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

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

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

Бакалавр
(назва освітнього ступеня)

на тему: Розробка інтернет-магазину з інтеграцією віджетів Elfsight на базі Node.js

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

_____ Єршов В. В.
(підпис) (прізвище та ініціали)

Керівник _____ Стефанишин В. М.
(підпис) (прізвище та ініціали)

Нормоконтроль _____ Стоянов Ю. А.
(підпис) (прізвище та ініціали)

Завідувач кафедри _____ Петрик М. Р.
(підпис) (прізвище та ініціали)

Рецензент _____ _____
(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 71, таблиць 3, рисунків 71, презентація.

Тема: Розробка інтернет-магазину з інтеграцією віджетів Elfsight на базі Node.js.

У даній кваліфікаційній роботі бакалавра представлено процес проєктування та реалізації інтернет-магазину, побудованого на платформі Node.js із застосуванням сторонніх інтерактивних рішень – віджетів Elfsight. Основною метою роботи є створення сучасного, масштабованого веб-додатку, який дозволяє ефективно реалізувати комерційні функції, забезпечуючи зручний користувацький досвід.

Проведено аналіз ринку електронної комерції, виявлено актуальні вимоги до сучасних веб-магазинів, проаналізовано платформи та інструменти, які дозволяють забезпечити інтерактивність та залученість користувачів. Elfsight було обрано як сервіс для швидкої інтеграції функціональних віджетів (відгуки, соціальні стрічки, контактні форми тощо), які підвищують ефективність взаємодії з клієнтами.

У процесі розробки створено архітектуру веб-додатку, реалізовано серверну частину за допомогою Node.js і Express.js, а також підключено базу даних MongoDB. Для клієнтської частини використано HTML, CSS та JavaScript. Інтеграція Elfsight виконувалася через вставку кастомізованих віджетів у шаблони сторінок, що забезпечило простоту реалізації без втрати продуктивності. Окрему увагу приділено безпеці, продуктивності сайту та його адаптивності для різних пристроїв.

Ключові слова: інтернет-магазин, веб-додаток, Node.js, MongoDB, Express.js, Elfsight, інтерактивність, веб-архітектура.

ABSTRACT

Bachelor's qualification work. Ivan Pulyuy Ternopil National Technical University, Department of Software Engineering, specialty 121 "Software Engineering". TNTU, 2025, Pages 71, tables 3, figures 71, presentation.

Topic: Development of an online store with integration of Elfsight widgets based on Node.js.

This bachelor's thesis presents the process of designing and implementing an online store built on the Node.js platform using third-party interactive solutions – Elfsight widgets. The main goal of the work is to create a modern, scalable web application that allows for the effective implementation of commercial functions while providing a convenient user experience.

An analysis of the e-commerce market was conducted, current requirements for modern web stores were identified, platforms and tools were analyzed that allow for interactivity and user engagement. Elfsight was chosen as a service for quick integration of functional widgets (reviews, social feeds, contact forms, etc.) that increase the efficiency of interaction with customers.

During the development process, the web application architecture was created, the server side was implemented using Node.js and Express.js, and the MongoDB database was connected. HTML, CSS, and JavaScript were used for the client side. Elfsight integration was performed by inserting customized widgets into page templates, which ensured ease of implementation without loss of performance. Special attention is paid to security, site performance and its adaptability for different devices.

Keywords: online store, web application, Node.js, MongoDB, Express.js, Elfsight, interactivity, web architecture.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

Інтернет-магазин – веб-сайт, що дозволяє користувачам купувати товари або послуги онлайн.

UI (User Interface) – інтерфейс користувача, що визначає зовнішній вигляд та взаємодію з сайтом.

GUI (Graphical User Interface) – графічний інтерфейс користувача, що включає візуальні елементи (кнопки, іконки, меню) для полегшення взаємодії з сайтом.

UX (User Experience) – користувацький досвід, який включає всі аспекти взаємодії з сайтом та його зручність.

HTML/CSS – основні технології для створення веб-сторінок та їх стилізації.

JavaScript – мова програмування для створення інтерактивних елементів на сайті.

API (Application Programming Interface) – спосіб взаємодії між програмними компонентами (наприклад, для інтеграції з Elfsight).

Респонсивний дизайн – адаптивне оформлення веб-сайту для коректного відображення на різних пристроях.

Elfsight – платформа для створення та інтеграції віджетів без необхідності програмування.

Віджет – невеликий інтерактивний елемент веб-сайту, що додає функціональність (наприклад, чат, галерея, відгуки).

Код вставки (Embed Code) – спеціальний фрагмент коду для додавання віджета на веб-сторінку.

Веб-хостинг – послуга розміщення сайту на сервері, що забезпечує його доступність в Інтернеті.

Фреймворк – набір готових бібліотек і шаблонів для спрощення розробки веб-додатків (React, Angular, Vue.js).

Гейміфікація – впровадження ігрових механік (бали, рівні, нагороди) для підвищення залученості користувачів у процесі покупок.

ВСТУП

Інтернет-магазини стали невід’ємною складовою сучасного бізнесу, адже електронна комерція стрімко розвивається та відкриває нові можливості для підприємців. У цифрову епоху наявність зручного, швидкого та ефективного інструменту для продажу товарів і послуг є ключовим чинником успіху компаній, незалежно від їхнього розміру чи сфери діяльності.

Ефективний інтернет-магазин – це не лише платформа для купівлі-продажу, а й важливий інструмент взаємодії з клієнтами, аналізу їхніх потреб та покращення користувацького досвіду. У цьому контексті важливу роль відіграє інтеграція сторонніх рішень, таких як віджети Elfsight, які дозволяють швидко та без значного кодування додавати функціональність – від форм зворотного зв’язку до соціальних відгуків і онлайн-чатів.

Успішна інтеграція таких інструментів дозволяє не лише підвищити зручність для користувача, а й створити конкурентну перевагу на ринку. В умовах жорсткої конкуренції важливо забезпечити клієнту позитивний досвід взаємодії із сайтом: зручну навігацію, швидке завантаження сторінок, адаптивність до мобільних пристроїв, а також наявність інтерактивних функцій, що викликають довіру.

Ключовим завданням є створення інтернет-магазину, який би поєднував сучасний дизайн, високу продуктивність, масштабованість та зручність у користуванні. Для реалізації цього проєкту обрано Node.js – потужну платформу для створення серверної частини веб-додатків, яка забезпечує високу швидкодію та гнучкість розробки. Використання Express.js для побудови REST API, MongoDB для зберігання даних і віджетів Elfsight для взаємодії з клієнтом дозволяє створити цілісну систему, орієнтовану на ефективне вирішення завдань бізнесу.

Node.js, як асинхронне середовище виконання JavaScript, дозволяє ефективно обробляти запити та забезпечувати стабільну роботу під високим навантаженням. Завдяки цьому інтернет-магазин здатен обслуговувати велику кількість

користувачів одночасно без втрати продуктивності. MongoDB як нереляційна база даних дозволяє зберігати інформацію у гнучкому форматі, що особливо зручно для швидко змінюваних проєктів.

Інтеграція Elfsight значно пришвидшує реалізацію користувацьких функцій, не вимагаючи складного програмування. За допомогою візуального налаштування віджетів та генерації коду для вставки, розробник може додати функціонал соціальних стрічок, спливаючих повідомлень, онлайн-чату, галерей, слайдерів тощо – без залучення сторонніх сервісів чи складної архітектури.

Таким чином, запропонований підхід дозволяє ефективно поєднати сучасні інструменти розробки, автоматизації та інтеграції, створивши продукт, який відповідає актуальним вимогам ринку. Результатом виконаної роботи є повнофункціональний інтернет-магазин, який може бути адаптований під різні потреби користувачів та бізнес-моделі.

У даній роботі розглянуто як технічні, так і концептуальні аспекти розробки інтернет-магазину з інтеграцією Elfsight. Особливу увагу приділено побудові архітектури, вибору технологій, забезпеченню продуктивності та безпеки. Саме ці питання, а також кінцевий результат розробки, і є предметом дослідження в межах даної кваліфікаційної роботи.

1 ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд конкурентів

На ринку електронної комерції існує велика кількість платформ для продажу товарів, зокрема в сфері геймерської продукції. Одним із яскравих прикладів є інтернет-магазин Gamestop, що давно зарекомендував себе як один із лідерів серед геймерських рітейлерів у США.

Gamestop пропонує широкий асортимент геймерських товарів: від відеоігор та консолей до аксесуарів і мерчу (рис. 1.1). Платформа має зручну структуру каталогу, що дозволяє користувачеві швидко знаходити потрібну продукцію. Важливою перевагою є гейміфіковані елементи, як-от бонусна система PowerUp Rewards, яка стимулює повторні покупки та утримує клієнта. Також Gamestop використовує інтеграцію з відгуками, рекомендаційними системами та акційними банерами, що підвищує рівень залученості користувачів. Сайт має адаптивний дизайн, що забезпечує комфортну навігацію з будь-якого пристрою.

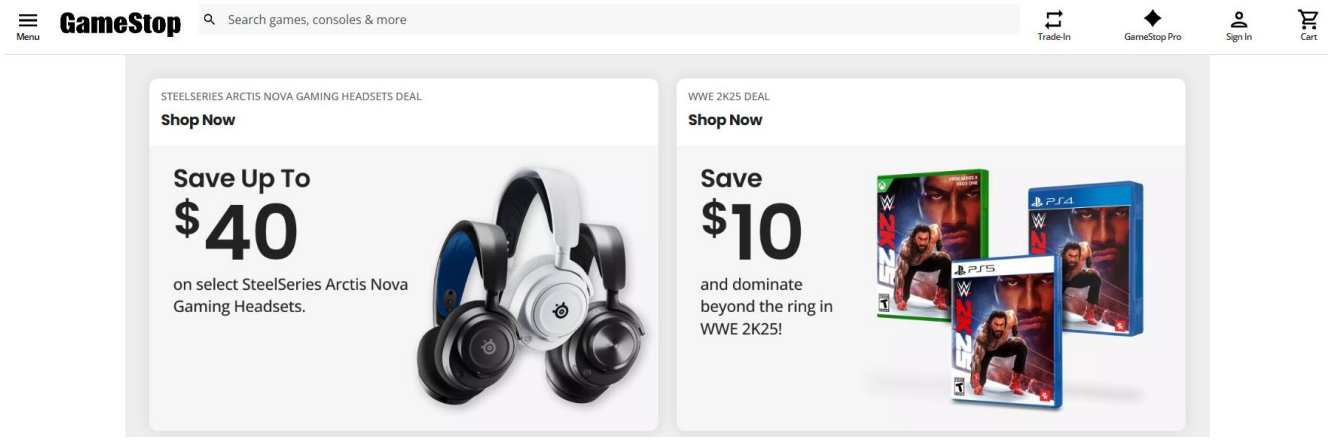

Рис. 1.1 – Головна сторінка Gamestop [6]

Іншим помітним конкурентом є NZXT BLD – онлайн-сервіс для конфігурації та покупки персоналізованих ігрових ПК (рис. 1.2). Цей проєкт демонструє приклад вузькоспеціалізованого інтернет-магазину, орієнтованого на конкретну аудиторію – геймерів, які бажають зібрати потужну ігрову машину.

Користувач має можливість налаштувати конфігурацію ПК відповідно до бажаних ігор, бюджету або технічних вимог. NZXT використовує інтерактивні віджети, що дозволяють в реальному часі бачити вартість, продуктивність системи та приблизний FPS в обраних іграх. Такий підхід забезпечує глибоке занурення в процес покупки, що підвищує рівень задоволення клієнта.

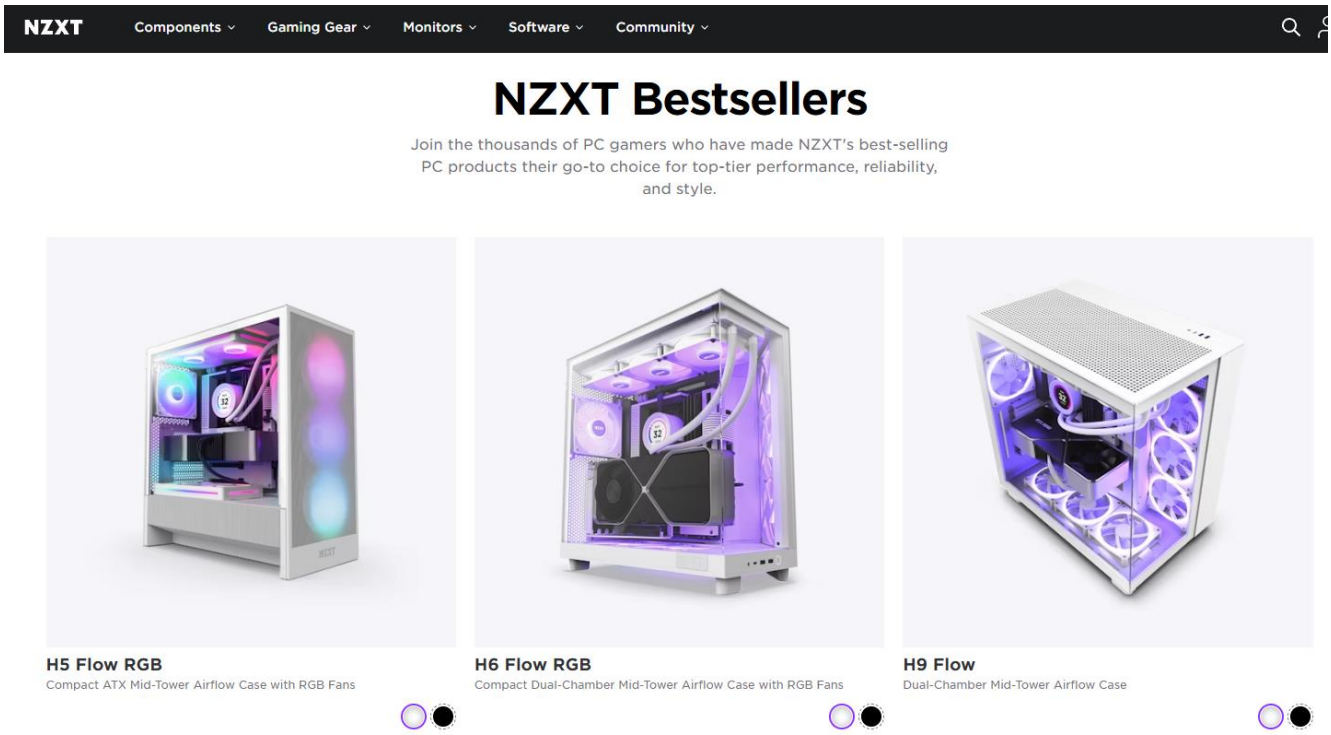

Рис. 1.2 – Інтерфейс конструктора NZXT BLD [5]

Також варто згадати Epic Games Store. Хоча це більше цифрова платформа для дистрибуції ігор, вона активно застосовує елементи маркетингу, схожі до інтернет-магазинів. Epic Games Store [7] пропонує користувачам ексклюзивні пропозиції, бонуси за підписку, акційні розіграші та знижки. Все це – частина стратегії гейміфікації, яка підвищує зацікавленість аудиторії. Платформа має зручний UI, інтеграцію з обліковими записами Epic, і регулярно оновлюється згідно з очікуваннями спільноти.

Усі зазначені конкуренти використовують сучасні технології для реалізації своїх продуктів, зокрема REST API, SSR/SPA підходи, адаптивний дизайн, а також гейміфіковані елементи, що сприяють збільшенню лояльності клієнтів. Проте

більшість із них мають монолітну архітектуру або закриті API, що ускладнює інтеграцію нових сервісів. Саме тому створення нового інтернет-магазину з відкритою архітектурою на Node.js із можливістю легкої інтеграції сторонніх віджетів (наприклад, Elfsight), персоналізації інтерфейсу та використання гейміфікації може стати гнучкішою, сучаснішою альтернативою існуючим рішенням.

1.2 Обґрунтування вибору напрямку дослідження

Створення сучасного інтернет-магазину потребує використання інструментів, які забезпечують гнучкість, масштабованість та простоту інтеграції сторонніх сервісів. З огляду на це, обрано напрямок дослідження, пов’язаний із розробкою серверної частини на основі середовища виконання JavaScript – Node.js, у поєднанні з Elfsight для покращення користувацького досвіду.

Node.js (рис. 1.3) дозволяє створювати високопродуктивні вебзастосунки завдяки неблокуючій архітектурі та подієвоорієнтованій моделі. Ця платформа ідеально підходить для електронної комерції, де важлива швидкість реакції, обробка запитів у режимі реального часу та гнучке управління даними. Крім того, велика кількість доступних модулів через npm дозволяє швидко реалізувати необхідну функціональність – від баз даних до аутентифікації користувачів.

Рис. 1.3 – Логотип node.js [1]

Для розробки клієнтської частини обрано зв’язку HTML, CSS та JavaScript, яка є стандартом у веброзробці. HTML забезпечує структуру вебсторінки, CSS

відповідає за стильове оформлення й адаптивність інтерфейсу, а JavaScript – за інтерактивність і динамічну взаємодію з користувачем. Така комбінація дозволяє створити сучасний, зручний і привабливий інтерфейс магазину, який легко масштабувати та підтримувати.

У якості системи управління базами даних використовується MongoDB – документоорієнтована NoSQL база даних, яка ідеально поєднується з Node.js завдяки гнучкості в роботі з JSON-подібними структурами даних. MongoDB забезпечує високу продуктивність, горизонтальне масштабування та зручність у зберіганні інформації про товари, користувачів, замовлення та інші сутності інтернет-магазину.

Особливу увагу в дослідженні приділено інтеграції віджетів Elfsight. Це готові інструменти, що дозволяють без зайвого кодування додавати на сайт такі компоненти, як відгуки з Google, Instagram-галереї, спливаючі банери, чати підтримки та інші елементи, які підвищують залучення користувачів та довіру до бренду. Elfsight (рис. 1.4) відомий своєю простотою інтеграції, адаптивністю до різних платформ та можливістю швидко змінювати вміст без редагування коду.

Рис. 1.4 – Логотип elfsight [2]

Окрему роль у дослідженні відіграє впровадження елементів гейміфікації, що є актуальною тенденцією в індустрії електронної комерції, особливо у сфері продажу геймерських товарів. Серед таких елементів передбачено використання інтерактивних лічильників досягнень, програми лояльності, колеса фортуни, балів за активність, бейджів за покупки та персоналізованих пропозицій. Впровадження гейміфікації сприяє підвищенню залученості користувачів, зростанню повторних продажів та створенню позитивного іміджу бренду серед цільової аудиторії.

Вибір саме цієї технологічної зв’язки – Node.js для серверної частини, HTML/CSS/JavaScript для фронтенду, MongoDB для бази даних, Elfsight для розширення клієнтського функціоналу та гейміфікаційних рішень – забезпечує сучасний, продуктивний підхід до створення інтернет-магазину геймерських товарів. Це дозволяє зосередитися на розвитку бізнес-логіки, не витрачаючи зайві ресурси на ручну реалізацію вторинних компонентів. У підсумку, обраний напрямок забезпечує швидку розробку, високу якість інтерфейсу, масштабованість проєкту та ефективну взаємодію з користувачем.

1.3 Технічний аспект проблеми

Розробка сучасного інтернет-магазину потребує врахування широкого спектра технічних аспектів, зокрема високої продуктивності, масштабованості, безпеки, інтеграції сторонніх сервісів і зручного користувацького інтерфейсу. Особливої уваги потребують проєкти, орієнтовані на сучасну цільову аудиторію – з елементами інтерактивності, персоналізації та гейміфікації.

З технічної точки зору, проєкт охоплює побудову повноцінної веб-платформи зі зручною клієнтською частиною, серверною логікою на основі Node.js, базою даних MongoDB та інтеграцією сторонніх віджетів через Elfsight. Крім того, передбачається впровадження додаткового функціоналу, такого як динамічна система бонусів, інтерактивних елементів тощо. Це створює підвищене навантаження на систему як з боку фронтенду, так і бекенду, вимагаючи злагодженої структури, гнучкості та добре організованого процесу розробки.

Щоб ефективно організувати технічну реалізацію таких завдань, у рамках даного проєкту обрано Agile-підхід, зокрема його популярну реалізацію – Scrum-методологію. Цей підхід орієнтований на гнучку, ітеративну розробку, що особливо цінно в умовах змінних вимог, потреби в швидкому тестуванні і впровадженні нового функціоналу (рис. 1.5).

Рис. 1.5 – Приклад процесу методології scrum [11]

Суть Scrum-методології полягає в:

- Ітераційності – робота над проєктом поділяється на короткі цикли (спринти) тривалістю 1–2 тижні. Кожен спринт завершується створенням повністю функціонального та протестованого фрагменту системи.
- Пріоритезації завдань – усі задачі фіксуються в беклозі проєкту, з якого формується план кожного спринту відповідно до актуальних потреб і пріоритетів.
- Постійній перевірці та адаптації – після кожного спринту проводиться аналіз результатів і визначення шляхів оптимізації процесу.
- Зосередженості на цінності для користувача – реалізується насамперед той функціонал, який має найбільше значення для кінцевого користувача, а не абстрактні або «в ідеалі потрібні» компоненти.

Технічні переваги використання Scrum:

- Рання валідація технічних рішень – завдяки коротким спринтам, архітектурні рішення можна перевірити й адаптувати ще на ранньому етапі, що знижує ризики серйозних помилок у майбутньому.

- Гнучка інтеграція сторонніх сервісів (наприклад, Elfsight) – завдання з інтеграції віджетів можуть бути винесені в окремий спринт, протестовані й за необхідності змінені без порушення загальної структури проєкту.
- Чітка модульність розробки – кожен компонент (реєстрація, каталог товарів, кошик, гейміфікація) реалізовується як самостійна частина, що спрощує підтримку і масштабування.
- Тестування на кожному етапі – Scrum сприяє регулярному ручному або автоматизованому тестуванню, що дозволяє своєчасно виявляти баги.
- Прозорість у технічному прогресі – завдяки щоденним мітингам (у командній роботі) або власній ретроспективі (у сольній) є чітке бачення, що вже зроблено, а що ще потребує реалізації.

У межах цього проєкту Scrum дозволяє реалізовувати інтернет-магазин поступово, фокусуючись на ключовому функціоналі:

- Спочатку – серверна архітектура Node.js, підключення MongoDB, базові маршрути.
- Далі – розробка каталогу, кошика, реєстрації, обробки замовлень.
- Окремим спринтом – інтеграція Elfsight-віджетів.
- Завершальні етапи – впровадження гейміфікаційної системи та тестування.

Завдяки такому підходу технічна реалізація проєкту відбувається поетапно, контрольовано та із збереженням гнучкості. Scrum забезпечує баланс між стабільністю розробки та можливістю швидко реагувати на зміни – що є критичним чинником успіху у сучасному веброзробленні, особливо в контексті електронної комерції.

2 РОЗРОБКА МОДЕЛІ ТА ПРОГРАМНОГО КОМПЛЕКСУ

2.1 Проєктування вебсистеми геймерського інтернет-магазину

Проєктування – це важливий етап у створенні будь-якої інформаційної системи, зокрема й вебзастосунку. На цьому етапі визначається, як саме працюватиме система, які її основні частини, як вони взаємодіятимуть між собою, і що саме буде реалізовано. Також на цьому етапі формуються технічні рішення, які стануть основою для написання коду. У межах проєктування створюються різні моделі, які допомагають краще уявити майбутню систему ще до початку програмування. Сюди входить модель предметної області, бізнес-модель та архітектура застосунку. Усі ці елементи дозволяють зрозуміти, як повинна виглядати і функціонувати система в цілому.

2.1.1 Розробка моделі предметної області

На цьому підетапі основна увага приділяється тому, щоб визначити головні частини системи, з чого вона складається, і як ці частини пов’язані між собою. Така модель допомагає краще зрозуміти, які об’єкти будуть використовуватись у системі, які між ними є зв’язки, і як це все працюватиме разом. Це потрібно для того, щоб уникнути плутанини під час подальшої реалізації.

Для кращого розуміння внутрішньої структури вебсистеми була створена діаграма пакетів (рис. 2.1) (Package Diagram), яка показує основні модулі (пакети) програми, їхню роль та залежності між ними. Завдяки цій діаграмі можна візуально побачити, як логічно поділена система, які частини за що відповідають, і як вони взаємодіють.

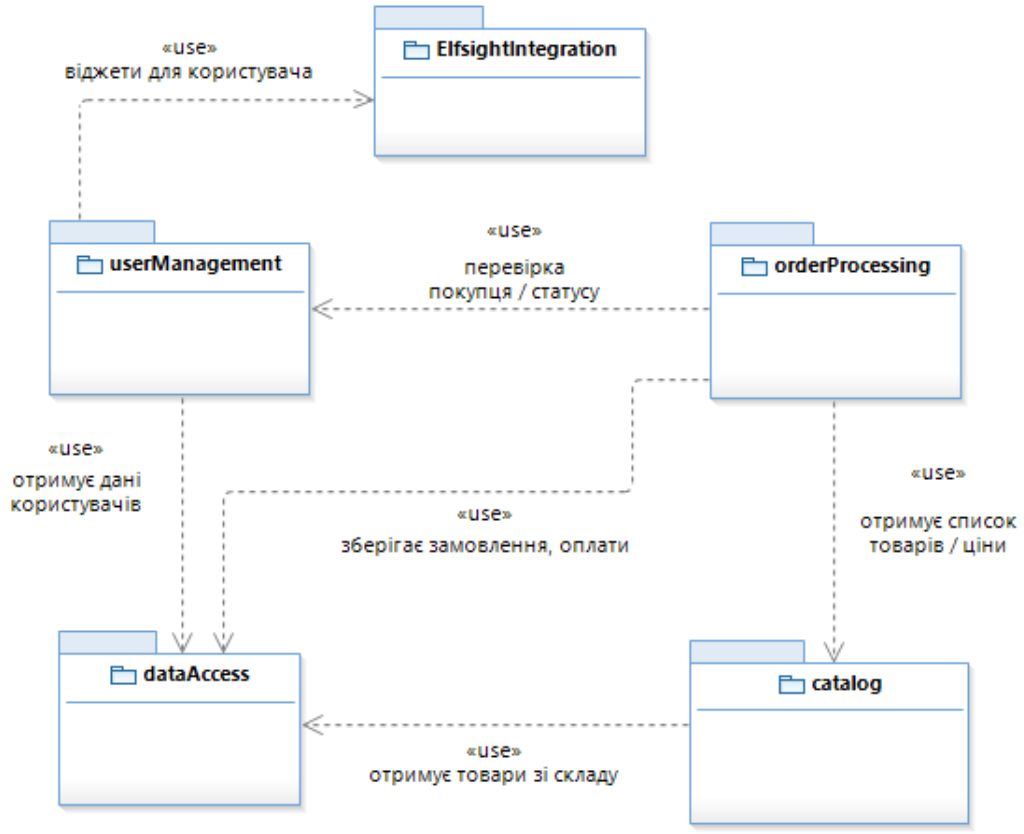

Рис. 2.1 – Діаграма пакетів ключових функціональних частин

На діаграмі виділено наступні пакети:

- userManagement – відповідає за керування користувачами (реєстрація, авторизація, профіль);
- orderProcessing – обробка замовлень, перевірка їх статусу;
- catalog – управління каталогом товарів, доступ до списку товарів та їх характеристик;
- dataAccess – централізований доступ до бази даних, збереження інформації про користувачів, замовлення, товари;
- ElfsightIntegration – інтеграція з віджетами стороннього сервісу Elfsight, які використовуються для відображення додаткової інформації (відгуки, соціальні мережі, онлайн-чат тощо).

2.1.2 Розробка бізнес моделі

Для кращого розуміння функціональності веб-застосунку та взаємодії користувачів із системою було побудовано діаграму (рис. 2.2) варіантів використання (Use Case Diagram). Вона візуалізує ключові сценарії поведінки для двох основних акторів – користувача та адміністратора. Діаграма допомагає сформувати чітке уявлення про основні дії, які можуть виконуватись у системі, та залежності між ними.

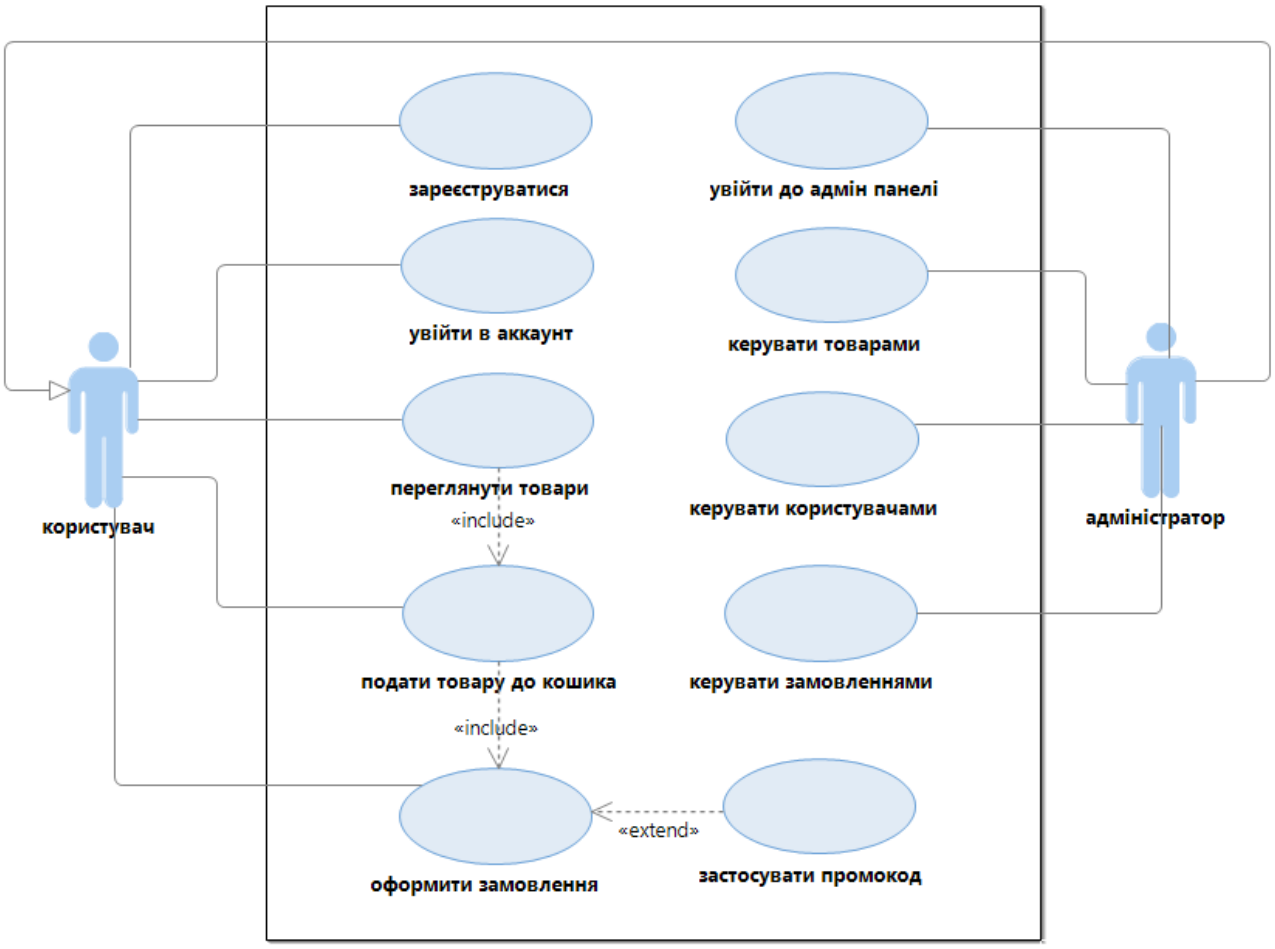

Рис. 2.2 – Діаграма варіантів використання ключових сценаріїв

Учасники системи (Actors):

- Користувач – звичайний відвідувач/покупець веб-застосунку.

- Адміністратор – користувач із розширеними правами доступу, що керує платформою.

Основні варіанти використання для користувача:

- Зареєструватися – створити новий обліковий запис.
- Увійти в акаунт – авторизуватись у системі.
- Переглянути товари – отримати список доступних товарів.
- Подати товар до кошика – додати вибраний товар у кошик.
- Оформити замовлення – завершити покупку та передати її на обробку. Включає додавання товарів у кошик (include). Може доповнюватись застосуванням промокоду (extend).
- Застосувати промокод – опційна дія під час оформлення замовлення для знижки.

Основні варіанти використання для адміністратора:

- Увійти до адмін панелі – авторизуватись як адміністратор.
- Керувати товарами – додавати, редагувати або видаляти товари.
- Керувати користувачами – переглядати або редагувати облікові записи.
- Керувати замовленнями – обробляти замовлення користувачів.

2.1.3 Проєктування архітектури

У сучасній розробці веб-застосунків однією з найпоширеніших моделей є клієнт-серверна архітектура (рис. 2.3). Вона забезпечує розподіл відповідальності між різними компонентами системи та дозволяє централізовано керувати даними й логікою обробки. У межах клієнт-серверної моделі клієнтська частина (frontend) виступає посередником між користувачем та сервером, виконуючи запити та відображаючи отриману інформацію. Серверна частина (backend) відповідає за обробку цих запитів, виконання бізнес-логіки, збереження та модифікацію даних, а також взаємодію з базами даних і зовнішніми сервісами.

Рис. 2.3 – Зв’язки у клієнт-серверній архітектурі [12]

Застосунок, що реалізується в межах даного проєкту, базується на трирівневій клієнт-серверній архітектурі, що складається з наступних рівнів:

- Клієнтський рівень (Presentation layer) – інтерфейс користувача, реалізований у вигляді адаптивного веб-інтерфейсу з інтегрованими Elfsight-віджетами (відгуки, галереї, форми). Цей рівень забезпечує зручну взаємодію з користувачем та передає запити до серверу за допомогою HTTP-протоколу. Основні технології – HTML, CSS, JavaScript, Elfsight SDK.
- Серверний рівень (Application logic layer) – основна бізнес-логіка застосунку, реалізована на платформі Node.js з використанням фреймворку Express.js. Сервер приймає запити від клієнта, виконує валідацію, обробляє дані, викликає відповідні сервіси або звертається до бази даних, а також взаємодіє з API Elfsight. Цей рівень реалізує REST API, а також контролює доступ користувачів, управління замовленнями, бонусними програмами тощо.
- Рівень даних (Data layer) – база даних MongoDB, де зберігаються всі основні об'єкти предметної області: користувачі, товари, замовлення, відгуки, налаштування віджетів тощо. База даних є гнучкою та масштабованою, що дозволяє зберігати як стандартні структуровані дані, так і вкладені об'єкти, властиві JSON-документам.

Архітектура також враховує можливість майбутньої інтеграції додаткових сторонніх сервісів або модулів гейміфікації. Elfsight-віджети підключаються динамічно через серверні налаштування або за допомогою спеціального адміністративного інтерфейсу.

Після визначення архітектури системи було побудовано концептуальну UML діаграму класів (рис. 2.4), яка відображає ключові компоненти бізнес-логіки інтернет-магазину. Діаграма демонструє основні програмні сутності, їхні атрибути, методи та зв’язки між ними.

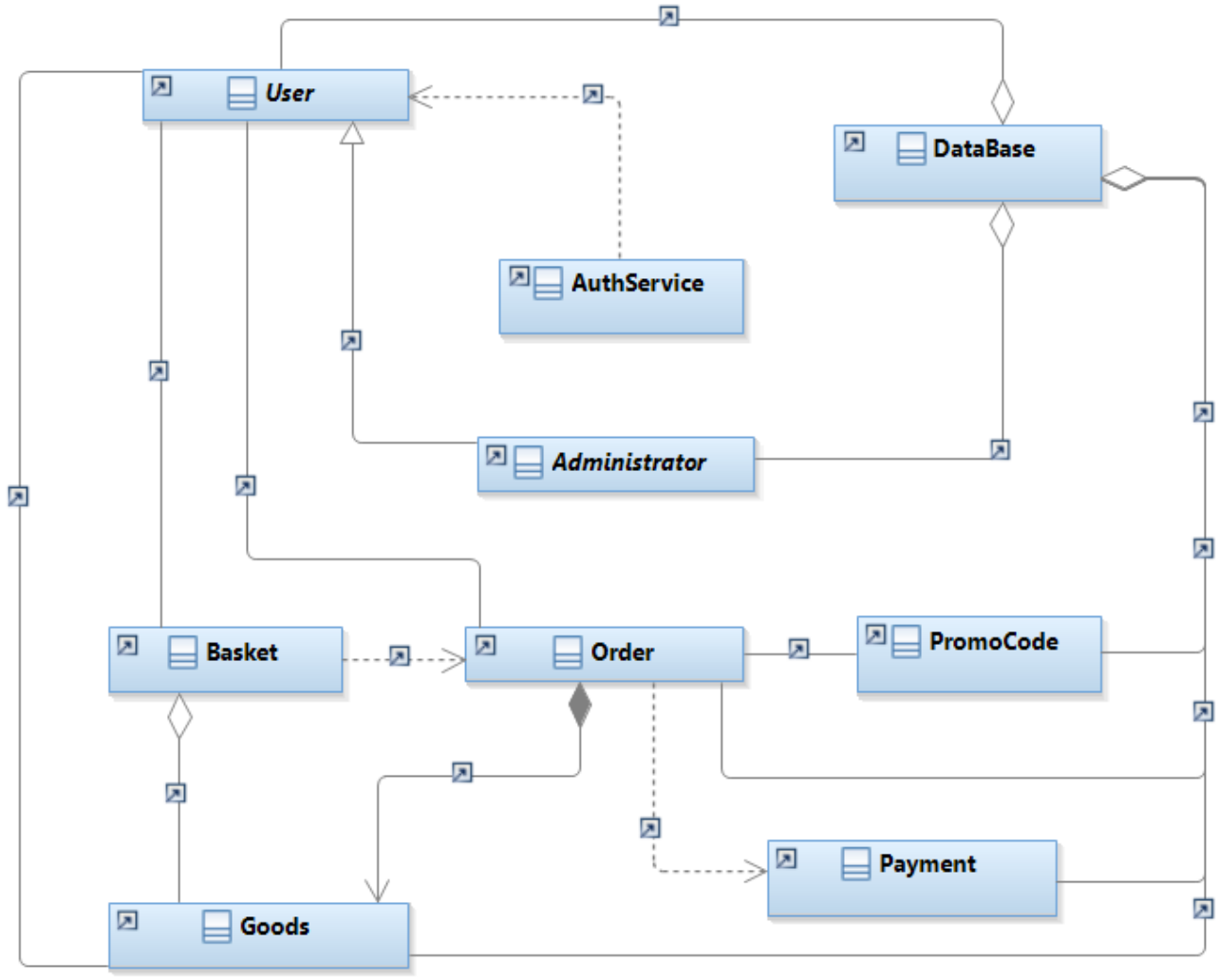

Рис. 2.4 – Концептуальна діаграма класів інтернет-магазину

На діаграмі класів зображено структуру програмного забезпечення інтернет-магазину, включаючи основні бізнес-сутності (користувач, товар, замовлення, кошик тощо), а також допоміжні компоненти для обробки платежів, промокодів, адміністрування та взаємодії з базою даних. Відображено зв’язки між класами, їх атрибути та ключові методи.

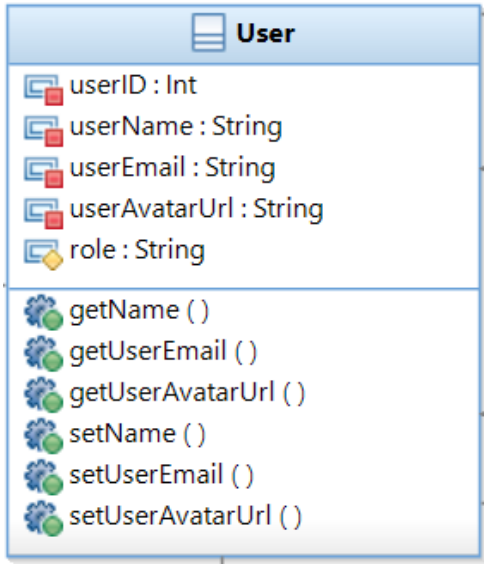

Рис. 2.5 – Клас User

Клас User (рис. 2.5): Зберігає інформацію про користувача: ім’я, електронну пошту, аватар, роль у системі. Дозволяє отримати та оновити персональні дані.

Взаємодія:

- Має один кошик (Basket) для зберігання обраних товарів.
- Може створювати замовлення (Order) і переглядати їх.
- Має доступ до списку товарів (Goods) – наприклад, переглядає асортимент.

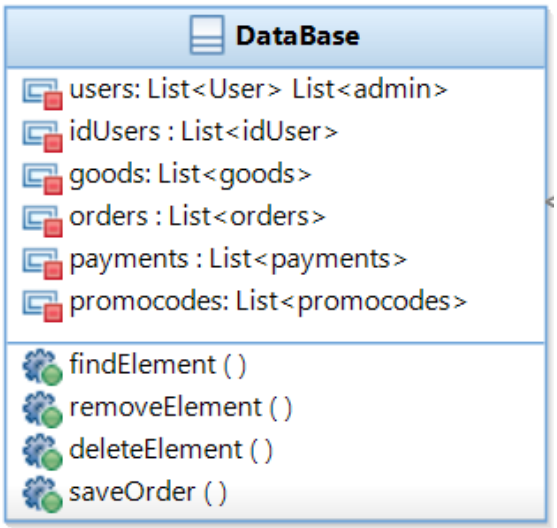

Рис. 2.6 – Клас Database

Клас Database (рис. 2.6): Універсальний клас-репозиторій. Зберігає всі об’єкти системи у вигляді списків та реалізує базові CRUD-операції (пошук, видалення, збереження). Пов’язаний з усіма ключовими класами: User, Administrator, Goods, Order, Payment, PromoCode.

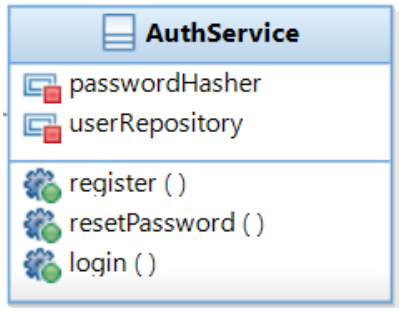

Рис. 2.7 – Клас AuthService

Клас AuthService (рис. 2.7): Забезпечує аутентифікацію та авторизацію користувачів. Містить логіку реєстрації, входу в систему та скидання пароля. Використовує User та userRepository.

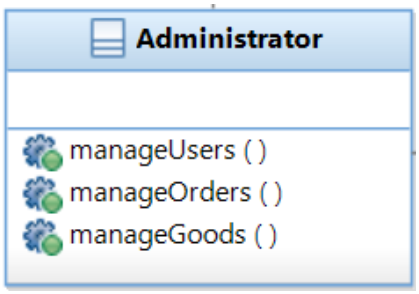

Рис. 2.8 – Клас Administrator

Клас Administrator (рис. 2.8): Представляє користувача з розширеними правами. Відповідає за управління даними в системі.

Взаємодія:

- Взаємодіє з DataBase для додавання/редагування/видалення товарів, замовлень і користувачів.
- Наслідує функціональність User.

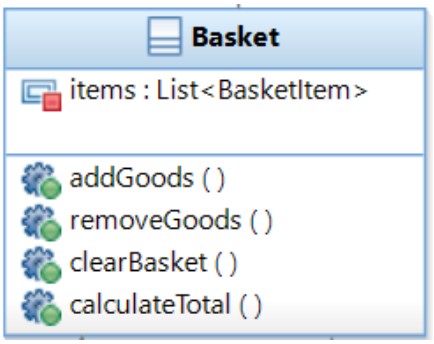

Рис. 2.9 – Клас Basket

Клас Basket (рис. 2.9): Зберігає список обраних товарів користувача. Реалізує функції додавання, видалення товарів та очищення кошика. Може обчислювати загальну суму замовлення.

Взаємодія:

- Належить до одного User.
- Містить список об’єктів Goods.

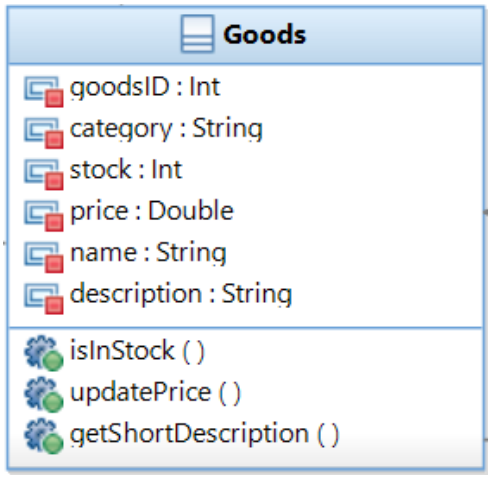

Рис. 2.10 – Клас Goods

Клас Goods (рис. 2.10): Представляє товар: назву, категорію, опис, ціну, наявність на складі. Дозволяє оновлювати інформацію про товар.

Взаємодія:

- Доступний для перегляду користувачем (User).
- Може бути доданий у Basket або включений до Order.
- Може редагуватися через Administrator.

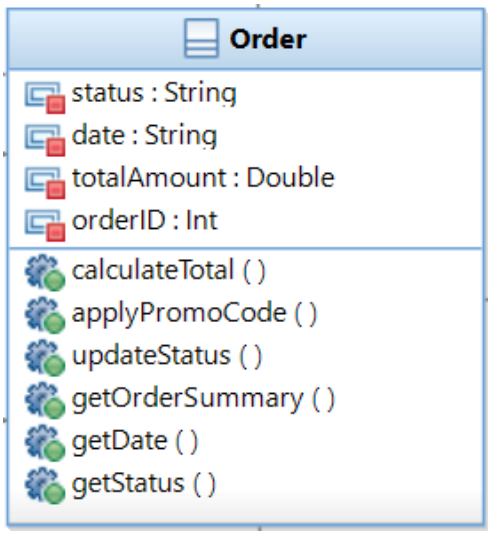

Рис. 2.11 – Клас Order

Клас Basket (рис. 2.11): Зберігає інформацію про замовлення: дата, статус, сума. Містить методи для застосування промокодів, оновлення статусу та підрахунку вартості.

Взаємодія:

- Створюється користувачем (User) на основі кошика (Basket).
- Містить посилання на список товарів (Goods) і пов’язаний з Payment та PromoCode.

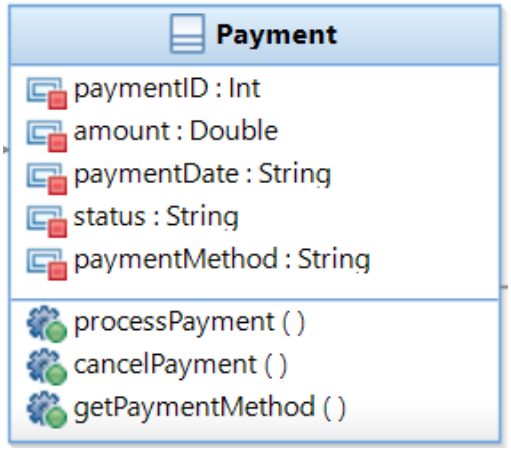

Рис. 2.12 – Клас Payment

Клас Payment (рис. 2.12): Відповідає за обробку платежів. Містить інформацію про дату, суму, статус і спосіб оплати.

Взаємодія:

- Пов'язаний з певним замовленням (Order).
- Реалізує обробку, скасування і перевірку способу оплати.

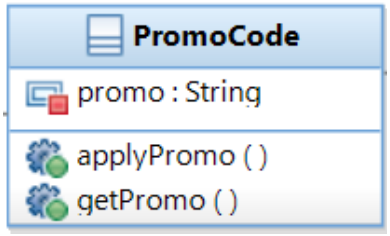

Рис. 2.13 – Клас Promocode

Клас Promocode (рис. 2.13): Дозволяє користувачу застосовувати знижки до замовлення. Прив’язується до Order, впливаючи на фінальну суму.

Після побудови діаграми класів, яка описує статичну структуру системи та взаємозв’язки між основними об’єктами, наступним кроком є моделювання динамічної поведінки системи. уло створено діаграму послідовності (рис. 2.14), що ілюструє взаємодію між об’єктами під час процесу оформлення замовлення.

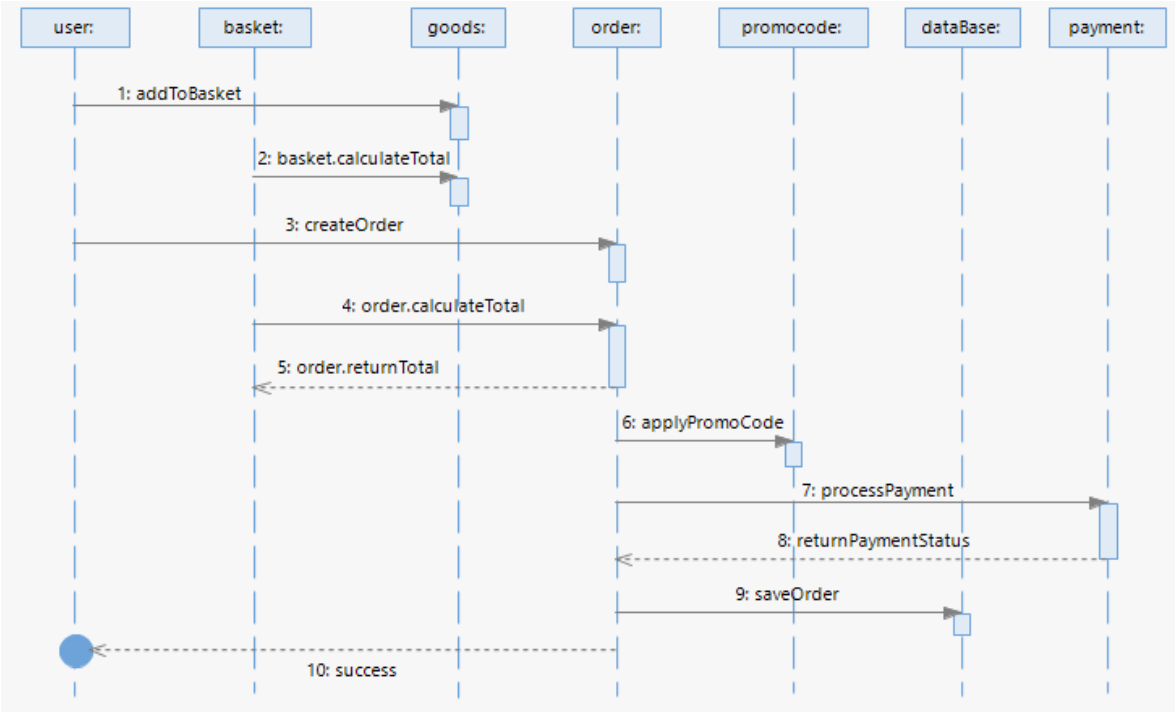

Рис. 2.14 – Діаграма послідовності

У даному сценарії розглядається процес оформлення замовлення користувачем у системі електронної комерції. Спочатку користувач додає товар до кошика за допомогою методу addToBasket. Далі об'єкт basket викликає метод calculateTotal, щоб обчислити загальну вартість товарів, для чого звертається до об'єкта goods з метою отримання актуальних даних про товари. Після цього виконується створення нового замовлення через метод createOrder, який ініціює об'єкт order.

Об'єкт order проводить додаткове обчислення загальної суми з урахуванням можливих податків чи доставки, використовуючи метод calculateTotal, і повертає підсумкову суму через returnTotal. Далі замовлення перевіряє можливість застосування знижки – викликається метод applyPromoCode об’єкта promocode. Якщо промокод застосовується, сума замовлення коригується відповідно до умов акції.

Після цього система переходить до етапу оплати. Метод processPayment передає дані в платіжну систему через об'єкт payment. Після обробки транзакції система повертає результат оплати методом returnPaymentStatus, який підтверджує успішність або неуспішність платежу. У разі успішної оплати замовлення зберігається в базу даних за допомогою методу saveOrder, який взаємодіє з об’єктом dataBase. Завершальним кроком є повідомлення користувача про успішне оформлення замовлення – система повертає відповідь success.

2.2 Конструювання e-commerce платформи на Node.js з Elfsight

Етап конструювання включає безпосередню реалізацію програмного продукту відповідно до спроєктованих моделей. Цей крок охоплює написання коду, розробку інтерфейсу користувача, підключення до бази даних, створення API для взаємодії клієнта з сервером, а також проведення тестування і оцінку

працездатності системи. У результаті отримано повноцінно функціонуючий застосунок, готовий до використання користувачами.

2.2.1 Реалізація ключових класів

У процесі розробки застосунку було реалізовано як серверну частину (бекенд), так і клієнтську частину (фронтенд). Бекенд був розгорнутий на платформі Node.js з використанням фреймворку Express.js, а базу даних MongoDB було підключено через бібліотеку Mongoose. Після реалізації всієї логіки застосунок успішно запущено: сервер було підключено до бази даних, а клієнтська частина почала взаємодіяти з сервером через HTTP-запити.

Основне завдання серверної частини полягає в обробці запитів від клієнта, керуванні ресурсами (користувачами, товарами, замовленнями), реалізації механізмів автентифікації користувачів та забезпеченні зберігання і зчитування даних з бази даних. Структура бекенду (рис. 2.15) організована відповідно до шаблону MVC (Model-View-Controller), що дозволило чітко структурувати програмну логіку, підвищити масштабованість системи та полегшити її супровід.

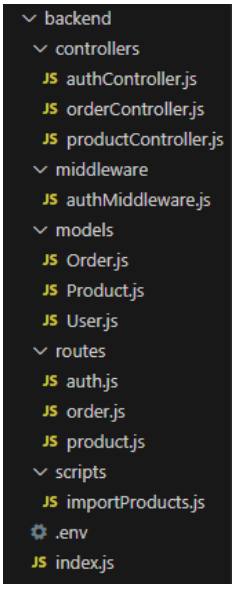

Рис. 2.15 – Дерево файлів JS використані у бекенді

Ось наведений опис основних js файлів та їх призначення. Файл controllers/authController.js – містить логіку для реєстрації та входу користувачів. Обробляє введені дані, виконує перевірку автентичності, генерує JWT-токени для авторизованих сесій.

```
 5 > const registerUser = async (req, res) => {···
22   };
23 > const loginUser = async (req, res) => {···
49   };
50 > const getUserProfile = async (req, res) => {···
60   };
61 > const updateProfile = async (req, res) => {···
90   };
91 > const getAllUsers = async (req, res) => {···
98   };
99 > const updateUserByAdmin = async (req, res) => {···
118  };
119
120  module.exports = { registerUser, loginUser, getUserProfile, updateProfile, getAllUsers, updateUserByAdmin};
```

Рис. 2.16 – Основні функції контролеру аутентифікації

Файл controllers/orderController.js – відповідає за створення нових замовлень та отримання існуючих. Отримує дані з моделі Order і взаємодіє з нею для збереження чи вибірки замовлень, які прив’язані до конкретного користувача.

```
1   // controllers/orderController.js
2   const Order = require('../models/Order');
3
4 > const createOrder = async (req, res) => {···
21  };
22 > const getUserOrders = async (req, res) => {···
29  };
30 > const payOrder = async (req, res) => {···
63  };
64
65  module.exports = { createOrder, getUserOrders, payOrder };
```

Рис. 2.17 – Основні функції контролеру замовлень

Файл controllers/productController.js – реалізує функціонал для створення, читання, оновлення та видалення товарів. Забезпечує взаємодію з моделлю Product та виконує перевірки валідності введених даних.

```
const Product = require('../models/Product');

> const addManyProducts = async (req, res) => {···
};
> const getAllProducts = async (req, res) => {···
};
> const getProductById = async (req, res) => {···
};
> const addProduct = async (req, res) => {···
};
> const deleteProduct = async (req, res) => {···
};
> const updateProduct = async (req, res) => {···
};
> const addCommentToProduct = async (req, res) => {···
};

module.exports = { addManyProducts, getAllProducts, getProductById,
  addProduct, deleteProduct, updateProduct, addCommentToProduct};
```

Рис. 2.18 – Основні функції контролеру товару/товарів

Файл middleware/authMiddleware.js – проміжний обробник, який перевіряє наявність та дійсність JWT-токена у запиті. Забезпечує захист приватних маршрутів, доступ до яких мають лише авторизовані користувачі.

```
const jwt = require('jsonwebtoken');
const User = require('../models/User');

const protect = async (req, res, next) => {
  let token;

  if (req.headers.authorization?.startsWith('Bearer')) {
    try {
      token = req.headers.authorization.split(' ')[1];
      const decoded = jwt.verify(token, process.env.JWT_SECRET);
      const user = await User.findById(decoded.id).select('-password');

      if (!user) return res.status(401).json({ message: 'Користувача не знайдено' });

      req.user = user;
      next();
    } catch (error) {
      console.error('JWT помилка:', error.message);
      res.status(401).json({ message: 'Невалідний токен' });
    }
  } else {
    res.status(401).json({ message: 'Немає токена' });
  }
};

> const isAdmin = (req, res, next) => {···
};

module.exports = { protect, isAdmin };
```

Рис. 2.19 – Функціонал проміжного обробнику

Файл models/User.js – визначає схему користувача з такими полями як email, пароль (зашифрований), роль. Містить методи для перевірки пароля та генерації токена.

```
const userSchema = new mongoose.Schema({
  username: String,
  email: { type: String, unique: true },
  password: String,
  avatarUrl: String,
  address: String,
  paymentMethod: String,
  role: { type: String, enum: ['user', 'admin'], default: 'user' }
});

// Хешування паролю
userSchema.pre('save', async function (next) {
  if (!this.isModified('password')) return next();
  this.password = await bcrypt.hash(this.password, 10);
  next();
});
```

Рис. 2.20 – Модель юзера

Файл models/Product.js – описує структуру товару: назва, опис, ціна, категорія, зображення. Визначає вимоги до типів і обов’язковості полів.

```
const commentSchema = new mongoose.Schema({
  username: String,
  avatarUrl: String,
  text: String,
  createdAt: { type: Date, default: Date.now }
});

const productSchema = new mongoose.Schema({
  title: { type: String, required: true },
  description: String,
  price: { type: Number, required: true },
  category: String,
  image: String,
  images: [String],
  rating: Number,
  sales: Number,
  manufacturer: String,
  origin: String,
  model: String,
  specs: { type: Map, of: String },
  comments: [commentSchema],
  hasDetails: { type: Boolean, default: false }
}, { timestamps: true });
```

Рис. 2.21 – Модель товару

Файл models/Order.js – містить схему замовлення, що включає користувача (reference на User), список товарів, дату та статус замовлення. Може зберігати додаткову інформацію, як-от сума чи спосіб оплати.

```
const orderSchema = new mongoose.Schema({
  user: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },
  items: [ ···
  ],
  address: { ···
  },
  total: Number,
  originalTotal: Number,
  status: { ···
  },
  paymentMethod: String,
  trackingNumber: String,
  createdAt: { type: Date, default: Date.now }
});
```

Рис. 2.22 – Модель замовлення

Файл routes/auth.js – налаштовує маршрути для реєстрації (/register) і входу користувачів (/login). Передає запити у відповідні методи з authController.

```
const express = require('express');
const router = express.Router();
const { registerUser, loginUser, getUserProfile,
    updateProfile, getAllUsers, updateUserByAdmin } = require('../controllers/authController');
const { protect, isAdmin } = require('../middleware/authMiddleware');

// Звичайні юзери
router.post('/register', registerUser);
router.post('/login', loginUser);
router.get('/profile', protect, getUserProfile);
router.patch('/profile', protect, updateProfile);

// Адмінські
router.get('/admin/users', protect, isAdmin, getAllUsers);
router.put('/admin/users/:id', protect, isAdmin, updateUserByAdmin);
```

Рис. 2.23 – Маршрути аутентифікації

Файл routes/order.js – маршрути для створення нового замовлення та отримання замовлень користувача. Захищені middleware’ом, що перевіряє токен.

```
const express = require('express');
const router = express.Router();
const { createOrder, getUserOrders, payOrder} = require('../controllers/orderController');
const { protect } = require('../middleware/authMiddleware');

router.post('/', protect, createOrder);
router.get('/', protect, getUserOrders);
router.put('/:id/pay', protect, payOrder);

module.exports = router;
```

Рис. 2.24 – Маршрути замовлень

Файл routes/product.js – містить маршрути для створення (POST), отримання (GET), оновлення (PUT) та видалення (DELETE) товарів. Деякі з них захищені авторизацією (для адміністратора).

```
const express = require('express');
const router = express.Router();
const { addManyProducts, getAllProducts, getProductById, addProduct,
    deleteProduct, updateProduct, addCommentToProduct} = require('../controllers/productController');
const { protect, isAdmin } = require('../middleware/authMiddleware');

router.get('/', getAllProducts);
router.post('/add-many', protect, isAdmin, addManyProducts);
router.get('/:id', getProductById);
router.post('/', protect, isAdmin, addProduct);
router.post('/:id/comments', protect, addCommentToProduct);
router.delete('/:id', protect, isAdmin, deleteProduct);
router.put('/:id', protect, isAdmin, updateProduct);
```

Рис. 2.25 – Маршрути товарів

Файл scripts/importProducts.js – скрипт для імпорту тестових даних товарів у базу даних. Автоматизує початкове наповнення системи продуктами.

```
mongoose.connect(process.env.MONGO_URI, {
  useNewUrlParser: true,
  useUnifiedTopology: true,
}).then(async () => {
  console.log('✅ Підключено до MongoDB');
  await Product.deleteMany();
  await Product.insertMany(products);
  console.log('✅ Товари імпортовано');
  process.exit();
}).catch(err => {
  console.error('❌ Помилка підключення:', err);
  process.exit(1);
});
```

Рис. 2.26 – Скрипт імпорту початкових товарів

Файл index.js – головний файл запуску сервера. Ініціалізує Express-додаток, підключає middleware, маршрути, встановлює з’єднання з MongoDB і запускає сервер на вказаному порту.

```
// Middleware
app.use(cors());
app.use(express.json());

// Підключення до MongoDB
mongoose.connect(process.env.MONGO_URI, {···
})
.then(() => console.log('✅ Підключено до MongoDB'))
.catch((err) => console.error('❌ Помилка підключення до MongoDB:', err));

// Роути
app.use('/api/auth', authRoutes);
app.use('/api/products', productRoutes);
app.use('/api/orders', orderRoutes);

// Старт сервера
app.listen(PORT, () => {···
});
```

Рис. 2.27 – Код запуску серверу та встановлення з’єднань

2.2.2 Розробка GUI

Продовжуючи реалізацію серверної частини, було створено клієнтську частину (фронтенд) застосунку. Інтерфейс користувача реалізовано за допомогою технологій HTML, CSS та JavaScript.

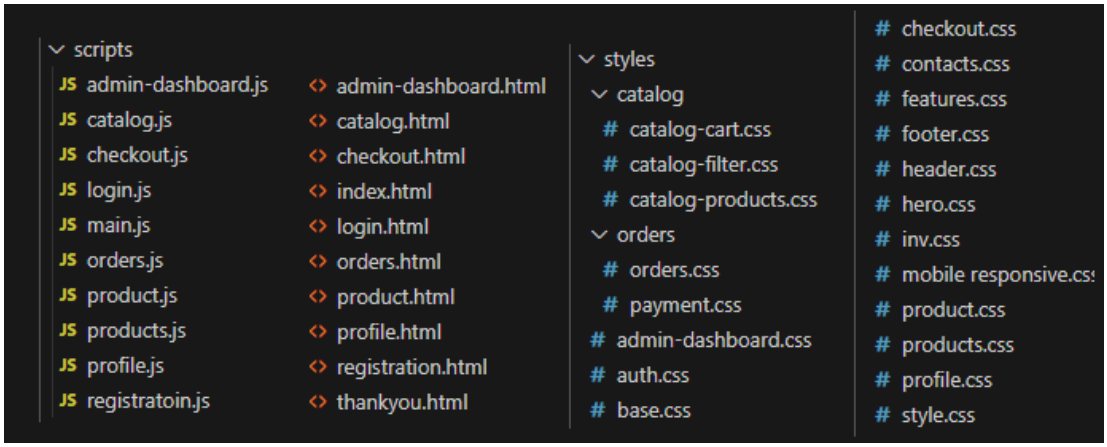

Рис. 2.28 – Дерева файлів проєкту

Результати розробки інтерфейсу/дизайну будуть показані у пункті 2.2.3. Опис взаємодії файлів та сторінок:

- Головна сторінка index.html формує перше враження про застосунок і містить логотип, головну навігацію, секцію з вітальним банером, кнопкою переходу до каталогу, секцію з перевагами, популярними товарами, контактною інформацією, інтерактивною картою, формою зворотного зв'язку та футером. Уся візуальна частина оформлена за допомогою окремих стилів у файлах hero.css, features.css, contacts.css, footer.css, header.css і base.css, що забезпечують адаптивність, відступи, анімації й кольорову палітру. Функціональна логіка реалізована в main.js (рис. 2.29), де відбувається перевірка наявності авторизації, динамічне завантаження товарів, активація промокодів та керування переходами між секціями сторінки.

```
document.addEventListener("DOMContentLoaded", () => {···
});
document.addEventListener('DOMContentLoaded', () => {
  const observer = new MutationObserver(() => {
    const promoEl = document.querySelector('.Coupon__Value-sc-11a9188b-4');
    const buttons = document.querySelectorAll('.ButtonBase__Overlay-sc-c65ab2ce-4.gzBANg');

    if (promoEl && buttons.length > 0) {
      const promoCode = promoEl.textContent.trim();
      console.log('🎯 Промокод знайдено:', promoCode);
      let clicked = false;
      buttons.forEach(button => {···
      });
      if (clicked) {
        observer.disconnect();
      }
    }
  });
  observer.observe(document.body, { childList: true, subtree: true });
});
```

Рис. 2.29 – Скрипт найдених промокодів у віджеті

- Сторінка авторизації login.html включає форму входу з двома полями – email і пароль, а також кнопку підтвердження та посилання на реєстрацію. За візуальне оформлення відповідає файл auth.css, що стилізує форму з урахуванням адаптивності та загального дизайну платформи. Взаємодія з сервером реалізована в login.js (рис. 2.30), де після заповнення форми відбувається перевірка даних через

API, збереження токена авторизації у localStorage та автоматичне перенаправлення до особистого кабінету в разі успіху.

```
  try {
    const res = await fetch('http://localhost:5000/api/auth/login', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ email, password })
    });
    const data = await res.json();
    if (res.ok) {
      localStorage.setItem('token', data.token);
      window.location.href = 'profile.html';
    } else {···
    }
  } catch (err) {
    console.error('Помилка входу:', err);
    showMessage('Сталася помилка при вході', 'error');
  }
});

function showMessage(text, type) {···
}
const token = localStorage.getItem('token');
if (token) {···
}
```

Рис. 2.30 – Скрипт логіну

- Реєстраційна сторінка registration.html побудована подібно до сторінки входу й використовує ті самі стилі auth.css. Вона містить розширену форму з кількома полями – ім’я, email, пароль і підтвердження пароля. Скрипт registration.js (рис. 2.31) відповідає за валідацію введених даних, надсилання їх до сервера та виведення результату на екран або повідомлення про успішну реєстрацію/помилку.

```
if (password !== confirmPassword) {
  return alert('Паролі не співпадають!');
}
try {
  const res = await fetch('http://localhost:5000/api/auth/register', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({ username, email, password })
  });

  const data = await res.json();
  alert(data.message || 'Реєстрація успішна!');
  window.location.href = 'login.html';
} catch (err) {
  console.error('Помилка реєстрації:', err);
  alert('Сталася помилка при реєстрації');
}
```

Рис. 2.31 – Скрипт реєстрації

- Особистий кабінет profile.html відображає інформацію про користувача, включаючи фото, ім’я, пошту, статус, прогрес заповнення профілю, адресну та платіжну інформацію, дату створення акаунта, а також активність. Візуальна частина реалізована через файли profile.css і style.css, що відповідають за макет блоків, адаптивність та колірну схему. Уся логіка обробки даних реалізована в profile.js (рис. 2.32), де скрипт отримує інформацію про користувача з API, відображає її на сторінці, дозволяє редагування через модальне вікно, зберігає зміни та керує гостьовим режимом, якщо користувач не авторизований.

```
> if (token) {···
> } else {···
  }
  async function loadProfile() {
    try {
      const res = await fetch('http://localhost:5000/api/auth/profile', {
        headers: { Authorization: `Bearer ${token}` },
      });

      if (!res.ok) throw new Error('Помилка авторизації');
      const user = await res.json();

      usernameEl.textContent = user.username;
      emailEl.textContent = user.email;
      avatarEl.src = user.avatarUrl || avatarEl.src;
      addressEl.textContent = user.address || 'Не вказано';
      paymentEl.textContent = user.paymentMethod || 'Не вказано';
      document.querySelector('.user-status').textContent = 'Активний';
>     if (user.role === 'admin') {···
      } else {
        document.getElementById('user-buttons').style.display = 'block';
      }
>   } catch (err) {···
    }
  }
> editButton.onclick = () => {···
  };
> cancelBtn.onclick = () => {···
  };
> saveBtn.onclick = async () => {···
  };
```

Рис. 2.32 – Скрипти та функції профілю

- Каталог товарів, який реалізовано у catalog.html, надає користувачеві можливість переглядати всі товари з фільтрами, пошуком і додаванням у кошик. Дизайн сторінки створено за допомогою файлів catalog-products.css, catalog-filter.css та catalog-cart.css, які забезпечують візуальну структуру карток товарів,

фільтрів за ціною й категоріями, а також вигляд кошика. Основна логіка розміщена в catalog.js (рис. 2.33, 2.34), який динамічно підтягує список товарів з сервера, формує DOM-елементи, обробляє додавання у кошик, відображення модальних вікон і взаємодію з localStorage.

Рис. 2.33 – Функціонал сторінки каталогу

Рис. 2.34 – Продовження коду сторінки каталогу

- Сторінка окремого товару product.html показує детальну інформацію про конкретний товар – велика галерея фото, назва, ціна, опис, технічні характеристики, рейтинг, відгуки та кнопка додавання у кошик. Візуальне оформлення виконується через product.css, який відповідає за структуру блоків, шрифти, відображення зображень і стилізацію форм. Функціональність реалізована

в product.js (рис. 2.35), який витягує ID товару з URL, робить запит до API, рендерить деталі товару, керує рейтингом, формою відгуків, перемикає фото в галереї, та забезпечує взаємодію з кошиком.

```
document.addEventListener('DOMContentLoaded', async () => {···
});
let currentIndex = 0;
let images = [];
function updateGallery(index) {···
}
images = product.images && product.images.length ? product.images : [product.image];
updateGallery(0);
const thumbnailsContainer = document.getElementById('thumbnails');
thumbnailsContainer.innerHTML = images.map((img, index) => `···
`).join('');
document.querySelector('.gallery-nav.prev').addEventListener('click', () => {···
});
document.querySelector('.gallery-nav.next').addEventListener('click', () => {···
});
```

Рис. 2.35 – Скрипт витягу товарів з БД та їх відображення

- Сторінка оформлення замовлення checkout.html дає змогу переглянути вміст кошика, ввести адресу доставки, підтвердити замовлення та завершити покупку. Дизайн сторінки виконано через checkout.css, де стилізуються всі елементи – блок товарів, форма доставки, загальна сума, повідомлення про помилки. Логіка в checkout.js (рис. 2.36) перевіряє, чи користувач авторизований, витягує список товарів, дозволяє ввести необхідні дані, надсилає замовлення на сервер і після підтвердження очищає кошик і перенаправляє на сторінку подяки.

```
cart.forEach(item => {
  const li = document.createElement('li');
  li.classList.add('cart-item');
  li.innerHTML = `
    <img src="${item.image}" alt="${item.title}" class="cart-item-image" />
    <div class="cart-item-info">
      <h3>${item.title}</h3>
      <p>${item.price} ₴</p>
    </div>
  `;
  checkoutCart.appendChild(li);
  total += item.price;
});
checkoutTotalPrice.textContent = `Сума: ${total} ₴`;
document.querySelector('.checkout-form').addEventListener('submit', async (e) => {···
});
```

Рис. 2.36 – Скрипт кошику для витягу товарів та підсумків суми

- Сторінка замовлень orders.html призначена для перегляду історії покупок користувача, де відображається список замовлень з інформацією про оплату, статус, дату та склад. За зовнішній вигляд відповідають стилі orders/orders.css і orders/payment.css, які формують таблиці, блоки з деталями, модальні вікна для оплати. Весь функціонал забезпечується через orders.js (рис. 2.37), який після перевірки авторизації отримує замовлення через API, розподіляє їх за статусом, дозволяє користувачу ввести платіжні дані в модальному вікні, активувати промокод і оновити статус після оплати.

```
document.addEventListener('DOMContentLoaded', async () => {···
});
document.getElementById('ordersList').addEventListener('click', event => {···
});
function openPaymentModal(orderId) {···
}
function closePaymentModal() {···
}
function applyPromo() {···
}
function handlePayment() {···
}

const confirmPaymentBtn = document.getElementById('confirm-payment-btn');
let currentOrderId = null;

confirmPaymentBtn.addEventListener('click', async () => {···
});
```

Рис. 2.37 – Скрипти та функціонал для оформлення замовлення

- Адміністративна панель admin-dashboard.html забезпечує доступ лише для користувачів з відповідним статусом і дозволяє керувати товарами та користувачами. Візуальна частина стилізується через admin-dashboard.css, що формує layout з формами, таблицями, кнопками редагування, модальними вікнами та панеллю фільтрації. У скрипті admin-dashboard.js (рис. 2.38) реалізовано перевірку прав доступу, завантаження даних з API, можливість додавати нові товари з динамічним додаванням характеристик, редагування та видалення існуючих товарів, оновлення інформації про користувачів у реальному часі.

Рис. 2.38 – Функціонал адмін-панелі

- Допоміжний скрипт products.js (рис. 2.39) містить масив об’єктів із базовими характеристиками товарів: id, назва, опис, фото, категорія, ціна та інше. Використовувався для початкового заповнення каталогу на клієнтській частині, а також для локального тестування без звернення до API.

```
const products = [
    {
      id: 1,
      title: "Logitech G102",
      description: "Точна сенсорна миша з RGB підсвіткою",
      price: 899,
      category: "mice",
      image: "https://encrypted-tbn0.gstatic.com/shopping?q
      hasDetails: true,
    },
    {
      id: 2,
      title: "HyperX Cloud Stinger",
      description: "Легка гарнітура з якісним мікрофоном",
      price: 1599,
      category: "headsets",
      image: "https://encrypted-tbn2.gstatic.com/shopping?q
      hasDetails: true,
    },
```

Рис. 2.39 – Масив товарів початкового відображення

2.2.3 Результати розробки

Нижче подано візуально реалізовані сторінки користувацького інтерфейсу. Кожна з них ілюструє практичну реалізацію відповідного функціоналу, описаного раніше.

На рисунку 2.40 зображено головну сторінку застосунку з навігаційним меню, фоном і секціями з популярними товарами та віджетом зміни мови.

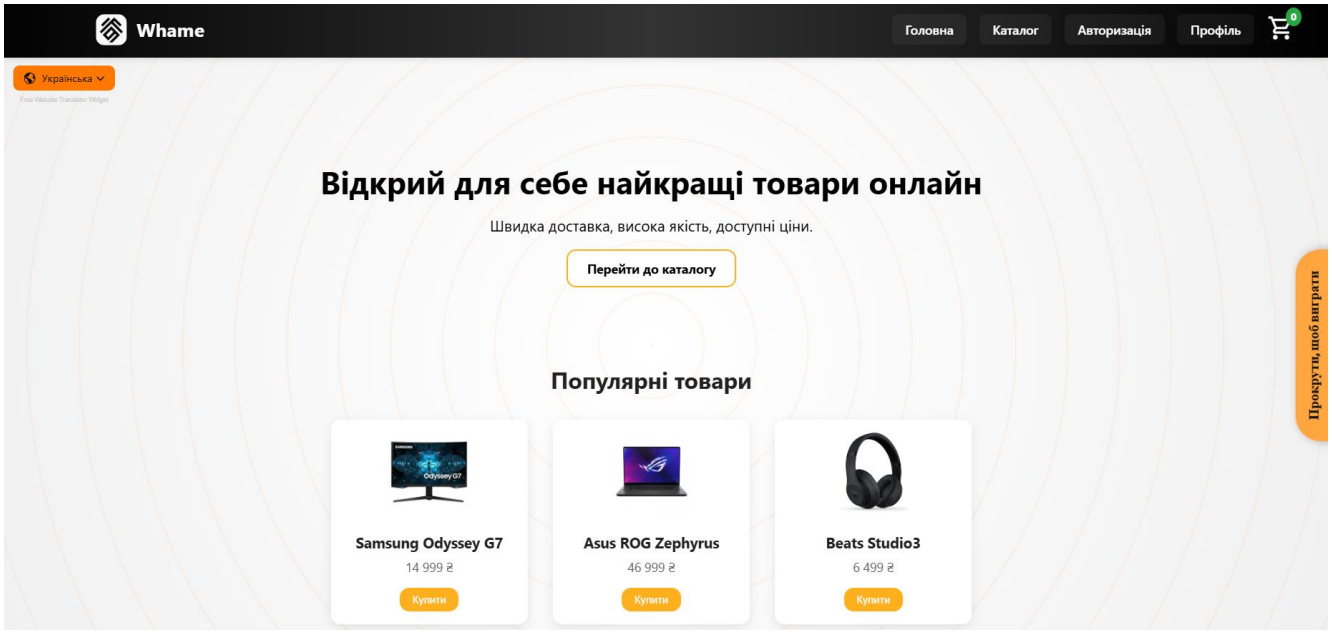

Рис. 2.40 – Перша частина головної сторінки сайту та хедер

На середній частині (рис. 2.41) показано секцію переваг. Вона складається з трьох візуально виділених блоків, які коротко інформують користувача про основні сильні сторони сервісу: швидку доставку, регулярні знижки та безпечну оплату.

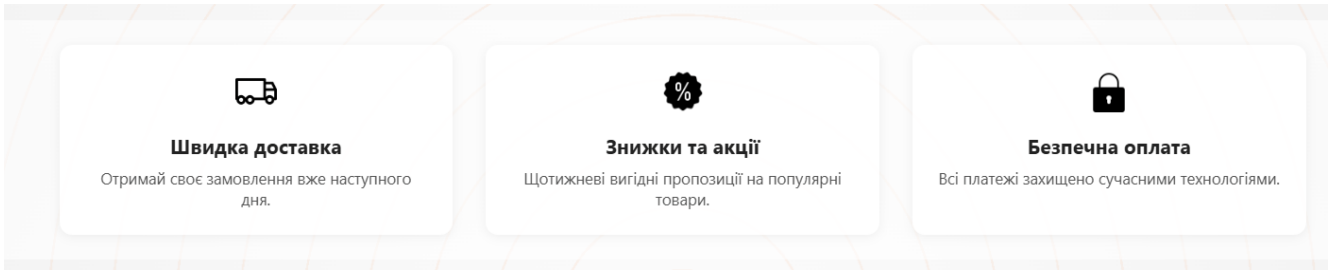

Рис. 2.41 – Друга частина головної сторінки сайту

На нижній частині (рис. 2.42) реалізовано футер, інтерактивні блоки з формою зворотного зв’язку, описом Whame та інтеграцією мапи.

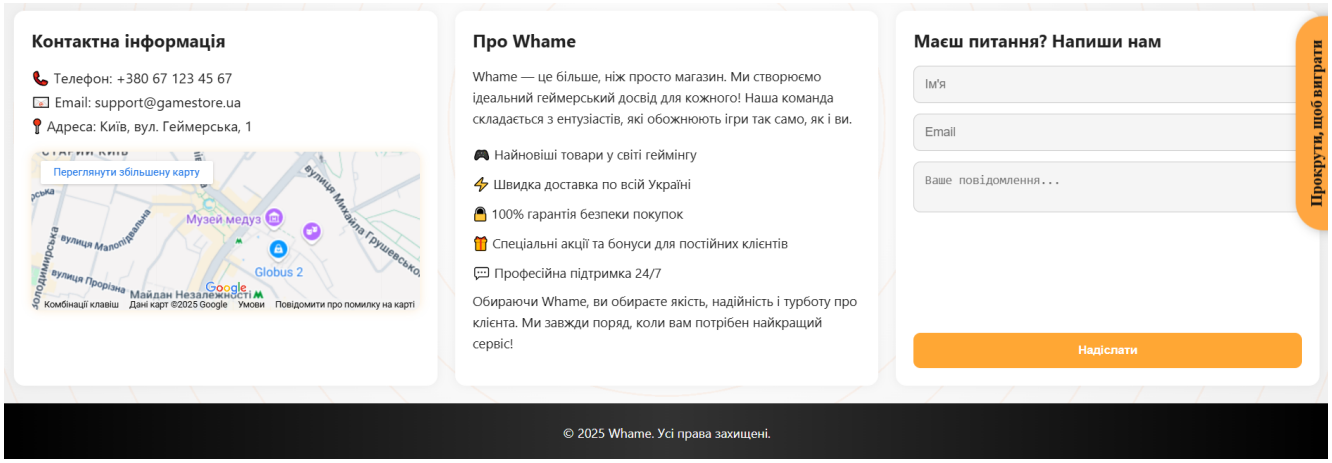

Рис. 2.42 – Третя частина головної сторінки сайту та футер

Тут користувач вводить свої дані для авторизації (рис. 2.43). Інтерфейс містить форму з валідацією введення та повідомленнями про помилки. Після успішного входу здійснюється перехід до особистого кабінету.

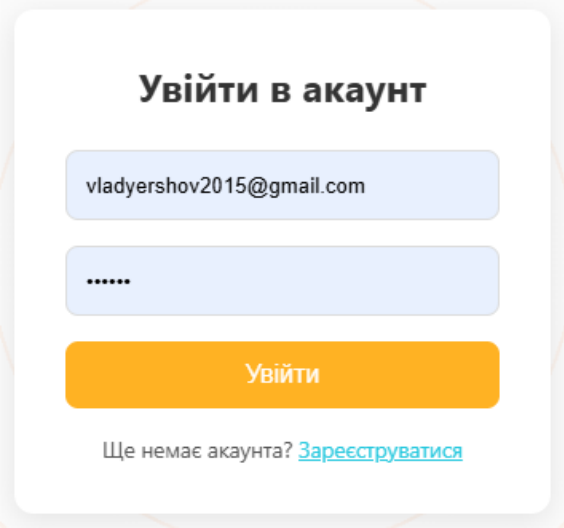

Рис. 2.43 – Форма входу в аккаунт

Далі реалізовано форму для створення облікового запису (рис. 2.44), включно з полями для імені, електронної пошти та підтвердження пароля. Реалізована перевірка правильності введених даних до надсилання на сервер.

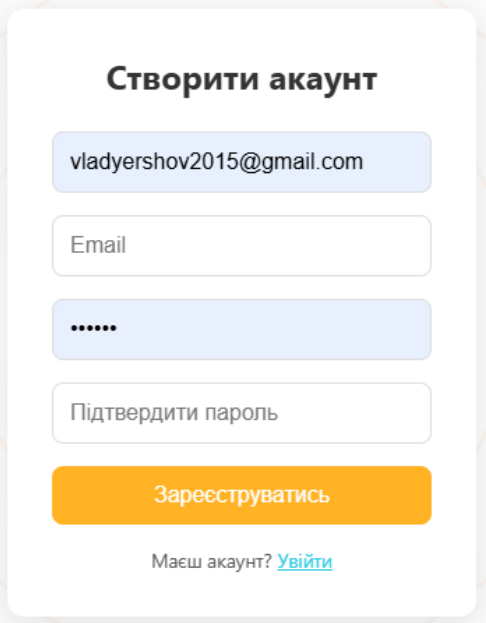

Рис. 2.44 – Форма реєстрації нового користувача

Користувач може переглядати та редагувати особисту інформацію. У профілі (рис. 2.45) відображено ім’я, електронну пошту, адресу доставки, історію активності та бейджики ролі.

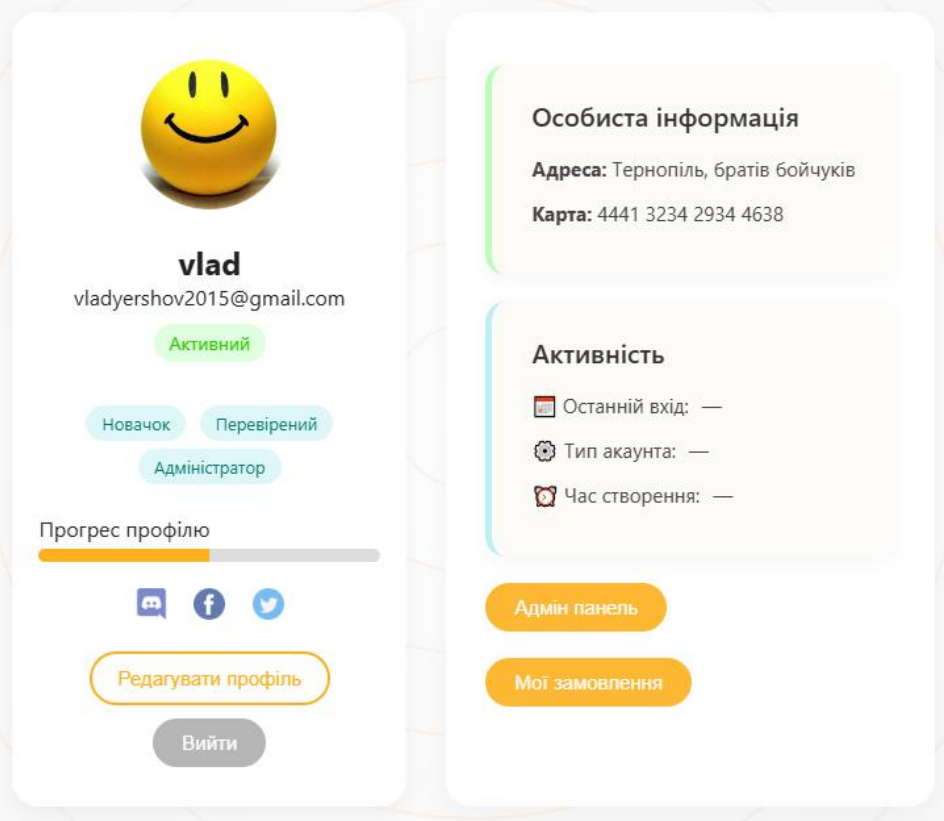

Рис. 2.45 – Профіль адміністратора сайту

У профілі також доступна функція редагування (рис. 2.46) через модальне вікно. (Дані оновлюються та зберігаються в БД)

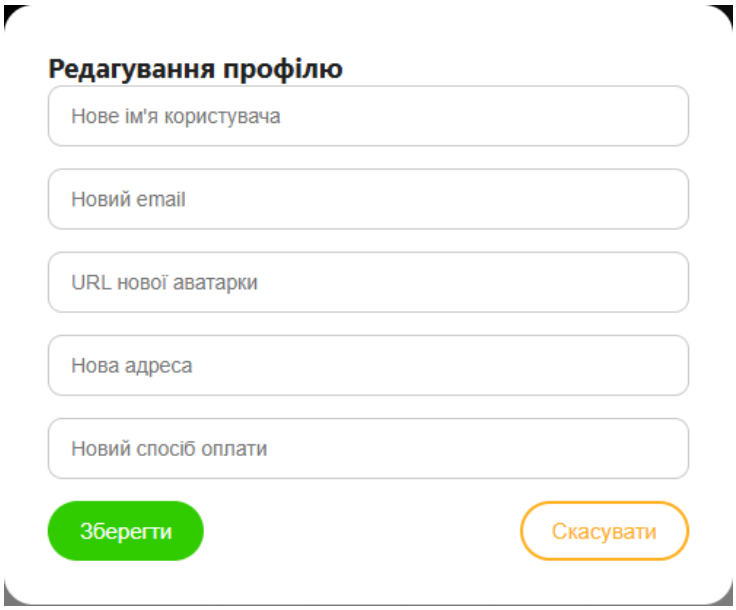

Рис. 2.46 – Модульне вікно для редагування профілю

На сторінці каталогу (рис. 2.47) відображається список товарів. Користувач може додавати товари до кошика.

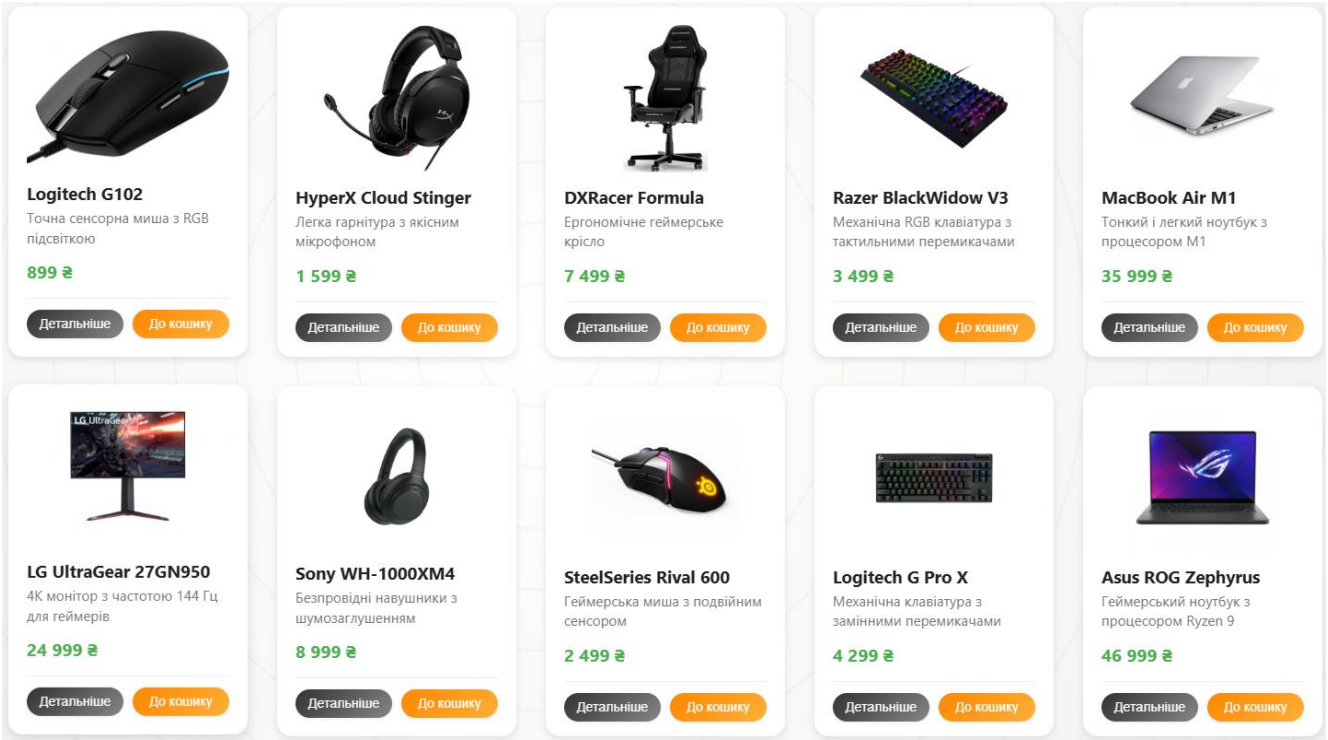

Рис. 2.47 – Каталог товарів

Оформлена можливість фільтрації (рис. 2.48) за категоріями та ціною.

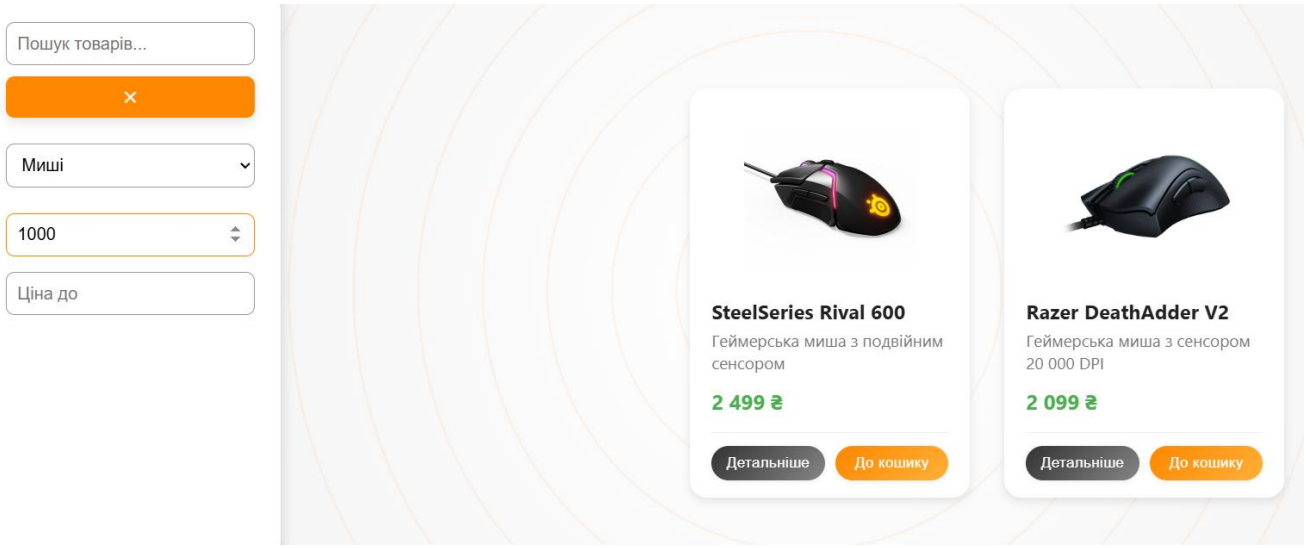

Рис. 2.48 – Приклад фільтрації пошуку по назві товару та ціні

Сторінка товару (рис. 2.49) відображає розширену інформацію про окремий товар, включаючи галерею фото, технічні характеристики, рейтинг.

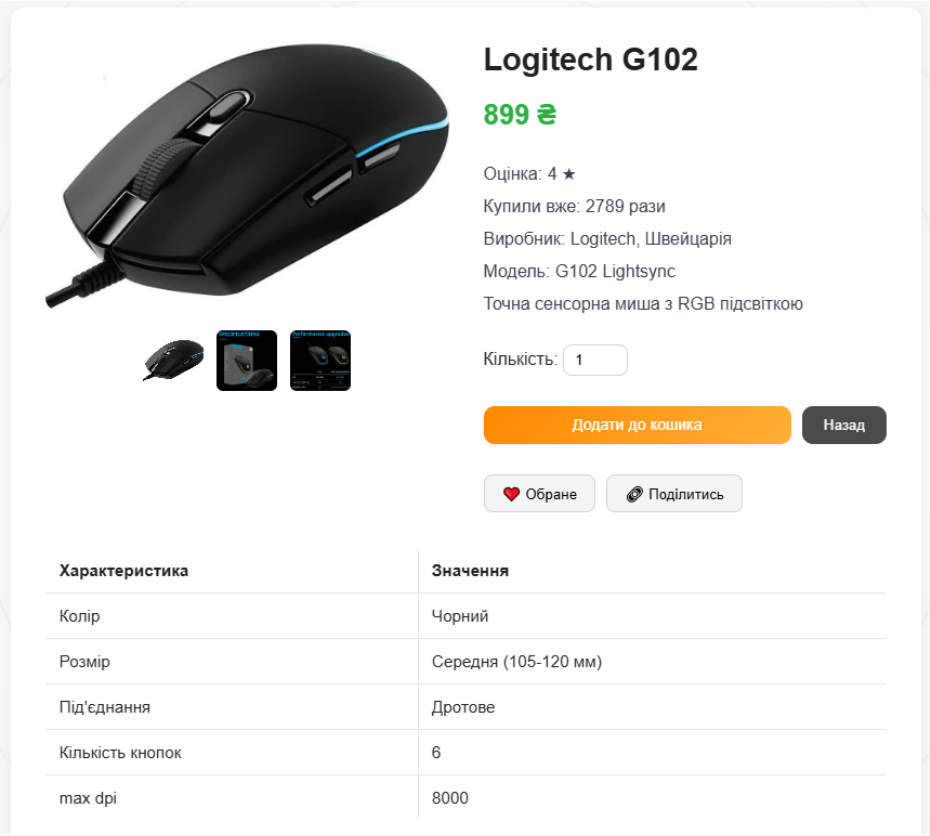

Рис. 2.49 – Перша частина відображення сторінки товару

Відгуки коментарів зберігаються та відображаються до видалення (рис. 2.50).

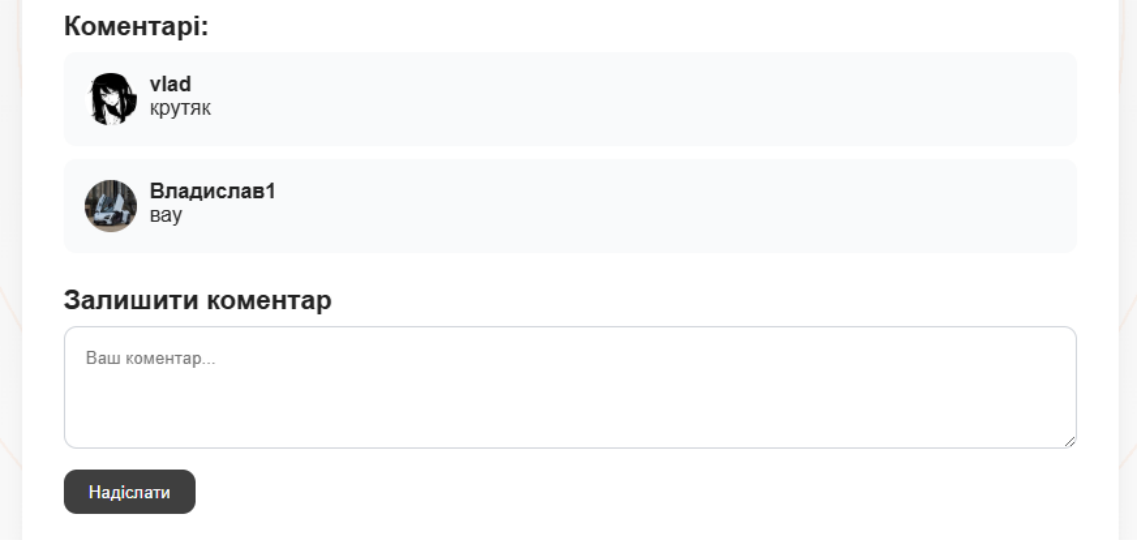

Рис. 2.50 – Друга частина відображення сторінки товару

Реалізований кошик (рис. 2.51), спочатку обрані товари попадають туди.

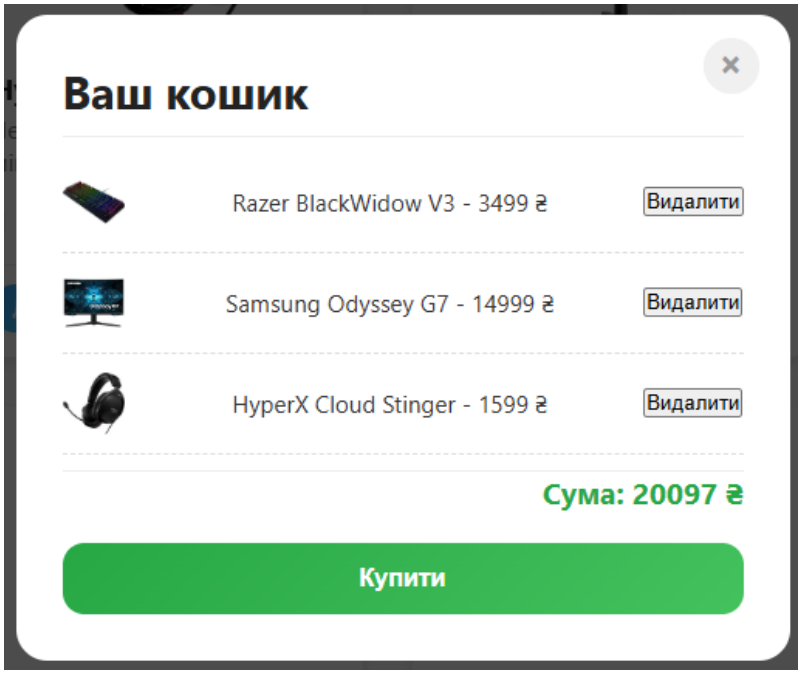

Рис. 2.51 – Модульне вікно кошику

Після підтвердження складу кошика, відкривається форма оформлення (рис. 2.52, 2.53) з введенням адреси доставкою, вибором способу оплати. Після завершення –відправлення замовлення на сервер і очищення кошика.

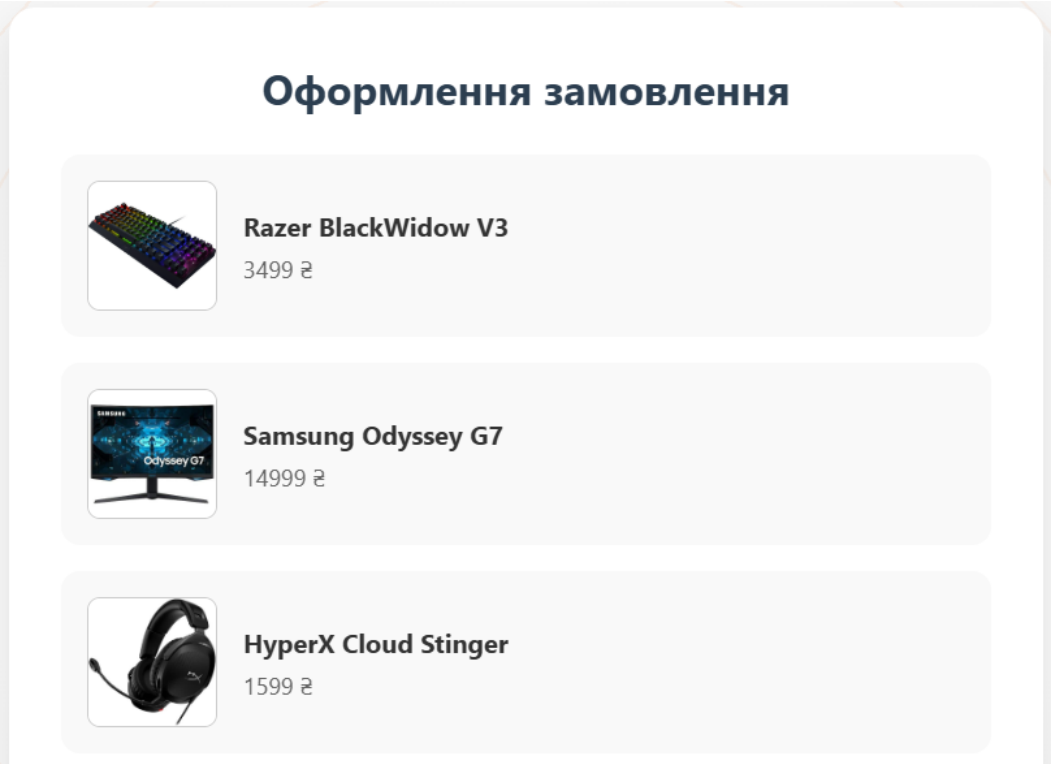

Рис. 2.52 – Перша частина сторінки оформлення замовлення (мод. вікно)

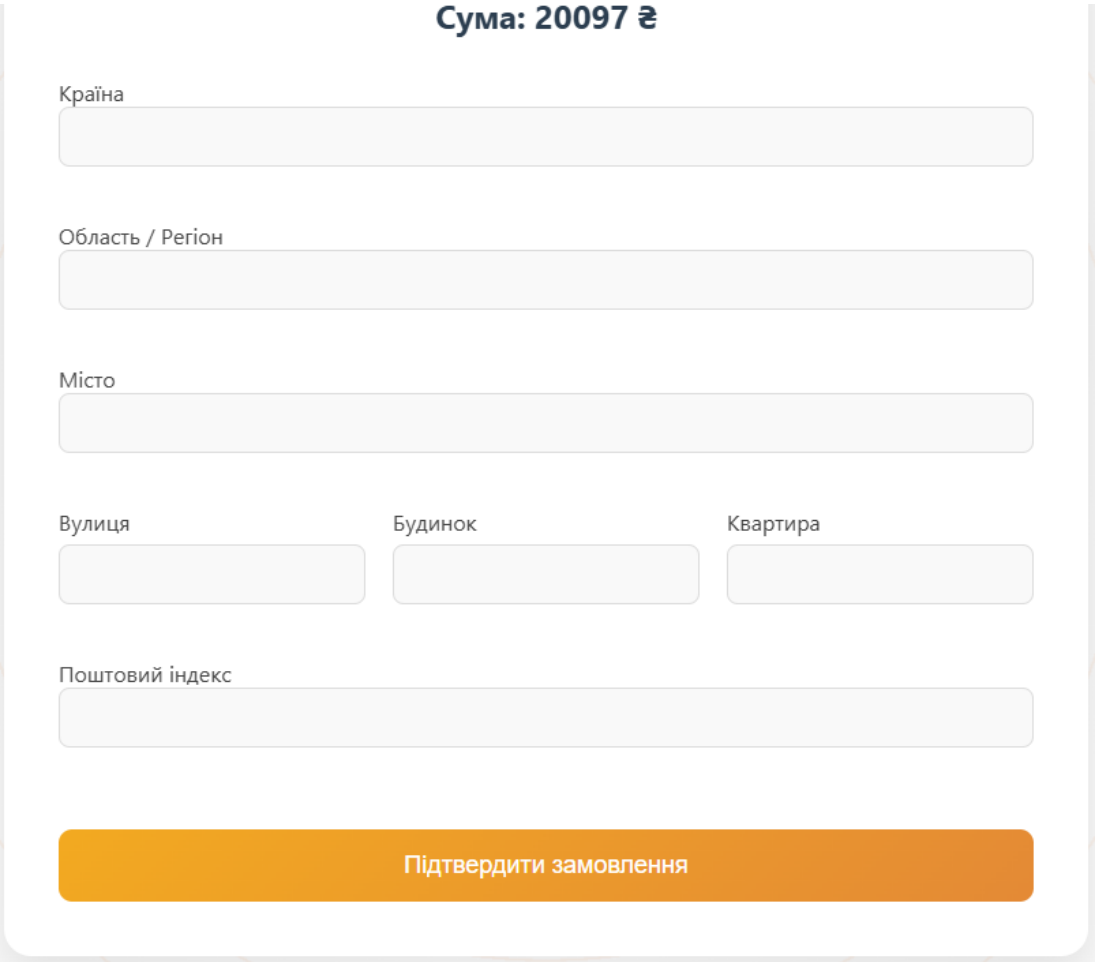

Рис. 2.53 – Друга частина форми оформлення замовлення

Після підтвердження замовлення користувач може переглядати оформлені замовлення (оплачені/не оплачені) (рис. 2.54), переглядати деталі кожного.

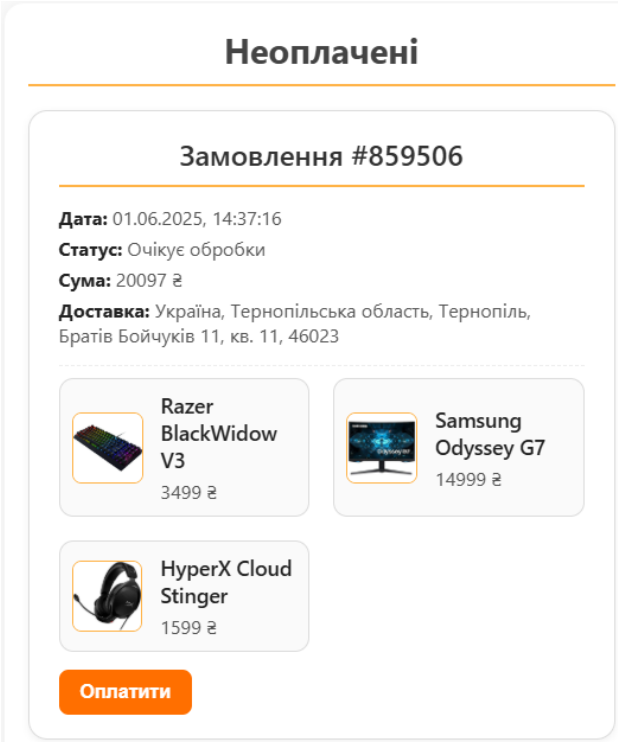

Рис. 2.54 – Неоплачені сформовані замовлення

Створена можливість оплати замовлення (рис. 2.55) з активацією промокоду.

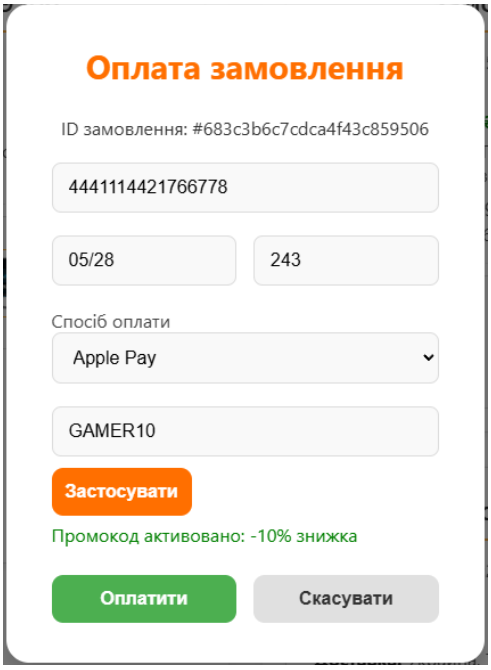

Рис. 2.55 – Модульне вікно оплати замовлення

Після оплати замовлення, воно переходить у розділ оплачених (рис. 2.56). Промокод знімає процент від загальної вартості товару. Появляється трек-номер, по якому можна відслідкувати замовлення на трек-сайтах.

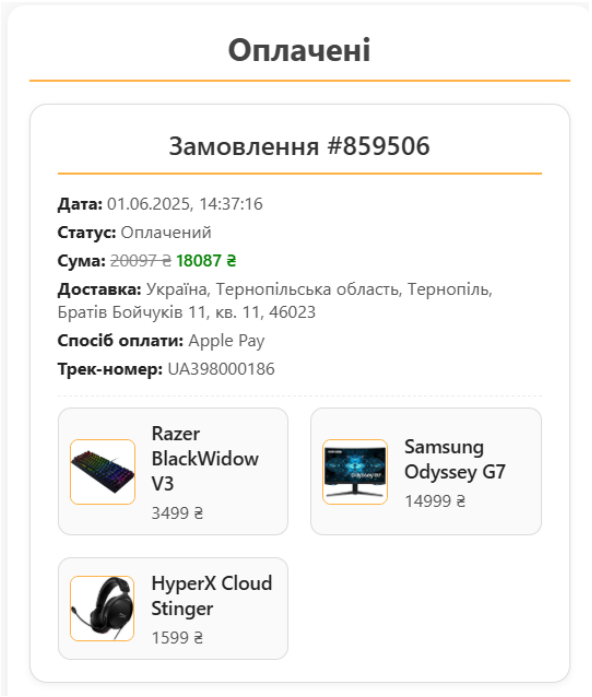

Рис. 2.56 – Оплачені замовлення

Бонуси, промокоди можна получити у віджеті колеса фортуни (рис. 2.57).

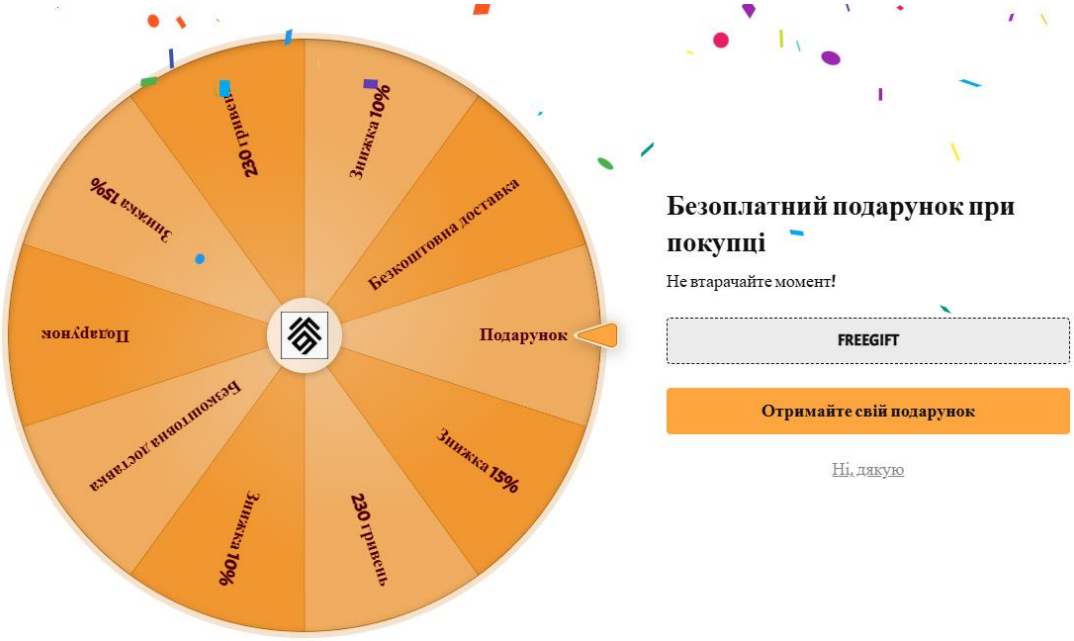

Рис. 2.57 – Колесо фортуни elfsight

Для користувачів з правами адміністратора відкрито окрему панель керування товарами (рис. 2.60) та користувачами (рис. 2.58). На головному екрані видно таблиці з усіма зареєстрованими користувачами, їхніми email та ролями.

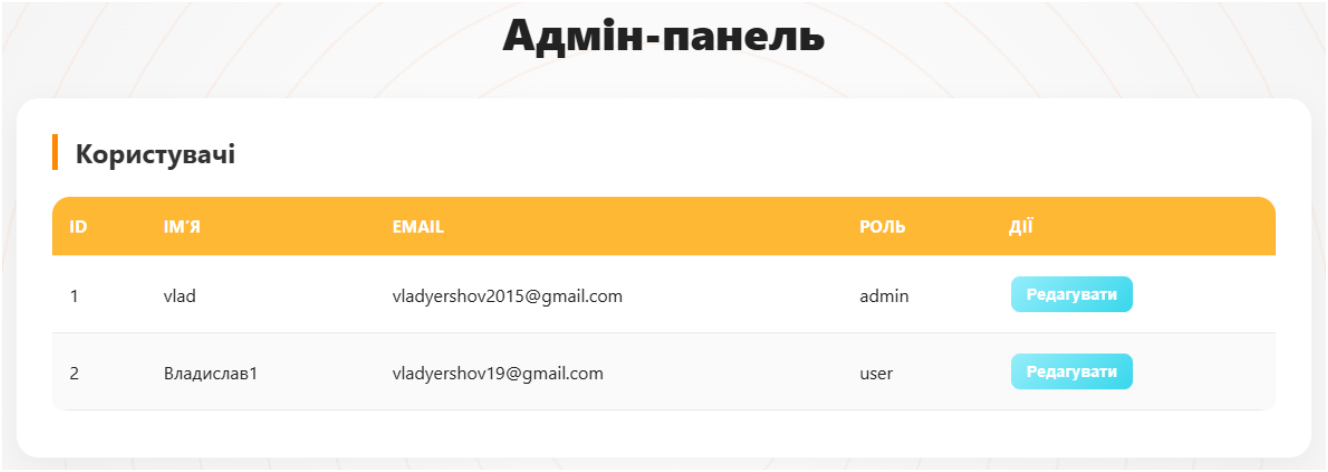

Рис. 2.58 – Список існуючих користувачів

Також доступні кнопки та модуль для редагування даних (рис. 2.59).

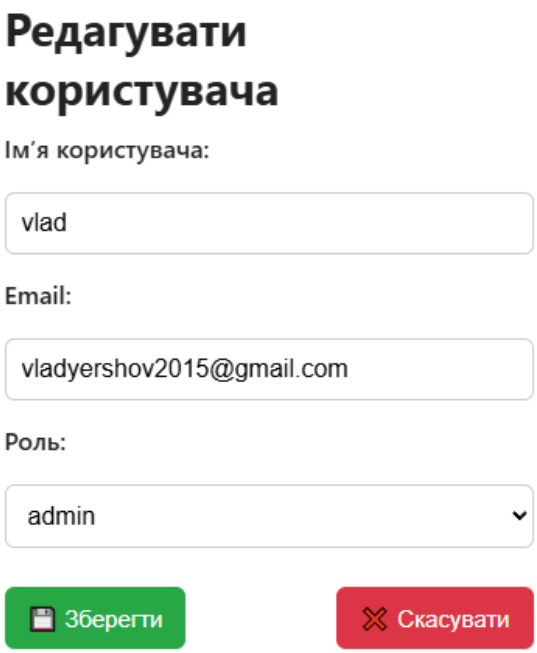

Рис. 2.59 – Модульне вікно редагування користувачів

У таблиці товарів можна переглядати, редагувати, добавляти або видаляти кожен продукт (рис. 2.60).

Товари

ID	НАЗВА	ЦІНА	КАТЕГОРІЯ	ОПИС	ЗОБРАЖЕННЯ	ДІЇ
682df44a664dcbc27d1abef7	Logitech G102	899 грн	mice	Точна сенсорна миша з RGB підсвіткою		
682df44a664dcbc27d1abef8	HyperX Cloud Stinger	1599 грн	headsets	Легка гарнітура з якісним мікрофоном		

Рис. 2.60 – Приклад відображення товарів

Нижче розміщена форма для додавання або оновлення товару (рис. 2.61) з усіма потрібними полями: назва, ціна, опис, зображення, характеристики. Є можливість додати технічні характеристики.

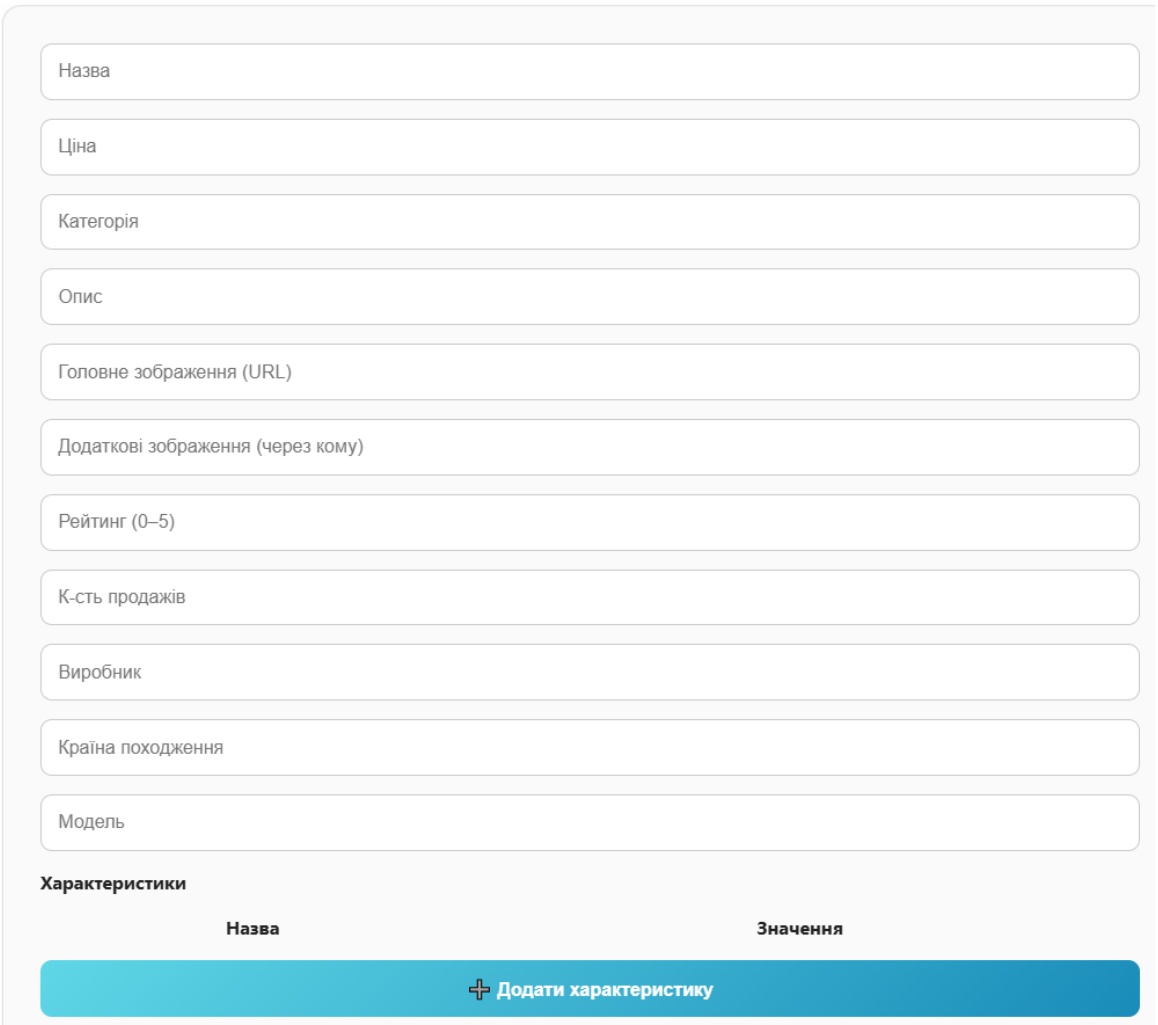

Рис. 2.61 – Форма редагування/додавання товару

У базі зберігається інформація про користувачів, замовлення та товари. Дані з форм (реєстрації, профілю, товарів, замовлень) на сайті потрапляють у базу даних і можуть бути переглянуті або змінені через MongoDB Compass – графічний інтерфейс для роботи з MongoDB.

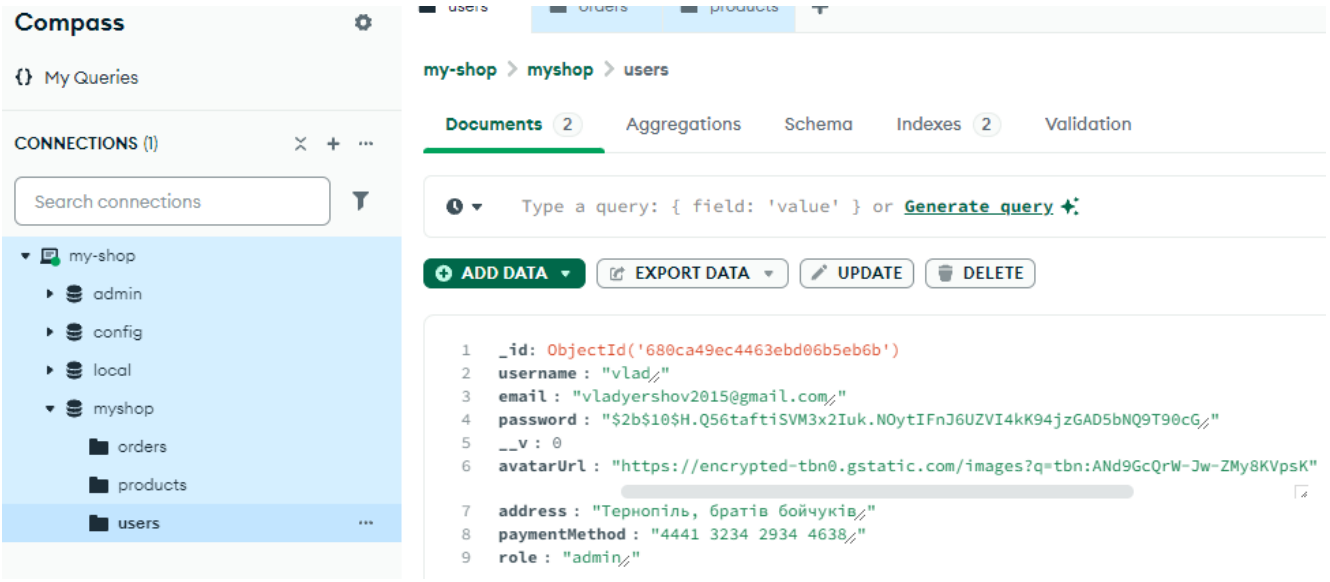

Рис. 2.62 – Дерево БД та відображення даних про юзерів

На рисунках 2.62 та 2.63 показано приклад зв’язку вебсайту з базою даних та збереження даних за допомогою MongoDB.

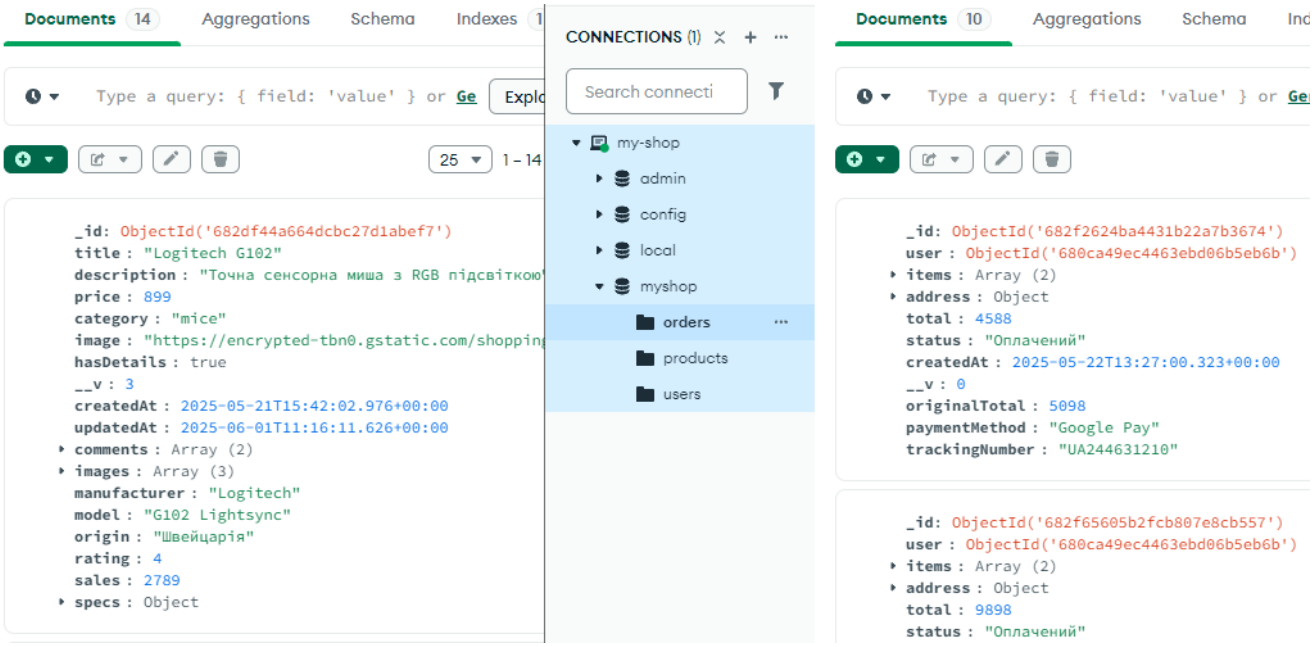

Рис. 2.63 – Відображення даних про замовлення та товар

2.2.4 Тестування програмного забезпечення та оцінка якості

У межах цього розділу буде проведено відслідковування багів, навантажувальне тестування для перевірки стабільності системи під високим навантаженням, а також автоматизоване тестування для підвищення ефективності перевірки функціональності.

Нижче наведено приклади баг-репортів у таблиця 2.1, 2.2, 2.3, що були зафіксовані в процесі відслідковування помилок

Таблиця 2.1 – Баг-репорт "Некоректна валідація зворотного зв’язку"

Summary	Форма зворотного зв’язку приймає некоректно введені email-адреси.
Severity	S.2
Priority	P.2
Description	У формі зворотного зв’язку не спрацьовує валідація при введенні неправильного формату електронної пошти. Це може призвести до втрати користувацьких звернень або спаму. Preconditions: • Відкрита сторінка з формою зворотного зв’язку. • Підключення до Інтернету. • Користувач не авторизований у системі. • JavaScript увімкнено в налаштуваннях браузера. Steps to Reproduce: 1) Перейти на сторінку "Контакти". 2) Ввести у поле "Email" неправильний формат, наприклад: test@ або test.com. 3) Заповнити інші обов’язкові поля. 4) Натиснути кнопку "Надіслати".
Expected Result	Користувач повинен побачити повідомлення про помилку в форматі email, і форма не повинна відправлятися.
Actual Result	Форма успішно надсилається, навіть якщо email введено у некоректному форматі.

Таблиця 2.2 – Баг-репорт "Кнопка віджета активується до натискання"

Summary	Кнопка активації віджета Elfsight стає активною до того, як користувач її натисне.
Severity	S.2
Priority	P.2
Description	При переході до віджета Elfsight кнопка активації автоматично активується без взаємодії користувача, що може збивати з пантелику і призводити до некоректної роботи інтерфейсу. Preconditions: • Відкрито сторінку з віджетом Elfsight. • Підключення до Інтернету. Steps to Reproduce: 1) Відкрити сторінку з віджетом Elfsight. 2) Перейти до блоку віджета. 3) Спостерігати стан кнопки активації без натискання на неї.
Expected Result	Кнопка активації повинна ставати активною лише після натискання користувачем.
Actual Result	Кнопка активації активується автоматично при переході до віджета, до натискання.

Таблиця 2.3 – Баг-репорт " Не зберігається активність користувача"

Summary	У блоці "Активність" не відображаються дані про останній вхід, тип акаунта та час створення.
Severity	S.3
Priority	P.2
Description	У розділі "Активність" відсутня інформація щодо останнього входу користувача, типу акаунта та часу створення. Preconditions: • Користувач увійшов у систему. • Профіль відкритий для перегляду. Steps to Reproduce: 4) Відкрити сторінку з віджетом Elfsight. 5) Перейти до блоку віджета. 6) Спостерігати стан кнопки активації без натискання на неї.

Продовження таблиці 2.3

Expected Result	Усі поля блоку "Активність" мають бути заповнені актуальними даними (дата останнього входу, тип акаунта, час створення).
Actual Result	Усі поля містять прочерки (–), тобто дані відсутні.

Етапи виявлення та фіксації багів присутні. Далі тестування було проведено у автоматизованому режимі з використанням katalon recorder, що дозволить ефективно перевірити ключову функціональність системи.

Katalon Recorder – це безкоштовне розширення для браузерів Chrome створене компанією Katalon Inc. як сучасна альтернатива класичному Selenium IDE.

Основна мета цього інструменту – забезпечити користувачам зручний спосіб запису, редагування та запуску автоматизованих тестів без необхідності писати код вручну. Katalon Recorder дозволяє тестувальникам фіксувати дії користувача на вебсторінках, зберігати їх як тести, а також експортувати їх у різні мови програмування. Завдяки простому інтерфейсу це розширення підходить як для новачків, так і для досвідчених автоматизаторів.

Для першого автоматизованого тестування була виконана дія авторизації користувача (рис. 2.64). Результат тестування показано на рисунку.

Command	Target	Value
open	http://127.0.0.1:5500/frontend/index.html	
click	link=Авторизація	
open	http://127.0.0.1:5500/frontend/login.html	
type	id=login-email	vladyershov2015@gmail.com
type	id=login-password	123456
click	xpath=//button[@type='submit']	
open	http://127.0.0.1:5500/frontend/profile.html	

Рис. 2.64 – Дані про успішний тест авторизації [10]

Бачимо, що все працює (по зеленому коліру).

У другому тесті створено подію добавлення товарів у кошик (рис. 2.65).

click	link=Каталог
open	http://127.0.0.1:5500/frontend/catalog.html
click	xpath=(.//*[normalize-space(text()) and normalize-space(.)='Logitech G102'])[1]/following::div[1]
click	xpath=(.//*[normalize-space(text()) and normalize-space(.)='Детальніше'])[2]/following::button[1]
click	xpath=(.//*[normalize-space(text()) and normalize-space(.)='Детальніше'])[2]/following::button[1]
click	xpath=//img[@alt='Кошик']
click	id=buyButton
open	http://127.0.0.1:5500/frontend/checkout.html

Рис. 2.65 – Дані про успішний тест додавання товарів у кошик [10]

Подію протестовано успішно. Далі тертій тест – оформлення замовлення з обраними товарами у кошику (рис. 2.66).

Command	Target	Value
click	id=buyButton	
open	http://127.0.0.1:5500/frontend/checkout.html	
click	name=country	
type	name=country	Украина
type	name=region	Тернопільська область
type	name=city	Тернопіль
type	name=street	Братів Бойчуків
type	name=postal	46023
click	name=country	
type	name=country	Україна
click	name=house	
type	name=house	1
click	name=apartment	
type	name=apartment	1
click	xpath=//button[@type='submit']	
assertAlert	✅ Замовлення успішно оформлено!	

Рис. 2.66 – Дані про успішний тест оформлення замовлення [10]

Також виконано, можна побачити повідомлення про успішне оформлення.

Після проведення функціонального та автоматизованого тестування, можна зробити висновок про відповідність розробленого інтернет-магазину заявленим вимогам. Усі основні функції, зокрема авторизація, додавання товарів у кошик та оформлення замовлення, працюють стабільно, що підтверджено результатами автоматизованих тестів (рис. 2.64 – 2.66).

Виявлені баги були задокументовані у вигляді баг-репортів (табл. 2.1 – 2.3), що свідчить про наявність процесу відслідковування та усунення помилок у рамках життєвого циклу розробки.

Автоматизоване тестування за допомогою Katalon Recorder дало змогу перевірити ключовий функціонал і виявити потенційні помилки без значних витрат часу. Це дозволяє зробити висновок про високу ефективність розробки з технічної точки зору та належний рівень якості кінцевого продукту.

Однак слід зауважити, що повноцінне навантажувальне тестування у реальному середовищі не було реалізоване у повному обсязі через відсутність стабільного хостингового оточення, яке б дозволило емулювати велику кількість одночасних користувачів.

Таким чином, хоча на поточному етапі реалізовані базові функції інтернет-магазину і підтверджено їхню працездатність, програмний продукт не є завершеним. Для повноцінного впровадження необхідно реалізувати підтримку платіжних систем, організувати хостингове середовище для навантажувального тестування та провести інтеграцію з сервісами аналітики й безпеки.

3 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

3.1 Психологія безпеки праці в загальній проблемі психології

Кожна конкретна праця, наприклад розробка інтернет-магазину, вимагає певних фізичних зусиль, нервово-психічних витрат, емоційної напруги і відбувається за різних санітарно-гігієнічних та кліматичних умов [15]. Все це впливає на виконавця праці – людину.

Тому завдання наук, що займаються проблемами праці, є вивчення зі своїх позицій багатогранних зв’язків між людиною та об’єктивними факторами праці. До цієї науки відноситься: психологія праці.

Психологія праці вивчає особливості трудової діяльності людини та процеси формування у неї якостей, професійно важливих для підвищення продуктивності праці.

Окремим напрямом у межах цієї дисципліни виступає психологія безпеки праці – наука, що досліджує вплив психічних станів, особистісних якостей та соціальних взаємин на дотримання правил охорони праці. Вона вивчає фактори, які можуть сприяти або, навпаки, знижувати готовність працівника діяти безпечно.

Відповідно до Типового положення про порядок проведення навчання і перевірки знань з питань охорони праці, затвердженого наказом Держнаглядохоронпраці України від 26.01.2005 № 15 [17], тематика навчальних програм обов’язково включає питання психології безпеки праці. Зокрема, передбачається засвоєння знань про вплив психофізіологічного стану працівників на рівень безпеки, причинну роль стресу, монотонності, втоми, а також формування навичок поведінкової профілактики ризиків у виробничому середовищі. Цей підхід підкреслює важливість психологічного чинника у забезпеченні ефективної системи охорони праці.

Основними задачами психології безпеки праці є:

- аналіз психофізіологічних функцій (уваги, пам’яті, реакцій) у контексті безпечної поведінки;

- виявлення причин необережних або ризикованих дій;
- розробка психопрофілактичних програм (інструктажів, тренінгів, оцінки психічної готовності до виконання робіт);
- участь у формуванні системи мотивації до дотримання правил безпеки.

Психологія безпеки праці спрямована не лише на попередження травм, але й на створення системного підходу до навчання працівників поведінці у небезпечних умовах, підвищення їх відповідальності за власну безпеку та безпеку колег.

Психологія безпеки на робочому місці – це окремий, прикладний напрям, що вивчає вплив психологічного клімату, взаємин у колективі, емоційного добробуту на загальну атмосферу безпеки в організації. У рамках психології безпеки розглядаються такі питання, як тренінги з безпечної поведінки, підтримка працівників у разі травм чи криз, аналіз та запобігання можливим небезпекам тощо.

У праці [16] важливу роль відіграють властивості й особливості психіки і свідомості. Характер трудової діяльності людини визначається не тільки фізичним навантаженням, а й величиною нервового та емоційного напруження, ритмом і темпом роботи, її монотонності, об’єму сприймання і перероблення інформації. Від цього залежить встановлення раціонального режиму праці і відпочинку, організація робочого місця, проведення професійного добору, професійної орієнтації тощо.

На безпеку праці людини впливає її психічний стан: наявність конфліктів, втома, захворювання, особливості психіки людини, нервове перенапруження, стрес. При наявності небезпечних чинників (рухомі деталі машин, погане освітлення тощо) та пригніченому стані психіки людини можуть виникати нещасні випадки.

Аналіз виробничого травматизму показує, що основна причина травм і загибелі людей на робочих місцях – це поганий психічний стан працівників під час виконання трудових обов’язків.

Наприкінці робочого дня і тижня в організмі людини нагромаджується втома та дратівливість. Тому в ці періоди треба бути дуже уважним і розсудливим при виконанні робіт.

Психологічна безпека на робочому місці означає створення умов, за яких працівники почуваються захищеними від психологічних загроз та стресів, які можуть виникати у робочому оточенні. Це включає запобігання конфліктам, підтримку та допомогу колег, повагу особистих кордонів та переваг кожного працівника, а також забезпечення конфіденційності та безпеки в робочому середовищі. Психологічна безпека на робочому місці сприяє підвищенню продуктивності, покращенню відносин між колегами та створенню комфортного робочого клімату.

Для забезпечення психологічної безпеки на робочому місці необхідно:

1) Забезпечити доброзичливу та підтримуючу атмосферу на робочому місці, де працівники можуть почуватися комфортно та впевнено.

2) Проводити інструктаж та тренінги щодо попередження конфліктів, управління стресом та емоційним станом [17]. «Навчання з питань охорони праці є обов’язковим для всіх працівників і проводиться відповідно до вимог чинного законодавства. Згідно з Порядком проведення навчання і перевірки знань з питань охорони праці, працівники проходять первинне навчання, інструктажі, а також періодичну перевірку знань не рідше одного разу на три роки. Відповідальність за організацію навчального процесу покладається на роботодавця. Перевірка знань завершується оформленням протоколу, а працівникам, які успішно її пройшли, видається відповідне посвідчення. Цей підхід сприяє зменшенню виробничого травматизму, підвищенню рівня безпеки та формує відповідальне ставлення до виконання службових обов’язків».

3) Навчати керівників та працівників навичкам ефективного спілкування та управління емоціями для запобігання конфліктам та непорозумінням.

4) Слухати та враховувати думку працівників, реагувати на їх звернення та проблеми, щоб створити довірчі стосунки.

5) Створити умови для роботи над професійним зростанням та розвитком працівників, щоб вони мали зрозумілі кар’єрні перспективи.

6) Підтримувати відкритість та прозорість у комунікації, щоб уникнути чуток і пліток, які можуть негативно позначитися на психологічному кліматі у колективі.

7) Надавати працівникам можливість для відпочинку та релаксації, щоб вони могли знайти баланс між роботою та особистим життям.

8) Створювати спільні заходи та ініціативи для зміцнення командного духу та співробітництва, щоб працівники відчували себе частиною спільної справи та спільноти.

3.2 Профілактика захворювань, які викликані напруженням органів, систем організму та невірним положенням тіла при роботі

Профілактика захворювань, викликаних напруженням органів, систем організму та неправильним положенням тіла під час виконання трудових обов’язків, є важливою складовою системи охорони праці. Одним із основних ризиків на робочому місці є тривале перебування працівника в статичній або нефізіологічній позі, що може спричинити захворювання опорно-рухового апарату, зорову втому, головний біль, порушення кровообігу, а в довгостроковій перспективі – професійні патології.

З метою зменшення шкідливого впливу факторів, пов’язаних із роботою за комп’ютером, слід дотримуватись чинних нормативних вимог охорони праці. Зокрема, НПАОП 0.00-7.15-18 «Вимоги щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями» [14] встановлює правила організації робочого місця, тривалості роботи, освітлення та перерв. Згідно з нормативом, тривалість безперервної роботи за комп’ютером повинна бути обмеженою, а протягом робочого дня мають передбачатися технологічні перерви для зменшення зорового та психоемоційного навантаження. Також рекомендовано розміщувати монітор на відстані 60–70 см від очей, а робоча поза повинна

забезпечувати пряму поставу з опорою на спинку стільця та можливістю зміни положення тіла.

Відповідно до ДСН 3.3.6.042-99 [15], у виробничих приміщеннях повинна бути забезпечена комфортна температура, відносна вологість повітря та швидкість його руху. Відхилення від встановлених параметрів можуть спричинити фізіологічний дискомфорт, втому, зниження працездатності та, як наслідок, погіршення стану здоров'я. Недотримання нормативного мікроклімату часто стає причиною головного болю, дратівливості та загального перевтомлення організму.

Закон України «Про охорону праці» [13] вказує на обов'язок роботодавця забезпечити безпечні та нешкідливі умови праці. У цьому контексті до обов'язків належить не лише технічне забезпечення, а й організаційні заходи: впровадження режимів праці і відпочинку, надання працівникам можливості змінювати вид діяльності, проводити фізкультхвилинки, навчання з основ ергономіки та формування навичок самоконтролю за поставою. Законодавчо також закріплена відповідальність за порушення вимог охорони праці, які призводять до професійних захворювань або втрати працездатності.

Окрему роль у профілактиці відіграє державний контроль, який здійснюється відповідними структурами, зокрема Державною службою України з питань праці та Державною службою України з надзвичайних ситуацій. Їхні функції включають перевірку дотримання нормативних вимог, інформування роботодавців і працівників, а також участь у розробці профілактичних програм. Дані служби активно публікують рекомендації з питань ергономіки та профілактики перевтоми на своїх офіційних ресурсах.

Таким чином, ефективна профілактика захворювань, пов'язаних із фізичним напруженням і неправильним положенням тіла на робочому місці, ґрунтується на виконанні нормативних вимог, дотриманні санітарних стандартів, організації належного робочого процесу та відповідальному ставленні як з боку роботодавця, так і самого працівника.

ВИСНОВКИ

У межах цієї кваліфікаційної роботи було реалізовано інтернет-магазин із клієнт-серверною архітектурою на базі Node.js з інтеграцією віджетів Elfsight. Одним із ключових елементів проєкту стало створення інтерактивного віджета «Колесо фортуни», що автоматично генерує промокоди та подарунки, які активуються під час оплати замовлення. Така функціональність сприяє підвищенню лояльності клієнтів та залученню нових користувачів.

На першому етапі було проведено аналіз конкурентів і визначено актуальні напрями розвитку в галузі електронної комерції. На основі цього обґрунтовано вибір Node.js як основної серверної платформи, а також MongoDB як бази даних завдяки її гнучкості, масштабованості та зручній роботі з JSON-структурами. Для фронтенд-частини застосовано стандартні вебтехнології – HTML, CSS та JavaScript, що дозволило створити адаптивний і зручний інтерфейс. Уся розробка велася у середовищі Visual Studio Code.

Розробка велась за методологією Agile Scrum, що дало змогу ефективно керувати завданнями, контролювати прогрес і вносити покращення на всіх етапах. У роботі створено модель предметної області, бізнес-модель, а також архітектуру системи, підтверджену діаграмами класів, послідовностей, варіантів використання та пакетів. Усі компоненти системи були логічно структуровані та протестовані.

Особливу увагу приділено тестуванню, яке охоплювало як функціональні, так і автоматизовані перевірки. Було впроваджено систему баг-репортів для фіксації помилок, що дозволило оперативно їх усувати. У результаті система продемонструвала стабільну роботу та продуктивність.

Розроблений інтернет-магазин відповідає поставленим цілям і технічним вимогам, є майже готовим до впровадження в реальні умови та може служити основою для подальшого розширення функціональності. Отримані результати підтверджують ефективність обраних рішень і застосованих технологій.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. NodeJS [Електронний ресурс] – Режим доступу до ресурсу: https://nodejs.org/en - Дата доступу: 17.03.2025.

2. Elfsight [Електронний ресурс] – Режим доступу до ресурсу: https://elfsight.com/ - Дата доступу: 28.03.2025.

3. MongoDB Compass [Електронний ресурс] – Режим доступу до ресурсу: https://www.mongodb.com/products/tools/compass - Дата доступу: 02.04.2025.

4. Visual Studio Code [Електронний ресурс] – Режим доступу до ресурсу: https://code.visualstudio.com/ - Дата доступу: 22.09.2022.

5. NZXT [Електронний ресурс] – Режим доступу до ресурсу: https://nzxt.com/ - Дата доступу: 14.03.2025.

6. GameStop [Електронний ресурс] – Режим доступу до ресурсу: https://www.gamestop.com/ - Дата доступу: 14.03.2025.

7. Epic games store [Електронний ресурс] – Режим доступу до ресурсу: https://store.epicgames.com/ - Дата доступу: 14.03.2025.

8. MongoDB [Електронний ресурс] – Режим доступу до ресурсу: https://www.mongodb.com/ - Дата доступу: 02.04.2025.

9. IBM RSAD [Електронний ресурс] – Режим доступу до ресурсу: https://www.ibm.com/products/rational-software-architect-designer - Дата доступу: 10.09.2024.

10. Katalon recorder [Електронний ресурс] – Режим доступу до ресурсу: https://katalon.com/ - Дата доступу: 14.05.2025.

11. Partners [Електронний ресурс] – Режим доступу до ресурсу: https://www.pm-partners.com.au/ - Дата доступу: 18.04.2025.

12. Dou [Електронний ресурс] – Режим доступу до ресурсу: https://dou.ua/forums/topic/44636/ - Дата доступу: 19.04.2025.

13. Закон України «Про охорону праці» [Електронний ресурс] – Режим доступу до ресурсу: https://zakon.rada.gov.ua/laws/show/2694-12 – Дата доступу: 05.06.2025.

14. НПАОП 0.00-7.15-18. Вимоги щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями [Електронний ресурс] – Режим доступу: https://zakon.rada.gov.ua/laws/show/z0508-18 – Дата доступу: 08.06.2025.

15. ДСН 3.3.6.042-99. Державні санітарні норми мікроклімату виробничих приміщень [Електронний ресурс] – Режим доступу до ресурсу: https://zakon.rada.gov.ua/rada/show/v0042282-99 – Дата доступу: 05.06.2025.

16. Охорона праці : підручник / за ред. В. П. Кучерявого. Львів : Оріяна-Нова, 2007. 368 с.

17. Закон України «Про охорону праці», Порядок проведення навчання і перевірки знань з питань охорони праці [Електронний ресурс] – Режим доступу: https://zakon.rada.gov.ua/laws/show/z0231-05#Text – Дата доступу: 08.06.2025.

ДОДАТКИ

ДОДАТОК А. Публікація у науковій конференції

VIII Міжнародна студентська науково - технічна конференція
"ПРИРОДНИЧІ ТА ГУМАНІТАРНІ НАУКИ. АКТУАЛЬНІ ПИТАННЯ"

УДК 004.4
Єршов В. – ст. гр. СП-41
Тернопільський національний технічний університет імені Івана Пулюя

ВИКОРИСТАННЯ NODE.JS ТА ELFSIGHT ДЛЯ РОЗРОБКИ ФУНКЦІОНАЛЬНОГО ІНТЕРНЕТ-МАГАЗИНУ

Науковий керівник: Стефанишин В.М.

Yershov V.
Ternopil Ivan Puluj National Technical University

USING NODE.JS AND ELFSIGHT TO DEVELOP A FUNCTIONAL ONLINE STORE

Supervisor: Stefanyshyn V.M.

Ключові слова: Node.JS, elsight, інтернет-магазин.
Key words: Node.JS, elsight, online store.

У реаліях сьогодення, коли бізнес дедалі більше переходить у цифровий формат, створення інтернет-магазинів стає не просто трендом, а необхідністю. Онлайн-торгівля активно розвивається, і від вибору технологій значною мірою залежить те, наскільки зручною та ефективною буде взаємодія з користувачами. Однією з перевірених часом технологій є Node.JS — середовище для запуску JavaScript на сервері. Завдяки своїй швидкодії, асинхронній обробці запитів та великій кількості доступних бібліотек, ця технологія користується популярністю серед розробників. Одним із ефективних способів підвищення функціональності веб-магазину є інтеграція віджетів Elfsight — гнучких, настроюваних елементів інтерфейсу, які забезпечують розширені можливості без потреби у складному програмуванні. Вони дозволяють швидко додати на сайт відгуки клієнтів, інтерактивні форми, онлайн-чати, елементи гейміфікації та інші інструменти, що сприяють підвищенню конверсії та покращенню користувацького досвіду. Серверне середовище виконання JS, у поєднанні з Express.js, надає зручну архітектуру для створення масштабованого та гнучкого бекенду, що дозволяє ефективно управляти маршрутами, сесіями та запитами до бази даних. Завдяки цьому можна легко реалізувати динамічне завантаження віджетів Elfsight через API або інтеграцію в шаблони сторінок.

Впровадження Elfsight у проєкт дозволяє зменшити витрати на розробку індивідуальних компонентів, підвищити швидкість розробки та забезпечити адаптивність до потреб клієнтів. Це особливо актуально для малих та середніх підприємств, які прагнуть швидко вийти на ринок з конкурентним продуктом.

Крім того, Node.JS чудово поєднується з сучасними підходами до розробки — такими як мікросервісна архітектура, DevOps-практики та CI/CD. Завдяки цьому розробники можуть автоматизувати процеси розгортання, тестування та оновлення системи, що критично важливо для e-commerce платформ, де стабільність і безперервність роботи є пріоритетними. JavaScript-орієнтований стек дозволяє також швидко підключати платіжні сервіси та аналітику, допомагаючи адаптувати магазин до змін. Отже, розробка інтернет-магазину на базі Node.JS із використанням Elfsight є сучасним, адаптивним та ефективним рішенням, яке дозволяє скоротити час виходу на ринок, підвищити рівень залучення клієнтів і забезпечити якісний користувацький досвід на всіх етапах взаємодії з продуктом.

ДОДАТОК В. Диск з файлом проекту