МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА

МИХАЙЛЯК МАРИНА ОЛЕКСАНДРІВНА

Допускається до захисту:
в.о. завідувача кафедри
інформаційних технологій
канд. техн. наук, доцент
_____ О. В. Зелінська
«____»_____ 20__ р.

РОЗРОБКА МАРКЕТПЛЕЙСУ ПЕРСОНАЛЬНИХ ЗАСОБІВ БЕЗПЕКИ

Спеціальність 122 Комп’ютерні науки

Кваліфікаційна (бакалаврська) робота

Керівник:
Т. В. Січко, доцент кафедри
інформаційних технологій,
к. т. н., доцент

_____

Оцінка: _____ / _____ / _____
(бали/за шкалою ЄКТС/за національною шкалою)

Голова ЕК: _____

Вінниця – 2025

АНОТАЦІЯ

Михайляк М. О. Розробка маркетплейсу персональних засобів безпеки. Спеціальність 122 «Комп’ютерні науки», освітня програма «Комп’ютерні науки». Донецький національний університет імені Василя Стуса, Вінниця 2025.

У кваліфікаційній (бакалаврській) роботі розроблено вебдодаток-маркетплейс індивідуальних засобів захисту. Реалізовано функції перегляду та фільтрації товарів, особистого кабінету, оформлення замовлень і додавання товарів продавцями. Для розробки використано HTML, CSS, JavaScript, FastAPI та PostgreSQL. Показано зручність використання додатка для користувачів і потенціал для розвитку онлайн-торгівлі в цій сфері.

Ключові слова: вебдодаток, веброзробка, маркетплейс, Python, FastAPI, PostgreSQL.

63 ст., 1 табл., 34 рис., 2 дод., 40 джерел.

ABSTRACT

Mykhailiak M. O. Development of a Marketplace for Personal Protective Equipment. Specialty 122 "Computer Science", educational program "Computer Science". Vasyl Stus Donetsk National University, Vinnytsia, 2025.

The bachelor’s qualification work presents the development of a web application – a marketplace for personal protective equipment. The application implements features such as product browsing and filtering, user account management, order processing, and product submission by sellers. The development was carried out using HTML, CSS, JavaScript, FastAPI, and PostgreSQL. The convenience of the application for users and its potential for advancing online commerce in this domain are demonstrated.

Keywords: web application, web development, marketplace, Python, FastAPI, PostgreSQL.

ЗМІСТ

ВСТУП

Питання особистої безпеки стає все актуальнішим, адже рівень загроз життю та здоров'ю людей підвищується. Розвиток технологій сприяє появі нових рішень сфері безпеки, зокрема через створення цифрових платформ, котрі забезпечують легкий доступ до придбання необхідних засобів захисту.

Наразі користувачам часто доводиться витрачати чимало часу на пошук та купівлю персональних засобів безпеки через розрізненість інформації та відсутність єдиної спеціалізованої платформи. Це ускладнює процес придбання, оскільки складно віднайти надійні продукти або перевірених постачальників.

Розробка вебдодатку маркетплейсу персональних засобів безпеки автоматизує вибір, замовлення й доставку товарів, а також надає користувачам повну та актуальну інформацію. Впровадження пошуку й фільтрації товарів значно полегшує купівлю та підвищує ефективність взаємодії з клієнтами.

Отже, створення спеціалізованого вебдодатку для реалізації засобів захисту є важливим кроком у покращенні доступності цих товарів та оптимізації процесу їх придбання.

Мета дослідження – розробка вебдодатку маркетплейсу індивідуальних засобів захисту, для залучення цільової аудиторії та зручного і доступного процесу купівлі. Рішення надає користувачам змогу знаходити, порівнювати та купувати різні засоби захисту, оцінювати їх, а також читати рекомендації й відгуки інших користувачів.

Завдання дослідження полягають у:

- вивченні ринку персональних засобів захисту;
- аналізі потреб споживачів щодо купівлі таких товарів в інтернет-магазинах ;
- визначенні ключових вимог до функціоналу вебдодатку;
- аналізі наявних рішень на ринку;
- підборі відповідних інструментів для розробки вебдодатку;

- створенні основних компонентів вебдодатку, що забезпечать зручний інтерфейс та ефективну взаємодію з користувачами.

Об'єктом дослідження є процес розробки вебзастосунку для маркетплейсу персональних засобів захисту, що включає проектування, розробку і впровадження функціональних частин проєкту.

Предметом дослідження є архітектура та функціональні можливості вебдодатку для онлайн замовлень персональних засобів безпеки.

Розробка маркетплейсу персональних засобів безпеки є важливою як з теоретичної, так і з практичної точки зору. В теоретичному аспекті дослідження сприяє глибшому розумінню особливостей функціонування онлайн-майданчиків для продажу специфічних товарів, на кшталт засобів безпеки, а також дозволяє визначити ключові фактори, що впливають на вибір технічних рішень для створення таких платформ. Практичне значення дослідження полягає у створенні реального інструменту для бізнесу, котрий дає змогу продавати персональні засоби безпеки онлайн. Розроблений вебдодаток допомагає підприємствам організувати ефективний процес продажу, управління продуктами, а також забезпечить зручний процес покупки для кінцевих споживачів.

Структура кваліфікаційної роботи складається зі вступу, трьох розділів та висновків. У вступній частині аргументується актуальність теми, окреслюються мета, завдання, об'єкт та предмет дослідження. Перший розділ містить теоретичний аналіз наявних рішень, другий – методику створення вебдодатку, третій – практичне втілення маркетплейсу. Висновки узагальнюють головні результати, а також пропонують перспективи для майбутніх досліджень. Також, робота включає перелік використаної літератури та додатки з допоміжними матеріалами, такими як ілюстрації, таблиці й програмний код.

РОЗДІЛ 1
ТЕОРЕТИЧНІ ОСНОВИ РОЗРОБКИ МАРКЕТПЛЕЙСУ ПЕРСОНАЛЬНИХ ЗАСОБІВ БЕЗПЕКИ

1.1 Постановка задачі

У межах дослідження насамперед необхідно здійснити вивчення ринку персональних засобів захисту. Це дозволить краще зрозуміти, які саме товари є найбільш затребуваними серед споживачів, які тенденції спостерігаються в останні роки, а також якими каналами найчастіше відбувається їх реалізація. Такий аналіз допоможе визначити, які продукти доцільно включити до початкового асортименту маркетплейсу.

Наступним важливим кроком є дослідження потреб потенційних користувачів – як покупців, так і продавців. Слід з’ясувати, які труднощі вони найчастіше зустрічають під час онлайн-купівлі засобів захисту, що саме вони очікують від зручного вебзастосунку, які функції вважають пріоритетними. На основі цього буде сформульовано чіткі вимоги до функціональності вебдодатку, його інтерфейсу та логіки взаємодії.

Паралельно необхідно провести аналіз уже існуючих рішень на ринку: великих маркетплейсів, нішевих інтернет-магазинів, мобільних застосунків тощо. Це дозволить визначити їхні сильні сторони, виявити недоліки, які можуть бути усунені в новому продукті, а також уникнути повторення вже реалізованих, але неефективних підходів.

Після цього постане завдання вибору технологічного стеку – мови програмування, фреймворків, бази даних, інструментів для створення користувацького інтерфейсу. Ці рішення повинні відповідати сучасним вимогам до продуктивності, безпеки, масштабованості й зручності підтримки проєкту.

Основним етапом роботи стане безпосередня реалізація вебдодатку. Серед ключових компонентів – система реєстрації та автентифікації користувачів, каталог товарів із фільтрами та пошуком, можливість розміщення товарів продавцями, оформлення замовлень, залишення відгуків та перегляду історії

покупок. Особливу увагу буде приділено інтерфейсу – він має бути адаптивним і зрозумілим як для досвідчених користувачів, так і для новачків.

Також важливо впровадити систему ролей, що передбачає розмежування доступу та функціональності для покупців, продавців і адміністраторів. Необхідно реалізувати захист персональних даних, включаючи шифрування паролів і використання токенів для безпечної аутентифікації.

Завершальним етапом стане тестування застосунку – як на функціональному, так і на технічному рівнях. Перевірятиметься коректність виконання основних операцій, стійкість до навантаження, безпечність зберігання даних. На основі отриманих результатів можна буде сформулювати варіанти покращення проєкту, зокрема можливість інтеграції з чат-ботами, підключення платіжної системи та створення мобільної версії додатку.

1.2 Аналіз предметної області та сучасного стану ринку персональних засобів безпеки

У наш час усе більше людей задумуються про особисту безпеку – і це не дивно. Ми хочемо почуватися захищеними як фізично, так і в цифровому просторі. Через це зростає інтерес до персональних засобів безпеки. І мова йде не тільки про класичні газові балончики чи електрошокери. Сьогодні на ринку багато різних сучасних рішень – від GPS-трекерів і відеореєстраторів до смарт-брелоків та мобільних додатків, які можуть викликати допомогу миттєво.

Сучасні персональні засоби безпеки стрімко розвиваються – і все це завдяки новітнім досягненням в електроніці, мобільному зв’язку та інтернеті речей (IoT). Нові пристрої стають все меншими, зручнішими та краще взаємодіють з іншими технологіями, які ми щодня використовуємо.

Наприклад, сучасні GPS-трекери – це вже не просто засіб стеження. Вони мають датчики руху, акселерометри і навіть можуть автоматично надіслати сигнал, якщо людина раптово впала або зник зв’язок.

Розумні браслети та прикраси з функцією SOS перетворюються на стильні аксесуари, які, у разі небезпеки, дозволяють подати сигнал тривоги одним

натисканням. Вони працюють у парі з мобільними додатками чи месенджерами – все просто і швидко.

Додатки для безпеки також стають розумнішими – завдяки штучному інтелекту вони можуть помітити щось незвичне: зміну маршруту, підозрілу поведінку або інші ознаки можливої загрози.

Завдяки таким інноваціям засоби безпеки стають не лише надійними, а й максимально зручними, непомітними й адаптованими до повсякденного життя кожної людини.

Ринок персональної безпеки зараз на підйомі. У 2023 році його обсяг склав близько 60 мільярдів доларів, а до 2030 року прогнозують зростання до понад 100 мільярдів. Щороку ринок зростає в середньому на 8–10%. І це логічно, адже попит зростає. Найчастіше такими засобами користуються такі категорії населення як:

- жінки й підлітки, які хочуть почуватися спокійніше на вулиці;
- люди похилого віку, яким важливо швидко викликати допомогу;
- батьки, що піклуються про безпеку своїх дітей;
- кур’єри, таксисти та інші працівники, які часто залишаються самі.

Найбільший попит – у США, Європі й Азії. Але й в Україні ця тема стає все більш актуальною. Причинами є війна, постійне відчуття тривоги, нестабільність.

Що стимулює ріст ринку:

- зростання злочинності;
- здешевлення електроніки – тобто більше людей можуть собі це дозволити;
- онлайн-магазини, де все можна замовити за пару кліків;
- постійна напруга в кризових регіонах;
- активна медіакампанія, яка нагадує про важливість безпеки.

У цій сфері працюють як великі міжнародні компанії, так і локальні виробники, які пропонують більш доступні варіанти. Приблизна структура ринку наведена у таблиці 1.1.

Таблиця 1.1 – Структура ринку за категоріями продуктів, %

Категорія	Частка ринку, %
Газові балончики	25
Електрошокери	15
GPS-трекери	20
Тривожні кнопки	10
Розумні замки/сигналізації	15
Інше (аксесуари, додатки)	15

Коли мова заходить про покупку засобів безпеки, більшість людей звертаються до великих маркетплейсів – таких як Amazon, eBay чи Rozetka [2].

На українському ринку є кілька онлайн-платформ, де користувачі можуть знайти товари для персональної безпеки. Найбільш відомі з них – Rozetka, Prom.ua та Bezpeka.club.

Rozetka – великий універсальний маркетплейс, де представлено широкий вибір товарів, включно з газовими балончиками, сигналізаціями, трекерами тощо. Платформа зручна, з багатьма фільтрами й відгуками. Проте вона не спеціалізується саме на безпекових товарах, тому важче знайти саме те, що потрібно, серед загального асортименту.

Рисунок 1.1 – Інтернет-магазин Rozetka

Prom.ua також пропонує великий вибір товарів від різних продавців. Там можна знайти персональні засоби безпеки, однак платформа більше орієнтована на дрібний бізнес і не завжди має чітку структуру або зручну навігацію для цієї категорії товарів.

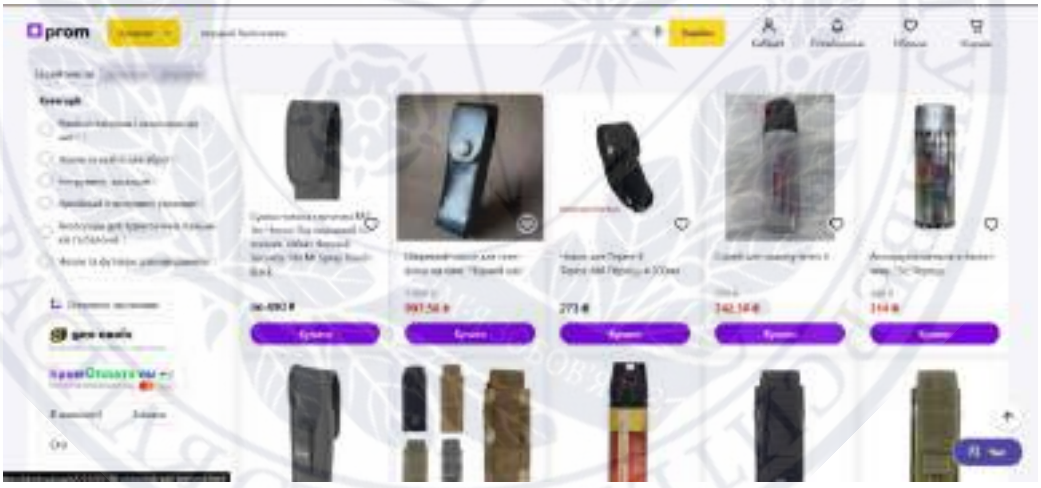

Рисунок 1.2 – Інтернет-магазин Prom.ua

Bezpeka.club – єдина з платформ, що спеціалізується саме на тематиці безпеки. Тут можна знайти інформаційні матеріали, огляди, а також придбати певні товари. Однак платформа має низку недоліків: обмежений асортимент, не завжди зручний інтерфейс, відсутність мобільного додатку та персоналізованих рекомендацій. Також не всі товари мають докладні описи або відгуки.

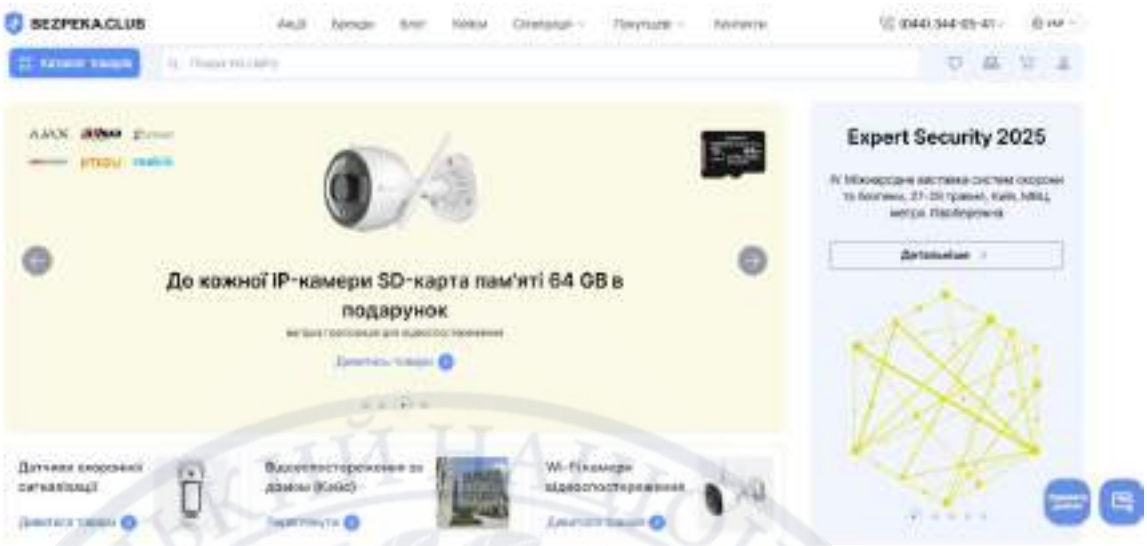

Рисунок 1.3 – Інтернет-магазин Bezpeka.club

Саме ці недоліки можна врахувати при створенні маркетплейсу, який поєднає функціональність, зручність і вузьку тематичну спрямованість. Це відкриває перспективи для створення спеціалізованого маркетплейсу, орієнтованого виключно на товари персональної безпеки – з фільтрами, рейтингами, відгуками й простим інтерфейсом [5].

Попри популярність теми, на ринку досі немає повноцінної спеціалізованої платформи, яка б зосередилася саме на товарах для персональної безпеки. І хоча є окремі виробники чи додатки, які пропонують точкові рішення, комплексного підходу бракує.

Наразі існують три типи конкурентів:

1. Великі міжнародні компанії на кшталт Garmin, Sabre, Life360 – вони розвивають технології безпеки, але продають переважно через універсальні канали.
2. Стартапи, які створюють оригінальні рішення – наприклад, Invisawear (прикраси з тривожними кнопками), Safelet (браслети з SOS), Wearsafe (сигнали з GPS).
3. Місцеві виробники, що орієнтуються на доступність – наприклад, тривожні кнопки з SIM-картами, недорогі GPS-трекери.

Але ні один із них не надає зручної платформи для користувачів з перевіреними товарами, реальними відгуками, підтримкою й гарантіями. Це саме те, що може запропонувати новий спеціалізований маркетплейс.

Тема актуальна як ніколи. Ринок активно розвивається, а люди дедалі більше хочуть простих, зрозумілих і доступних способів подбати про свою безпеку й безпеку близьких. Ідея створити окрему платформу для таких товарів – це не просто про бізнес. Це відповідь на запит часу й можливість запропонувати суспільству щось справді корисне.

1.3 Методи та підходи до створення маркетплейсів

Створення маркетплейсу – це не просто написання коду. Це розуміння людей: чого вони шукають, як обирають, що викликає довіру. Особливо, коли мова йде про таку сферу, як особиста безпека. Тут важливо не просто розмістити товари на сайті – потрібно враховувати багато деталей: тип продукції, наявність сертифікації, зручність доставки та навіть можливість швидкої реакції у критичних ситуаціях. Щоб вебзастосунок працював і приносив реальну користь, спершу потрібно визначитися з бізнес-моделлю. Тобто – як саме відбуватиметься взаємодія між продавцями й покупцями.

Є кілька популярних форматів, і кожен має свої переваги залежно від того, на кого орієнтована платформа.

- B2C (Business to Consumer) – коли компанії продають свої товари напряму людям. У цьому випадку це може бути, наприклад, виробник трекерів або постачальник газових балончиків, які напряму реалізують продукцію користувачам через платформу. Це зручно, бо гарантує якість і підтримку;
- C2C (Consumer to Consumer) – коли користувачі можуть продавати щось один одному. Наприклад, хтось купив захисний гаджет, але не користувався ним – і тепер хоче передати іншому. Такий підхід дозволяє дати речам друге життя й робить застосунок доступнішим для всіх;
- B2B (Business to Business**)** – коли бізнес працює з бізнесом. Наприклад, компанія, що займається охороною, хоче замовити одразу 100 брелоків-

тривожних кнопок для своїх працівників. Такий формат підходить для оптових закупівель і корпоративного сегмента.

Рисунок 1.4 – Основні моделі електронної комерції: B2C, C2C, B2B

Для розробки маркетплейсу персональних засобів безпеки була обрана бізнес-модель B2C. Такий формат передбачає продаж товарів безпосередньо від виробників або офіційних постачальників до кінцевих споживачів. Це дозволяє забезпечити зручний доступ до перевіреної продукції, гарантії якості та відповідність сертифікаційним вимогам.

Однак у перспективі платформа може бути розширена до моделі C2C, що дасть змогу користувачам продавати або обмінювати між собою вже використані засоби безпеки, які залишаються придатними до використання. Це розширення підвищить гнучкість сервісу, дозволить охопити ширшу аудиторію та сприятиме розвитку культури повторного використання якісних товарів.

Другим кроком стало створення структури сайту. Щоб користувачам було легко орієнтуватися на платформі, важливо було продумати, як саме буде виглядати "каркас" сайту – які сторінки будуть основними, як вони пов’язані між собою, і як зробити шлях до покупки максимально простим. Було визначено

ключові розділи, як-от головна сторінка, каталог товарів, картка товару, кошик, особистий кабінет, інструменти для зручного пошуку й фільтрації. Усе це – щоб кожен відвідувач міг швидко знайти потрібне та без проблем оформити замовлення.

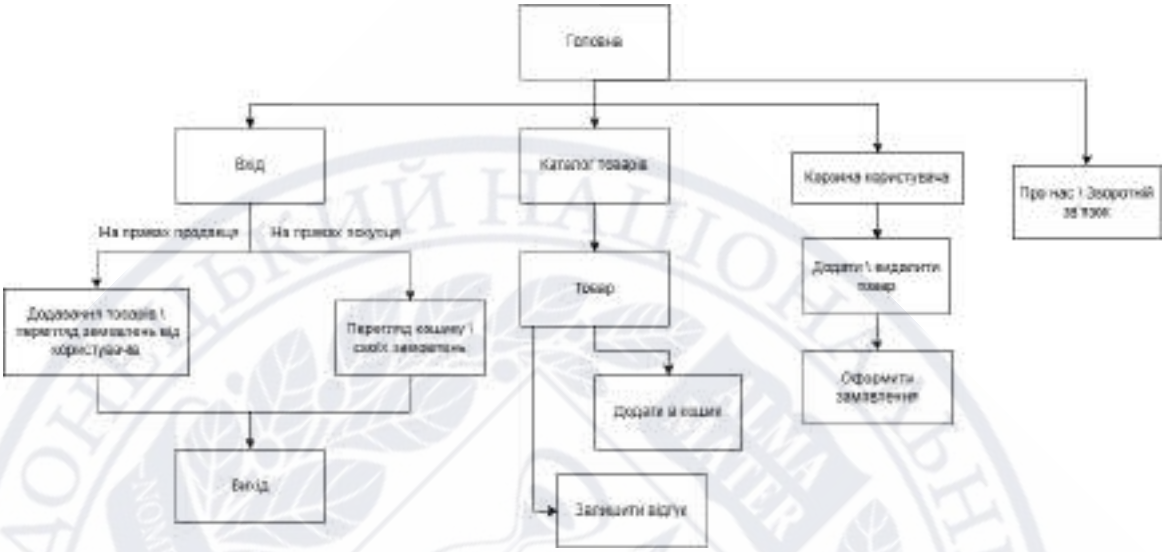

Рисунок 1.5 – Структура вебдодатку

Наступним кроком є проектування інтерфейсу користувача. Інтерфейс відіграє ключову роль у взаємодії з платформою, адже саме він формує перше враження та впливає на зручність користування. Для маркетплейсу персональних засобів безпеки важливо створити інтуїтивно зрозуміле, логічне та комфортне середовище, в якому користувач швидко знайде потрібний товар і без зайвих зусиль оформить замовлення.

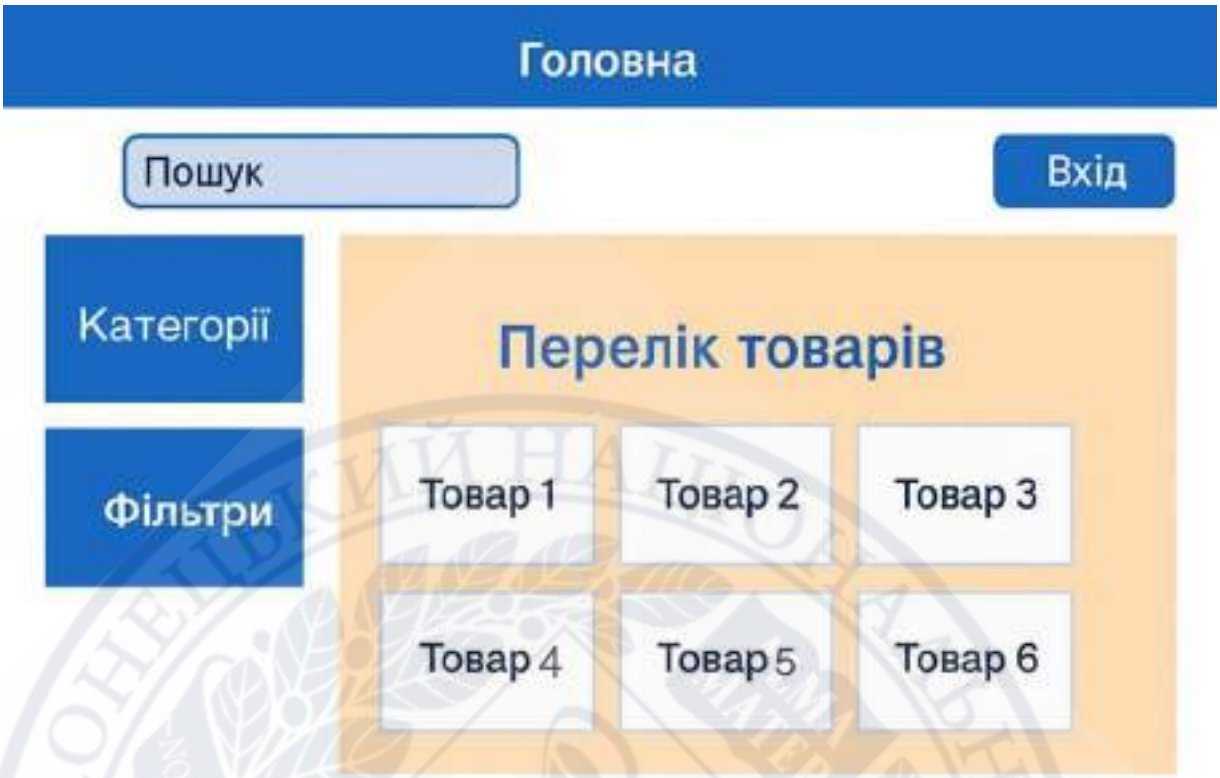

Рисунок 1.6 – Макет головної сторінки

Під час проєктування інтерфейсу було враховано такі основні принципи:

- зручна навігація – користувач має змогу легко орієнтуватися в платформі завдяки чітко структурованим категоріям і зручним фільтрам. Наприклад, передбачено можливість сортування товарів за типом, що значно полегшує пошук;
- адаптивний дизайн – інтерфейс може коректно відображатися як на комп'ютерах, так і на мобільних пристроях. Це важливо, адже значна частина користувачів здійснює покупки саме через смартфони або планшети;
- простота процесу покупки – особлива увага приділялася зручності оформлення замовлення. Користувач може швидко додати товар до кошика та оформити замовлення.

Загалом, інтерфейс розроблявся з урахуванням потреб різних категорій користувачів, щоб забезпечити максимальний комфорт та ефективність взаємодії з маркетплейсом.

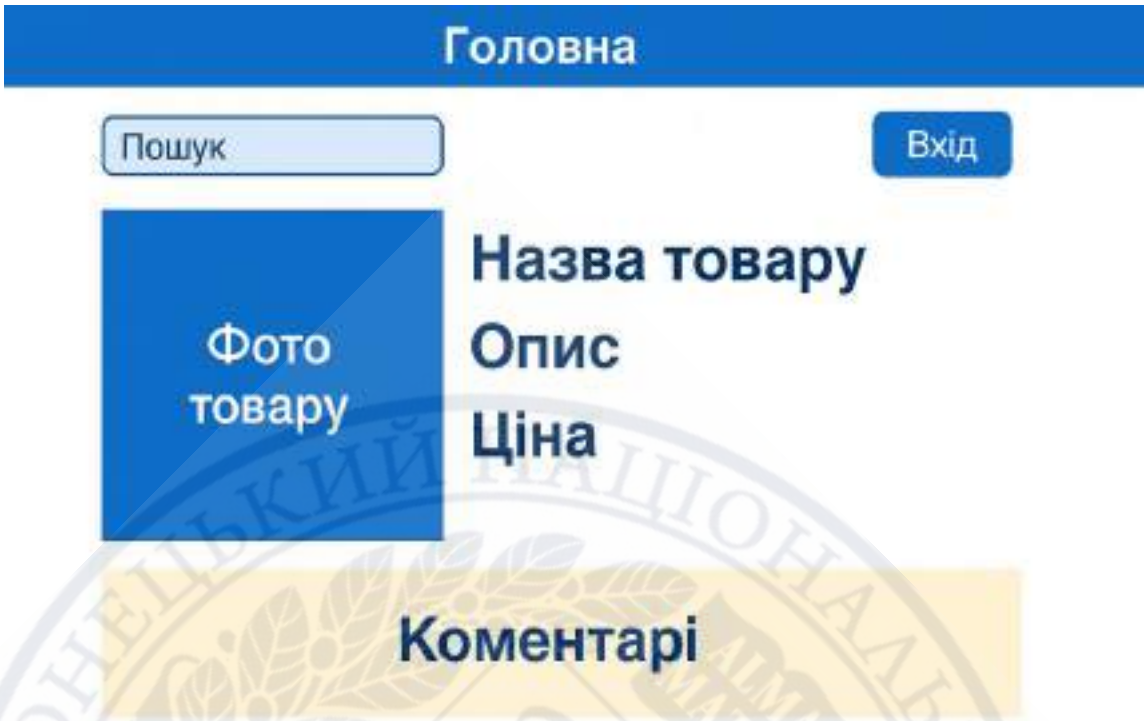

Рисунок 1.7 – Макет сторінки товару

Четвертим кроком є безпека користувачів – це не просто технічна вимога, а одна з ключових умов довіри до будь-якої онлайн-платформи. У випадку маркетплейсу, що спеціалізується на персональних засобах безпеки, ця тема набуває ще більшого значення, адже мова йде не лише про захист інформації, а й про відповідальність перед користувачами.

Одним із першочергових кроків стало впровадження шифрування при передачі даних. Завдяки використанню сучасних протоколів, інформація, яку користувач вводить на сайті, передається захищеним каналом, зводячи до мінімуму ризик її перехоплення сторонніми особами.

Не менш важливим є і те, як обробляються паролі користувачів. Для цього використовується хешування – спеціальний спосіб перетворення пароля в зашифрований вигляд, який неможливо зчитати або відновити у зворотному напрямку. Це означає, що навіть у випадку витоку бази даних, реальні паролі залишаються недоступними.

Усі ці заходи спрямовані на одне – зробити користування платформою безпечним, зберегти конфіденційність даних і створити відчуття впевненості для кожного, хто нею користується.

Наступним із найважливіших етапів у роботі маркетплейсу фізичних товарів – це організація доставки. Особливо це актуально для платформи, що спеціалізується на засобах особистої безпеки, адже покупці очікують не лише якісний товар, а й своєчасне та безпечне його отримання. Щоб задовольнити ці очікування, система доставки повинна бути максимально зручною, товари повинні доставлятись надійним перевізником до місцезнаходження клієнта. Крім того, можливість відстежувати замовлення на кожному етапі шляху надає впевненості та підвищує довіру до сервісу – адже завжди приємно знати, де саме знаходиться твій товар і коли він прибуде.

Рисунок 1.8 – Основні етапи виконання онлайн-замовлення

Для успішного залучення продавців і покупців маркетплейсу також необхідно застосовувати різноманітні маркетингові стратегії. Одним із важливих напрямків є SEO-оптимізація, яка допомагає підвищити видимість платформи у пошукових системах. Це включає не тільки правильний вибір ключових слів та мета-тегів, але й створення корисного контенту, який буде відповідати на запитання користувачів і допомагати знайти саме те, що вони шукають. Іншим ефективним інструментом є реклама в соціальних мережах. Платформи, як-от Facebook, Instagram та LinkedIn, дозволяють точно налаштовувати таргетинг, що

особливо корисно для просування товарів, пов’язаних із безпекою, та залучення цільової аудиторії. Крім того, варто розглянути партнерські програми, які допоможуть залучити нових продавців і покупців, створюючи вигідні умови для партнерів, що в свою чергу сприятиме розвитку платформи та збільшенню її популярності.

У кінцевому підсумку, всі ці методи та підходи відіграють важливу роль у розвитку маркетплейсу. Вони допомагають створити стабільну та ефективну платформу, де зручно і безпечно взаємодіють продавці та покупці. Важливо, щоб маркетплейс був помітним, доступним і привабливим для користувачів, і саме злагоджене застосування різних стратегій допомагає залучати нових учасників, підтримувати інтерес і стимулювати активність. Правильно організована комунікація з користувачами та ефективне просування платформи забезпечують її успіх і сталий розвиток у конкурентному середовищі.

1.4 Етапи розробки вебдодатку

Розробка вебдодатку для маркетплейсу персональних засобів безпеки відбувалась поетапно, що дозволило забезпечити стабільну, зручну та функціональну роботу сервісу. Кожен етап мав своє значення і впливав на якість кінцевого результату.

На початковому етапі було проведено аналітичне дослідження. Основною метою було визначення цільової аудиторії, розуміння потреб користувачів і формулювання задач, які має вирішувати вебдодаток. Ідея проєкту передбачала створення вебзастосунку, на якій продавці можуть розміщувати засоби безпеки, а покупці – ефективно знаходити і придбавати необхідні товари. Було проаналізовано функціонал схожих платформ, що дозволило врахувати як вдалий досвід, так і уникнути поширених помилок. Важливим результатом цього етапу стало формування технічних та функціональних вимог до системи.

Після завершення аналітичного етапу розпочалося проєктування. Створено логічну структуру вебдодатку: визначено необхідні сторінки, їхній вміст та можливості взаємодії. Продумано сценарії переходів між сторінками, а

також відображення інтерфейсу залежно від ролі користувача – гість чи зареєстрований користувач. Це дозволило сформувати цілісне уявлення про роботу системи й підготувати технічне підґрунтя для реалізації.

Наступним етапом стало створення бази даних – важливої складової будь-якого вебдодатку, що забезпечує надійне збереження та обробку інформації. У структурі бази передбачено зберігання даних про користувачів, товари, категорії, замовлення, доставку та інші ключові елементи системи.

Для цього було розроблено набір таблиць з чітко визначеною структурою та логічними зв’язками між ними. Такий підхід дозволив забезпечити цілісність даних і підвищити ефективність доступу до них. У якості системи керування базами даних було обрано PostgreSQL, так як це надійне та функціонально потужне рішення, яке добре підходить для складних вебпроєктів [3].

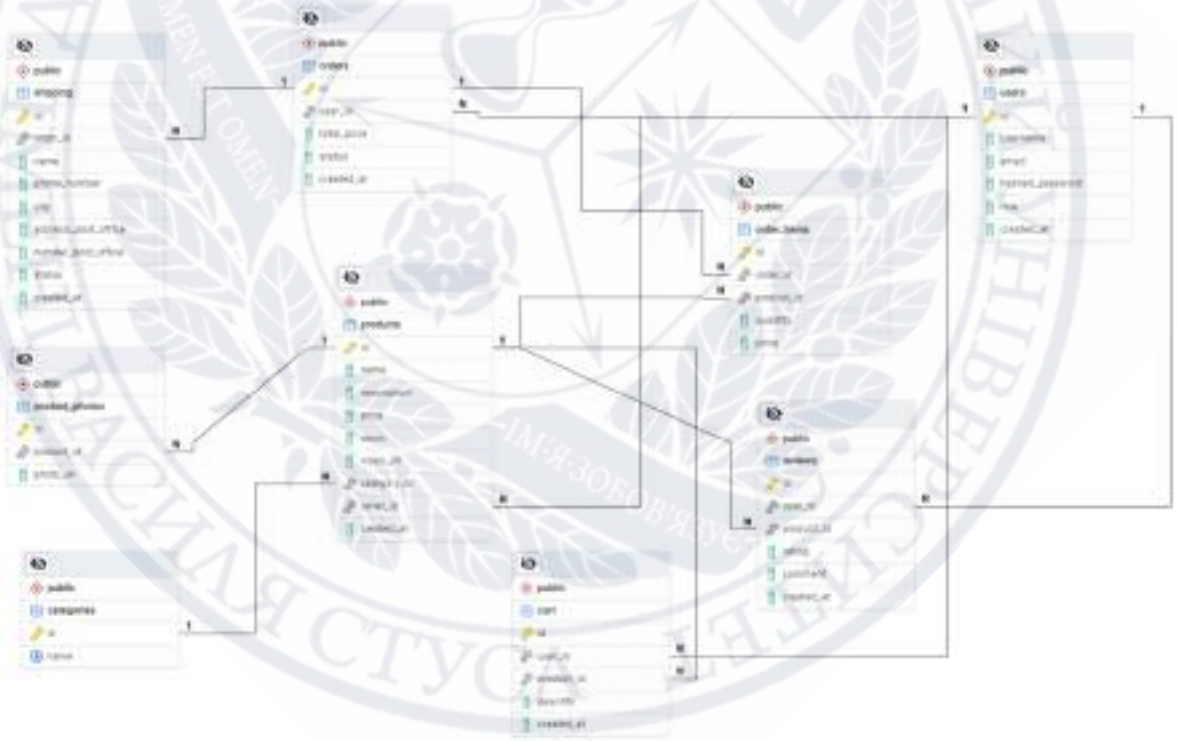

Рисунок 1.9 – ER-діаграма бази даних

Після завершення проєктування структури вебдодатку та моделювання бази даних було розпочато етап розробки користувацького інтерфейсу. Основною метою цього етапу стало створення інтуїтивно зрозумілого, візуально привабливого та зручного у використанні інтерфейсу. Особлива увага

приділялася простоті навігації, логічному розміщенню елементів та відсутності надмірного візуального навантаження.

Для реалізації зовнішнього вигляду застосовувалися стандартні вебтехнології – HTML і CSS, які забезпечують гнучкість у побудові структури сторінок та їх стилізації. З метою ефективної інтеграції динамічного вмісту з бекендом використовувалася система шаблонів Jinja2, що дозволяє відображати інформацію з бази даних у зручному форматі без втрати узгодженості дизайну.

Паралельно з розробкою інтерфейсу здійснювалося створення серверної логіки вебдодатку. Цей процес охоплював розробку функціональних механізмів взаємодії між користувачем, інтерфейсом та базою даних. Було реалізовано функціонал реєстрації та авторизації, обробки замовлень, керування товарами та користувацькими даними. Передача інформації між клієнтською та серверною частинами відбувалася у форматі, зручному для подальшої обробки та відображення.

Для серверної частини було обрано мову програмування Python у поєднанні з фреймворком FastAPI, який відзначається високою продуктивністю, сучасністю підходів та зручністю підтримки. Такий вибір дозволив реалізувати ефективну, масштабовану та стабільну архітектуру вебдодатку.

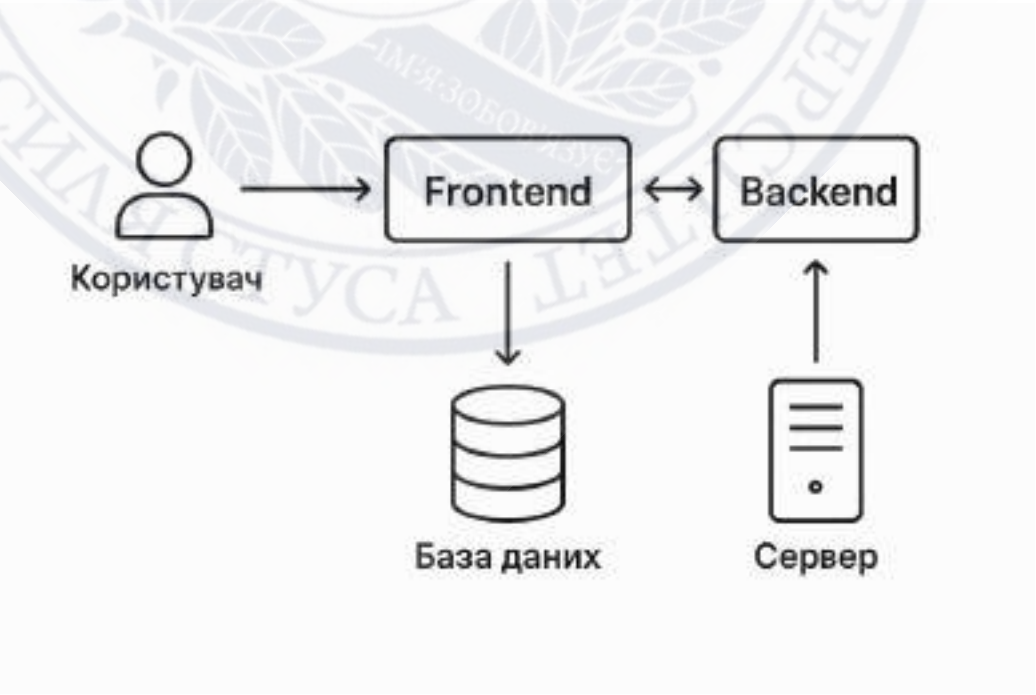

Рисунок 1.10 – Архітектурна схема вебдодатку

Після реалізації основної логіки системи розпочався етап тестування. Метою було перевірити, чи працює сайт стабільно, а всі функції – відображення товарів, оформлення замовлень, фільтрація, авторизація – виконуються коректно. Було проведено ручне тестування інтерфейсу та перевірку поведінки системи в разі помилок користувача. Це допомогло виявити та виправити кілька недоліків.

Після тестування було проведено оптимізацію. Окремі частини коду були спрощені або покращені. Також вдосконалили дизайн для покращення зручності користування.

У підсумку, розробка вебдодатку для маркетплейсу персональних засобів безпеки включала кілька ключових етапів: аналітику, проєктування структури, створення бази даних, розробку інтерфейсу та серверної частини, тестування та оптимізацію.

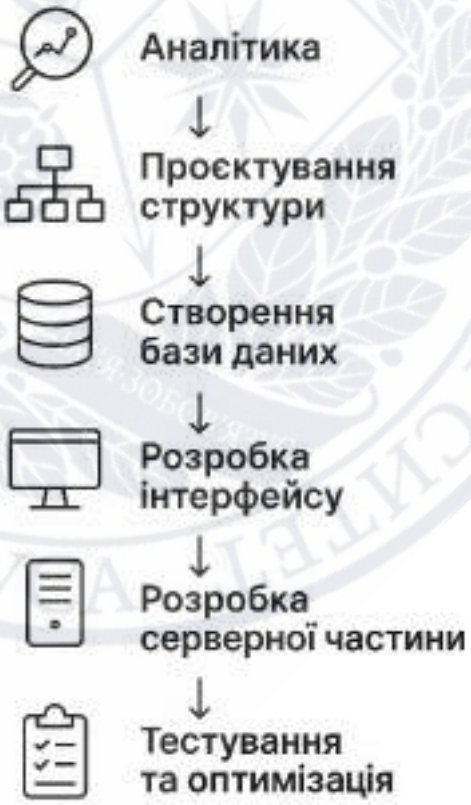

Рисунок 1.11 – Етапи розробки вебдодатку

Кожен етап був важливим для досягнення мети – створення стабільного, зручного та функціонального вебсервісу. Завдяки ретельному підходу до кожного етапу вдалося реалізувати ефективну платформу, що відповідає вимогам користувачів та забезпечує зручний досвід взаємодії.

РОЗДІЛ 2
ПРОЕКТУВАННЯ, МОДЕЛЮВАННЯ ТА ВИБІР ТЕХНОЛОГІЙ

2.1 Проектування архітектури вебдодатку

Архітектура вебдодатку – це, по суті, його “скелет”: вона визначає, як влаштована система зсередини, як її частини взаємодіють між собою та як саме обробляються запити від користувача. Саме від правильно побудованої архітектури залежить, наскільки ефективно працюватиме застосунок, чи буде він зручним у підтримці, безпечним і чи зможе зростати разом із навантаженням [1].

Найпоширеніший підхід – архітектура клієнт-сервер. Там клієнт (наприклад, браузер) надсилає запит, а сервер обробляє його і повертає відповідь. Така модель зрозуміла й надійна, тому її часто використовують для створення сайтів, інтернет-магазинів, CRM-систем, маркетплейсів та інших вебплатформ. Вона дозволяє чітко розділити обов’язки між клієнтом і сервером, а також дає змогу масштабувати систему в разі зростання кількості користувачів.

Ще один популярний варіант – трирівнева архітектура. У ній виділяють три основні шари: інтерфейс користувача (UI), сервер додатків, де вся бізнес-логіка і база даних. Такий підхід допомагає краще організувати код і робить систему більш захищеною – адже всі дані зберігаються окремо від логіки та візуальної частини. Цей тип архітектури часто використовують у великих компаніях, банках і фінансових системах.

Також у останні роки все більшої популярності набуває мікросервісна архітектура. У ній замість єдиного великого додатку створюється набір невеликих сервісів, кожен із яких виконує одну конкретну задачу. Ці сервіси спілкуються між собою через API. Головна перевага такого підходу – гнучкість: кожен сервіс можна оновлювати, масштабувати або замінювати окремо. Це дуже зручно для великих платформ із високими навантаженнями, наприклад, соцмереж або онлайн-банкінгу.

Проте для невеликих і середніх проєктів часто використовують монолітну архітектуру. Весь застосунок – це один спільний проєкт, але логіка в ньому

розділена на окремі модулі: робота з користувачами, управління товарами, збереження даних тощо. Такий підхід простіший у реалізації, що робить його зручним на початкових етапах. Єдине, про що варто пам’ятати це те, що з часом масштабування може вимагати додаткових зусиль.

Рисунок 2.1 – Різниці між архітектурами

Отже, вибір архітектури – це стратегічне рішення, яке залежить від цілей і масштабів проєкту. Важливо створити таку структуру, яка буде надійною, зрозумілою для команди та готовою до розвитку в майбутньому.

Під час створення маркетплейсу персональних засобів безпеки було обрано використовувати архітектуру «клієнт-сервер». Це надійний підхід, особливо ефективний для вебзастосунків. Поділ між клієнтом і сервером дозволяє чітко розмежувати функції: клієнт відповідає за зовнішній вигляд та взаємодію з користувачем, а сервер – за логіку, обробку запитів і доступ до бази даних.

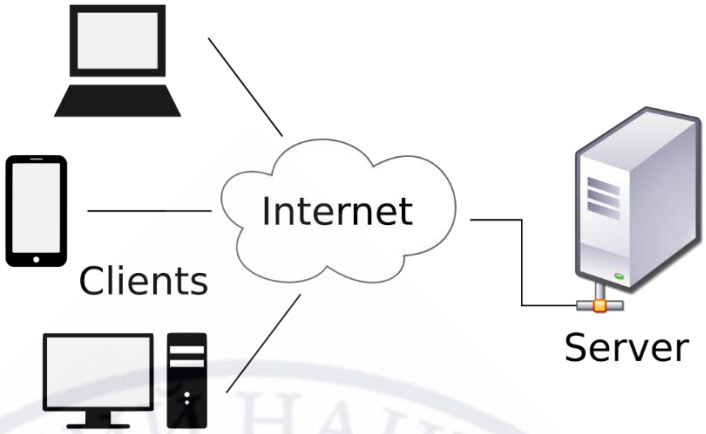

Рисунок 2.2 – Взаємодія клієнтів з сервером через мережу Інтернет

Серверну частину було реалізували на основі FastAPI – сучасного Python-фреймворку, який відомий своєю швидкістю, зручністю та підтримкою асинхронних запитів. Це дає змогу краще масштабувати проєкт, тобто додавати нові можливості без втрати продуктивності. Код поділений на окремі модулі: реєстрація й авторизація, управління та замовлення товарів – усе логічно структуровано.

Для зберігання даних було обрано PostgreSQL – надійну і потужну систему, яка добре справляється з великими обсягами інформації. Опис моделей бази даних зроблено через SQLAlchemy – це дозволяє працювати з таблицями так, ніби вони звичайні Python-класи. Такий підхід значно спрощує розробку та читається набагато краще.

Інтерфейс користувача побудований на основі HTML-шаблонів із використанням Jinja2. Це означає, що сторінки формуються динамічно – наприклад, список товарів або кошик покупця оновлюються в реальному часі на основі даних із сервера. Кожна сторінка генерується сервером і надсилається користувачеві вже у готовому вигляді.

Система безпеки базується на токенах: після авторизації користувач отримує унікальний токен, який дозволяє отримувати доступ до закритих розділів додатку. Це гарантує, що конфіденційна інформація й функції будуть доступні лише тим, хто має на це дозвіл, тобто був авторизований.

Загалом, така архітектура забезпечує гнучкість та зручність розвитку проєкту. Кожен компонент взаємодіє з іншими через чіткі інтерфейси, тому додавати нові функції або змінювати існуючі частини можна безболісно – це важливо для підтримки та оновлення системи в майбутньому.

2.2 Моделювання процесів та алгоритмів функціонування маркетплейсу

Моделювання процесів – це важлива складова при розробці програмного забезпечення, зокрема вебдодатків. Воно допомагає краще зрозуміти, як саме функціонуватиме система, ще до початку програмування. Як і у випадку з будівництвом, де спочатку створюються ескізи та креслення, а вже потім – сам проєкт, у розробці програм логічно спершу уявити роботу системи. Це дозволяє вчасно виявити слабкі місця, продумати взаємодію користувачів і підготувати основу для ефективного впровадження.

Для маркетплейсу, орієнтованого на продаж засобів безпеки, важливо змоделювати як ролі користувачів, так і основні дії, які вони виконують на платформі. До ключових користувачів належать:

- покупець, який переглядає товари, фільтрує їх за параметрами, додає до кошика, оформлює замовлення та залишає відгуки;
- продавець, що створює акаунт, додає та редагує товари, обробляє замовлення;
- адміністратор, який контролює зміст платформи, слідкує за активністю користувачів.

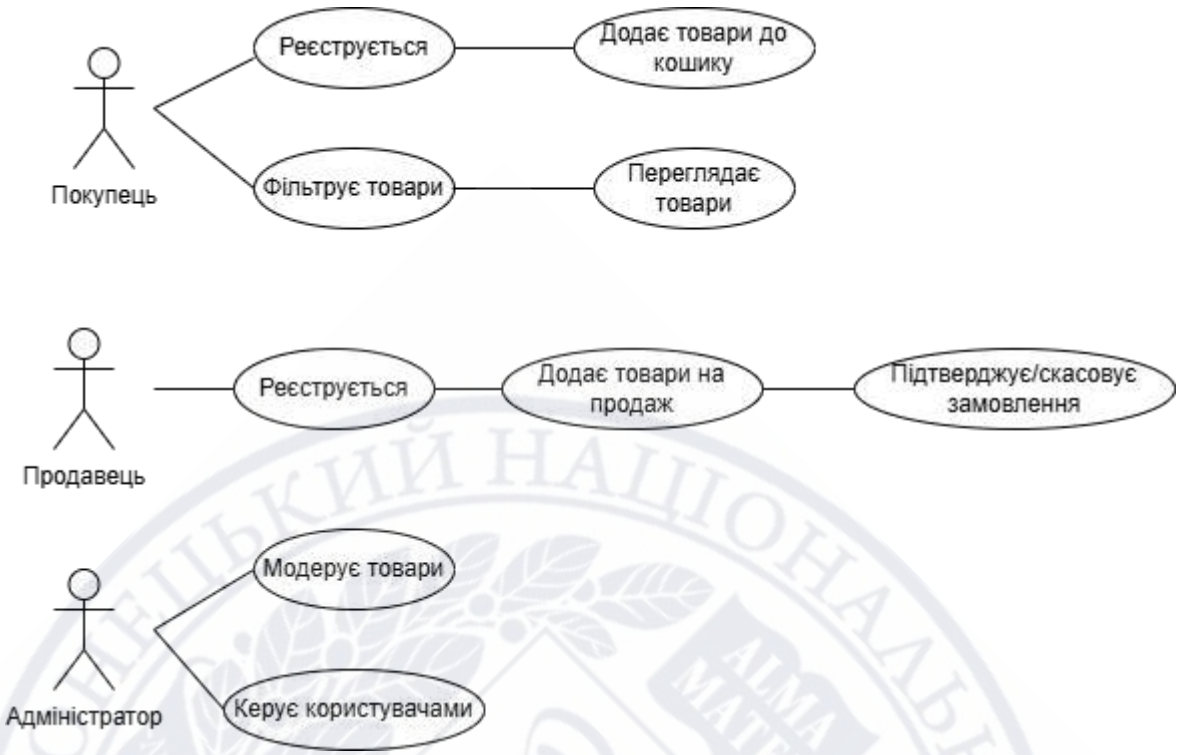

Рисунок 2.3 – UML діаграма варіантів використання системи

Серед основних процесів, які визначають роботу системи, можна виділити:

- реєстрацію та авторизацію користувачів;
- додавання товарів до каталогу;
- пошук і фільтрацію товарів;
- оформлення замовлень та обмін інформацією між покупцем і продавцем;
- подальшу обробку замовлень;
- оцінювання товарів та залишення відгуків.

Ці дії реалізуються за допомогою алгоритмів – покрокових інструкцій, які описують, як саме система реагує на дії користувача або зміни у базі даних. Наприклад:

- реєстрація користувача передбачає перевірку, чи вже існує обліковий запис із вказаним e-mail чи логіном. Якщо користувач новий, його пароль шифрується за допомогою bcrypt, генерується JWT-токен для авторизації, і вся інформація зберігається в базі даних. У відповідь користувач отримує токен або повідомлення про помилку;

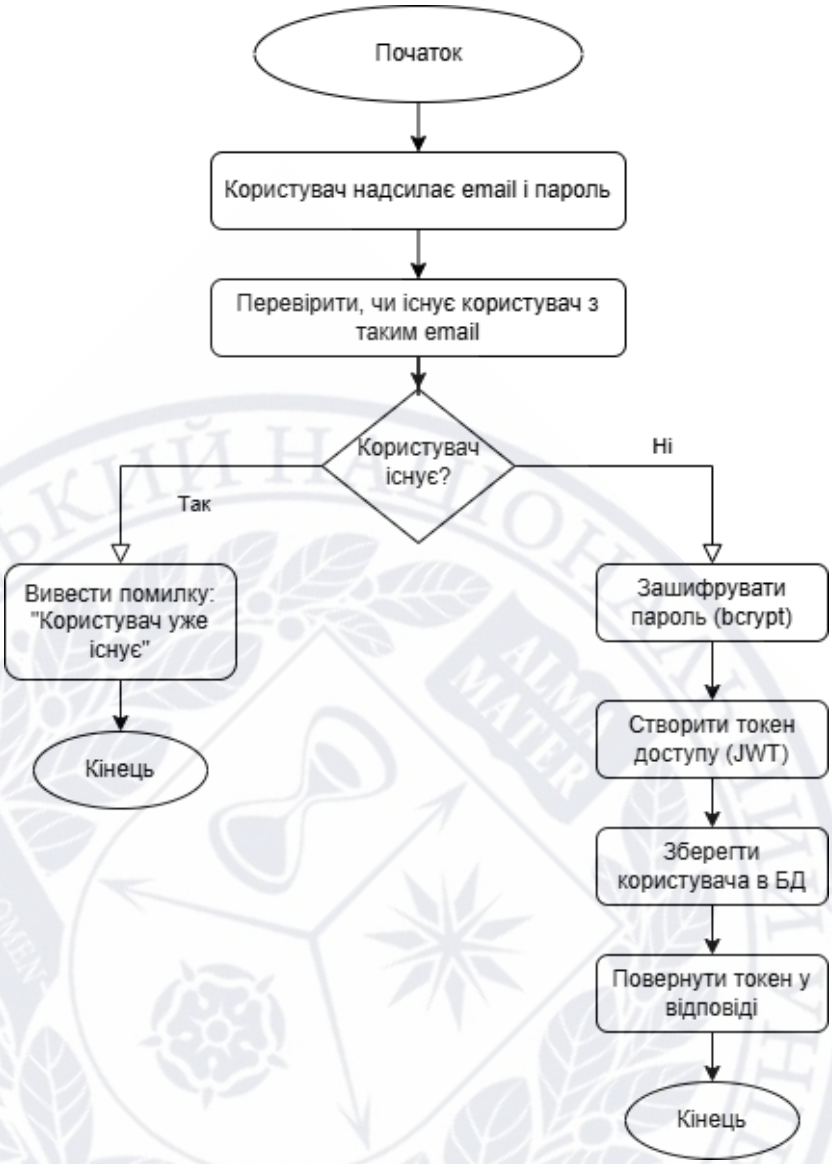

Рисунок 2.4 – Блок-схема алгоритму реєстрація користувача

- додавання товару продавцем вимагає перевірки прав доступу. Після підтвердження система приймає назву, опис, категорію, ціну й зображення товару, зберігає ці дані в базу і повертає повідомлення про успішне додавання або про помилку;

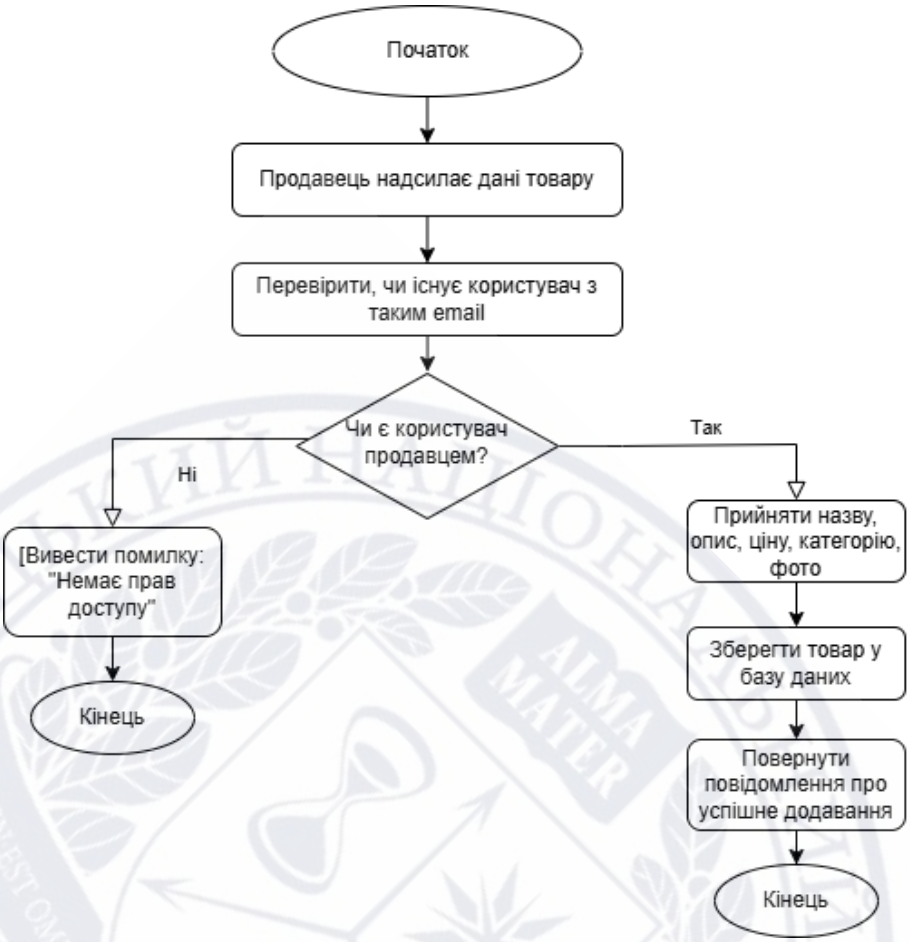

Рисунок 2.5 – Блок-схема алгоритму додавання товару продавцем

- пошук товарів реалізується шляхом прийому параметрів запиту (ключові слова, категорія, діапазон цін). На їх основі формується SQL-запит для бази даних, після чого користувачеві повертається список відповідних товарів;

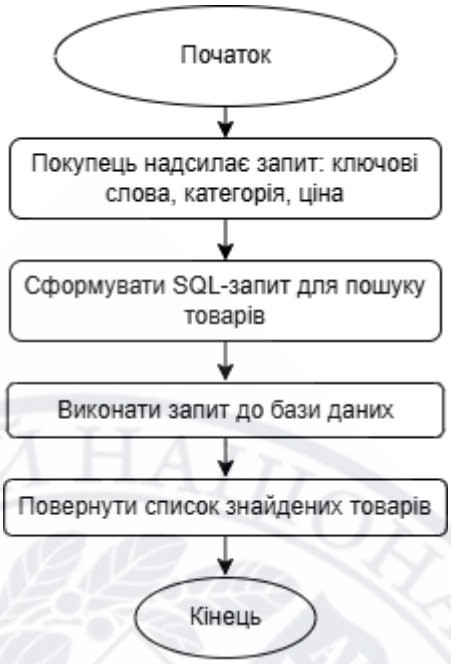

Рисунок 2.6 – Блок-схема алгоритму пошуку товарів

- оформлення замовлення відбувається після додавання товару до кошика. Система зберігає замовлення з прив’язкою до конкретного товару та покупця, призначаючи йому статус "у процесі";

Рисунок 2.7 – Блок-схема алгоритму оформлення замовлення

- робота продавця із замовленням включає можливість підтвердити або скасувати його. Система оновлює статус відповідно до дії продавця, а покупець одразу бачить зміну у своєму кабінеті;

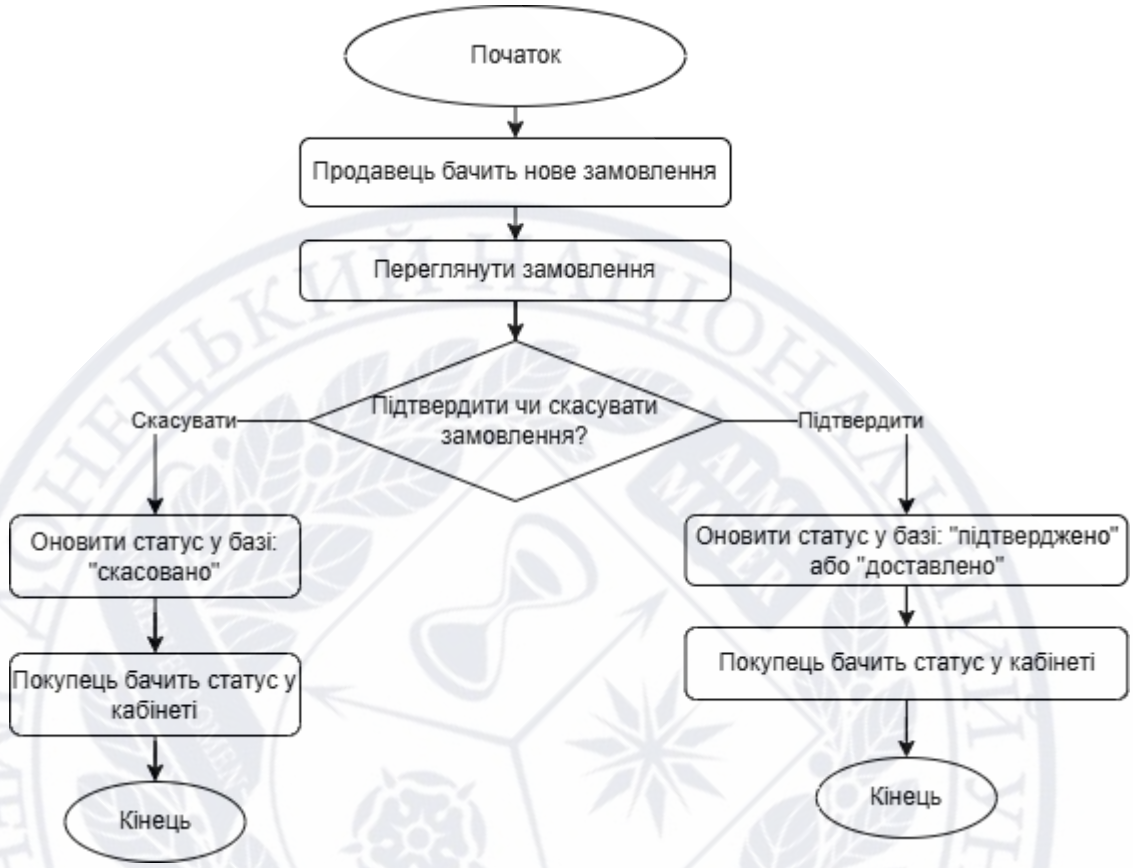

Рисунок 2.8 – Блок-схема алгоритму підтвердження або скасування замовлення продавцем

- залишення відгуку користувачем можливе на сторінці товару, там є поле для вводу текстового коментаря та рейтингу. Після цього рейтинг товару автоматично оновлюється.

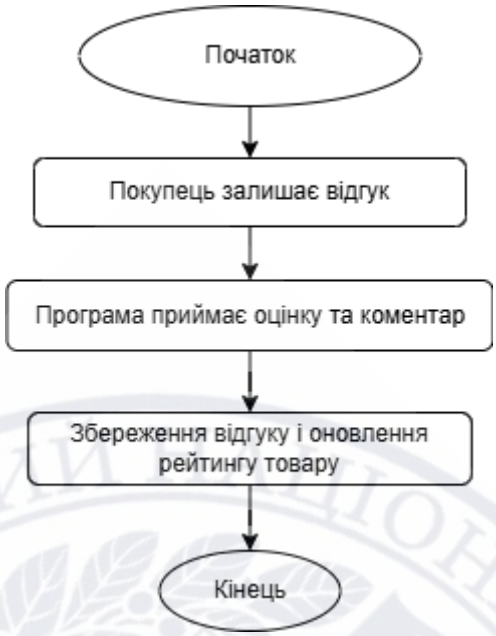

Рисунок 2.9 – Блок-схема алгоритму залишення відгуку

Усі описані алгоритми реалізуються за допомогою фреймворку FastAPI, де кожна дія користувача обробляється окремим endpoint'ом. Вся логіка обробки даних винесена в окремі функції, а сама база даних побудована на PostgreSQL із використанням SQLAlchemy як ORM, що дозволяє працювати з таблицями як з об'єктами Python. Завдяки такій структурі маркетплейс працює узгоджено, передбачувано й зручно для всіх користувачів платформи.

2.3 Вибір програмного забезпечення та інструментів для розробки

Для розробки маркетплейсу персональних засобів безпеки були обрані сучасні, зручні та ефективні інструменти. Головна мета – створити стабільний, швидкий і зрозумілий для користувача вебзастосунок.

Одним із ключових рішень при створенні будь-якого вебзастосунку є вибір мови програмування. Саме від цього рішення залежить не лише швидкість розробки, а й зручність подальшої підтримки, продуктивність системи, рівень безпеки та легкість у навчанні для майбутніх учасників команди.

Під час розробки маркетплейсу персональних засобів безпеки було розглянуто кілька найпопулярніших мов програмування, які широко використовуються у веброзробці.

- JavaScript – основна мова для створення інтерфейсів (фронтенду). Також її можна використовувати на сервері завдяки Node.js. Основна перевага – одна мова для всього застосунку. Це спрощує розробку та зменшує потребу в перемиканні між мовами. Вона дуже гнучка, має велику спільноту та безліч бібліотек.
- PHP довгий час був стандартом у світі веброзробки. Він чудово підходить для генерації HTML-сторінок і використовується в таких платформах, як WordPress. Для невеликих сайтів або проектів зі статичним контентом це – робоче рішення. Однак у сучасній розробці, особливо для створення API та більш складних вебзастосунків, PHP поступається новішим підходам за зручністю, швидкістю і підтримкою
- Java – потужна і безпечна мова, яка часто використовується у великих корпоративних системах. Вона забезпечує хорошу продуктивність і масштабованість. Але вимагає написання великої кількості коду навіть для простих речей, що уповільнює розробку. Для невеликих команд і проектів із середньою складністю Java може бути надмірною.
- Мова C# у поєднанні з платформою ASP.NET дозволяє створювати стабільні та масштабовані вебзастосунки. Вона має сучасні можливості, високу швидкість виконання та зручні інструменти для розробників. Проте C# найкраще працює в межах екосистеми Microsoft, що може обмежити вибір серверів і технологій.
- Python – мова, яка швидко завоювала популярність завдяки своїй простоті, легкості у вивченні та гнучкості. Вона активно використовується у веброзробці, аналізі даних, штучному інтелекті та автоматизації. У веброзробці Python добре поєднується з сучасними фреймворками.

Для реалізації маркетплейсу персональних засобів безпеки було прийнято рішення використовувати Python як основну мову програмування [31]. Основними причинами такого вибору стали простота синтаксису, швидкість

розробки, наявність великої кількості бібліотек та активна спільнота розробників.

Python дозволяє ефективно реалізовувати як базовий функціонал, так і більш складну логіку взаємодії між користувачами, обробки замовлень, роботи з базами даних та авторизації. Завдяки цьому мова є чудовим вибором як для створення прототипу, так і для подальшого масштабування застосунку.
Такий підхід забезпечує гнучкість, надійність і можливість розширення системи у майбутньому без необхідності повної зміни технологічного стеку.

Наступний крок – вибір каркаса проєкту. Фреймворк – це набір готових інструментів і правил, який допомагає набагато швидше розробляти вебдодатки. Він прописує структуру розробки проекту і зазвичай вирішує багато стандартних завдань: маршрутизацію, доступ до бази даних, авторизацію, обробку запитів і т.д.

Існує кілька популярних фреймворків Python. Кожен має свої переваги та недоліки та призначений для різних застосувань [32]:

1. Django – найвідоміший і найстаріший фреймворк Python. Його головна перевага полягає в тому, що він має майже все в комплекті, як панель адміністратора, зручний ORM для роботи з базою даних, систему авторизації, маршрутизацію, що робить його дуже зручним, особливо при роботі з великими сайтами або класичними веб-додатками. Але коли потрібно створювати лише API, тобто без інтерфейсу, може здатися, що з Django важко працювати.
2. Flask – легкий і мінімалістичний фреймворк. Він забезпечує максимальну свободу, маючи в собі лише базову функціональність, а все інше додається поступово зовнішніми бібліотеками. Це робить його сприятливим варіантом для невеликих або нестандартних проектів, які потребують більшого контролю над кодом.
3. FastAPI – сучасний і дуже швидкий фреймворк. Він з’явився нещодавно, але вже став дуже популярним. Працює на асинхронних запитах, що дозволяє обслуговувати багато користувачів одночасно.

Має такі переваги як:

- автоматично створює документацію до API (Swagger, Redoc);
- перевіряє правильність вхідних даних;
- підтримує підказки типів прямо у коді;
- працює дуже швидко, майже як JavaScript.

FastAPI добре підходить для створення API, які використовують сайти, мобільні додатки або чат-боти.

Після порівняння фреймворків було обрано саме FastAPI. Він поєднує швидкість, простоту та сучасні можливості. Завдяки цьому вдалося швидко створити зручний і гнучкий API для маркетплейсу. До нього легко підключити інші сервіси і масштабувати проєкт у майбутньому.

Також одним із найважливіших аспектів будь-якого вебзастосунку є база даних. Саме там зберігається вся важлива інформація: користувачі, товари, замовлення, повідомлення тощо. Тому при створенні маркетплейсу важливо вибрати надійну, гнучку і зручну СУБД [34].

1. SQLite – найпростіший варіант. Це база даних, яка зберігається у вигляді одного файлу і не потребує сервера. Вона дуже зручна для тестів або маленьких проєктів. Але для постійно працюючого маркетплейсу вона не підійде. Якщо багато людей будуть користуватись сайтом одночасно, можуть виникнути конфлікти й проблеми з продуктивністю.
2. MySQL – це одна з найпопулярніших баз даних. Вона швидка, надійна, і працює на більшості хостингів. Ця СУБД добре підходить для великих проєктів і має багато інструментів для безпеки й резервного копіювання. Але іноді працювати з нестандартними даними (як-от JSON або складними зв’язками) не так зручно.
3. PostgreSQL – потужна і сучасна база даних з відкритим кодом. Вона підтримує не лише звичайні таблиці, а й більш складні типи даних: JSON, масиви, координати, повнотекстовий пошук тощо. Це ідеальний варіант для складних проєктів. PostgreSQL легко справляється з великим навантаженням,

має зручну мову запитів і дозволяє створювати гнучку структуру з хорошими зв’язками між таблицями.

4. MongoDB – це база, яка працює не з таблицями, а з документами у форматі JSON. Вона дуже гнучка і зручна, якщо дані часто змінюються або мають вкладену структуру. Тому її часто використовують у стартапах. Але MongoDB не має класичних зв’язків між таблицями. А для маркетплейсу, де багато пов’язаних сутностей (товари, користувачі, замовлення), це може створити складнощі.

Для розробки маркетплейсу було обрано PostgreSQL [21]. Вона добре підходить для такого застосунку, який має рости й розвиватися. Вона швидка, стабільна та зручно працює з Python. Її можливості дозволяють будувати гнучкі зв’язки між даними, працювати з різними типами полів і витримувати велике навантаження. Це надійна основа для будь-якого серйозного проєкту.

Для взаємодії з базою даних використовується SQLAlchemy. Це ORM-бібліотека, яка дозволяє працювати з базою не через сирі SQL-запити, а зручніше – у вигляді об’єктів Python. Так легше писати і підтримувати код.

Також фронтенд є важливою складовою вебдодатків – це все, що бачить користувач. Кнопки, сторінки, форми – саме через них користувач взаємодіє з сайтом. У маркетплейсі це особливо важливо. Людина має легко знаходити потрібний товар, оформлювати замовлення без зайвих труднощів.

Щоб створити зручний інтерфейс використовувались HTML-шаблони. Це звичайні HTML-сторінки, в які можна вставляти динамічні дані, наприклад, ім’я користувача або список товарів. У Python дуже зручно працювати з шаблонізатором Jinja2. Він простий у використанні і добре взаємодіє з FastAPI, який використовується в проєкті.

Шаблони дозволяють створити гарні сторінки без складного JavaScript. До них легко підключити CSS або готові стилі з бібліотек, наприклад, Bootstrap чи Tailwind.

Щоб інтерфейс не був статичним, було додано трохи JavaScript. Завдяки цьому можна оновлювати дані без перезавантаження сторінки, надсилати запити

до API, передавати токени тощо. Не використовувались важкі фреймворки – лише прості скрипти на основі fetch().

Це дозволило зробити сторінку більш "живою", але не ускладнювало проєкт.

Для стилів було обрано CSS. Стилі писались вручну – задаючи кольори, розміри, відступи, шрифти. Такий підхід дав повний контроль над виглядом сторінок. Він не залежить від сторонніх бібліотек і добре підходить для проєкту.

У розробці вебдодатку маркетплейсу поєднано простоту HTML-шаблонів з можливостями Jinja2, CSS і невеликої кількості JavaScript. Це дозволило створити зручний інтерфейс для користувачів, не перевантажуючи проєкт. Такий підхід ідеально підходить для MVP та невеликих проєктів.

Врешті-решт, щоб писати код швидко й без зайвих нервів, потрібно мати зручне середовище розробки. Воно має підсвічувати синтаксис, підказувати, що писати далі, знаходити помилки та дозволяти все запускати прямо з редактора.

Було обрано Visual Studio Code – безкоштовний редактор коду від Microsoft, яким користуються програмісти в усьому світі. Він працює з різними мовами програмування, а завдяки плагінам – підходить майже під будь-який проєкт.

VS Code має такі переваги як:

- підсвічує код і пропонує автодоповнення для Python, HTML, SQL;
- має багато плагінів – наприклад, для FastAPI, SQLAlchemy, Git, PostgreSQL, Jinja2;
- дозволяє запускати сервер і відслідковувати помилки прямо в редакторі;
- зручно працює з Git – можна бачити зміни, комітити, відкочуватися;
- має вбудований термінал – не потрібно перемикатися між вікнами.

Крім того, у VS Code приємний інтерфейс, є автозбереження, теми оформлення, підказки – усе, щоб працювати було комфортно.

РОЗДІЛ 3
ПРАКТИЧНА РОЗРОБКА ВЕБДОДАТКУ

3.1 Розробка бекенду: функціональні можливості вебдодатку

У межах розробки вебдодатку маркетплейсу персональних засобів безпеки було створено бекенд за допомогою сучасного Python-фреймворку FastAPI, який забезпечує високу продуктивність, зручну інтеграцію з базами даних та автоматичну генерацію документації API.

Насамперед потрібно почати з реалізації системи аутентифікації та авторизації – це один з ключових елементів будь-якого сучасного вебдодатку, особливо коли йдеться про маркетплейс. Вона дозволяє забезпечити безпечний доступ до особистого кабінету користувача, захистити його дані та контролювати, хто і до чого має доступ.

У вебдодатку для маркетплейсу реалізовано два основні механізми безпеки:

- аутентифікація – це підтвердження особи користувача, тобто процес входу в систему;
- авторизація – це визначення, які саме дії може виконувати користувач після входу, залежно від його ролі.

Користувачі можуть зареєструватися через спеціальний API-запит, де вводять свій логін, електронну пошту, пароль. Щоб убезпечити облікові записи, паролі не зберігаються у відкритому вигляді. Замість цього використовується хешування за допомогою алгоритму bcrypt. Тобто кожен пароль проходить додатковий захист перед тим, як потрапити до бази даних.

Після реєстрації користувач може увійти до системи, ввівши свої дані. У разі успішного входу сервер створює токен доступу (JWT), який передається клієнту. Надалі цей токен додається до кожного запиту – ідентифікуючи користувача та дозволяючи йому взаємодіяти з захищеним функціоналом.

JSON Web Token (JWT) – це сучасний і безпечний спосіб передачі зашифрованої інформації між клієнтом і сервером. Після успішного входу

користувача створюється токен, який містить ID користувача та строк його дії. Цей токен не передається в заголовках запиту, як зазвичай, а зберігається у cookie – спеціальному файлі, який автоматично надсилається браузером при кожному запиті до сервера. Такий підхід робить взаємодію з API зручнішою для користувача і дозволяє краще захищати токен від доступу зі стороннього JavaScript (наприклад, шляхом встановлення флагів HttpOnly та Secure). Сервер перевіряє токен при кожному запиті. Якщо він дійсний – запит виконується. Якщо ні – система одразу відмовляє у доступі. Такий підхід суттєво підвищує безпеку.

```
@auth_router.post('/login')
def login (response: Response, form_data: OAuth2PasswordRequestForm = Depends(),
          db: Session = Depends(app.database.get_db)):
    user = db.query(app.models.User).filter(app.models.User.email == form_data.username).first()

    if not user or not verify_password(form_data.password, user.hashed_password):
        raise HTTPException(status_code=400, detail='invalid password')

    token = create_access_token(data={'sub' : user.email, 'id' : user.id})
    response.set_cookie(key='Authorization', value=f'Bearer {token}',
                        httponly=True, secure=True, path='/')

    return {'access_token' : token, 'token_type' : 'bearer'}
```

Рисунок 3.1 – Приклад коду для аутентифікації користувача

У додатку реалізовано рольову модель доступу, що визначає, які саме функції доступні користувачеві. Є три основні ролі:

- покупець – переглядає товари, додає їх до кошика, оформлює замовлення, залишає відгуки;
- продавець – додає свої товари, редагує або видаляє їх, переглядає замовлення;
- адміністратор (опціонально) – має повний доступ до системи: модерація товарів, управління користувачами, перегляд усіх замовлень та відгуків.

Роль зберігається в базі даних.

Завдяки поєднанню FastAPI, bcrypt і JWT вдалося створити зручну та безпечну систему входу і доступу. Вона надійно захищає персональні дані, дає

можливість гнучко керувати доступом і легко масштабувати систему в майбутньому – наприклад, додавши нові ролі або двофакторну аутентифікацію.

Сторінка кожного товару на маркетплейсі є важливою частиною платформи, що забезпечує користувачів всією необхідною інформацією про продукт і дає змогу взаємодіяти з ним. Вона не тільки надає деталі товару, але й дозволяє користувачам додавати товар у кошик. Для реалізації такої сторінки на бекенді важливо правильно організувати зберігання та обробку даних товарів, їх відображення, а також інтеграцію з іншими частинами системи, такими як кошик, замовлення та система відгуків.

Кожен товар на платформі має свій окремий запис у базі даних, який можна обробляти через API. Коли користувач відкриває сторінку товару, бекенд надає всю необхідну інформацію для його відображення:

- дані про товар: це основна інформація, яка включає назву, опис товару, його ціну та зображення;
- кількість товару в наявності: бекенд відслідковує кількість товару, що є в наявності, і передає цю інформацію користувачу, щоб він міг бачити, скільки одиниць товару доступно для покупки;
- рейтинг товару: рейтинг товару обчислюється на основі оцінок користувачів, які залишили їх після покупки. Бекенд зберігає ці оцінки в базі даних і регулярно оновлює середній рейтинг товару. Актуальний рейтинг передається на фронтенд через API, щоб користувач міг бачити середню оцінку товару, що допомагає йому прийняти рішення про покупку;
- коментарі до товару: всі коментарі до товару зберігаються в окремій таблиці бази даних. Кожен коментар містить текстову оцінку товару та ідентифікатор користувача, який залишив відгук. Бекенд забезпечує можливість користувачам залишати нові коментарі після покупки. Коментарі передаються через API на фронтенд, де вони відображаються разом з оцінкою та іншою інформацією про товар.

```
def get_all_products(db: Session, page=None, per_page=None):
    if page is None:
        all_products = db.query(models.Product).all()
    else:
        all_products = db.query(models.Product).offset((page - 1) * per_page).limit(per_page).all()

    result = []
    for product in all_products:
        prod_photo = db.query(models.ProductPhoto).filter(models.ProductPhoto.product_id == product.id).first()
        result.append({product : prod_photo})

    return result
```

Рисунок 3.2 – Приклад коду для отримання даних про товар

Коли користувач додає товар до кошика, на бекенд передаються ідентифікатор товару та його кількість. Ці дані зберігаються в таблиці кошика. Користувач може змінювати кількість одиниць товару або видаляти його з кошика. Для цього на бекенді реалізовано відповідні функції.

Наступним із базових функціональних модулів розробленого вебдодатку маркетплейсу персональних засобів безпеки є система управління товарами. Вона забезпечує можливість створення, редагування, перегляду та видалення товарних позицій, причому ці дії можуть виконувати лише користувачі з відповідними правами доступу. Цей функціонал реалізовано на бекенді за допомогою фреймворку FastAPI, з інтеграцією з базою даних через ORM-бібліотеку SQLAlchemy.

Функція створення товарів передбачена виключно для користувачів, які мають роль продавця. Для реалізації цього механізму створено окремий API-ендпоінт, який приймає основні характеристики товару: назву, опис, ціну, категорію, кількість у наявності, зображення (у вигляді URL). Після проходження валідації введених даних інформація зберігається у базі даних із прив’язкою до конкретного продавця. Такий підхід гарантує, що кожен продавець має змогу керувати лише власним товарним асортиментом.

Окрім звичайних продавців, можливість створювати товари має й адміністратор платформи. Це корисно, коли потрібно додати продукцію від імені самої системи – наприклад, для співпраці з офіційними постачальниками або щоб показати приклади товарів у тестовому режимі. Адміністратор додає товари

так само, як і продавець, але без прив'язки до конкретного акаунта. Така гнучкість дає змогу швидко реагувати на потреби ринку, самостійно наповнювати каталог і створювати нові пропозиції, які бачать усі користувачі маркетплейсу.

Механізм редагування та видалення товарів також обмежено правами доступу. Продавець має можливість у будь-який момент оновити інформацію про свої товари або видалити їх. При цьому система перевіряє, чи належить товар поточному користувачу. Якщо це підтверджується, то через відповідні запити (PUT/PATCH для редагування та DELETE для видалення) вносяться зміни до бази даних. Оновлена інформація одразу стає доступною у загальному каталозі. Функціонал перегляду товарів є відкритим для всіх користувачів, незалежно від їхньої ролі. Він дозволяє отримати повний список доступних товарів, а також переглянути детальну інформацію про кожну позицію, зокрема опис, зображення, ціну, рейтинг та кількість у наявності. За необхідності передбачено реалізацію пагінації, яка дозволяє розбити великий список товарів на зручні для перегляду сторінки.

З метою покращення зручності користувачів реалізовано механізм фільтрації та сортування товарів. Зокрема, реалізована можливість фільтрації за категорією, ціновим діапазоном. Це дозволяє кожному користувачу швидко знайти потрібний товар і ефективно взаємодіяти з платформою.

```
def filter_products(categories: list, min_price: float, max_price: float, db: Session):
    all_products = db.query(models.Product).filter(models.Product.price > min_price,
                                                   models.Product.price < max_price).all()

    result_products = [ product for product in all_products
                                if product.category.id in categories ]
    result_data = []
    for product in result_products:
        prod_photo = (db.query(models.ProductPhoto)
                      .filter(models.ProductPhoto.product_id == product.id).first())
        result_data.append({product : prod_photo})

    return result_data
```

Рисунок 3.3 – Приклад коду для отримання відфільтрованих товарів

Система управління товарами є одним із ключових компонентів маркетплейсу, який забезпечує гнучке та динамічне наповнення торговельного

каталогу. Завдяки використанню FastAPI та SQLAlchemy вдалося досягти високої продуктивності, масштабованості та розширюваності бекенд-частини застосунку, що є необхідною умовою для стабільної роботи системи в умовах зростання кількості користувачів і товарів.

Також однією з ключових частин будь-якого маркетплейсу є система замовлень. Вона забезпечує повний цикл взаємодії між покупцем, продавцем і платформою. Реалізована система на бекенді за допомогою FastAPI та взаємодіє з базою даних для збереження інформації про замовлення, їх статуси та користувачів.

Покупець може додавати товари до кошика, і цей кошик зберігається в базі даних у таблиці order_items. Там фіксуються ідентифікатори товарів та користувачів. У кошику покупець може побачити загальну суму замовлення та змінювати кількість товарів або видаляти їх, поки не оформить покупку.

```
def get_item_cart(db: Session, user_id):
    cart = (db.query(models.Cart).filter(models.Cart.user_id == user_id)
            .order_by(models.Cart.id).all())

    item_in_cart = [
                    { 'product_id' : i.product_id,
                      'maindata' : (db.query(models.Product)
                                    .filter(models.Product.id == i.product_id).first()),
                      'quantity' : i.quantity}
                    for i in cart
                    ]

    return item_in_cart
```

Рисунок 3.4 – Приклад коду для отримання відфільтрованих товарів

Кожен товар, доданий до кошика, пов'язаний з конкретним користувачем і товаром через їх ідентифікатори в таблиці. Користувач може додавати будь-яку кількість товарів або видаляти їх. Всі зміни в кошику оновлюються на фронтенді, що дає можливість зручно коригувати замовлення перед оформленням.

Якщо покупець задоволений вибором, він переходить до оформлення замовлення. Система зберігає інформацію про замовлені товари в таблиці orders, де вказуються:

- ідентифікатор користувача;

- ідентифікатори товарів для доставки;
- загальна сума замовлення на момент оформлення.

Після оформлення кожне замовлення отримує статус «очікує обробки». Далі статус може змінюватися в залежності від етапу виконання:

- «очікує» – замовлення чекає на обробку;
- «виконано» – замовлення підтверджене, товар відправлений або доставлений;
- «відмінено» – замовлення скасоване покупцем або адміністратором.

Зміна статусу здійснюється через API-ендпоінти, які дають продавцям та адміністраторам можливість контролювати виконання замовлення. Наприклад, після перевірки товару на складі продавець може оновити статус на «виконано».

Зареєстровані користувачі можуть переглядати історію своїх замовлень. Для цього є окремий ендпоінт, який повертає список усіх замовлень з такими даними:

- номер замовлення;
- загальна сума;
- поточний статус.

Це дає можливість покупцям відслідковувати свої покупки.

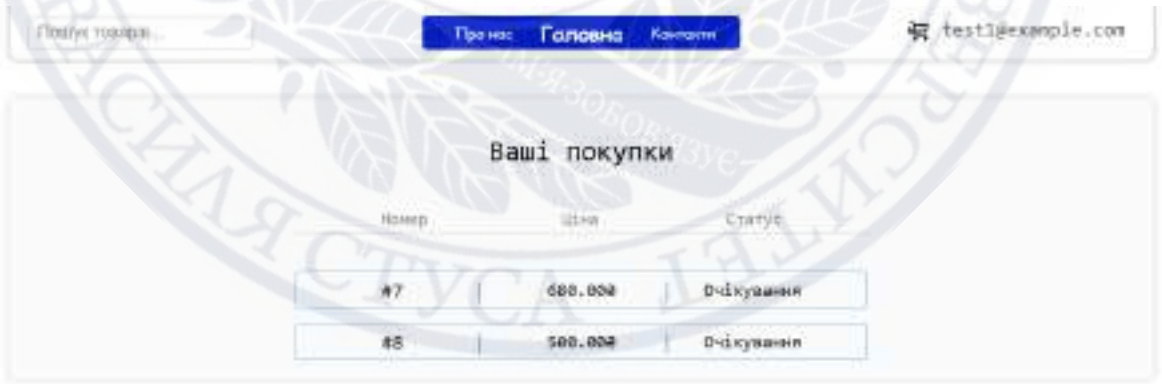

Рисунок 3.5 – Вигляд сторінки історії замовлень користувача

Система замовлень є основою функціонування маркетплейсу, адже вона забезпечує простий і зручний процес покупки. Покупці можуть легко змінювати кількість товарів у кошику, бачити кінцеву суму та оформляти замовлення. Продавці та адміністратори контролюють процес виконання замовлень за допомогою системи статусів. Це створює гнучкість у процесах покупки, доставки та відстеження товарів, роблячи платформу зручною та ефективною для всіх учасників.

Для реалізації інтеграції з базою даних маркетплейсу було використано SQLAlchemy – потужний ORM для Python. Це дозволило зручно працювати з реляційними базами даних, виконуючи операції над таблицями через об'єкти Python, що робить роботу з даними гнучкою і масштабованою.

У проекті визначено основні моделі, які відображають ключові елементи маркетплейсу. Кожна модель відповідає певній сутності, і для кожної з них SQLAlchemy створює відповідні таблиці в базі даних. Також між цими моделями налаштовуються зв’язки, які дозволяють обробляти складні відносини між даними.

У кожного користувача є базова інформація: ім’я, електронна пошта, хешований пароль та роль (покупець, продавець або адміністратор). Користувач може мати багато замовлень, відгуків. У цій моделі визначається зв'язок один-до-багатьох між користувачем та замовленнями/відгуками.

```python
class User(Base):
    __tablename__ = "users"

    id = Column(Integer, primary_key=True, index=True)
    username = Column(String, unique=True, index=True, nullable=False)
    email = Column(String, unique=True, index=True, nullable=False)
    hashed_password = Column(String, nullable=False)
    created_at = Column(TIMESTAMP, server_default=func.now())

    orders = relationship('Order', back_populates='user')
    reviews = relationship("Review", back_populates="user")
    carts = relationship("Cart", back_populates="user")
```

Рисунок 3.6 – Приклад коду для створення таблиці «users»

Модель для товарів включає такі дані як назва, опис, ціна, наявність, категорія тощо. Товари можуть мати багато відгуків, а також бути частиною кількох замовлень, що реалізується через зв'язок багато-до-багатьох з таблицею замовлень.

Замовлення містить ідентифікатор користувача, що його створив, статус замовлення (наприклад, "очікує" або "виконано") і список товарів, які були придбані. Замовлення має зв'язок один-до-багатьох з користувачами, а також зв'язок багато-до-багатьох з товарами через проміжну таблицю, яка зберігає зв'язок між замовленнями та товарами.

Відгуки містять інформацію про оцінку товару, текст відгуку і дату його створення. Кожен відгук належить певному товару та користувачу, тому тут встановлено зв'язок багато-до-одного з товарами та користувачами.

Типи зв’язків між таблицями:

Один-до-багатьох (One-to-Many)

- Користувач – Замовлення: один користувач може створити кілька замовлень, але кожне замовлення належить лише одному користувачеві.
- Користувач – Відгуки: один користувач може залишити кілька відгуків, але кожен відгук належить одному користувачу.
- Товар – Відгуки: один товар може мати багато відгуків від різних користувачів.

Багато-до-багатьох (Many-to-Many)

- Замовлення – Товар: одне замовлення може включати кілька товарів, і кожен товар може бути частиною кількох замовлень. Це реалізується через проміжну таблицю, наприклад, order_items, яка зберігає зв’язок між замовленнями і товарами.

Завдяки SQLAlchemy можна виконувати стандартні CRUD (Create, Read, Update, Delete) операції з моделями даних. Кожна сутність – користувач, товар, замовлення, відгук – має відповідні операції для додавання, читання, оновлення або видалення записів.

- Create – додавання нових записів в базу даних;

- Read – отримання даних з бази;
- Update – оновлення наявних записів;
- Delete – видалення записів з бази.

Для реалізації всіх цих операцій був створений окремий файл crud.py, у якому зосереджені функції для роботи з базою даних. Такий підхід дозволяє розділити логіку доступу до даних від основного коду застосунку, зробити проєкт більш структурованим і зручним для підтримки.

```
def get_all_products(db: Session, page=None, per_page=None):···

def filter_products(categories: list, min_price: float, max_price: float, db: Session):···

def get_all_category(db: Session):···

def get_detail_product(db: Session, product_id):···

def search_items(db: Session, query: str):···

def add_product_to_cart(db: Session, product: schemas.CartCreate, user_id):···

def add_review(db: Session, data: dict):···
```

Рисунок 3.7 – Приклад функцій для роботи маркетплейсу в файлі crud.py

Ця структура дозволяє ефективно організувати взаємодію з базою даних у маркетплейсі, гарантуючи, що дані товарів, користувачів, замовлень, відгуків будуть зберігатися коректно та зручно для доступу. Це основа для побудови масштабованого та надійного бекенду для будь-якої торгової платформи.

Усі основні функціональні можливості вебдодатку маркетплейсу реалізовані через HTTP-запити. Кожен аспект системи – від реєстрації користувача до створення товару, оформлення замовлення або управління відгуками – обробляється окремими кінцевими точками RESTful API, що працюють за допомогою FastAPI.

FastAPI – це потужний фреймворк для створення вебдодатків на Python, який дозволяє ефективно розробляти REST API [25]. Одна з головних переваг FastAPI – це автоматичне генерування документації через Swagger для всіх доступних кінцевих точок. Така документація дозволяє розробникам і

тестувальникам легко переглядати всі доступні методи, їхні параметри та типи запитів і відповідей. І найкраще те, що документація генерується автоматично, що значно економить час на її написання та оновлення.

Swagger-документація надає:

- повний перелік усіх доступних кінцевих точок API, включаючи методи (GET, POST, PUT, DELETE), URL та детальні описи ;
- приклади запитів і відповідей для кожної кінцевої точки;
- можливість тестувати API безпосередньо через інтерфейс документації, відправляючи запити і отримуючи відповіді.

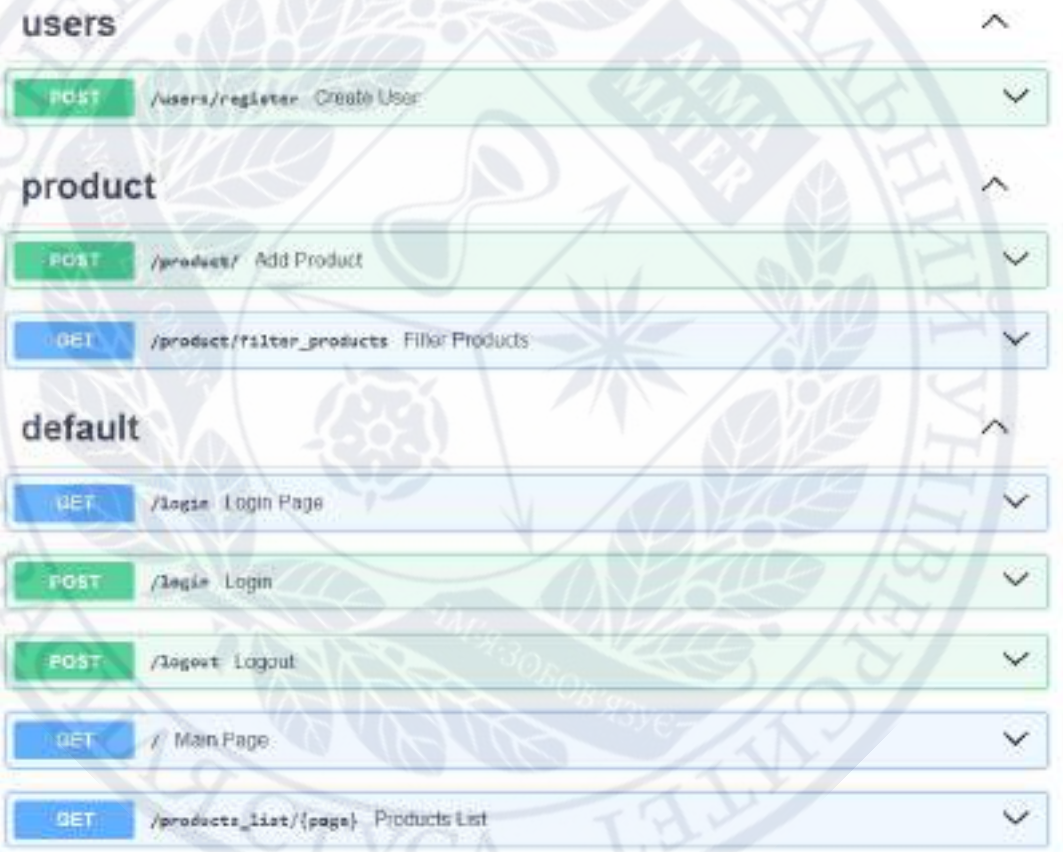

Рисунок 3.8 – Вебінтерфейс автоматично згенерованої документації FastAPI

Це не лише дозволяє інтегрувати API в інші частини системи, а й спрощує тестування. Можливість тестувати API прямо з інтерфейсу документації робить процес взаємодії з API ще більш зручним.

Повний лістинг програми знаходиться в Додатку А.

У підсумку, розробка бекенду для маркетплейсу гарантує стабільну роботу всіх необхідних функціональних можливостей, простоту інтеграції з базою даних і легкість у тестуванні API. Усі основні процеси – від управління товарами до обробки замовлень – реалізовані з урахуванням безпеки, зручності та ефективності для користувачів.

3.2 Розробка фронтенду: інтерфейс та механізми взаємодії користувачів

Розробка фронтенду маркетплейсу персональних засобів безпеки зосереджена на створенні простого та зрозумілого інтерфейсу для користувачів. Вебдодаток розроблений за сучасними технологіями, що забезпечують зручність, адаптивність та безпеку.

Основні частини інтерфейсу:

1. Головна сторінка: це перша сторінка, яку бачить користувач. Вона включає каталог товарів, фільтри, кнопку для входу. Після авторизації з’являються кнопки для доступу до особистого кабінету та кошика.
2. Каталог товарів: тут користувачі можуть переглядати засоби безпеки. Є фільтри для сортування товарів за категоріями, ціною. Кожен товар має картку з фото, описом та ціною.
3. Пошуковий блок: допомагає швидко знайти потрібний товар за допомогою пошукового рядка.
4. Кошик: іконка кошика завжди в правому верхньому куті. Користувач може побачити додані товари, їх кількість та загальну суму. Є можливість редагувати кількість товарів або видалити їх.
5. Профіль користувача: авторизовані користувачі можуть переглядати свої дані, кошик, а також історію замовлень.
6. Сторінка товару: при натисканні на товар відкривається детальна інформація: характеристики, опис, відгуки та рейтинг. Користувач може додати товар до кошика.

7. Форма реєстрації та входу: Для нових користувачів є форма реєстрації, а для зареєстрованих – форма входу.
8. Панель навігації: Це меню, яке дає доступ до основних розділів: каталог, кошик, профіль.

Рисунок 3.9 – Головна сторінка маркетплейсу

Повний огляд вебдодатку знаходиться у Додатку Б.

Фронтенд маркетплейсу зручний і зрозумілий. Розробка гарантує безпечну взаємодію з системою, що важливо для онлайн-шопінгу.

Інтерфейс зроблений на HTML, CSS і JavaScript. Сторінки мають логічну структуру, а елементи реагують на дії користувача – наприклад, натискання кнопок, введення тексту чи перехід між сторінками.

Оформити замовлення можна через просту покрокову форму, яка відкривається з кошика.

Кожен етап, наприклад, введення адреси має свій розділ із відповідними полями: текстові поля, списки. Усе оформлено з використанням тегів <form>, <input>, <select> тощо. Коли натискаєш кнопку "Оформити замовлення", JavaScript перевіряє, чи все заповнено правильно. Потім з’являється повідомлення про успішне замовлення.

```
<form id="placeAnOrder">
    <input type="name" id="name" name="name" placeholder="Прізвище Ім'я По батькові" required>
    <input type="phone_number" id="phone_number" name="phone_number" placeholder="Номер телефон
    <input type="city" id="city" name="city" placeholder="Місто" required>
    <input type="address_post_office" id="address_post_office" name="address_post_office" place
    <input type="number_post_office" id="number_post_office" name="number_post_office" placehol
    <button type="submit">Замовити</button>
</form>
```

Рисунок 3.10 – Приклад HTML коду для оформлення замовлення

Фільтри – це чекбокси представлені як <input type="checkbox">, випадаючі списки або повзунки. Наприклад, можна обрати категорію або діапазон цін. Коли користувач щось вибирає, JavaScript автоматично надсилає запит і перезавантажує сторінку з оновленим списком товарі. Товари показані у вигляді карток у сітці, реалізовані як блок із CSS-класом.

```
<form id="filterForm" style="display: grid; width: 220px;">
    <fieldset>
        <legend>Категорії:</legend>
        {% for category in all_category %}
            <label><input type="checkbox" name="category" value="{{ category.id }}"> {{ category.name }}</label><br>
        {% endfor %}
    </fieldset>

    <div class="container-filter-price">
        <label class="label-filter-price" for="min_price">Мін. ціна:</label>
        <input type="number" id="min_price" name="min_price" min="0" max="10000">

        <label class="label-filter-price" for="max_price">Макс. ціна:</label>
        <input type="number" id="max_price" name="max_price" min="0" max="10000">
    </div>
    <button class="filter-button" type="submit">Застосувати</button>
</form>
```

Рисунок 3.11 – Приклад HTML коду для фільтрації товарів

На кожній картці товару є кнопка "Додати до кошика" представлена тегом <button type="submit">. Натискаючи – товар додається. У верхньому куті є іконка кошика – вона одразу показує нову кількість товарів.

Після входу користувач може зайти в профіль – посилання на нього є вгорі сторінки. У профілі є розділи: особиста інформація, історія замовлень, кошик. Інформація редагується через просту форму з кнопкою "Зберегти". Історія замовлень – це таблиця з номерами та статусами.

Також на сайті наявні інші елементи інтерфейсу, як-от:

- підказки з’являються при наведенні (наприклад, в списках замовлень з’являється відповідний значок, якшо не була надана особиста інфрормація для доставки);

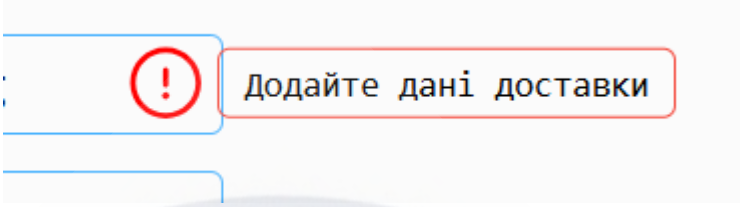

Рисунок 3.12 – Приклад підказки для користувачів

- меню навігації адаптивне – на мобільних телефонах воно стає "бургером".

Завдяки HTML, CSS і JavaScript користувачі можуть легко фільтрувати товари, додавати їх до кошика, оформлювати замовлення та керувати своїм профілем.

JavaScript використовується для забезпечення інтерактивності інтерфейсу на клієнтській стороні. Основна логіка взаємодії реалізована через базові функції, що обробляють події користувача – зокрема, натискання на кнопки, введення даних у форми та взаємодію з елементами інтерфейсу.

Одним із ключових підходів є використання функції addEventListener, яка дозволяє «слухати» події на конкретних елементах, таких як кнопки додавання до кошика, оформлення замовлення, застосування фільтрів тощо. Наприклад, для кнопки додавання товару до кошика, після кліку відбувається виклик функції, яка надсилає асинхронний запит на сервер.

```
document.getElementById('filterForm').addEventListener('submit', async function(event) {
    event.preventDefault();

    const formData = new FormData();

    let selectedCategories = Array.from(document.querySelectorAll('input[name="category"]:checked'))
    .map(input => input.value);

    selectedCategories.forEach(category => {
        formData.append('categories', category);
    });

    if (document.getElementById('min_price').value === '') {
        formData.append('min_price', 0.0);
    } else {
        formData.append('min_price', document.getElementById('min_price').value);
    };

    if (document.getElementById('max_price').value === '') {
        formData.append('max_price', 10000.0);
    } else {
        formData.append('max_price', document.getElementById('max_price').value);
    };
```

Рисунок 3.13 – Приклад HTML коду для фільтрації товарів

Передача даних на бекенд здійснюється через метод fetch, що дозволяє виконувати HTTP-запити без перезавантаження сторінки. Таким чином, обробка дій користувача відбувається динамічно, що значно покращує загальний користувацький досвід.

Використання fetch особливо актуальне для таких операцій:

- додавання або видалення товарів із кошика;
- надсилання форми реєстрації чи авторизації;
- фільтрація товарів за параметрами.

Всі відповіді сервера обробляються на клієнтській стороні, і за необхідності виводяться відповідні повідомлення для користувача.

3.3 Варіанти покращення та перспективи розвитку вебдодатку

Розроблений вебдодаток виконує основні функції маркетплейсу персональних засобів безпеки, однак існують численні можливості для його подальшого вдосконалення з урахуванням потреб користувачів, зростання аудиторії та розширення функціоналу. У цьому розділі наведено основні напрями розвитку системи:

1. Один із важливих кроків у розвитку вебдодатку – підключення до API логістичної компанії, зокрема «Нової Пошти». Це найпопулярніша служба доставки в Україні, тож її інтеграція зробить маркетплейс зручнішим і функціональнішим.

На етапі, на якому перебуває маркетплейс зараз, користувачі самостійно вводять адресу доставки. Це займає час, може призводити до помилок і не дає можливості відстежувати посилку. Завдяки API можна значно полегшити цей процес. Наприклад:

- показувати список доступних відділень одразу після вибору міста;
- реалізувати пошук за адресою;
- дати можливість обрати тип доставки – на відділення, кур’єром чи у поштомат;

На фронтенді це реалізується через зручні форми та випадаючі списки. Вони оновлюються динамічно за допомогою JavaScript та fetch. Можна навіть додати інтерактивну мапу, щоб обирати відділення візуально.

Сьогодні інтеграція з логістичними сервісами – це вже стандарт для інтернет-магазинів. Це дозволяє зробити процес замовлення простим, швидким і зручним – незалежно від того, де знаходиться користувач.

2. Сучасні вебдодатки мають добре працювати не лише на комп’ютерах, а й на смартфонах. Багато користувачів роблять замовлення саме з телефону, тому адаптивність дуже важлива. У маркетплейсі вже є базова підтримка адаптивного дизайну. Але далі планується ще більше вдосконалити мобільну версію.

Рисунок 3.14 – Приклад вигляду вебдодатку на телефонах

З технічного боку, верстка базується на гнучких сітках (flex, grid) і відсоткових розмірах. Це дає змогу елементам підлаштовуватись під екран користувача. Повна адаптація – це не лише про зручність. Вона покращує враження від користування сайтом, підвищує довіру до сервісу й навіть позитивно впливає на пошукову оптимізацію.

3. Фільтри – це важлива частина будь-якого маркетплейсу. Вони допомагають користувачам швидко знаходити потрібні товари, особливо коли

вибір великий. У вебзастосунку вже є базові фільтри за категорією і ціною. Але щоб зробити пошук ще зручнішим, варто додати нові можливості.

Серед корисних фільтрів:

- виробник – дозволяє обрати товари конкретного бренду;
- сертифікація – показуватимуться лише товари з підтвердженням якості;
- особливості – наприклад, матеріал, рівень захисту, одноразовість тощо.

Загалом, розширені фільтри допоможуть покупцеві зекономити час і зробити процес пошуку набагато зручнішим.

4. Оплата товарів прямо на сайті – це стандарт для будь-якого сучасного онлайн-магазину. Це зручно, швидко і викликає довіру. На даний момент після оформлення замовлення користувач просто чекає відповіді продавця. Але інтеграція платіжних систем дозволить одразу оплатити покупку.

Користувач зможе обрати зручний варіант: банківську картку, Apple Pay, Google Pay чи інші популярні сервіси.

Для цього потрібно:

- вибрати платіжного провайдера (наприклад, LiqPay, Fondy, WayForPay);
- налаштувати захищене з'єднання (HTTPS);
- створити зручну форму для введення платіжних даних або перенаправляти користувача на сторінку оплати;
- обробити результат транзакції (успішна/невдала оплата);
- оновити статус замовлення в базі даних.

У підсумку, онлайн-оплата – це можливість зробити процес покупки повністю автоматизованим, а також пришвидшити доставку і підвищити довіру до сайту.

5. Щоб платформою могли користуватися не лише українці, варто додати підтримку англійської мови. Це зробить сайт зрозумілим для ширшої аудиторії, включаючи іноземців або тих, хто не розуміє українську. Багатомовність – це простий, але дуже важливий крок для зростання платформи.

6. Ще одна корисна ідея – створити зручну панель для адміністратора. У вигляді дашборду можна показувати важливу статистику: скільки було продажів,

які товари найпопулярніші, як поводяться користувачі на сайті. Такі дані допомагають краще розуміти ситуацію й ухвалювати правильні рішення для розвитку платформи.

Отже, усі запропоновані покращення зроблять платформу зручнішою, функціональнішою та більш привабливою для користувачів. Якщо реалізовувати їх поступово, вебдодаток з часом перетвориться на повноцінний і сучасний маркетплейс, яким легко та приємно користуватись.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи вдалося досягти поставленої мети – створено вебдодаток маркетплейсу індивідуальних засобів захисту, який забезпечує користувачам зручний, зрозумілий та доступний процес пошуку й купівлі необхідних товарів.

У процесі дослідження було систематизовано актуальну інформацію про ринок персональних засобів безпеки, а також проаналізовано поведінку споживачів в умовах онлайн-торгівлі. Це дало змогу чітко окреслити вимоги до функціоналу майбутнього вебзастосунку, з урахуванням потреб та очікувань користувачів.

Аналіз існуючих рішень на ринку показав, що спеціалізовані онлайн-магазини, орієнтовані саме на продаж індивідуальних засобів захисту, практично відсутні. Більшість доступних платформ є загальними маркетплейсами, які не повною мірою задовольняють потреби користувачів у швидкому пошуку, порівнянні характеристик та оцінюванні саме таких товарів. Це стало аргументом на користь розробки нового маркетплейсу з акцентом на зручність використання та якісну взаємодію з інтерфейсом.

У ході роботи було досліджено сучасні технології створення вебдодатків, що дало змогу обґрунтовано обрати стек Python, Flask та PostgreSQL як надійну та ефективну основу для реалізації проєкту.

Розроблено ключові компоненти вебзастосунку: головну сторінку, сторінку товарів, функціонал кошика, систему оформлення замовлення, особистий кабінет користувача та інші елементи, які забезпечують базову, але повноцінну функціональність маркетплейсу.

Здійснено ручне тестування, результати якого підтвердили працездатність реалізованих функцій на базовому рівні. Виявлені недоліки й можливості покращення лягли в основу подальших рекомендацій щодо розвитку проєкту.

Серед основних напрямів удосконалення вебзастосунку можна виокремити кілька ключових аспектів. Насамперед, доцільним є впровадження

інтеграції з API логістичних служб, що дозволить автоматизувати процес оформлення та відстеження доставки замовлень. Важливим кроком також є покращення адаптивності інтерфейсу, аби забезпечити комфортне користування вебдодатком на всіх мобільних пристроях. Крім того, доцільно розширити систему фільтрів, що значно полегшить користувачам пошук і порівняння товарів. Ще одним перспективним напрямом є реалізація функціоналу онлайн-оплати безпосередньо через сайт, що зробить процес замовлення ще зручнішим та швидшим.

Таким чином, створений вебзастосунок має прикладне значення: він може стати основою для подальшого комерційного розвитку або масштабування, з урахуванням запитів ринку та сучасних тенденцій у сфері електронної комерції.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Граф М. С., Кузьменко О. В. Архітектура, проєктування та безпека веб-орієнтованих інформаційних та комп'ютерних систем : навч. посіб. 179 с.
2. Дражниця С. А., Забурмеха Є. М. Електронна комерція: світові тренди та прогноз розвитку в Україні. Вісник Хмельницького національного університету. Серія: Економічні науки. 2018. № 6. С. 69-73
3. Завадський І.О. Основи баз даних: навч. посіб. Київ, 2011. 192 с.
4. Зелінська О. В., Січко Т. В., Потапова Н. А., Якубич К. О. Методичні рекомендації до виконання, оформлення та захисту кваліфікаційної (бакалаврської) роботи. Вінниця: ДонНУ імені Василя Стуса, 2024. 27 с.
5. Іваненко Л.М. Маркетплейси як об’єктивний наслідок розвитку електронної комерції. Економіка і організація управління. 2021. №4(44). – С. 178-187.
6. Краус К.М., Краус Н.М., Манжура О.В. Електронна комерція та інтернет-торгівля: навчально-методичний посібник. Київ: Аграр Медіа Груп, 2021. 456 с.
7. Лебеденко С. О. Навчально-методичний комплекс дисципліни «Електронна комерція». Київ: КПІ ім. Ігоря Сікорського, 2021. 73 с.
8. Манако В., Манако Д., Данилова О., Войченко О. Основи будування сайтів. 2006. 120 с.
9. Ніколаєв І. В., Загреба М. М., Вишневська В. А. Інформаційні послуги електронних торговельних майданчиків у маркетинговій діяльності. Центральноукраїнський науковий вісник. Економічні науки. 2022. №8(41). С. 56-68.
10. Павлов Д.Л., Січко Т.В. Принцип роботи Web API та його застосування. Збірник тез доповідей Всеукраїнської науково-практичної конференції «Прикладні інформаційні технології». Вінниця: ДонНУ імені Василя Стуса, 2023. С. 166-168.

11. Петрик М., Петрик О. Моделювання програмного забезпечення: науково-методичний посібник. Тернопіль: ТНТУ, 2016. 94 с.
12. Просович О. П., Боцман Ю. С. Маркетплейс як дієвий інструмент цифрового маркетингу. Вісник Національного університету «Львівська політехніка». 2018. № 914. С. 32–38.
13. Сидоренко В.В., Константинова Л.В., Смірнов С.А. Організація баз даних: навч. посіб. Кропивницький: ЦНТУ, 2018. 274 с.
14. Ткачук Н. О., Січко Т.В. Застосування big data у бізнесі. Комп'ютерні технології обробки даних: матеріали всеукр. наук.-практ. конф. Вінниця, 2020. С. 103-106.
15. Anders M. Python 3 Web Development. Birmingham: Packt Publishing, 2011. 321 с.
16. Brambilla M., Ceri S., Fraternali P., Manolescu I. Process Modeling in Web Applications. 2006.
17. Domingues A. L. S., Bianchini S., Costa M. L. S., Ferrari F., Maldonado J. C. Web Application Development Methods: A Comparison. 2007.
18. Juba S., Vannahme A., Volkov A. Learning PostgreSQL. 2015. 356 с.
19. Madeyski L., Stochmialek M. Architectural Design of Modern Web Applications. 2005.
20. Martin P. Чиста архітектура: мистецтво розробки програмного забезпечення. 2019. 416 с.
21. Obe R., Hsu L. PostgreSQL: Up and Running: A Practical Guide to the Advanced Open Source Database. 2017. 281 с.
22. Svekis L., Putten M. JavaScript from Beginner to Professional. 2021. 140 с.
23. An Overview of Electronic Commerce. Jain V., Malviya B., Arya S. 2021. URL: https://www.researchgate.net/publication/351775073_An_Overview_of_Electronic_Commerce_e-Commerce

24. Common web application architectures. URL: https://learn.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/common-web-application-architectures
25. Documentation FastAPI. URL: https://fastapi.tiangolo.com/
26. Documentation Jinja2. URL: https://jinja.palletsprojects.com/en/stable/
27. Electronic Commerce: Theory and Practice. Isoraite M., Miniotiene N. – 2018. URL: https://www.researchgate.net/publication/329704574_Electronic_Commerce_Theory_and_Practice
28. Flavian C., Gurrea R., Orus C. Web design: A key factor for the website success. 2009. URL: https://www.researchgate.net/publication/221679095_Architectural_Design_of_Modern_Web_Applications
29. How Web Works - Web Application Architecture for Beginners. URL: https://www.geeksforgeeks.org/how-web-works-web-application-architecture-for-beginners/
30. Monolithic vs. Microservices Architecture. URL: https://www.geeksforgeeks.org/monolithic-vs-microservices-architecture/
31. Python 3.13.3 documentation. URL: https://docs.python.org/3/
32. Python Frameworks vs. Python Libraries. URL: https://fullscale.io/blog/python-frameworks-vs-python-libraries/
33. The Role of Open Web Standards for Website Development Adhering to the One Web Vision. URL: https://www.researchgate.net/publication/267324997_The_Role_of_Open_Web_Standards_for_Website_Development_Adhering_to_the_One_Web_Vision
34. The top 4 Python backend frameworks for building entry level AI projects. URL: https://pieces.app/blog/the-top-4-python-back-end-frameworks-for-your-next-project
35. Top 12 Best Databases for Web Application Development in 2024. URL: https://positiwise.com/blog/best-databases-for-web-application-development

36. Top 8 Tech Stacks: Choosing the Right Tech Stack. URL: https://fullscale.io/blog/top-5-tech-stacks/
37. Understanding Web Application Architectures: A Comprehensive Overview of Infrastructure Models and Components. URL: http://medium.com/@gwenilorac/layout-of-web-applications-795b3e8e4c1b
38. Using FastAPI to Build Python Web APIs. URL: https://realpython.com/fastapi-python-web-apis/
39. Web Standards Model. URL: https://developer.mozilla.org/en-US/docs/Learn_web_development/Getting_started/Web_standards/The_web_standards_model
40. Which Is the Best Python Web Framework: Django, Flask, or FastAPI? URL: https://blog.jetbrains.com/pycharm/2025/02/django-flask-fastapi/

ДОДАТКИ

ДОДАТОК А

Лістинг програми

```
@app.get('/')
def main_page(request: Request, db: Session = Depends(get_db), user: dict
= Depends(get_current_user)):
    all_products = crud.get_all_products(db)
    all_category = crud.get_all_category(db)
    return templates.TemplateResponse('main_page.html', {'request' :
request,
                                                         'all_products' :
all_products,
                                                         'current_page' :
1,
                                                         'all_category' :
all_category,
                                                         'user' : user})
```

Ця функція відповідає за рендеринг головної сторінки застосунку. Вона отримує всі товари та категорії з бази даних, а також передає поточного користувача для персоналізації інтерфейсу.

```
@app.get("/products_list/{page}", response_class=HTMLResponse)
def products_list(
    request: Request,
    page: int,
    db: Session = Depends(get_db),
    user: dict = Depends(get_current_user),
    categories: list[int] = Query([], alias="categories"),
    min_price: Optional[float] = Query(None),
    max_price: Optional[float] = Query(None)
):
    all_category = crud.get_all_category(db)

    if categories or max_price or min_price:
        all_products_for_page = products.filter_products(categories,
min_price, max_price, db)

        return templates.TemplateResponse("products_list.html", {
            "request": request, "all_products": all_products_for_page,
"current_page": None,
            'user': user, 'all_category': all_category, "total_pages":
1})

    else:
        per_page = 8
        total_products = db.query(models.Product).count()
        all_products_for_page = crud.get_all_products(db, page, per_page)
```

```
        return templates.TemplateResponse("products_list.html", {
            "request": request,
            "all_products": all_products_for_page,
            "current_page": page,
            'user': user,
            'all_category': all_category,
            "total_pages": (total_products // per_page) + (1 if
total_products % per_page else 0),
        })
```

Функція відповідає за вивід списку товарів з пагінацією. Також вона дозволяє фільтрувати товари за категоріями, мінімальною та максимальною ціною. Якщо фільтри активні – повертається відфільтрований список, інакше — товари за сторінками.

```
@app.get("/search", response_class=HTMLResponse)
def search_products(request: Request, query: Union[str, int] = Query(...,
min_length=1, description="Пошуковий запит"),
                    user: dict = Depends(get_current_user), db: Session =
Depends(get_db)):

    result_items = crud.search_items(db, str(query))
    all_category = crud.get_all_category(db)

    return templates.TemplateResponse("products_list.html", {
            "request": request,
            "all_products": result_items,
            "current_page": None,
            'user': user,
            'all_category': all_category,
            "total_pages": 1,
    })
```

Реалізує пошук товарів у базі даних за ключовим словом або числовим значенням. Результати виводяться на шаблон зі списком товарів. Пошук доступний лише авторизованим користувачам.

```
@app.get("/login", response_class=HTMLResponse)
def login_page(request: Request, user: dict = Depends(get_current_user)):
    return templates.TemplateResponse("login.html", {"request": request,
'user' : user})
```

Повертає HTML-сторінку для входу в обліковий запис. Якщо користувач уже авторизований, функція все одно дозволяє переглянути сторінку, що може бути корисно для повторного входу.

```
@app.get('/profile')
def profile(request: Request, db: Session = Depends(get_db), user: dict =
Depends(get_current_user)):
    if user is None or user['id'] == None:
        return RedirectResponse(url="/login")
    user_info = crud.get_user_by_email(db, user['username'])
    return templates.TemplateResponse("profile.html", {"request":
request, 'user' : user_info})
```

Виводить сторінку профілю поточного користувача. Якщо користувач не авторизований — його перенаправляють на сторінку входу. Дані профілю отримуються з бази за email-адресою.

```
@app.get('/user_purchase')
def purchase(request: Request, db: Session = Depends(get_db), user: dict
= Depends(get_current_user)):
    if user is None or user['id'] == None:
        return RedirectResponse(url="/login")

    user_orders = crud.user_purchase(db, user['id'])
    print(user_orders)
    return templates.TemplateResponse('purchase.html', {'request' :
request,
                                                         'user' : user,
                                                         'orders' :
user_orders})
```

Показує історію замовлень користувача. Якщо користувач не авторизований — відбувається перенаправлення на сторінку входу. Дані замовлень витягуються з бази через відповідну CRUD-функцію.

```
@app.get('/detail_product/{product_id}')
def detail_product(request: Request, product_id: int, db: Session =
Depends(get_db), user: dict = Depends(get_current_user)):
    detail, prod_photo, prod_reviews = crud.get_detail_product(db,
product_id)
    print(user)
    return templates.TemplateResponse('detail_product.html', {'request' :
request,
```

```
                                                    'user' :
user,
                                                    'detail' :
detail,
                                                    'prod_photo
' : prod_photo,
                                                    'prod_revie
ws' : prod_reviews})
```

Виводить детальну інформацію про конкретний товар за його ID. Також завантажуються пов’язані фотографії та відгуки. Доступна як для авторизованих, так і неавторизованих користувачів.

```
@app.get('/register')
def register(request: Request, user: dict = Depends(get_current_user)):
    return templates.TemplateResponse('register.html', {'request' :
request, 'user' : user,})
```

Відображає сторінку реєстрації нового користувача. Функція також передає поточного користувача, якщо такий існує (наприклад, для перевірки або повідомлення).

```
@app.get('/user_cart')
def user_cart(request: Request, db: Session = Depends(get_db), user: dict
= Depends(get_current_user)):
    if user is None or user['id'] == None:
        return RedirectResponse(url="/login")

    cart = crud.get_item_cart(db, user['id'])
    total_price = sum(i["maindata"].price * i["quantity"] for i in cart)

    return templates.TemplateResponse('user_cart.html', {'request' :
request,
                                                         'cart' : cart,
                                                         'user' : user,
                                                         'total_price' :
total_price})
```

Відповідає за відображення товарів у кошику користувача. Якщо користувач не авторизований, відбувається перенаправлення на сторінку входу. Також рахується загальна сума всіх товарів.

```
@app.get('/place_an_order/{order_id}')
def place_an_order(request: Request, order_id:int, db: Session =
Depends(get_db), user: dict = Depends(get_current_user)):
    if user is None or user['id'] == None:
        return RedirectResponse(url="/login")

    return templates.TemplateResponse('delivery_data.html', {'request' :
request,
                                                             'order_id' :
order_id,
                                                             'user' :
user})
```

Відкриває сторінку оформлення доставки для вибраного замовлення за його ID. Доступна лише для авторизованих користувачів. Використовується для переходу до заповнення даних доставки.

```
@app.post("/add_review")
async def add_review(request: Request, db: Session = Depends(get_db)):
    data = await request.json()
    new_review = crud.add_review(db, data)
    return new_review
```

Приймає JSON-дані з відгуком через запит POST та додає їх до бази даних за допомогою відповідної функції в CRUD-модулі. Використовується для додавання нових відгуків на товари.

```
@auth_router.post('/login')
def login (response: Response, form_data: OAuth2PasswordRequestForm =
Depends(),
           db: Session = Depends(app.database.get_db)):
    user = db.query(app.models.User).filter(app.models.User.email ==
form_data.username).first()

    if not user or not verify_password(form_data.password,
user.hashed_password):
        raise HTTPException(status_code=400, detail='invalid password')

    token = create_access_token(data={'sub' : user.email,
'id' : user.id})
    response.set_cookie(key='Authorization', value=f'Bearer {token}',
                        httponly=True, secure=True, path='/')

    return {'access_token' : token, 'token_type' : 'bearer'}
```

Ця функція реалізує автентифікацію користувача через логін-форму, що використовує стандарт OAuth2PasswordRequestForm. Вона приймає логін (у вигляді email) та пароль, перевіряє їх у базі даних, і у випадку успіху генерує JWT-токен, який зберігається у браузері користувача як HttpOnly cookie. Завдяки httponly=True, токен недоступний JavaScript, що підвищує безпеку.

```
@auth_router.post('/logout')
async def logout(response: Response):
    response.delete_cookie(key='Authorization', path='/')
    return {"message": "Logged out successfully"}
```

Відповідає за завершення сеансу користувача. При виклику функції cookie з токеном (Authorization) видаляється за допомогою response.delete_cookie. Після цього токен більше не передається у запитах, і доступ до захищених маршрутів блокується.

```
async def get_current_user(request: Request):
    token = request.cookies.get('Authorization')
    if token is None:
        return None

    token = token.split(' ')[1]
    try:
        payload = jwt.decode(token, SECRET_KEY, algorithms=[ALGORITHM])
        username: str = payload.get('sub')
        user_id: str = payload.get('id')
        if username is None or user_id is None:
            raise HTTPException(status_code=401,  detail='second')
        return {'username' : username, 'id' : user_id}
    except JWTError:
        return {'username' : None, 'id' : None}
```

Допоміжна функція, яка витягує токен з cookie запиту та декодує його. Після розшифрування JWT перевіряє наявність необхідних полів (sub і id). У разі успішного розпізнавання токена повертає словник з email (username) та id користувача. Якщо токен недійсний або відсутній — повертає None, що дозволяє контролювати доступ до приватних сторінок.

```
def hash_password(password: str) -> str:
    return pwd_context.hash(password)

def verify_password(plain_password: str, hashed_password: str) -> bool:
    return pwd_context.verify(plain_password, hashed_password)
```

Ці функції відповідають за хешування паролів при реєстрації користувача та перевірку паролів при вході. Паролі ніколи не зберігаються у відкритому вигляді — тільки у вигляді хешу з використанням безпечного алгоритму (наприклад, bcrypt через passlib).

```
def create_access_token(data: dict, expires_delta: int =
int(ACCESS_TOKEN_EXPIRE_MINUTES)):
    to_encode = data.copy()
    expire = datetime.utcnow() + timedelta(minutes=expires_delta)
    to_encode.update({'exp' : expire})
    return jwt.encode(to_encode, SECRET_KEY, algorithm=ALGORITHM)
```

Створює новий JWT-токен на основі переданих даних (sub, id) і встановлює час дії (exp). Токен підписується секретним ключем (SECRET_KEY) і передається користувачу для подальшого доступу до приватних маршрутів.

```
def verify_access_token(token: str):
    try:
        payload = jwt.encode(token, SECRET_KEY, algorithm=ALGORITHM)
        return payload
    except JWTError:
        return None
```

Призначена для перевірки дійсності JWT-токена.

Наведені вище функції є ключовими компонентами, необхідними для ефективного функціонування маркетплейсу. Вони забезпечують базові процеси взаємодії між продавцями та покупцями, управління товарами, обробку замовлень.

ДОДАТОК Б

Огляд сайту

Головна сторінка вебзастосунку була продемонстрована в основній частині кваліфікаційної роботи як приклад реалізації інтерфейсу користувача та початкової навігації по системі.

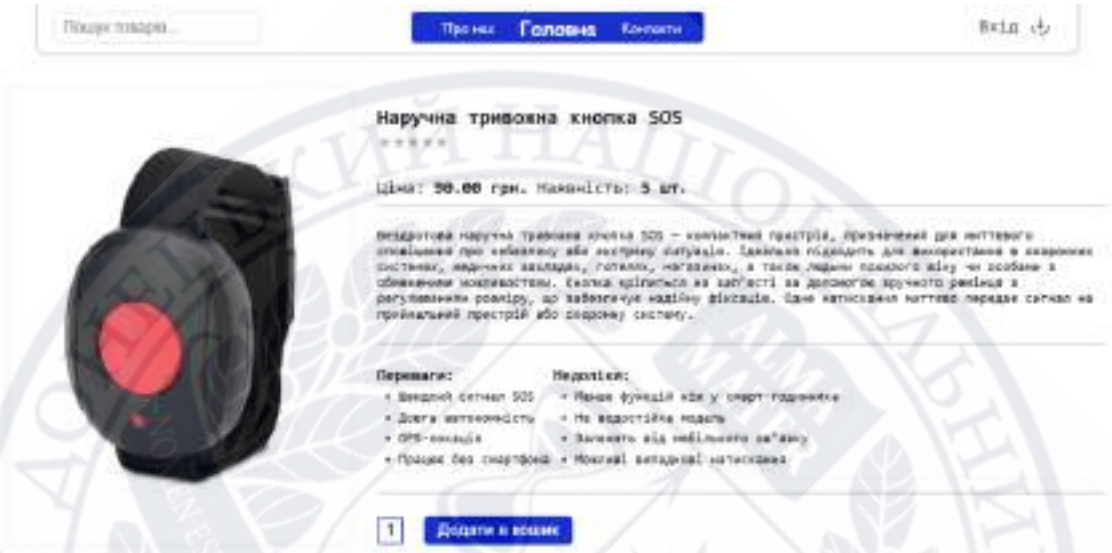

Рисунок Б.1 – Сторінка товару

На зображенні представлено сторінку з детальним описом товару в маркетплейсі. Виведено інформацію про тривожну кнопку зокрема зображення, назву, рейтинг у вигляді зірочок, ціну, опис характеристик та кнопку для додавання товару до кошика з можливістю вибору кількості. У нижній частині сторінки розміщено блок для відгуків користувачів.

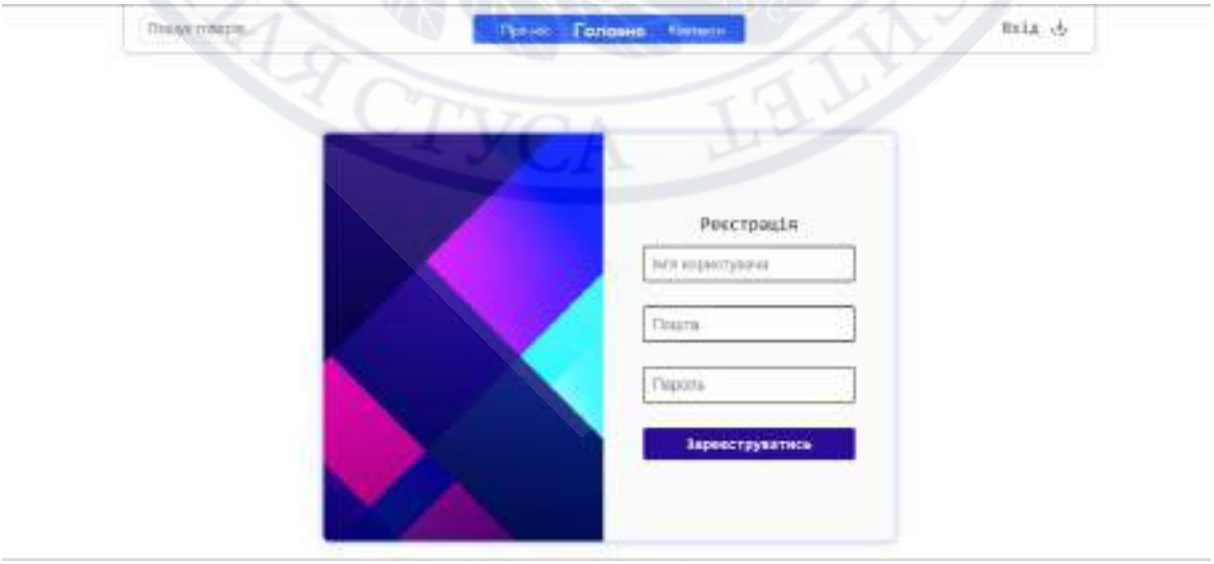

Рисунок Б.2 – Сторінка реєстрації

На зображенні зображено сторінку реєстрації користувача на вебсайті. У центрі розташовано форму, поділену на дві частини: ліву — з абстрактним фоновим зображенням, і праву — з полями для введення даних. Користувач може ввести ім’я, електронну пошту та пароль, після чого натиснути кнопку «Зареєструватись».

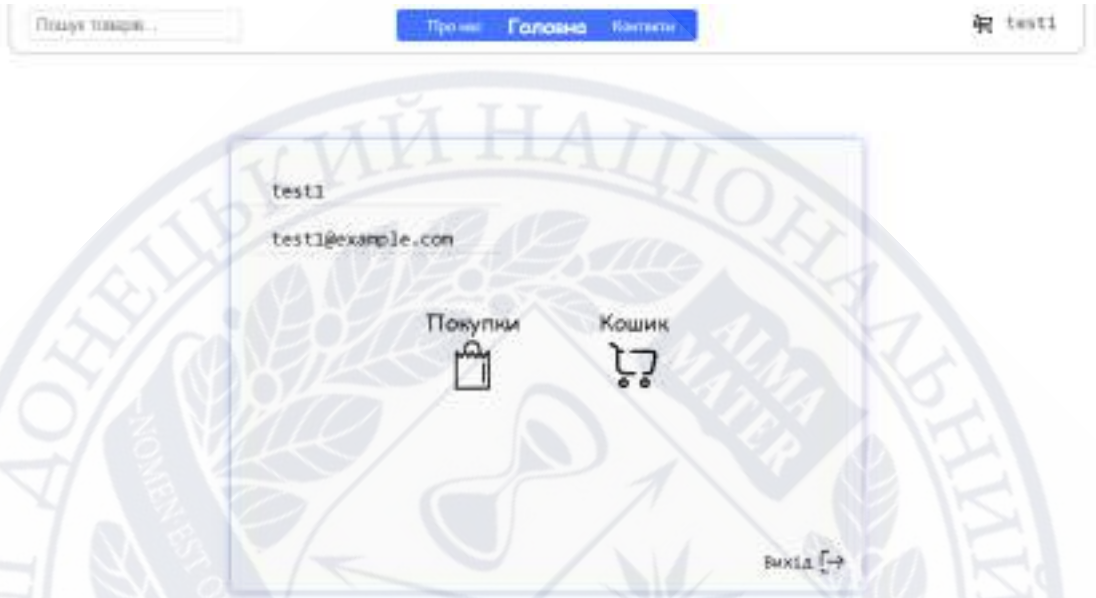

Рисунок Б.3 – Особистий кабінет користувача

Це сторінка профілю користувача після входу, на якій відображено ім’я (test1), email (test1@example.com), кнопки для перегляду покупок і кошика з відповідними іконками, а також кнопка «Вихід» у правому нижньому куті.

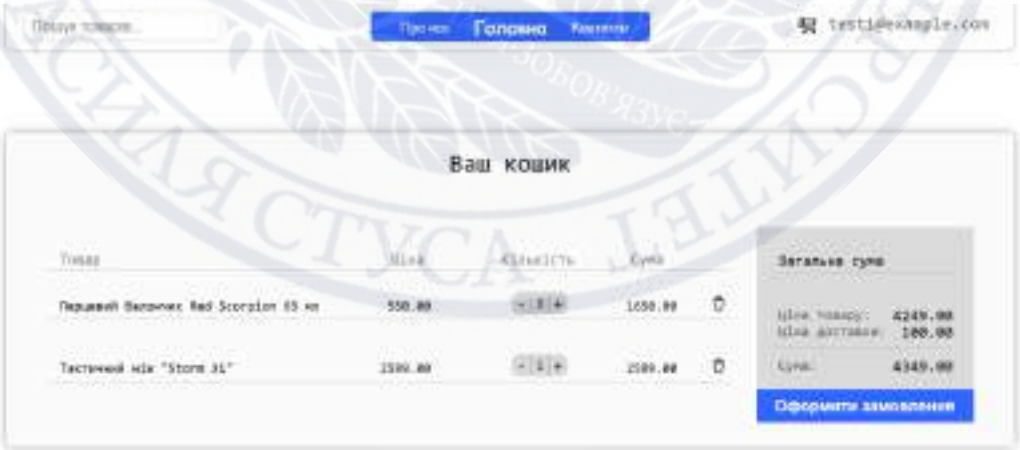

Рисунок Б.4 – Сторінка кошика користувача

Сторінка кошика містить список обраних товарів, для кожного з яких відображаються назва, ціна за одиницю, вибрана кількість і підсумкова вартість

за цей товар. Справа наведено загальну суму всіх товарів, до якої додається вартість доставки.

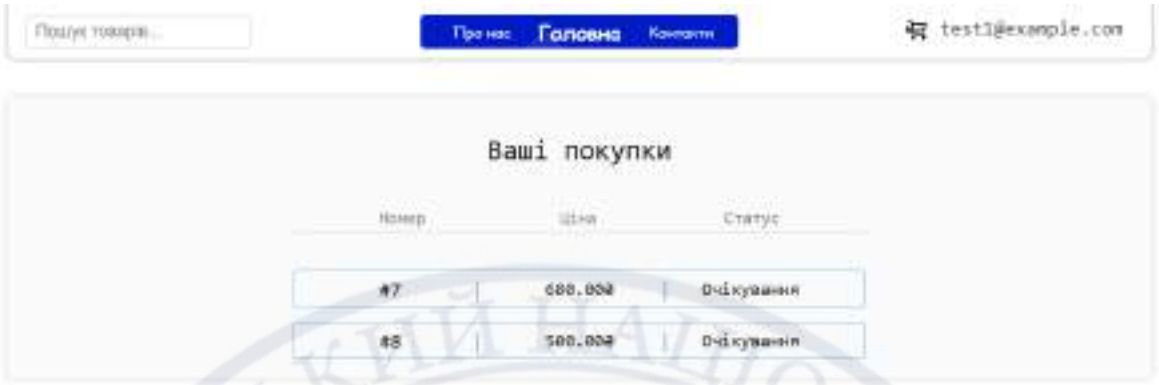

Рисунок Б.5 – Сторінка покупок користувача

Сторінка зі списком покупок містить перелік усіх замовлень користувача, де для кожного замовлення вказано його номер, загальну суму покупки та поточний статус (наприклад, "Очікується", "Виконано" або "Скасовано"), що дозволяє швидко переглянути історію придбань і стан обробки кожного з них.

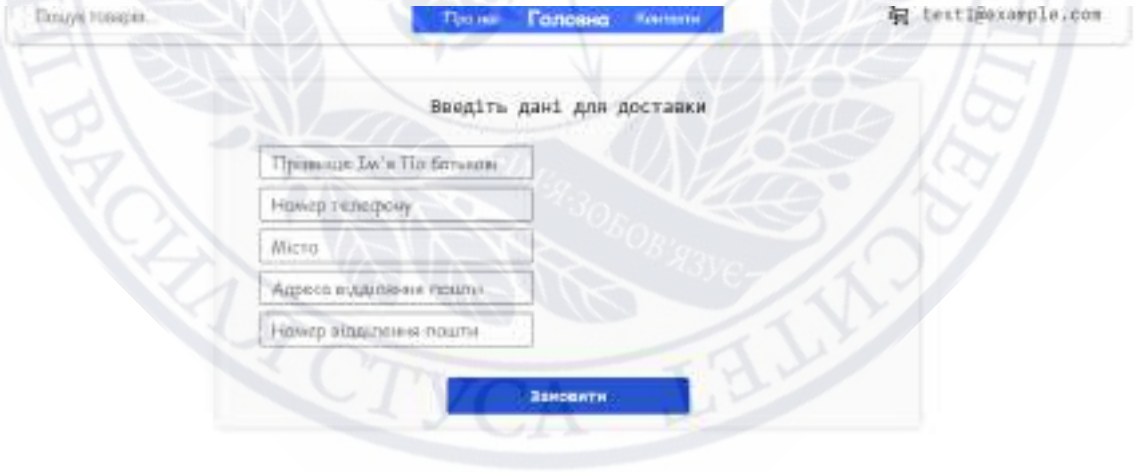

Рисунок Б.6 – Сторінка для введення даних доставки

Сторінка введення даних доставки містить форму, де користувач повинен заповнити поля з інформацією для доставки замовлення: ім’я та прізвище, номер телефону, адреса (вулиця, будинок, квартира), місто, область, поштовий індекс і, за потреби, коментар до замовлення; також може бути вибір способу доставки (наприклад, кур’єром або до відділення пошти).

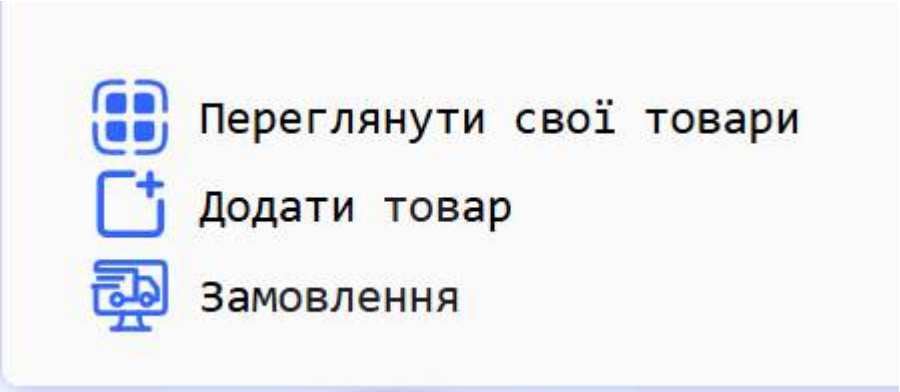

Рисунок Б.7 – Функції для продавця

Додаткові функції, які є у користувача, якщо він має статус продавця.

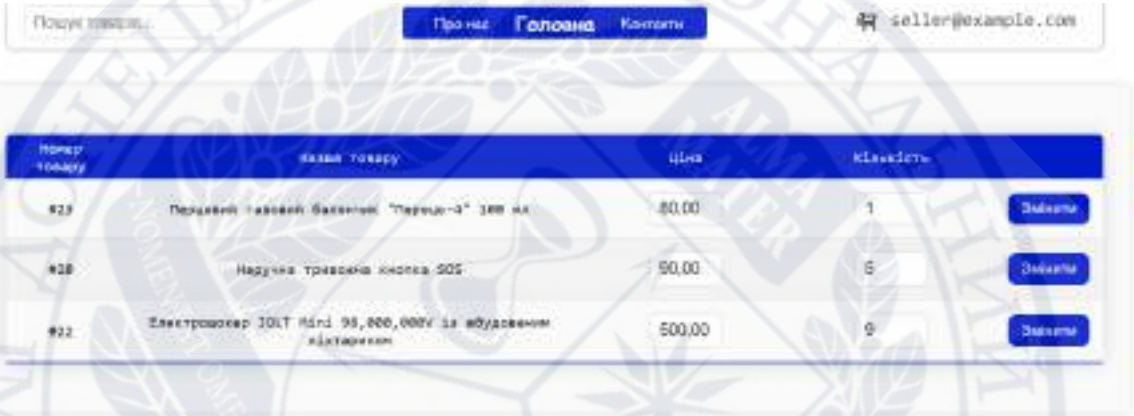

Рисунок Б.8 – Сторінка товарів продавця

На сторінці представлено таблицю зі списком товарів, що містить такі стовпці: номер товару, назва товару, ціна, кількість, а також кнопку для редагування «Змінити». Кожен рядок таблиці відповідає окремому товару з можливістю змінити його кількість та ціну.

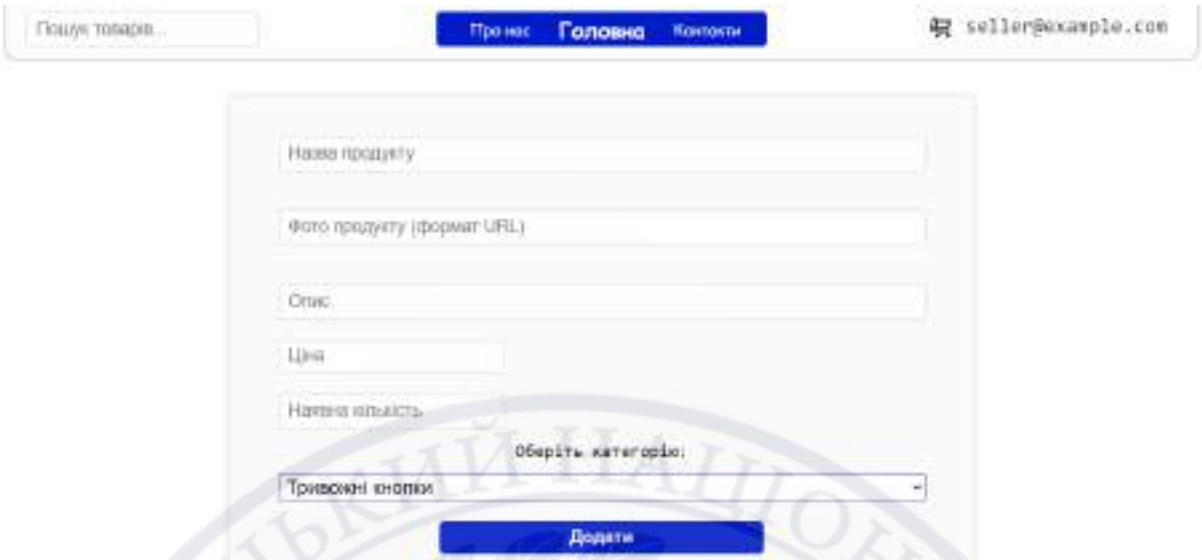

Рисунок Б.9 – Сторінка для додавання товарів

На сторінці розміщено форму для додавання нового товару. Вона складається з полів для введення назви, посилання на фото, опису, ціни, наявної кількості, а також випадаючого списку для вибору категорії. Нижче форми розміщено синю кнопку «Додати», яка призначена для підтвердження введених даних. Форма має чітку структуру, усі елементи розташовані вертикально, що забезпечує зручність заповнення.

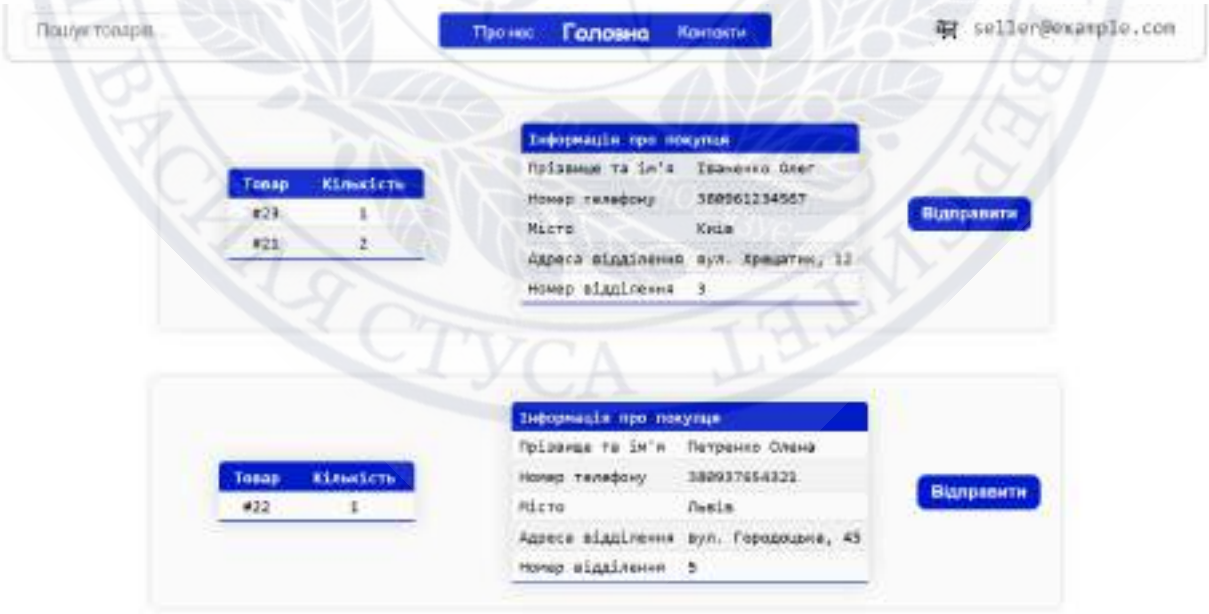

Рисунок Б.10 – Сторінка для перегляду замовлень

На зображенні показано інтерфейс сторінки продавця з переліком замовлень, де кожне замовлення містить таблицю з товаром, інформацією про

покупця (ПІБ, телефон, місто, адреса, відділення) та кнопкою для оновлення статусу, надсилання замовлення.