та серйозності наслідків.

Для зниження рівня ризику застосовують комплекс заходів, які умовно поділяють на технічні, організаційні, адміністративні та економічні. Технічні заходи можуть включати оновлення обладнання, автоматизацію процесів, модернізацію конструкцій. Організаційні впровадження регламентів, проведення інструктажів, планування процедур. Адміністративні заходи передбачають видання відповідних наказів, положень, розпоряджень. Економічні охоплюють страхування, систему компенсацій, ліцензування або штрафи [1, с. 32].

У процесі управління ризиком важливу роль відіграє аналіз співвідношення витрат і доцільності заходів. Як підкреслюється у посібнику, "іноді легше усунути менш небезпечну загрозу, яка піддається контролю, ніж боротися зі складною високоризиковою ситуацією" [1, с. 32]. Таким чином, прийняття рішень про втручання має базуватись не лише на рівні ризику, а й на вартості його усунення.

Оцінювання ризику проводиться за допомогою різноманітних методів. До них належать: інженерні (наприклад, побудова дерев подій, аналіз відмов), статистичні (на основі даних про нещасні випадки), модельні (з моделюванням впливу на людину), експертні (залучення фахівців), соціологічні (опитування) та комбіновані методи, які поєднують кілька підходів [1, с. 30].

Ключовим поняттям у системі управління є "прийнятний ризик" — це такий рівень, який не перевищує встановлених нормативами меж, може бути допущений без порушення вимог безпеки [1, с. 33]. Впровадження ризик-орієнтованого підходу в сучасній системі охорони праці означає систематичний аналіз ризиків на всіх етапах діяльності, їхній моніторинг, оцінювання, встановлення допустимих рівнів і застосування заходів зі зниження ризику [1, с. 33].

3.2 Правила техніки безпеки при експлуатації обладнання

Правила техніки безпеки при експлуатації обладнання базуються на чинному законодавстві України. Згідно зі ст. 13 Закону України «Про охорону праці», роботодавець зобов’язаний створити на робочому місці умови праці, які відповідають нормативно-правовим актам з охорони праці, забезпечити функціонування системи управління охороною праці, а також підтримувати технічні засоби у безпечному стані [2, с. 4].

До робіт підвищеної небезпеки належать, зокрема, електрозварювальні, газополум’яні, роботи з вибухонебезпечними речовинами, обслуговування обладнання під тиском, а також роботи в охоронних зонах ліній електропередач. Для таких робіт встановлюються спеціальні вимоги до стану обладнання, порядку проведення інструктажу, рівня підготовки персоналу та наявності засобів індивідуального захисту [3, с. 2–4].

Навчання працівників, які виконують роботи підвищеної небезпеки, проводиться згідно з Типовим положенням про порядок проведення навчання та перевірки знань з питань охорони праці (НПАОП 0.00-4.36-05). Усі працівники перед початком роботи проходять спеціальне навчання та перевірку знань, яка здійснюється комісією, до складу якої входять представники служби охорони праці, керівництва та профспілки. Результати фіксуються в протоколах, які зберігаються не менше 5 років [4, с. 12].

Важливу роль у забезпеченні безпеки відіграє розроблення інструкцій з охорони праці, які є обов’язковими локальними нормативними документами підприємства. Згідно з НПАОП 0.00-4.15-98, інструкції розробляє служба охорони праці або інженер (спеціаліст), який добре обізнаний із технологічними процесами та умовами виконання робіт [5, с. 4].

Під час розроблення інструкцій фахівці керуються:

- положеннями чинного законодавства (Закон України «Про охорону праці»);
- типовими інструкціями для відповідних професій;
- міжгалузевими та галузевими нормативними актами;
- технологічною документацією підприємства [5, с. 5].

Інструкція повинна мати таку структуру: загальні положення; вимоги безпеки перед початком роботи; під час виконання; у разі аварійних ситуацій; після завершення роботи [5, с. 6]. Інструкції затверджуються наказом керівника підприємства, погоджуються з керівниками відповідних структурних підрозділів, службою охорони праці, а в разі потреби — з юридичною службою.

Після затвердження всі інструкції обов’язково реєструються у Журналі реєстрації інструкцій з охорони праці. Працівники зобов’язані ознайомлюватися з ними під час проходження вступного, первинного, повторного, позапланового чи цільового інструктажу, що фіксується в Журналі реєстрації інструктажів з питань охорони праці [5, с. 5].

Інструкції підлягають періодичному перегляду не рідше одного разу на п’ять років, а також у разі змін законодавства, впровадження нових технологій або змін умов праці [5, с. 6].

Також підприємства можуть вести графік перегляду інструкцій, журнал видачі інструкцій підрозділам, а також журнал перевірки знань з охорони праці. Ведення цієї документації забезпечує системний контроль за станом охорони праці, підвищує рівень підготовки персоналу і є юридичним доказом виконання обов’язків роботодавцем у разі перевірок або нещасних випадків [5, с. 6].

ВИСНОВКИ

У процесі виконання дипломної роботи було розроблено веб-маркетплейс для комп'ютерних аксесуарів, який відповідає сучасним вимогам до електронної комерції. Основним завданням було створення надійної, безпечної та зручної платформи для взаємодії продавців та покупців на ринку комп'ютерних аксесуарів.

Архітектура системи побудована на основі сучасного стеку технологій, включаючи Node.js, Express.js та MySQL. Це дозволило створити масштабовану та продуктивну систему, здатну обробляти велику кількість запитів та зберігати значні обсяги даних. Модульна структура проекту забезпечує легкість підтримки та розширення функціоналу.

Безпека системи забезпечується комплексом заходів, включаючи надійну систему аутентифікації, захист від CSRF-атак та валідацію вхідних даних. Особливу увагу приділено захисту персональних даних користувачів та безпеці форм введення інформації.

Тестування системи проводилося на локальному хості з використанням Selenium IDE, що дозволило автоматизувати процес перевірки функціональності та забезпечити стабільність роботи системи. Було проведено комплексне тестування форм, валідації даних та перевірку безпеки системи.

Локальне тестування системи показало її ефективність та надійність. Система успішно обробляє дані, забезпечує безпечне зберігання інформації та надає зручний інтерфейс для користувачів. Адаптивний дизайн забезпечує комфортне використання на різних пристроях.

Перспективи розвитку системи включають розширення функціоналу, впровадження додаткових можливостей для продавців, покращення системи пошуку та фільтрації товарів. Система має потенціал для масштабування та адаптації до змінних потреб ринку комп'ютерних аксесуарів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Node.js Documentation. [Електронний ресурс]. - Режим доступу: https://nodejs.org/en/docs/ – дата доступу: 01.05.2025
2. Express.js Documentation. [Електронний ресурс]. - Режим доступу: https://expressjs.com/ – дата доступу: 01.05.2025
3. MySQL Documentation. [Електронний ресурс]. - Режим доступу: https://dev.mysql.com/doc/ – дата доступу: 01.05.2025
4. Passport.js Documentation. [Електронний ресурс]. - Режим доступу: http://www.passportjs.org/docs/ – дата доступу: 01.05.2025
5. Sequelize Documentation. [Електронний ресурс]. - Режим доступу: https://sequelize.org/ – дата доступу: 01.05.2025
6. EJS Documentation. [Електронний ресурс]. - Режим доступу: https://ejs.co/ – дата доступу: 01.05.2025
7. Nodemailer Documentation. [Електронний ресурс]. - Режим доступу: https://nodemailer.com/ – дата доступу: 01.05.2025
8. Selenium IDE Documentation. [Електронний ресурс]. - Режим доступу: https://www.selenium.dev/selenium-ide/docs/en/introduction/getting-started – дата доступу: 01.05.2025
9. REST API Design Best Practices. [Електронний ресурс]. - Режим доступу: https://restfulapi.net/ – дата доступу: 01.05.2025
10. JavaScript ES6+ Features. [Електронний ресурс]. - Режим доступу: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference – дата доступу: 01.05.2025
11. Web Application Architecture. [Електронний ресурс]. - Режим доступу: https://developer.mozilla.org/en-US/docs/Web/Architecture – дата доступу: 01.05.2025
12. Database Design for E-commerce. [Електронний ресурс]. - Режим доступу: https://www.mysql.com/why-mysql/ – дата доступу: 01.05.2025
13. Selenium automates browsers. [Електронний ресурс]. - Режим доступу: https://www.selenium.dev/ – дата доступу: 01.05.2025

14. Information System for Design of Thin Multilayer Film Processes Parameters Management based on Diffusion. Mykhaylo Petryk, Vitalii Chyzh, Halyna Tsupryk, Oksana Petryk, ITTAP'2024: 4th International Workshop on Information Technologies: Theoretical and Applied Problems, October 23- 25, 2024, Ternopil, Ukraine, Opole, Poland, 2024, pp. 486-493 (Scopus). [Електронний ресурс]. - Режим доступу: https://ceur-ws.org/Vol-3896 – дата доступу: 08.05.2025

15. Methods of constructing algorithms for comparative test statistical verification of mathematical models of bioobject responses to low-intensity stimuli / Bohdan Yavorskyy, Evhenia Yavorska, Halyna Tsupryk, Roman Kinash // Scientific Journal of TNTU. — Tern.: TNTU, 2023. — Vol 112. — No 4. — P. 82–90. (фахове видання України) – дата доступу: 08.05.2025

16. Яворська, ЄБ, Кінаш, РВ, Цуприк, ГБ, Николайчук ВІ Проблеми виявлення біосигналів у медичних інформаційних системах/ЄБ Яворська, Кінаш РВ, ГБ Цуприк, ВІ Николайчук//IV Міжнародна науково-практична конференція «Інформаційні системи та технології в медицині»(ICM–2021)[Текст]: зб. наук. пр.– Харків: Нац. аерокосм. ун-т ім. МЄ Жуковського «Харків. авіац. ін-т», 2021.–25-26 с. – дата доступу: 08.05.2025

17. Bibliographic description: Pasika DR, Tsupryk HB (2023) Rozrobka odnostorinkovoho veb-zastosunku z vykorystanniam VUE. JS i REACT: porivnialnyi analiz produktyvnostib korystuvatskoho dosvidu ta dotsilnosti [Developing a single-page web application using VUE. JS and REACT: a comparative analysis of user experience performance and expediency]. IMSTT (Tern., 13-14 December 2023), pp. 218-219 [in Ukrainian]. [Електронний ресурс]. - Режим доступу: https://scholar.google.com/citations?view_op=view_citation&hl=uk&user=9pNcfCkAAAAJ&sortby=pubdate&citation_for_view=9pNcfCkAAAAJ:JQOojiI6XY0C – дата доступу: 08.05.2025

18. Осельський С. В. Розробка реактивного фронтенд фреймворку для односторінкових додатків з власною системою реактивності : кваліфікаційна робота на здобуття освітнього ступеня магістр за спеціальністю „121 — інженерія програмного забезпечення“ / С. В. Осельський. — Тернопіль: ТНТУ, 2023. — 99 с.

[Електронний ресурс]. - Режим доступу: http://elartu.tntu.edu.ua/handle/lib/44477 – дата доступу: 08.05.2025

19. Технічні особливості взаємодії між клієнтом та сервером у реальному часі. О Остапчук, Г Цуприк Матеріали X науково-технічної конференції “Інформаційні моделі, системи та технології” Тернопільського національного технічного університету імені Івана Пулюя. — Тернопіль: ТНТУ, 2022. — 124 с. – дата доступу: 08.05.2025

20. Розробка сучасного On-line сервісу із застосуванням технологій Веб-програмування ВВ Залізняк, ГБ Цуприк - Матеріали VIII науково-технічної конференції “Інформаційні моделі, системи та технології” Тернопільського національного технічного університету імені Івана Пулюя. — Тернопіль: ТНТУ, 2020. — 161 с. – дата доступу: 08.05.2025

21. Мелех Л.В. Безпека життєдіяльності та охорона праці: навч. посіб./Мелех Л.В. – Львів: ЛДУ внутрішніх справ. 2022. 219 с.

22. Закон про охорону праці. Офіційний вебпортал парламенту України. URL: https://zakon.rada.gov.ua/laws/show/2694-12#Text (дата звернення: 14.06.2025).

23. Про перелік робіт з підвищеною небезпекою (НПАОП 0.00-4.36-05): наказ Держгірпромнагляду України від 26.01.2005 р. № 15 (зі змінами). – URL: https://zakon.rada.gov.ua/laws/show/z0231-05 (дата звернення: 14.06.2025).

24. Про типове положення про порядок проведення навчання та перевірки знань з питань охорони праці (НПАОП 0.00-4.36-05): наказ Держгірпромнагляду України від 26.01.2005 р. № 15 (із змінами від 30.01.2017 р.). – URL: https://zakon.rada.gov.ua/laws/show/z0231-05 (дата звернення: 14.06.2025).

25. Про затвердження Правил безпечної експлуатації електроустановок споживачів (ДНАОП 0.00-1.21-98). Офіційний вебпортал парламенту України. URL: https://zakon.rada.gov.ua/laws/show/z0093-98 (дата звернення: 14.06.2025).

26. Дистанційний курс «Кваліфікаційні роботи бакалаврів» сайту дистанційного навчання ТНТУ [Електронний ресурс]. – Режим доступу: URL: https://dl.tntu.edu.ua/bounce.php?course=5329

ДОДАТКИ

Додаток А Структура Проєкту

Лістинг коду файлу index.js:

```
const express = require("express");
const csrf = require("csurf");
const stripe = require("stripe")(process.env.STRIPE_PRIVATE_KEY);
const Product = require("../models/product");
const Category = require("../models/category");
const { Cart, CartItem } = require("../models/cart");
const Order = require("../models/order");
const middleware = require("../middleware");
const router = express.Router();

const csrfProtection = csrf();
router.use(csrfProtection);


router.get("/", async (req, res) => {
  try {
    const products = await Product.findAll({
      order: [["createdAt", "DESC"]],
      include: [{ model: Category, as: 'category' }]
    });
    const categories = await Category.findAll({ order: [["title", "ASC"]]
});
    res.render("shop/home", {
      pageName: "Home",
      products,
      categories,
      csrfToken: req.csrfToken()
    });
  } catch (error) {
    console.log(error);
    res.redirect("/");
  }
});


router.post("/shopping-cart/add-to-cart/:id", async (req, res) => {
  const productId = req.params.id;
  try {
    let cart;
    if (req.user) {

      [cart] = await Cart.findOrCreate({
        where: { userId: req.user.id },
        include: [{ model: CartItem, as: 'items' }]
      });
    } else if (req.session.cart) {

      cart = await Cart.findByPk(req.session.cart.id, {
        include: [{ model: CartItem, as: 'items' }]
      });
      if (!cart) {
```

```
        cart = await Cart.create();
      }
    } else {

      cart = await Cart.create();
    }


    const product = await Product.findByPk(productId);
    if (!product) {
      req.flash("error", "Product not found");
      return res.redirect("/");
    }

    let cartItem = await CartItem.findOne({
      where: {
        CartId: cart.id,
        productId: product.id
      }
    });

    if (cartItem) {

      cartItem.qty += 1;
      cartItem.price = cartItem.qty * product.price;
      await cartItem.save();
    } else {

      cartItem = await CartItem.create({
        CartId: cart.id,
        productId: product.id,
        qty: 1,
        price: product.price,
        title: product.title,
        productCode: product.productCode
      });
    }


    cart.totalQty = await CartItem.sum('qty', { where: { CartId: cart.id }
});
    cart.totalCost = await CartItem.sum('price', { where: { CartId: cart.id
} });
    await cart.save();


    req.session.cart = cart;

    req.flash("success", "Item added to the shopping cart");
    res.redirect(req.headers.referer);
  } catch (err) {
    console.log(err);
    req.flash("error", "Error adding item to cart");
    res.redirect("/");
  }
});


router.get("/shopping-cart", async (req, res) => {
  try {
```

```
    let cart;
    if (req.user) {

      cart = await Cart.findOne({
        where: { userId: req.user.id },
        include: [{ model: CartItem, as: 'items' }]
      });
    } else if (req.session.cart) {

      cart = await Cart.findByPk(req.session.cart.id, {
        include: [{ model: CartItem, as: 'items' }]
      });
    }

    if (!cart || !cart.items || cart.items.length === 0) {
      return res.render("shop/shopping-cart", {
        products: null,
        totalPrice: 0,
        pageName: "Кошик"
      });
    }


    const products = await productsFromCart(cart);

    res.render("shop/shopping-cart", {
      products: products,
      totalPrice: cart.totalCost,
      pageName: "Кошик"
    });
  } catch (err) {
    console.log(err);
    req.flash("error", "Помилка при завантаженні кошика");
    res.redirect("/");
  }
});


router.get("/removeAll/:id", async function (req, res, next) {
  const productId = req.params.id;
  try {
    let cart;
    if (req.user) {
      cart = await Cart.findOne({
        where: { userId: req.user.id },
        include: [{ model: CartItem, as: 'items' }]
      });
    } else if (req.session.cart) {
      cart = await Cart.findByPk(req.session.cart.id, {
        include: [{ model: CartItem, as: 'items' }]
      });
    }

    if (cart) {
      console.log('Видалення товару з ID:', productId);

      await CartItem.destroy({
        where: {
          CartId: cart.id,
          productId: productId
```

```
        }
      });


      cart.totalQty = await CartItem.sum('qty', { where: { CartId: cart.id
} }) || 0;
      cart.totalCost = await CartItem.sum('price', { where: { CartId: cart.id
} }) || 0;
      await cart.save();


      req.session.cart = cart;

      req.flash('success', 'Товар успішно видалено з кошика');


      if (cart.totalQty <= 0) {
        if (req.user) {
          await cart.destroy();
        }
        req.session.cart = null;
      }
    }

    res.redirect('/shopping-cart');
  } catch (err) {
    console.error('Помилка при видаленні товару:', err);
    req.flash("error", "Помилка при видаленні товару з кошика");
    res.redirect("/shopping-cart");
  }
});


router.get("/reduce/:id", async function (req, res, next) {
  const productId = req.params.id;
  try {
    let cart;
    if (req.user) {
      cart = await Cart.findOne({
        where: { userId: req.user.id },
        include: [{ model: CartItem, as: 'items' }]
      });
    } else if (req.session.cart) {
      cart = await Cart.findByPk(req.session.cart.id, {
        include: [{ model: CartItem, as: 'items' }]
      });
    }

    if (cart) {

      const cartItem = await CartItem.findOne({
        where: {
          CartId: cart.id,
          productId: productId
        }
      });

      if (cartItem) {

        cartItem.qty -= 1;
```

```
        if (cartItem.qty <= 0) {

          await cartItem.destroy();
        } else {

          const product = await Product.findByPk(productId);
          cartItem.price = cartItem.qty * product.price;
          await cartItem.save();
        }


        cart.totalQty = await CartItem.sum('qty', { where: { CartId: cart.id
} }) || 0;
        cart.totalCost = await CartItem.sum('price', { where: { CartId:
cart.id } }) || 0;
        await cart.save();


        req.session.cart = cart;


        if (cart.totalQty <= 0) {
          if (req.user) {
            await cart.destroy();
          }
          req.session.cart = null;
        }
      }
    }

    res.redirect('/shopping-cart');
  } catch (err) {
    console.log(err);
    req.flash("error", "Error reducing item quantity");
    res.redirect("/shopping-cart");
  }
});


router.get("/checkout", middleware.isLoggedIn, async (req, res) => {
  const errorMsg = req.flash("error")[0];

  if (!req.session.cart) {
    return res.redirect("/shopping-cart");
  }

  cart = await Cart.findById(req.session.cart._id);

  const errMsg = req.flash("error")[0];
  res.render("shop/checkout", {
    total: cart.totalCost,
    csrfToken: req.csrfToken(),
    errorMsg,
    pageName: "Checkout",
  });
});


router.post("/checkout", middleware.isLoggedIn, async (req, res) => {
```

```
  if (!req.session.cart) {
    return res.redirect("/shopping-cart");
  }
  const cart = await Cart.findById(req.session.cart._id);
  stripe.charges.create(
    {
      amount: cart.totalCost * 100,
      currency: "usd",
      source: req.body.stripeToken,
      description: "Test charge",
    },
    function (err, charge) {
      if (err) {
        req.flash("error", err.message);
        console.log(err);
        return res.redirect("/checkout");
      }
      const order = new Order({
        user: req.user,
        cart: {
          totalQty: cart.totalQty,
          totalCost: cart.totalCost,
          items: cart.items,
        },
        address: req.body.address,
        paymentId: charge.id,
      });
      order.save(async (err, newOrder) => {
        if (err) {
          console.log(err);
          return res.redirect("/checkout");
        }
        await cart.save();
        await Cart.findByIdAndDelete(cart._id);
        req.flash("success", "Successfully purchased");
        req.session.cart = null;
        res.redirect("/user/profile");
      });
    }
  );
});


router.get("/shopping-cart/count", async function (req, res) {
  try {
    let cart;
    if (req.user) {
      cart = await Cart.findOne({
        where: { userId: req.user.id }
      });
    } else if (req.session.cart) {
      cart = await Cart.findByPk(req.session.cart.id);
    }

    if (!cart) {
      return res.json({ count: 0 });
    }

    res.json({ count: cart.totalQty || 0 });
  } catch (err) {
```

```
    console.log(err);
    res.json({ count: 0 });
  }
});


async function productsFromCart(cart) {
  let products = []; // array of objects
  for (const item of cart.items) {
    let foundProduct = await Product.findByPk(item.productId, { include: [{
model: Category, as: 'category' }] });
    if (!foundProduct) continue;
    let foundProductObj = foundProduct.toJSON();
    foundProductObj["qty"] = item.qty;
    foundProductObj["totalPrice"] = item.price;
    products.push(foundProductObj);
  }
  return products;
}

module.exports = router;
```

Лістинг коду файлу product.js:

```
const express = require("express");
const router = express.Router();
const Product = require("../models/product");
const Category = require("../models/category");
const { Op } = require("sequelize");
var moment = require("moment");
const csrf = require("csurf");

const csrfProtection = csrf();
router.use(csrfProtection);


router.get("/", async (req, res) => {
  const successMsg = req.flash("success")[0];
  const errorMsg = req.flash("error")[0];
  const perPage = 8;
  let page = parseInt(req.query.page) || 1;
  try {
    const { count, rows: products } = await Product.findAndCountAll({
      order: [["createdAt", "DESC"]],
      include: [{ model: Category, as: "category" }],
      offset: perPage * (page - 1),
      limit: perPage
    });
    res.render("shop/index", {
      pageName: "Всі товари",
      products,
      successMsg,
      errorMsg,
      current: page,
      breadcrumbs: null,
      home: "/products/?",
      pages: Math.ceil(count / perPage),
      csrfToken: req.csrfToken()
```

```
    });
  } catch (error) {
    console.log(error);
    res.redirect("/");
  }
});


router.get("/search", async (req, res) => {
  const perPage = 8;
  let page = parseInt(req.query.page) || 1;
  const successMsg = req.flash("success")[0];
  const errorMsg = req.flash("error")[0];

  try {
    const { count, rows: products } = await Product.findAndCountAll({
      where: {
        title: { [Op.like]: `%${req.query.search || ''}%` }
      },
      order: [["createdAt", "DESC"]],
      include: [{ model: Category, as: "category" }],
      offset: perPage * (page - 1),
      limit: perPage
    });
    res.render("shop/index", {
      pageName: "Search Results",
      products,
      successMsg,
      errorMsg,
      current: page,
      breadcrumbs: null,
      home: "/products/search?search=" + req.query.search + "&",
      pages: Math.ceil(count / perPage),
      csrfToken: req.csrfToken()
    });
  } catch (error) {
    console.log(error);
    res.redirect("/");
  }
});


router.get("/:slug", async (req, res) => {
  const successMsg = req.flash("success")[0];
  const errorMsg = req.flash("error")[0];
  const perPage = 8;
  let page = parseInt(req.query.page) || 1;
  try {
    const foundCategory = await Category.findOne({ where: { slug:
req.params.slug } });
    if (!foundCategory) throw new Error('Category not found');
    const { count, rows: allProducts } = await Product.findAndCountAll({
      where: { categoryId: foundCategory.id },
      order: [["createdAt", "DESC"]],
      include: [{ model: Category, as: "category" }],
      offset: perPage * (page - 1),
      limit: perPage,
      attributes: ['id', 'title', 'price', 'imagePath', 'available',
'categoryId']
    });
```

```
    res.render("shop/index", {
      pageName: foundCategory.title,
      currentCategory: foundCategory,
      products: allProducts,
      successMsg,
      errorMsg,
      current: page,
      breadcrumbs: req.breadcrumbs,
      home: "/products/" + req.params.slug.toString() + "/?",
      pages: Math.ceil(count / perPage),
      csrfToken: req.csrfToken()
    });
  } catch (error) {
    console.log(error);
    return res.redirect("/");
  }
});


router.get("/:slug/:id", async (req, res) => {
  const successMsg = req.flash("success")[0];
  const errorMsg = req.flash("error")[0];
  try {
    const product = await Product.findByPk(req.params.id, { include: [{
model: Category, as: "category" }] });
    if (!product) throw new Error('Product not found');
    res.render("shop/product", {
      pageName: product.title,
      product,
      successMsg,
      errorMsg,
      moment: moment,
      csrfToken: req.csrfToken()
    });
  } catch (error) {
    console.log(error);
    return res.redirect("/");
  }
});

module.exports = router;
```

Лістинг коду файлу pages.js:

```
const express = require("express");
const csrf = require("csurf");
const nodemailer = require("nodemailer");
const router = express.Router();
const {
  userContactUsValidationRules,
  validateContactUs,
} = require("../config/validator");
const csrfProtection = csrf();
router.use(csrfProtection);


router.get("/about-us", (req, res) => {
  res.render("pages/about-us", {
    pageName: "About Us",
  });
});


router.get("/shipping-policy", (req, res) => {
  res.render("pages/shipping-policy", {
    pageName: "Shipping Policy",
  });
});


router.get("/careers", (req, res) => {
  res.render("pages/careers", {
    pageName: "Careers",
  });
});


router.get("/contact-us", (req, res) => {
  const successMsg = req.flash("success")[0];
  const errorMsg = req.flash("error");
  res.render("pages/contact-us", {
    pageName: "Contact Us",
    csrfToken: req.csrfToken(),
    successMsg,
    errorMsg,
  });
});


router.post(
  "/contact-us",
  [userContactUsValidationRules(), validateContactUs],
  (req, res) => {

    const smtpTrans = nodemailer.createTransport({
      host: "smtp.gmail.com",
      port: 587,
      secure: false,
      auth: {

        user: process.env.GMAIL_EMAIL,
```

```
        pass: process.env.GMAIL_PASSWORD,
      },
      tls: {
        rejectUnauthorized: false,
      },
    });


    const mailOpts = {
      from: req.body.email,
      to: process.env.GMAIL_EMAIL,
      subject: `Enquiry from ${req.body.name}`,
      html: `
      <div>
      <h2   style="color:   #478ba2;   text-align:center;">Client's   name:
${req.body.name}</h2>
      <h3 style="color: #478ba2;">Client's email: (${req.body.email})<h3>
      </div>
      <h3 style="color: #478ba2;">Client's message: </h3>
      <div style="font-size: 30;">
      ${req.body.message}
      </div>
      `,
    };


    smtpTrans.sendMail(mailOpts, (error, response) => {
      if (error) {
        req.flash(
          "error",
          "An error occured... Please check your internet connection and try
again later"
        );
        return res.redirect("/pages/contact-us");
      } else {
        req.flash(
          "success",
          "Email sent successfully! Thanks for your inquiry."
        );
        return res.redirect("/pages/contact-us");
      }
    });
  }
);

module.exports = router;
```

Лістинг коду файлу app.js:

```
require("dotenv").config();
if (!process.env.SESSION_SECRET) {
  console.error('SESSION_SECRET is not set in .env file');
  process.exit(1);
}
const createError = require("http-errors");
const express = require("express");
const path = require("path");
const cookieParser = require("cookie-parser");
const logger = require("morgan");
const session = require("express-session");
const MySQLStore = require('express-mysql-session')(session);
const passport = require("passport");
const flash = require("connect-flash");
const Category = require("./models/category");
const { connectDB, sequelize } = require("./config/db");
require('./models/associations');

const app = express();
require("./config/passport");


connectDB();


app.set("views", path.join(__dirname, "views"));
app.set("view engine", "ejs");


const adminRouter = require("./routes/admin");
app.use("/admin", adminRouter);

app.use(logger("dev"));
app.use(express.json());
app.use(express.urlencoded({ extended: false }));
app.use(cookieParser());
app.use(express.static(path.join(__dirname, "public")));


const sessionStore = new MySQLStore({
  host: process.env.DB_HOST || 'localhost',
  port: 3306,
  user: process.env.DB_USER || 'root',
  password: process.env.DB_PASSWORD || '',
  database: process.env.DB_NAME || 'bags_ecommerce'
});

app.use(
  session({
    secret: process.env.SESSION_SECRET,
    resave: false,
    saveUninitialized: false,
    store: sessionStore,
    cookie: { maxAge: 60 * 1000 * 60 * 3 },
  })
);
app.use(flash());
```

```
app.use(passport.initialize());
app.use(passport.session());


app.use(async (req, res, next) => {
  try {
    res.locals.login = req.isAuthenticated();
    res.locals.session = req.session;
    res.locals.currentUser = req.user;
    const categories = await Category.findAll({
      order: [['title', 'ASC']]
    });
    res.locals.categories = categories;
    next();
  } catch (error) {
    console.log(error);
    res.redirect("/");
  }
});


get_breadcrumbs = function (url) {
  var rtn = [{ name: "Home", url: "/" }],
    acc = "",
    arr = url.substring(1).split("/");

  for (i = 0; i < arr.length; i++) {
    acc = i != arr.length - 1 ? acc + "/" + arr[i] : null;
    rtn[i + 1] = {
      name: arr[i].charAt(0).toUpperCase() + arr[i].slice(1),
      url: acc,
    };
  }
  return rtn;
};
app.use(function (req, res, next) {
  req.breadcrumbs = get_breadcrumbs(req.originalUrl);
  next();
});


const indexRouter = require("./routes/index");
const productsRouter = require("./routes/products");
const usersRouter = require("./routes/user");
const pagesRouter = require("./routes/pages");
app.use("/products", productsRouter);
app.use("/user", usersRouter);
app.use("/pages", pagesRouter);
app.use("/", indexRouter);


app.use(function (req, res, next) {
  next(createError(404));
});


app.use(function (err, req, res, next) {

  res.locals.message = err.message;
  res.locals.error = req.app.get("env") === "development" ? err : {};
```

```
  res.status(err.status || 500);
  res.render("error");
});

var port = process.env.PORT || 3000;
app.set("port", port);
app.listen(port, () => {
  console.log("Server running at port " + port);
});

module.exports = app;
```

Лістинг коду файлу user.js:

```
const express = require("express");
const router = express.Router();
const csrf = require("csurf");
var passport = require("passport");
var LocalStrategy = require("passport-local").Strategy;
const Product = require("../models/product");
const Order = require("../models/order");
const Cart = require("../models/cart");
const middleware = require("../middleware");
const {
  userSignUpValidationRules,
  userSignInValidationRules,
  validateSignup,
  validateSignin,
} = require("../config/validator");
const csrfProtection = csrf();
router.use(csrfProtection);


router.get("/signup", middleware.isNotLoggedIn, (req, res) => {
  var errorMsg = req.flash("error")[0];
  res.render("user/signup", {
    csrfToken: req.csrfToken(),
    errorMsg,
    pageName: "Sign Up",
  });
});

router.post(
  "/signup",
  [
    middleware.isNotLoggedIn,
    userSignUpValidationRules(),
    validateSignup,
    passport.authenticate("local.signup", {
      successRedirect: "/user/profile",
      failureRedirect: "/user/signup",
      failureFlash: true,
    }),
  ],
  async (req, res) => {
    try {
```

```
      if (req.session.cart) {
        const cart = await new Cart(req.session.cart);
        cart.user = req.user._id;
        await cart.save();
      }

      if (req.session.oldUrl) {
        var oldUrl = req.session.oldUrl;
        req.session.oldUrl = null;
        res.redirect(oldUrl);
      } else {
        res.redirect("/user/profile");
      }
    } catch (err) {
      console.log(err);
      req.flash("error", err.message);
      return res.redirect("/");
    }
  }
);


router.get("/signin", middleware.isNotLoggedIn, async (req, res) => {
  var errorMsg = req.flash("error")[0];
  res.render("user/signin", {
    csrfToken: req.csrfToken(),
    errorMsg,
    pageName: "Sign In",
  });
});


router.post(
  "/signin",
  [
    middleware.isNotLoggedIn,
    userSignInValidationRules(),
    validateSignin,
    passport.authenticate("local.signin", {
      failureRedirect: "/user/signin",
      failureFlash: true,
    }),
  ],
  async (req, res) => {
    try {
      console.log("Користувач увійшов:", req.user.email);


      let cart = await Cart.findOne({ where: { userId: req.user.id } });
      console.log("Знайдений кошик користувача:", cart);


      if (req.session.cart && !cart) {
        console.log("Створення нового кошика з сесії");
        cart = await Cart.create({
          userId: req.user.id,
          items: req.session.cart.items || [],
          totalQty: req.session.cart.totalQty || 0,
          totalCost: req.session.cart.totalCost || 0
```

```
        });
      }


      if (cart) {
        console.log("Завантаження кошика в сесію");
        req.session.cart = cart;
      }


      if (req.session.oldUrl) {
        var oldUrl = req.session.oldUrl;
        req.session.oldUrl = null;
        res.redirect(oldUrl);
      } else {
        res.redirect("/user/profile");
      }
    } catch (err) {
      console.log("Помилка при вході:", err);
      req.flash("error", err.message);
      return res.redirect("/");
    }
  }
);

router.get("/profile", middleware.isLoggedIn, async (req, res) => {
  const successMsg = req.flash("success")[0];
  const errorMsg = req.flash("error")[0];
  try {
    console.log("Завантаження профілю для користувача:", req.user.email);

    allOrders = await Order.findAll({ where: { userId: req.user.id } });
    res.render("user/profile", {
      orders: allOrders,
      errorMsg,
      successMsg,
      pageName: "User Profile",
      currentUser: req.user
    });
  } catch (err) {
    console.log("Помилка при завантаженні профілю:", err);
    return res.redirect("/");
  }
});
router.get("/logout", middleware.isLoggedIn, (req, res) => {
  req.logout();
  req.session.cart = null;
  res.redirect("/");
});
module.exports = router;
```

Додаток Б Ілюстрації варіантів використання системи

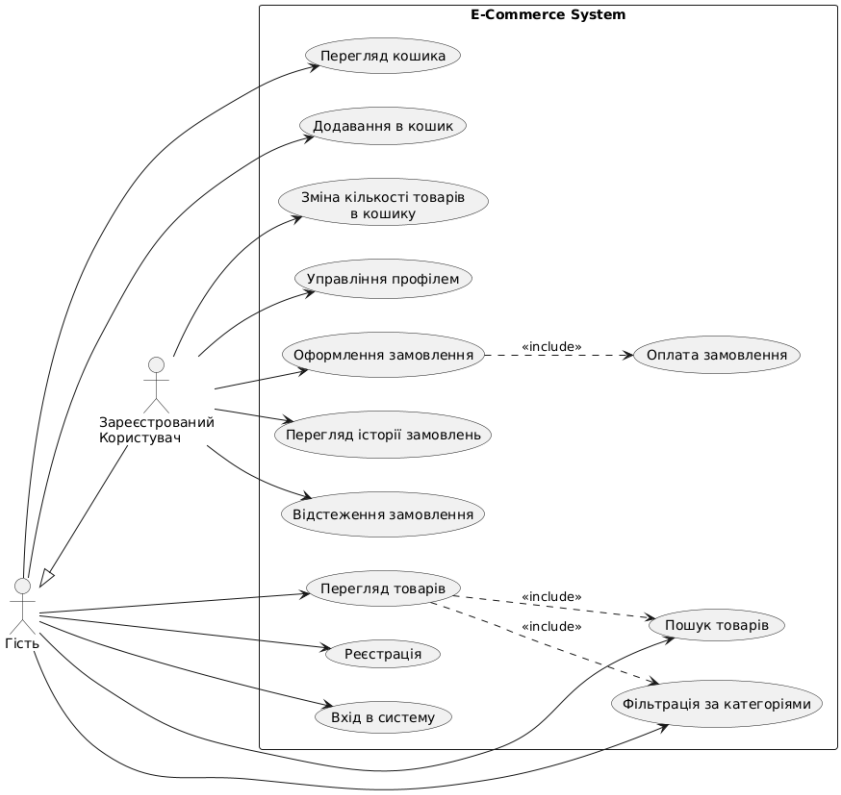

Рисунок 1.1.7.1 – Діаграма варіантів використання

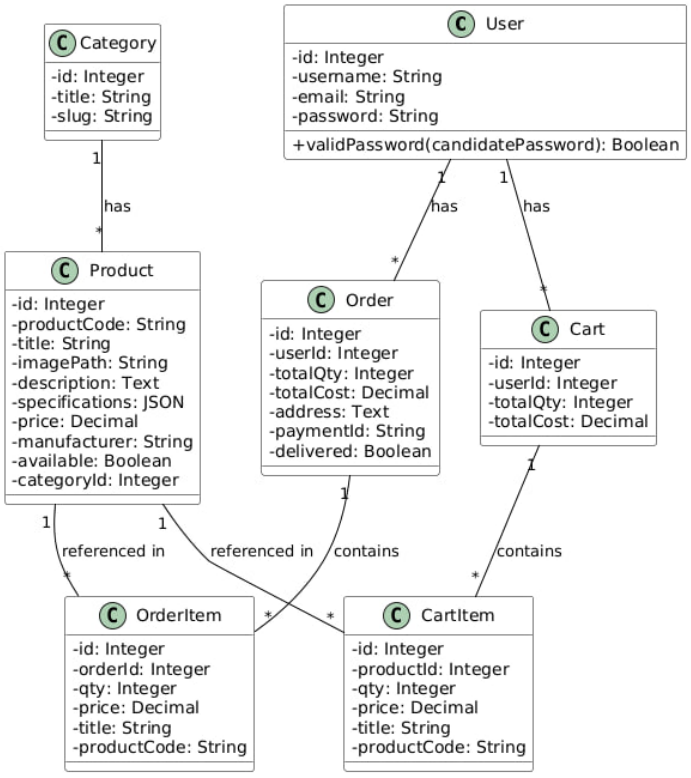

Рисунок 1.2.3.1 – Діаграма класів

Додаток В Публікація у науковій конференції

І.П. Вовк, канд. екон. наук, доц., А.Й. Матвіїшин, канд. техн. наук, доц., Ю.Я. Вовк, канд. техн. наук, доц. 184
ЦИФРОВІЗАЦІЯ ТА СТІЙКІСТЬ ЛАНЦЮГІВ ПОСТАЧАННЯ В УМОВАХ ГЛОБАЛЬНОЇ НЕСТАБІЛЬНОСТІ: ПОЄДНАННЯ ПІДХОДІВ ПРОЄКТНОГО АНАЛІЗУ ТА УПРАВЛІННЯ ЛАНЦЮГАМИ ПОСТАЧАННЯ

Р.Л. Голосинський 186
ПРОБЛЕМАТИКА ПРОГРАМНИХ РІШЕНЬ ДЛЯ КЕРУВАННЯ БЕЗПІЛОТНИКАМИ У ВІЙСЬКОВИХ КОНФЛІКТАХ

В.І.Гураль, Г.Б.Цуприк, к.т.н., доц. 188
ВИКОРИСТАННЯ СУЧАСНИХ ВЕБ-ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ ІНТЕРНЕТ-МАГАЗИНУ КОМП'ЮТЕРНИХ АКСЕСУАРІВ

Т. Денега – ст. гр. СП-41, д. ф-м. н. проф. М.Р.Петрик 189
РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ ДЛЯ НАДАННЯ ФРІЛАНС ПОСЛУГ З ВИКОРИСТАННЯМ ФРЕЙМВОРКІВ REACT ТА NODE.JS

Д.А.Дячишин, Г.Б.Цуприк, к.т.н., доц.. 190
ВИКОРИСТАННІ СУЧАСНИХ ІТ ТЕХНОЛОГІЙ ПРИ РОЗРОБЦІ СИСТЕМ ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ ОБРОБКИ ДАНИХ КОМЕРЦІЙНОГО ЗНАЧЕННЯ

Р.З. Золотий, к.т.н., доцент, Д.А. Коломийчук 192
ЕФЕКТИВНІСТЬ АТАК МЕРЕЖ МАШИННОГО НАВЧАННЯ

Д.В. Коваль, Г.Б. Цуприк, к.т.н., доц.. 193
МОДУЛЬНА РЕАЛІЗАЦІЯ ФУНКЦІОНАЛУ ФІНАНСОВОГО КОНТРОЛЮ НА ANDROID ІЗ ЗАСТОСУВАННЯМ ПРИНЦИПІВ ІНВЕРСІЇ УПРАВЛІННЯ ТА ШАБЛОНУ MVVM

А.М. Ковтко; В.Б. Савків, к.т.н., доц.; Г.В. Козбур, к.т.н., доц.; І.Р. Козбур 195
АНАЛІЗ ЕФЕКТИВНОСТІ МУТАЦІЙНОГО ТЕСТУВАННЯ

А. Конончук – ст. гр. СП-41, док. філософії. І. Мудрик 197
СТВОРЕННЯ УКРАЇНСЬКОЇ МОВНОЇ ГРИ НА ОСНОВІ EMBEDDING-МОДЕЛЕЙ

О.І. Крамар, к.ф.-м.н., доц.; Т.О. Крамар 198
МОЖЛИВОСТІ ТРАНСФОРМАЦІЇ ЦИФРОВОГО МУЗЕЮ ІВАНА ПУЛЮЯ В КОНТЕКСТІ ВИКОРИСТАННЯ VR-ГАРНІТУРИ META QUEST

Т.О. Крамар, А.В. Мельник 200
ВИКОРИСТАННЯ ШТУЧНОГО ІНТЕЛЕКТУ ДЛЯ СТВОРЕННЯ 3D-МОДЕЛЕЙ: ПЕРСПЕКТИВИ ТА СТАН ДОСЛІДЖЕНЬ

В.В. Круцяк, Г.Б. Цуприк, к.т.н., доц.. 202
РОЗРОБКА ПРИКЛАДНОГО СПЕЦІАЛІЗОВАНОГО ЗАСТОСУНКУ ДЛЯ ОПТИМІЗАЦІЇ ОПРАЦЮВАННЯ МАСИВІВ ДАННИХ З ЗАСТОСУВАННЯМ ПАРАДИГМ ТА ТЕХНОЛОГІЙ PYTHON, PYQT5 ТА SQLITE

Я. Курко, доцент, О. Босюк ст. викладач, Н. Вальчак ст. викладач, З. Кульчицький, ст. викладач, І. Казмірчук, ст. викладач. 204
ЗАСТОСУВАННЯ КОМП'ЮТЕРНОЇ ПРОГРАМИ "REACTION-TEST" ДЛЯ ВИЗНАЧЕННЯ СТАРТОВОЇ РЕАКЦІЇ СПОРТСМЕНІВ

Т. А. Липак, аспірант 206
ІНТЕГРАЦІЯ МЕТОДІВ ШТУЧНОГО ІНТЕЛЕКТУ В РОЗРОБКУ ОРІЄНТОВАНИХ НА КОРИСТУВАЧА ІНТЕРФЕЙСІВ

УДК 004.41

В.І.Гураль, Г.Б.Цуприк, к.т.н., доц.
Тернопільський національний технічний університет імені Івана Пулюя, Україна

ВИКОРИСТАННЯ СУЧАСНИХ ВЕБ-ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ ІНТЕРНЕТ-МАГАЗИНУ КОМП'ЮТЕРНИХ АКСЕСУАРІВ

V.I.Gural, H.B.Tsupryk, Ph.D., Assoc. Prof.

USING MODERN WEB TECHNOLOGIES FOR DEVELOPING AN ONLINE STORE FOR COMPUTER ACCESSORIES

Сучасний розвиток інформаційних технологій та зростаюча популярність онлайн-шопінгу створюють нові вимоги до систем електронної комерції. Особливо це стосується спеціалізованих магазинів, таких як магазини комп'ютерних аксесуарів, де користувачі потребують детальної інформації про продукцію та зручного інтерфейсу для пошуку специфічних товарів. Аналіз ринку показує, що існуючі рішення часто не відповідають цим вимогам, пропонуючи або надто спрощений функціонал, або надто складний інтерфейс, що ускладнює навігацію.

Метою даної роботи є створення сучасного інтернет-магазину комп'ютерних аксесуарів, який би поєднував у собі зручний користувацький інтерфейс, ефективну систему управління товарами та надійну систему безпеки. Особливу увагу приділено розробці інтуїтивно зрозумілої системи навігації та пошуку, що дозволить користувачам швидко знаходити потрібні товари серед великої кількості позицій.

Для реалізації проекту обрано сучасний стек технологій, що включає Node.js як серверну платформу та MySQ як систему управління базами даних. Такий вибір дозволяє забезпечити високу продуктивність системи, її масштабованість та надійність. Особливу увагу приділено аспектам безпеки системи, включаючи захист персональних даних користувачів та забезпечення безпеки транзакцій.

Розроблена система відрізняється від аналогів спеціалізованим каталогом товарів з детальною системою фільтрації, оптимізованим пошуком з урахуванням специфіки комп'ютерних аксесуарів, зручним інтерфейсом для продавців з можливістю швидкого додавання нових товарів, системою рейтингів та відгуків для підвищення довіри користувачів, а також інтеграцією з популярними платіжними системами.

Важливим аспектом розробки є забезпечення безпеки системи. Для цього реалізовано систему авторизації та аутентифікації користувачів, шифрування конфіденційних даних, захист від поширених веб-вразливостей та систему резервного копіювання даних.

Додатково, система включає функціонал для аналітики продажів, що дозволяє відстежувати популярні товари та тенденції на ринку. Це допомагає продавцям оптимізувати свій асортимент та цінову політику. Також реалізовано систему сповіщень для користувачів про нові надходження та акції, що підвищує залученості користувачів та стимулює продажі.

Система також підтримує інтеграцію з соціальними мережами, що дозволяє користувачам ділитися товарами з друзями та слідкувати за оновленнями через соціальні платформи. Це сприяє збільшенню аудиторії та покращенню взаємодії з користувачами.

Додатково, система включає функціонал для аналітики продажів, що дозволяє відстежувати популярні товари та тенденції на ринку. Це допомагає продавцям оптимізувати свій асортимент та цінову політику. Також реалізовано систему сповіщень для користувачів про нові надходження та акції, що підвищує залученості користувачів та стимулює продажі.

Система також підтримує інтеграцію з соціальними мережами, що дозволяє користувачам ділитися товарами з друзями та слідкувати за оновленнями через соціальні платформи. Це сприяє збільшенню аудиторії та покращенню взаємодії з користувачами.

Додатково, система включає функціонал для аналітики продажів, що дозволяє відстежувати популярні товари та тенденції на ринку. Це допомагає продавцям оптимізувати

свій асортимент та цінову політику. Також реалізовано систему сповіщень для користувачів про нові надходження та акції, що підвищує залученості користувачів та стимулює продажі.

Система також підтримує інтеграцію з соціальними мережами, що дозволяє користувачам ділитися товарами з друзями та слідкувати за оновленнями через соціальні платформи. Це сприяє збільшенню аудиторії та покращенню взаємодії з користувачами.

Додатково, система включає функціонал для аналітики продажів, що дозволяє відстежувати популярні товари та тенденції на ринку. Це допомагає продавцям оптимізувати свій асортимент та цінову політику. Також реалізовано систему сповіщень для користувачів про нові акції, що підвищує залученості користувачів.

Система також підтримує інтеграцію з соціальними мережами, що дозволяє користувачам ділитися товарами з друзями та слідкувати за оновленнями через соціальні платформи. Це сприяє збільшенню аудиторії та покращенню взаємодії з користувачами.

Література

1. Web Development with Node and Express: Leveraging the JavaScript Stack / Ethan Brown. 2nd Edition. O'Reilly Media, 2019. 332 p. URL: https://www.oreilly.com/library/view/web-development-with/9781492053507/ (дата звернення: 19.05.2025).

2. Node.js Design Patterns / Mario Casciaro, Luciano Mammino. 3rd Edition. Packt Publishing, 2020. 662 p. URL: https://www.packtpub.com/product/node-js-design-patterns-third-edition/9781839214110 (дата звернення: 19.05.2025).

3. OWASP Cheat Sheet Series: Node.js Security Cheat Sheet. OWASP Foundation. URL: https://cheatsheetseries.owasp.org/cheatsheets/Nodejs_Security_Cheat_Sheet.html (дата звернення: 19.05.2025).

4. Methods of constructing algorithms for comparative test statistical verification of mathematical models of bioobject responses to low-intensity stimuli / Bohdan Yavorskyy, Evhenia Yavorska, Halyna Tsupryk, Roman Kinash // Scientific Journal of TNTU. — Tern.: TNTU, 2023. — Vol 112. — No 4. — P. 82–90.

5. Бойко І.В., М.Р. Петрик, Г.Б. Цуприк. Інформаційні технології видобутку даних (Data mining, високопродуктивні обчислення у складних системах): навчальний посібник. Тернопіль: : ТНТУ 2020 – 62 с.

6. Яворська, Є.Б., Кінаш, Р.В., Цуприк, Г.Б., Николайчук В.І. Проблеми виявлення біосигналів у медичних інформаційних системах / Є.Б. Яворська, Кінаш Р.В., Г.Б. Цуприк, В.І. Николайчук // IV Міжнародна науково-практична конференція «Інформаційні системи та технології в медицині» (ICM–2021) [Текст] : зб. наук. пр. – Харків : Нац. аерокосм. ун-т ім. М. Є. Жуковського «Харків. авіац. ін-т», 2021. – 25-26 с.

Додаток Г Диск з роботою