МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ВЕТЕРИНАРНОЇ МЕДИЦИНИ
ТА БІОТЕХНОЛОГІЙ ІМЕНІ С.З.ҐЖИЦЬКОГО

Факультет механіки, енергетики та інформаційних технологій
Кафедра інформаційних технологій

КВАЛІФІКАЦІЙНА РОБОТА

першого (бакалаврського) рівня вищої освіти

на тему: **«Розробка Web застосунку магазину комп'ютерної техніки»**

Виконав: студент 4 курсу групи Іт-41

Спеціальності
126 «Інформаційні системи та технології»
(шифр і назва)

Фурдзин Денис Дмитрович
(прізвище і ініціали)

Керівник: к. е. н., доц. Шувар Б. І.
(прізвище і ініціали)

Рецензент: к.т.н., доцент Гуменюк Р.В.
(прізвище і ініціали)

ДУБЛЯНИ-2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ВЕТЕРИНАРНОЇ МЕДИЦИНИ ТА БІОТЕХНОЛОГІЙ ІМЕНІ С.З.ҐЖИЦЬКОГО

ФАКУЛЬТЕТ МЕХАНІКИ, ЕНЕРГЕТИКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Перший (бакалаврський) рівень вищої освіти
Спеціальність 122 «Комп'ютерні науки»

«ЗАТВЕРДЖУЮ»
Завідувач кафедри _____
_____
«____» _____ 202_ р.

ЗАВДАННЯ
на кваліфікаційну роботу студенту
Фурдзина Дениса Дмитровича
(прізвище, ім'я, по батькові)

1. Тема роботи: *«Розробка Web застосунку магазину комп'ютерної техніки»*
2. Керівник роботи *к.е.н., доцент, Шувар Богдан Іванович*
(наук.ступінь, вч. звання, прізвище, ініціали)

затверджені наказом Львівського НУП №123/К-с від 25.02.2025р.
Строк подання студентом роботи 09.06.2025р.
Вихідні дані до роботи: Перелік функціональних вимог до інтернет-магазину комп'ютерної техніки, сучасні технології розробки веб-застосунків (HTML, CSS, JavaScript, Flask, SQLite, Docker), а також вимоги до збереження даних, авторизації користувачів і роботи адміністративної панелі.

3. Зміст розрахунково-пояснювальної записки (перелік питань, які необхідно розробити)
Вступ, Розділ 1. Аналіз предметної області, Розділ 2. Постановка завдання та відбір архітектурних інструментів, Розділ 3. Проєктування інтернет-магазину з продажу комп'ютерної техніки, Розділ 4. Охорона праці, Висновки, Список використаних джерел, Додатки.
4. Перелік ілюстраційного матеріалу (з точним зазначенням обов'язкових схем та моделей): *Діаграма варіантів використання (Use-Case), фрагменти інтерфейсу інтернет-магазину (головна сторінка, каталог, кошик, профіль користувача), схема бази даних на основі SQLite, UML-діаграма класів системи, вигляд адміністративної панелі для керування товарами, таблиці результатів ручного тестування, вигляд Docker-контейнера, запуск системи командою docker-compose up.*

5. Консультанти з розділів:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	*Шувар Б. І., доцент кафедри інформаційних технологій*		
5	*Городецький І. М., доцент кафедри інженерної механіки*		

6. Дата видачі завдання 26.02.2025 р.

Календарний план

№ з/п	Назва етапів дипломного проєкту	Терміни виконання етапів роботи	Примітка
1.	*Написання першого розділу*	27.02.2025 – 15.03.2025	
2.	*Виконання другого розділу та аркушів ілюстраційного матеріалу до нього*	15.03.2025 – 02.04.2025	
3.	*Виконання третього, четвертого розділів та аркушів ілюстраційного матеріалу до нього*	03.04.2025 – 25.04.2025	
4.	*Написання розділу «Охорона праці»*	26.04.2025 – 28.04.2025	
5.	*Завершення оформлення розрахункового-пояснювальної записки та аркушів ілюстраційного матеріалу*	29.04.2025 – 29.05.2025	
6.	*Завершення роботи цілому*	30.05.2025 – 07.06.2025	

Студент _____, Фурдзин Д.Д.
(підпис) (прізвище, ініціали)

Керівник роботи _____, Шувар Б.І.
(підпис) (прізвище, ініціали)

УДК 004.738.5

Розробка Web застосунку магазину комп’ютерної техніки.

Фурдзин Денис Дмитрович. Кафедра ІТ. Дубляни, ЛНУВМБ ім. С.З.Ґжицького, 2025.

Кваліфікаційна робота: 60 ст. текст. част., 21 рис., 4 табл., 15 літ. джерел.

У кваліфікаційній роботі описано процес розробки веб-застосунку інтернет-магазину для продажу комп’ютерної техніки з використанням мови програмування Python, фреймворку Flask та бази даних SQLite. Система забезпечує можливість реєстрації користувачів, авторизації, перегляду товарного каталогу, додавання товарів до кошика, оформлення замовлення та керування товарами з боку адміністратора.

Здійснено аналіз аналогічних веб-магазинів, визначено ключові вимоги до інтерфейсу, функціоналу та швидкодії системи. Розроблено UML-діаграми, базову структуру бази даних, а також адміністративний інтерфейс для управління товарним контентом. Реалізовано Docker-розгортання для зручного хостингу застосунку.

Проведено ручне тестування функціональності з метою перевірки стабільності роботи, коректності взаємодії з базою даних та зручності інтерфейсу. Робота містить візуалізацію ключових компонентів системи та приклади реалізації коду.

Ключові слова: веб-застосунок, інтернет-магазин, Flask, Python, SQLite, Docker, каталог товарів, адміністрування, тестування, електронна комерція.

Key words: web application, online store, Flask, Python, SQLite, Docker, product catalog, administration, testing, e-commerce.

ЗМІСТ

ВСТУП

Сучасна епоха цифрової трансформації охоплює всі сфери людської діяльності, і торгівля не є винятком. Інтернет поступово витісняє традиційні форми реалізації товарів та послуг, що зумовлено не лише зручністю онлайн-покупок для кінцевого споживача, а й можливістю для бізнесу охопити значно ширшу аудиторію з мінімальними витратами на фізичну інфраструктуру. В результаті цього процесу інтернет-магазини стали невіддільною частиною комерційного простору, а розробка відповідних веб-застосунків — важливим напрямом у сфері інформаційних технологій.

Зростання популярності онлайн-магазинів зумовлює необхідність створення надійних, зручних, захищених і продуктивних веб-рішень. Особливої актуальності набуває розробка інтернет-магазинів для продажу комп'ютерної техніки — товару, що потребує детального опису, чіткої категоризації, можливості порівняння характеристик і безперебійної обробки замовлень. Споживачі очікують від таких сайтів зручного інтерфейсу, швидкого пошуку, ефективної фільтрації, адаптації до мобільних пристроїв та безпечної авторизації. Усе це висуває до розробника комплексні вимоги щодо проєктування та реалізації застосунку.

Розробка веб-застосунку в межах даної дипломної роботи передбачає використання сучасного технологічного стеку. Клієнтська частина реалізована за допомогою HTML, CSS та JavaScript — базових веб-технологій, що забезпечують побудову інтерфейсу та елементарну інтерактивність. Серверна логіка написана з використанням мікрофреймворку Flask — одного з найпопулярніших інструментів для створення легких, але функціональних веб-додатків на мові Python. База даних реалізована на основі SQLite — простої у використанні вбудованої реляційної СКБД, яка є зручною для малих і середніх проєктів. Для ізоляції середовища та зручності розгортання застосовано Docker-контейнери. Особливу увагу приділено безпеці користувацьких даних — для збереження паролів реалізовано шифрування із застосуванням бібліотеки bcrypt. Тестування

виконано вручну відповідно до попередньо сформованих сценаріїв, що дозволило оцінити функціональність і стабільність роботи програми.

Метою дипломної роботи є створення веб-застосунку інтернет-магазину комп’ютерної техніки, який забезпечуватиме повноцінну взаємодію користувача з системою: перегляд та пошук товарів, роботу з обліковим записом, авторизацію, збереження замовлень та їх обробку. Крім того, у проєкті реалізовано адміністративну частину, що дозволяє керувати базою товарів та користувачів.

У структурі роботи передбачено: аналіз предметної області та існуючих аналогів; постановку задачі та обґрунтування вибору архітектурних рішень; проєктування бази даних і логіки взаємодії між модулями; реалізацію застосунку та його функціональне тестування; а також розгляд аспектів охорони праці, пов’язаних із використанням комп’ютерної техніки під час розробки і експлуатації ПЗ.

Результати дослідження можуть бути використані як основа для створення повноцінного інтернет-магазину з реальним функціоналом, а також як навчальний приклад впровадження сучасного технологічного підходу до веб-розробки.

РОЗДІЛ 1.
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Опис предметної області

Інтернет-магазини є одним із найпоширеніших прикладів реалізації електронної комерції у веб-середовищі. Вони стали потужним інструментом для ведення бізнесу, що дозволяє здійснювати торгівлю товарами та послугами без необхідності відкриття фізичної точки продажу. На відміну від звичайних сайтів-візитівок, які лише інформують користувача, інтернет-магазини передбачають активну взаємодію з базою даних, користувачем, платіжними системами та адміністративною частиною системи. У своїй сутності інтернет-магазин поєднує веб-інтерфейс, інструменти керування контентом (товарами), механізми автентифікації та сервіси обробки транзакцій.

Згідно з визначенням, «електронна комерція — як діловій активності і вид суспільних відносин щодо купівлі-продажу товарів та послуг, що передбачає взаємодію сторін на основі інформаційних мереж і в результаті якого право власності чи право користування товаром або послугою передається від однієї особи до іншої» [5]. Таким чином, інтернет-магазин є спеціалізованим прикладом інформаційної системи, яка надає користувачу можливість здійснити всі основні етапи покупки — від перегляду товару до оплати й отримання квитанції. До цього додається можливість відстеження статусу замовлення, перегляд історії покупок, керування профілем та отримання повідомлень від магазину.

З точки зору архітектури програмного забезпечення, інтернет-магазин — це багаторівнева система. Клієнтська частина (frontend) забезпечує взаємодію з користувачем, включаючи форму реєстрації, авторизації, вітрину товарів, фільтри, кошик та оформлення замовлень. Серверна частина (backend), реалізована у цьому проєкті на Flask, відповідає за обробку запитів, перевірку

прав доступу, взаємодію з базою даних та забезпечення бізнес-логіки. Для зберігання даних використовується SQLite — реляційна база даних, яка зручно інтегрується з Flask і не потребує розгортання окремого серверного ПЗ. Крім того, реалізовано систему авторизації, де паролі користувачів зберігаються не у відкритому вигляді, а у хешованому форматі за допомогою бібліотеки bcrypt. Як зазначається в офіційній документації: «bcrypt is a password hashing function designed to build a cryptographically secure representation of a password string using a salt and a computationally expensive algorithm» [1].

Значну роль у побудові ефективного інтернет-магазину відіграє організація предметної області. У випадку торгівлі комп'ютерною технікою система повинна підтримувати багаторівневу категоризацію товарів, специфікації, технічні характеристики, варіанти конфігурацій, фільтри за параметрами (тип процесора, об'єм пам'яті, виробник тощо), а також наочне відображення даних у вигляді зображень, таблиць і порівнянь. Складність даних вимагає чіткого моделювання зв'язків між сутностями: «товар», «категорія», «замовлення», «користувач», «статус доставки», «оплата» тощо. Водночас важливо забезпечити гнучкість структури даних, оскільки асортимент техніки може часто оновлюватися, доповнюватися новими параметрами або серіями продуктів.

Ще однією особливістю є інтеграція з платіжними системами. Хоча у межах цієї дипломної роботи реалізовано лише формування та збереження замовлення, архітектура застосунку передбачає можливість подальшого додавання платіжних модулів. Це може бути реалізовано за допомогою сторонніх API — наприклад, LiqPay, Fondy або Stripe — з обов'язковою перевіркою статусу транзакцій, створенням платіжних сесій і повідомлень про оплату.

На рисунку представлено узагальнену логічну модель взаємодії основних елементів інтернет-магазину, яка охоплює ключові кроки обробки даних — від звернення користувача до формування та збереження замовлення в базі даних.(рис. 1.1)

Описана предметна область охоплює широке коло інформаційних процесів, реалізованих за допомогою програмних засобів. У межах цього проєкту інтернет-магазин виконує функції реєстрації та авторизації користувачів, виводу товарів за категоріями, додавання їх до кошика, формування замовлення та збереження його до бази даних. Таким чином, модель охоплює як логіку користувача, так і логіку адміністратора, що має змогу переглядати та обробляти замовлення, а також оновлювати асортимент товарів.

Загалом, предметна область інтернет-магазину комп'ютерної техніки поєднує в собі високий рівень структурованості, складні взаємозв'язки між даними та підвищені вимоги до безпеки й зручності користування. Її комплексність робить цю тему актуальною з точки зору навчальної практики і прикладного застосування.

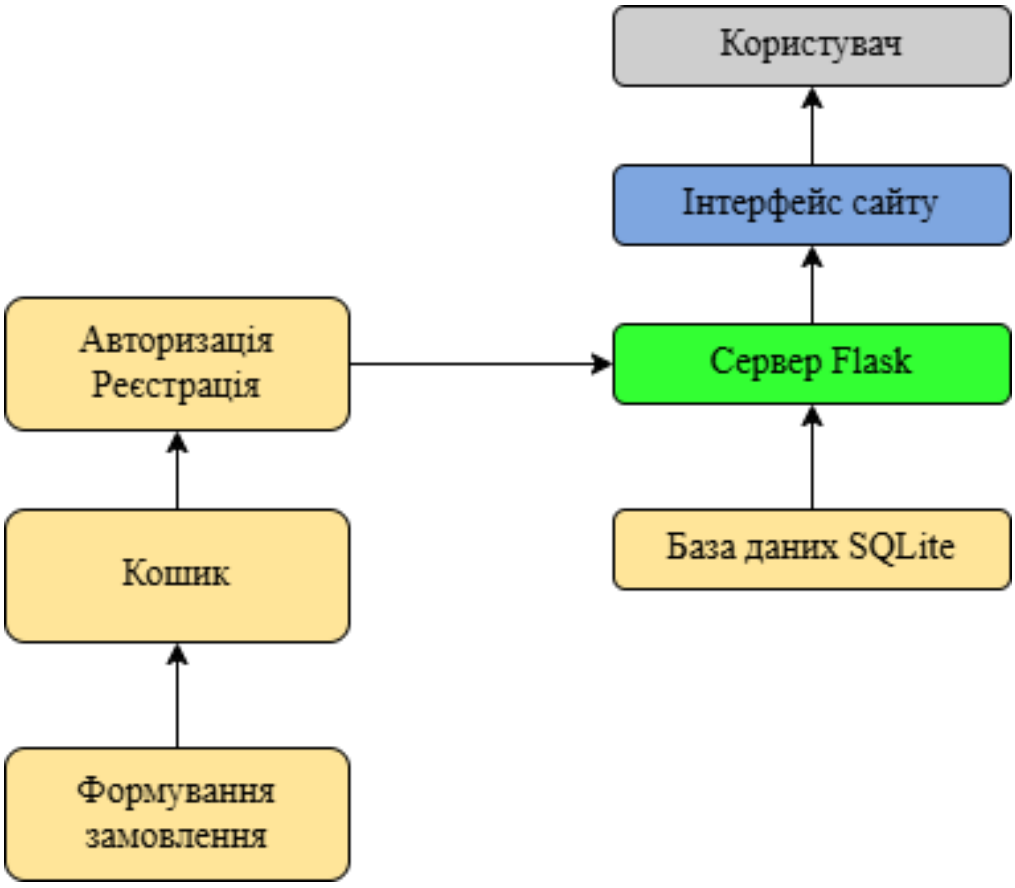

Рисунок 1.1 - Узагальнена модель взаємодії елементів інтернет-магазину

1.2. Огляд аналогів інтернет-магазинів

Перш ніж розпочати проєктування власного веб-застосунку інтернет-магазину комп'ютерної техніки, доцільно дослідити вже реалізовані рішення, що зарекомендували себе в українському інтернет-просторі. Аналіз їх функціональності, структури та технічної реалізації дозволяє не лише уникнути поширених помилок, але й успадкувати найкращі практики для впровадження у власному продукті. Такий підхід забезпечує створення конкурентоспроможного, зручного та надійного вебсайту.

У межах цього підрозділу було детально проаналізовано чотири платформи: Rozetka, MOYO, Brain та Comfy. Усі вони є провідними представниками e-commerce в Україні, однак мають відмінності в реалізації інтерфейсів, фільтраційних систем, адаптивності та технічних основ.

Аналіз проводився методом функціонального огляду — через безпосередню взаємодію з сайтом, огляд HTML-коду, перевірку сценаріїв навігації, швидкості відгуку, а також з урахуванням досвіду користувача. Результати зведено в таблицю 1.1.[8, 13,14,15]

Таблиця 1.1 - Порівняння інтернет-магазинів комп'ютерної техніки

Назва	**Інтерфейс**	**Фільтрація**	**Порівняння товарів**	**Мобільна версія**	**Завантаження**
Rozetka	Висока	Розширена	Так	Так	Висока
MOYO	Середня	Базова	Ні	Так	Середня
Brain	Середня	Базова	Так	Так	Середня
Comfy	Висока	Середня	Так	Так	Висока

На прикладі Rozetka бачимо ідеальну інтеграцію зручного інтерфейсу та високої продуктивності. У коді присутні елементи асинхронної завантажуваності, компоненти на React або подібних бібліотеках, що свідчить

про сучасну фронтенд-архітектуру. Навігація інтуїтивна, головна сторінка наповнена категоріями, фільтрами та акційними пропозиціями.

Comfy теж демонструє якісну реалізацію, хоча й дещо поступається гнучкістю фільтрів. Ймовірно, його побудовано на кастомній CMS з компонентами, які мають адаптивний інтерфейс і оптимізовані для мобільних пристроїв.

Натомість MOYO та Brain працюють на базі шаблонних систем. У MOYO простежуються ознаки OpenCart або Magento без значної кастомізації. Інтерфейс менш структурований, фільтри обмежені, хоча мобільна версія функціональна. Brain має дещо кращу організацію структури, але сайт важчий у навігації, а швидкість завантаження сторінок нижча.

Щодо платформ, то слід виокремити два підходи: використання CMS або повна кастомна розробка. CMS (як-от OpenCart, Magento, Bitrix) забезпечують швидкий запуск і просту підтримку, однак мають обмеження у продуктивності та адаптації під нестандартну логіку. Кастомні рішення — як Rozetka — вимагають більше зусиль, але надають повну свободу в архітектурі та масштабуванні.

Крім того, під час вивчення архітектури розглянутих сайтів було зафіксовано використання різних підходів до реалізації пошуку та фільтрації. Наприклад, Rozetka використовує динамічну фільтрацію з асинхронною підгрузкою результатів, що підвищує швидкість взаємодії. Також відзначається використання серверного рендерингу сторінок з подальшим кешуванням — такий підхід забезпечує як швидке завантаження, так і ефективну SEO-оптимізацію. У той час як у MOYO фільтрація реалізована за допомогою статичних запитів із повною перезавантажуваністю сторінки, що знижує зручність для користувача.

З технічного боку, Comfy і Brain демонструють застосування CDN (Content Delivery Network) для прискорення доставки вмісту, зокрема зображень і скриптів. Це позитивно впливає на загальну продуктивність і відповідає сучасним підходам у фронтенд-розробці. Rozetka, у свою чергу, застосовує також

систему lazy-load для оптимізації медіа-контенту, що дозволяє зменшити навантаження на мережу.

Окрему увагу привертає реалізація особистого кабінету користувача. На більшості платформ він передбачає перегляд історії замовлень, зміну особистих даних, налаштування повідомлень та збереження обраних товарів. Така функціональність підвищує лояльність користувачів і є обов'язковою для повноцінного інтернет-магазину, зокрема в сегменті комп'ютерної техніки, де товари зазвичай мають високу вартість і вибір потребує часу.

Таким чином, проведений аналіз засвідчує, що при створенні власного застосунку необхідно звертати увагу не лише на візуальну частину та навігацію, а й на технічну складову — ефективність завантаження, реалізацію пошукових механізмів, гнучкість архітектури та можливість масштабування. Саме ці аспекти ляжуть в основу наступних розділів, присвячених технічному проєктуванню майбутнього веб-застосунку.

1.3. Функціональність. Переваги та недоліки інтернет-магазину

Функціональність є визначальною характеристикою будь-якого інтернет-магазину, оскільки саме вона забезпечує повний цикл взаємодії користувача з системою — від ознайомлення з асортиментом товарів до оформлення та підтвердження покупки. Ефективна реалізація функціоналу дозволяє не лише підвищити зручність користування, а й створити конкурентну перевагу серед інших онлайн-сервісів.

У типовій реалізації інтернет-магазину функціональність включає декілька базових елементів. Насамперед це реєстрація та авторизація користувача, що забезпечує ідентифікацію клієнта та доступ до персонального кабінету. Далі — каталог товарів, структурований за категоріями та підкатегоріями з можливістю швидкого пошуку й застосування фільтрів. Надзвичайно важливою є функція кошика, де зберігаються вибрані товари до моменту оформлення замовлення. Також обов'язковим є механізм створення й підтвердження замовлення, який

повинен включати введення даних доставки, вибір способу оплати, підтвердження покупки та формування рахунку. У розширеному варіанті передбачено реалізацію історії замовлень, відгуків про товари, списків бажаного, системи знижок або акцій, а також інтеграцію з платіжними сервісами та службами доставки.

З точки зору переваг, інтернет-магазини мають низку незаперечних сильних сторін, що вигідно відрізняє їх від традиційної роздрібної торгівлі. Однією з головних переваг є доступність — користувач може здійснити покупку в будь-який момент часу, не обмежуючись годинами роботи магазину чи географічним розташуванням. Така можливість значно підвищує ймовірність завершення покупки, особливо серед зайнятих людей або клієнтів із віддалених населених пунктів.

Другою суттєвою перевагою є економічна ефективність. Застосування електронної комерції дозволяє значно зменшити витрати на обслуговування фізичної торгової точки та оптимізувати логістику. «Упровадження інтернет-торгівлі дозволяє підприємствам уникати значної частини витрат на оренду приміщень, оплату праці персоналу, комунальні послуги та інші традиційні витрати» [7]. Відсутність потреби в утриманні магазину, спрощення складського обліку та автоматизація документообігу сприяють зниженню операційних витрат і підвищують маржинальність бізнесу.

Ще однією вагомою перевагою є масштабованість. У разі розширення асортименту, зростання кількості користувачів чи потреби в нових функціях інтернет-магазин легко адаптується до нових умов завдяки гнучкості архітектури, особливо за умови правильного проєктування бази даних і серверної логіки.

Крім того, інтернет-магазини дозволяють автоматизувати численні процеси — ведення обліку товарів, відстеження замовлень, генерацію звітності, інтеграцію з CRM-системами тощо. Це суттєво зменшує людський фактор, прискорює обробку запитів і знижує ризик помилок.

Попри очевидні переваги, інтернет-магазини мають і низку недоліків або викликів, з якими стикається розробник. Одним із основних ризиків є небезпека витоку персональних даних. Оскільки користувачі залишають у системі чутливу інформацію — логіни, паролі, адреси доставки, а іноді й дані платіжних карток, важливо забезпечити захист на всіх рівнях. У цьому проєкті реалізовано шифрування паролів за допомогою бібліотеки bcrypt, що значно ускладнює несанкціонований доступ до облікових записів.

Ще одним обмеженням є відсутність фізичного контакту з товаром. Покупець не має змоги оглянути техніку вживу, перевірити якість збірки чи швидкодію. Це компенсується за допомогою якісних фотографій, описів, відеоогляді в і відгуків. У сфері комп'ютерної техніки це особливо актуально, адже товари мають складні технічні характеристики, і помилки у виборі можуть бути критичними.

Також необхідно враховувати технічні аспекти — підтримка сайту, оновлення, тестування, захист від атак тощо. Інтернет-магазин — це не разова розробка, а постійний процес підтримки та вдосконалення, що вимагає наявності фахівців або залучення зовнішніх сервісів.

У підсумку, інтернет-магазин є ефективним і сучасним інструментом для організації торгівлі, який надає широкий спектр можливостей як для бізнесу, так і для користувача. Водночас для його успішної реалізації необхідно враховувати технічні, безпекові й організаційні аспекти, що робить цю задачу багаторівневою та комплексною. У наступному підпункті буде розглянуто етапи створення інтернет-магазину як цілісного проєкту — від ідеї до реалізації.

1.4. Етапи створення інтернет-магазину

Створення інтернет-магазину — це комплексний процес, який поєднує різні галузі знань: від аналізу бізнес-процесів і UX-дизайну до розробки програмного забезпечення та адміністрування серверного середовища. Для досягнення високої якості кінцевого продукту розробка має ґрунтуватися на чітко

структурованій послідовності етапів, кожен з яких виконує окрему функцію в загальному життєвому циклі проєкту.

Першим етапом виступає аналіз предметної області та формулювання вимог, у межах якого здійснюється дослідження ринку, вивчаються потреби цільової аудиторії, визначається ціль проєкту та функціональні характеристики системи. Зокрема, формується уявлення про структуру каталогу товарів, ролі користувачів, необхідні інтерактивні елементи й сценарії взаємодії.

На другому етапі — проєктування архітектури та моделювання логіки функціонування — формалізуються бізнес-процеси за допомогою діаграм варіантів використання, визначається структура бази даних, обирається стек технологій. Це дозволяє забезпечити цілісність і узгодженість рішень на наступних етапах. Також формується уявлення про клієнтську та серверну частини, їхні зв’язки, способи обміну даними.

Третій етап — реалізація функціональності — охоплює безпосередню розробку програмного коду відповідно до обраної архітектури. На цьому етапі створюються модулі для реєстрації та авторизації, обробки замовлень, управління кошиком, взаємодії з базою даних, відображення контенту тощо. Також розробляється інтерфейс користувача, в якому реалізуються принципи зручності (usability) та адаптивності.

Четвертий етап — тестування системи. Його мета — забезпечити коректну роботу реалізованого функціоналу, перевірити відповідність очікуванням користувачів, виявити помилки або недоопрацювання до моменту запуску. У залежності від обраного підходу застосовується ручне або автоматизоване тестування.

П’ятий етап — розгортання інтернет-магазину та його підтримка. У сучасних умовах усе частіше використовуються контейнери (Docker) для створення ізольованого середовища, що дозволяє забезпечити стабільну роботу незалежно від конкретного обладнання чи ОС. Розгортання включає

налаштування серверної інфраструктури, баз даних, безпечного доступу, маршрутизації запитів та конфігурації хостингу.

Важливим також є етап післяпроєктного супроводу, який включає оновлення контенту, реалізацію додаткових функцій, підтримку безпеки та обслуговування клієнтів. З огляду на динаміку розвитку технологій, інтернет-магазин повинен бути гнучким до змін і масштабування.

Отже, дотримання загальноприйнятої методології створення веб-застосунків забезпечує підвищення надійності, зменшення кількості помилок, ефективність взаємодії між учасниками розробки та, зрештою, підвищення конкурентоспроможності електронного ресурсу.

РОЗДІЛ 2.
ПОСТАНОВКА ЗАВДАННЯ ТА ВІДБІР АРХІТЕКТУРНИХ ІНСТРУМЕНТІВ

2.1. Мета та постановка завдання проєкту

У сучасних умовах інтенсивного розвитку цифрових технологій електронна комерція набуває все більшого значення як повноцінна форма ведення бізнесу. Її практична реалізація базується на створенні інформаційних систем, які забезпечують ефективну взаємодію між користувачем і сервером, обробку замовлень, захист персональних даних та зручну навігацію. Метою цієї дипломної роботи є розробка веб-застосунку інтернет-магазину, що спеціалізується на продажу комп’ютерної техніки, із забезпеченням усіх ключових функцій електронної торгівлі — від реєстрації користувача до формування замовлення й адміністрування товарного асортименту.

На відміну від використання готових CMS-систем, які обмежують розробника в частині внутрішньої логіки, проєкт реалізовано з нуля за допомогою мови програмування Python і фреймворку Flask. Це дозволяє гнучко керувати структурою маршрутизації, взаємодією з базою даних, авторизацією користувачів і валідацією форм. В якості системи керування базами даних обрано SQLite, що є вбудованим реляційним механізмом зберігання, який дозволяє зручно працювати з локальними даними без потреби розгортання окремого серверного середовища. Клієнтська частина застосунку реалізована з використанням HTML, CSS і JavaScript, що забезпечує адаптивність і доступність інтерфейсу для користувачів із різних пристроїв. Задля підвищення безпеки реалізовано шифрування паролів за допомогою бібліотеки bcrypt, що дозволяє створювати хеші із сіллю та ускладнює несанкціонований доступ до облікових записів.

У межах реалізованого проєкту передбачено можливість реєстрації, входу в систему, перегляду асортименту продукції, використання технічних фільтрів, додавання товарів до кошика, оформлення замовлення та збереження всіх відповідних даних у базу. Окрім цього, реалізовано адміністративну частину, яка дозволяє змінювати дані про товари, видаляти або додавати нові позиції, а також переглядати надіслані замовлення з метою обробки.

Застосунок спроєктовано як модульна система з можливістю подальшого масштабування. Архітектура дозволяє розширити функціонал, додавши сторонні API, наприклад сервіси електронної оплати, служби доставки або CRM-системи. Це робить систему не лише навчальним проєктом, а й базою для реального впровадження у сферу електронної торгівлі. Кожен компонент, реалізований у проєкті, розглядається як частина єдиної інформаційної екосистеми, яка забезпечує повноцінне функціонування магазину в онлайн-середовищі.

Саме така форма взаємодії реалізована в межах створеного веб-застосунку, що підтверджує його відповідність теоретичним і практичним вимогам до сучасної інформаційної системи у сфері онлайн-торгівлі.

2.2. Організація процесу створення веб-застосунку

Розробка інтернет-магазину є складним багаторівневим процесом, що охоплює не лише програмування, а й ретельне планування, проєктування структури, забезпечення безпеки, тестування та налаштування середовища для розгортання. Щоб створити повноцінний, зручний та стабільний веб-застосунок, необхідно дотримуватись чіткої послідовності етапів. У межах цієї дипломної роботи реалізовано застосунок, орієнтований на продаж комп’ютерної техніки, що вимагає підтримки технічних фільтрів, багаторівневої категоризації товарів та безпечної обробки користувацьких даних.

Першим етапом розробки виступає аналіз потреб та постановка задачі, під час якого визначаються вимоги до функціоналу, майбутню структуру сайту, категорії товарів, а також основні ролі користувачів — клієнт, адміністратор,

гість. На цьому етапі важливо врахувати специфіку предметної області: для інтернет-магазину техніки значну увагу потрібно приділити точності опису товарів, структурі бази даних і можливості швидкої навігації.

Наступний етап — проєктування структури та архітектури системи — передбачає вибір технологічного стеку, розробку логічної структури бази даних, побудову маршрутів взаємодії між компонентами. Для розробки цього застосунку обрано Flask як гнучкий веб-фреймворк, SQLite як легку реляційну СКБД, HTML/CSS/JS для клієнтської частини та Docker для розгортання. Структурно розробляється загальна архітектура системи, яка охоплює інтерфейс користувача, серверну логіку, базу даних, зовнішні залежності та механізми автентифікації.

Після цього реалізується етап створення функціональних модулів. Спершу формується логіка реєстрації та авторизації користувача, де використовуються захищені методи хешування паролів (bcrypt). Далі реалізуються механізми перегляду товарів, пошуку, фільтрації за категоріями, додавання в кошик, оформлення замовлення. Вся функціональність реалізується на базі маршрутизованої логіки з обробкою форм, валідацією даних та записом до бази.

Окрему увагу приділяється етапу створення інтерфейсу користувача. Він повинен бути зручним, логічним і адаптивним до різних типів пристроїв. Важливо, щоб усі елементи інтерфейсу були зрозумілими без додаткових пояснень, а навігація — послідовною. У застосунку реалізовано розмітку HTML, стилізацію CSS, а також використано базовий JavaScript для клієнтської взаємодії з елементами сторінок.

Наступним етапом є тестування, яке проводиться з метою перевірки всіх реалізованих функцій у реальних сценаріях користування. У даному проєкті застосовано ручне тестування, що включало перевірку реєстрації, авторизації, заповнення кошика, обробки некоректних даних, навігації між сторінками, формування та збереження замовлень. Тестування дозволяє уникнути критичних

помилок на етапі запуску системи та сформувати початковий звіт про стабільність роботи.

Завершальний етап — розгортання та подальша підтримка — включає створення ізольованого серверного середовища з використанням Docker-контейнерів, що забезпечує однакову працездатність незалежно від платформи. Також налаштовується база даних, структура директорій, перевіряється коректність маршрутів, залежностей та портів. Цей етап є критично важливим для підготовки застосунку до подальшої експлуатації, оновлення або масштабування.

Узагальнену схему основних етапів розробки інтернет-магазину подано на рисунку 2.1.

Рисунок 2.1 - Основні етапи створення інтернет-магазину

Таким чином, поетапний підхід до створення інтернет-магазину забезпечує не лише логічну послідовність дій, а й мінімізацію ризиків, зменшення витрат часу та підвищення якості кінцевого продукту. У наступному розділі буде здійснено формалізацію поставленого завдання та розпочато технічне обґрунтування архітектурних рішень, необхідних для реалізації заявленого функціоналу.

2.2. Обґрунтування вибору архітектури веб-застосунку

Архітектура веб-застосунку є критично важливим аспектом у процесі його розробки, адже саме вона визначає логіку взаємодії між компонентами системи, їхню масштабованість, зручність підтримки та ефективність реалізації функціоналу. У межах дипломної роботи реалізовано інтернет-магазин комп'ютерної техніки, що функціонує на базі багаторівневої клієнт-серверної архітектури. Рішення ґрунтується на використанні фреймворку Flask для серверної частини, SQLite як вбудованої системи керування базою даних, HTML, CSS і JavaScript для клієнтської частини, а також Docker для ізольованого розгортання.

Однією з головних переваг обраної архітектури є її гнучкість і прозорість. Flask є мікрофреймворком на мові Python, що дозволяє швидко створювати REST-подібні веб-застосунки, при цьому зберігаючи повний контроль над маршрутизацією, обробкою запитів, валідацією форм і взаємодією з базою даних. Його мінімалістичний підхід і модульна структура дозволяють уникати надлишкової залежності від сторонніх рішень і реалізовувати логіку, що чітко відповідає специфіці предметної області.

В якості бази даних використовується SQLite — легка, локальна реляційна СКБД, яка забезпечує збереження структурованих даних без потреби в налаштуванні окремого серверного середовища. Це особливо доцільно в навчальному або невеликому комерційному проєкті, де важливо забезпечити простоту розгортання та високу продуктивність при обмеженій кількості одночасних запитів. Реалізована структура бази даних містить ключові сутності — користувачі, товари, замовлення, позиції в замовленнях — що дозволяє підтримувати повноцінну логіку взаємодії між клієнтською та серверною частинами.

Клієнтська частина створена з використанням HTML5, CSS3 і JavaScript, що забезпечує адаптивність інтерфейсу та доступність застосунку на різних типах пристроїв. Особливу увагу приділено питанням безпеки. Усі паролі

користувачів хешуються з використанням криптографічної бібліотеки bcrypt, яка забезпечує захист даних у разі витоку або компрометації сховища.

Із метою стандартизації розгортання застосунок упаковано у Docker-контейнери. Це дозволяє забезпечити однакове середовище виконання на різних пристроях та спростити процес інсталяції. Кожен компонент (сервер, проксі, сховище) виконується в окремому контейнері, що відповідає принципам контейнеризації та підвищує стабільність при оновленнях. У логічному поданні система поділяється на клієнтську частину, серверне середовище, базу даних та мережевий прошарок, що забезпечує взаємодію між компонентами. На рисунку 2.2 подано спрощену архітектурну схему веб-застосунку.

Обрана архітектура дозволяє досягти хорошого балансу між простотою реалізації, продуктивністю і можливістю масштабування. На відміну від готових CMS-платформ, таких як WordPress, Shopify або Payload CMS, реалізація з нуля на Flask надає повну свободу у визначенні структури застосунку, способів збереження даних, логіки авторизації та обробки запитів.

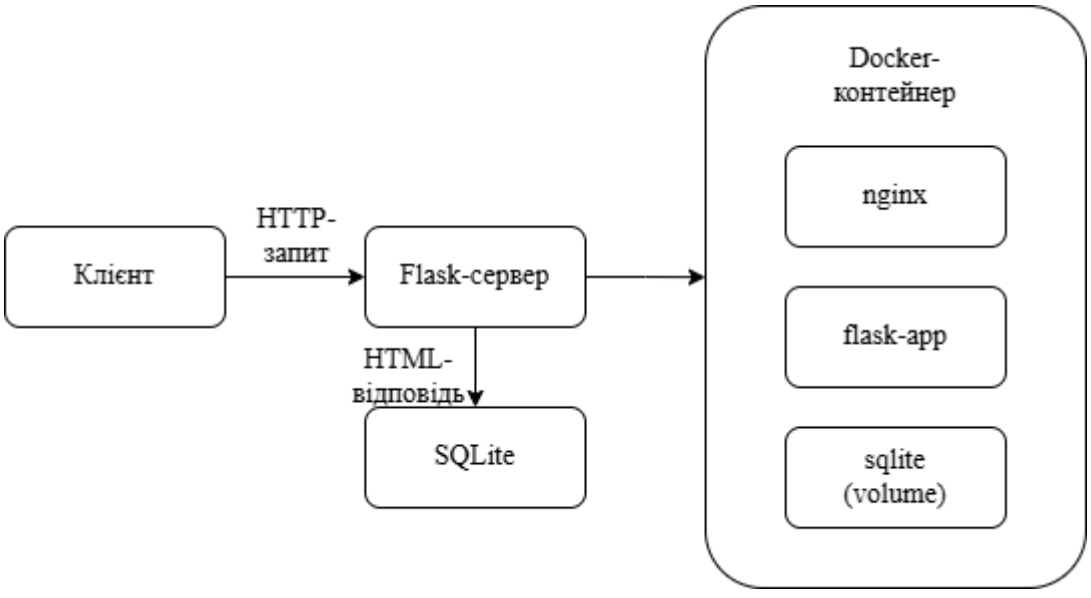

Рисунок 2.2 - Архітектура веб-застосунку інтернет-магазину комп’ютерної техніки

У таблиці 2.1 подано порівняльну характеристику основних архітектурних підходів до створення інтернет-магазинів.

З наведеного аналізу випливає, що обраний підхід — Flask + SQLite у контейнерному середовищі Docker — дозволяє досягнути високого рівня контролю над усіма етапами життєвого циклу застосунку. Він є оптимальним для цілей дипломної роботи, де необхідно не лише реалізувати функціонал інтернет-магазину, а й продемонструвати розуміння внутрішньої логіки та архітектурних принципів сучасної веб-розробки.

Таблиця 2.1 - Порівняння архітектурних підходів

Параметр	**Flask + SQLite (обраний варіант)**	**CMS (WordPress / Shopify)**	**SPA + Backend API (React + Node.js)**
Швидкість розробки	Висока	Дуже висока	Середня
Гнучкість кастомізації	Висока	Обмежена	Висока
Продуктивність	Висока для малих проєктів	Середня	Висока
Залежність від сторонніх рішень	Мінімальна	Висока	Середня
Безпека	Під контролем розробника	Залежить від плагінів	Потребує налаштування
Підходить для технічного магазину	Так	Частково (обмежені фільтри)	Так

2.3. Функціональна структура реалізованого веб-застосунку

Розроблений інтернет-магазин комп’ютерної техніки реалізований як класичний веб-застосунок на основі архітектури «клієнт-сервер». Це означає, що уся взаємодія з користувачем здійснюється через веб-інтерфейс (frontend), тоді як

обробка запитів, логіка перевірки доступу, операції з базою даних і формування відповідей виконуються на стороні сервера (backend).

Архітектура побудована на фреймворку Flask, що забезпечує обробку HTTP-запитів, маршрутизацію, роботу з шаблонами HTML і взаємодію з базою даних SQLite. На стороні клієнта застосовано стандартні технології HTML, CSS і JavaScript, що забезпечують інтуїтивно зрозумілий інтерфейс користувача. Оформлення виконано з урахуванням адаптивності для підтримки мобільних і десктопних пристроїв.

Застосунок логічно поділений на декілька модулів, кожен з яких виконує окрему функцію: реєстрація та авторизація користувачів, перегляд списку товарів, додавання товарів до кошика, оформлення замовлення, а також адміністративна частина для перегляду замовлень та оновлення асортименту. Для забезпечення безпеки паролі користувачів зберігаються у хешованому вигляді з використанням бібліотеки bcrypt, що унеможливлює доступ до справжніх значень навіть у разі компрометації бази.

Всі основні сторінки застосунку реалізовані з використанням шаблонів HTML, з'єднаних із серверною логікою за допомогою маршрутів Flask. Дані передаються між клієнтом і сервером у вигляді HTTP POST/GET запитів, що обробляються функціями-представленнями (views). Інформація зберігається в базі даних SQLite, яка організована у вигляді таблиць «users», «products», «orders», «cart» тощо. Кожна таблиця зв’язана ключами, що забезпечує референційну цілісність даних.

У якості платформи для розгортання обрано Docker, що дозволяє створити ізольоване серверне середовище. Усі залежності описано у файлі Dockerfile, а конфігурація компонентів — у docker-compose.yml, що спрощує запуск на будь-якому хості з підтримкою контейнеризації.

Загальна структура забезпечує швидке розширення функціоналу та гнучке масштабування. Як вказується у навчальному посібнику, «web-застосунки мають

гнучку архітектуру, яка дозволяє поступове розширення системи без переробки базової логіки» [5].

На рисунку 2.3 подано структурну модель основних функціональних блоків, реалізованих у застосунку.

Рисунок 2.3 - Функціональна структура веб-застосунку інтернет-магазину

2.4. Способи оплати інтернет-магазину

Хоча в межах реалізованого веб-застосунку функціонал оплати не був впроваджений повністю, архітектура системи передбачає можливість подальшої інтеграції з популярними платіжними сервісами. У сучасних інтернет-магазинах забезпечення безпечної й зручної оплати є ключовим етапом завершення покупки. Відсутність надійного платіжного модуля може суттєво знизити довіру користувачів, у той час як вдало реалізована система оплати здатна суттєво підвищити конверсію та задоволеність клієнтів.

Поширеною практикою в електронній комерції є використання сторонніх платіжних шлюзів, таких як LiqPay, Fondy, Stripe, Portmone. Ці сервіси дозволяють підключити форму оплати без зберігання платіжних даних на сервері магазину, що значно підвищує безпеку. Процес інтеграції передбачає обробку запиту з даними замовлення, переадресацію на сторінку оплати, отримання

відповіді про статус транзакції та оновлення запису в базі даних. Цикл такого процесу зображено на рисунку 2.4.

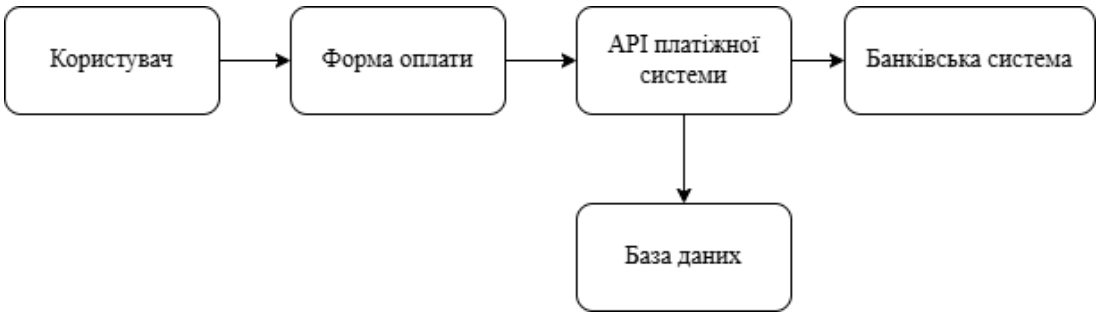

Рисунок 2.4 - Схема потенційної інтеграції з платіжною системою

На рисунку демонструється типовий механізм: користувач обирає спосіб оплати, після чого дані передаються через захищене з’єднання до платіжного API. Після проведення транзакції, платіжна система повертає відповідь, яку обробляє серверна частина сайту, оновлюючи статус відповідного замовлення в базі.

З урахуванням цього, для подальшого розширення функціоналу доцільно передбачити інтеграцію з платіжними системами, які підтримують API для Python, забезпечують тестовий режим (sandbox) та адаптацію під мобільні пристрої. Нижче подано порівняння трьох популярних сервісів, доступних на українському ринку.

Таблиця 2.2 - Порівняння популярних способів онлайн-оплати

Платіжна система	**Комісія для продавця**	**Підтримка Python API**	**Підтримка мобільних пристроїв**	**Можливість тестування**
LiqPay	~2.75%	Так	Так	Так
Stripe	~2.9% + $0.3	Так	Так	Так
Fondy	~2.5–3%	Так	Так	Так

У майбутньому достатньо буде реалізувати API-запит, зберігати токен транзакції та оновлювати статус замовлення після підтвердження сервера, що дозволить вивести сайт на рівень повноцінного комерційного рішення.

РОЗДІЛ 3.
ПРОЄКТУВАННЯ ІНТЕРНЕТ-МАГАЗИНУ ПРОДАЖУ КОМП'ЮТЕРНОЇ ТЕХНІКИ

3.1. Функціональне моделювання діяльності інтернет-магазину продажу комп'ютерної техніки

Процес розробки інформаційної системи, зокрема веб-застосунку для продажу комп’ютерної техніки, передбачає попереднє моделювання функціональної структури. Це дає змогу візуалізувати основні сценарії взаємодії користувача з системою, виявити ролі та їхні повноваження, а також структурувати логіку бізнес-процесів у межах інтернет-магазину.

Серед основних суб'єктів системи визначаються такі ролі, як гість (неавторизований користувач), зареєстрований клієнт та адміністратор. Гість має змогу переглядати асортимент, ознайомлюватися з описами товарів, користуватися пошуком і фільтрами. Зареєстрований користувач додатково отримує доступ до кошика, можливість створювати замовлення, зберігати історію покупок та редагувати особистий профіль. Адміністратор, своєю чергою, здійснює керування товарами, перегляд та обробку замовлень, модифікацію даних про товари та доступ до звітності.

Для візуалізації функціональної моделі системи розроблено діаграму варіантів використання (Use-Case), яка наведена на рисунку 3.1. У центрі діаграми зображено основні дії, які можуть виконувати користувачі системи. Зокрема, показано, як відбувається взаємодія з товарами, оформлення замовлення, доступ до профілю та адміністративна обробка замовлень.

У межах цього функціонального моделювання також важливим є аналіз логіки переходів між сторінками. У структурі інтерфейсу виділяються головна сторінка з каталогом товарів, сторінка авторизації/реєстрації, профіль

користувача, кошик та сторінка підтвердження замовлення. Кожен перехід між цими елементами має бути логічно обґрунтованим і зручним для користувача.

Рисунок 3.1 - Діаграма варіантів використання для інтернет-магазину комп’ютерної техніки

На рисунку 3.2 представлено фрагмент головної сторінки реалізованого інтерфейсу, який відображає категорії товарів, доступні для перегляду неавторизованим користувачем.

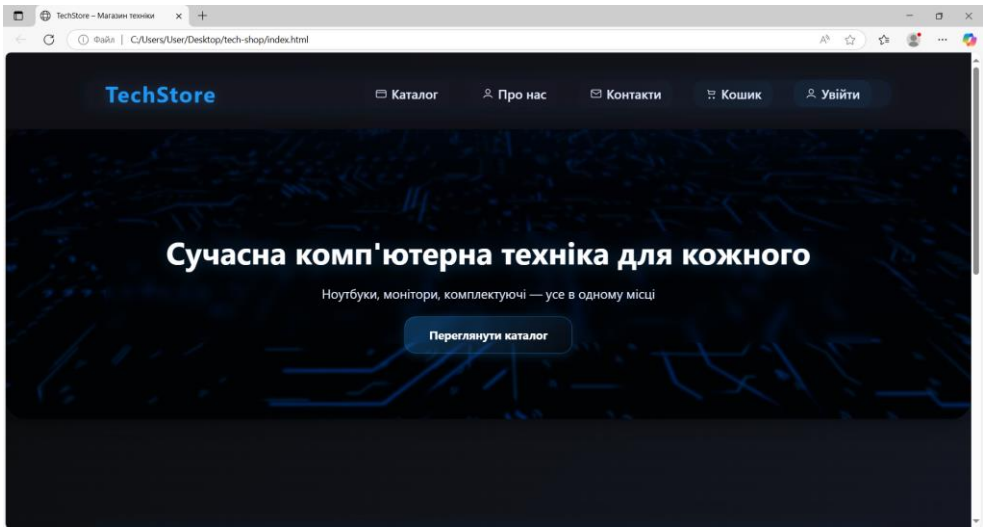

Рисунок 3.2 - Головна сторінка інтерфейсу інтернет-магазину

Функціональна модель дозволяє перевірити повноту реалізованих сценаріїв, виявити потенційні вузькі місця у логіці взаємодії, а також забезпечити подальше логічне проектування бази даних та серверної частини. З урахуванням цієї моделі, у наступному підрозділі буде розглянуто конкретну реалізацію структури бази даних із використанням SQLite.

Крім базових сценаріїв, до функціональної моделі також можна включити допоміжні дії, які не завжди знаходяться на поверхні взаємодії, але є важливими для загальної зручності користувача. Серед них — перегляд інформаційних сторінок (наприклад, «Про нас», «Доставка і оплата»), підписка на розсилку новин або контактна форма для звернень. Ці елементи не обов'язково мають прямий вплив на процес купівлі, але формують загальне враження від сервісу та підвищують лояльність користувачів.

Окремо варто підкреслити роль захисту інформації у функціональній моделі. Система передбачає авторизацію з використанням зашифрованих паролів, обмеження доступу до адміністративних дій лише для користувачів з відповідними правами, а також захист від некоректного введення даних. Усі ці моменти враховувались під час побудови сценаріїв взаємодії і проєктування бізнес-логіки.

Також важливою складовою є підтримка коректної навігації та відсутності «мертвих» сторінок. Перевірка логіки переходів дозволяє забезпечити, щоб користувач після завершення замовлення міг без труднощів повернутися на головну сторінку або переглянути інші товари. Це підвищує загальну зручність використання веб-застосунку та знижує кількість відмов.

Функціональне моделювання на цьому етапі дало змогу не лише описати структуру взаємодії, але й закласти основу для перевірки реалізації: після завершення програмування кожен сценарій міг бути перевірений на відповідність моделі. Такий підхід значно зменшує ризик логічних помилок, дозволяє краще планувати розвиток системи в майбутньому та забезпечує високу якість з точки зору користувацького досвіду.

Ефективне функціонування веб-застосунку інтернет-магазину безпосередньо залежить від правильної структури бази даних, яка забезпечує збереження, доступ і логічне пов’язання інформації між користувачами, товарами, замовленнями та іншими об’єктами системи. У цьому проєкті для реалізації зберігання даних було обрано систему керування базами даних SQLite, яка відзначається простотою, легкістю в налаштуванні та високою швидкістю роботи в умовах помірного навантаження.

Розроблена база даних відповідає структурі, необхідній для підтримки функціонування онлайн-магазину техніки. Її логіка побудована відповідно до нормалізованої моделі, що дозволяє уникнути надмірного дублювання інформації та забезпечити референційну цілісність. Загальна структура передбачає п’ять основних таблиць: users, products, categories, orders та order_items.

Таблиця users зберігає інформацію про зареєстрованих користувачів, включаючи їх логін, електронну пошту, хешований пароль та базову контактну інформацію. Таблиця products містить дані про товари: назву, опис, категорію, ціну, залишок на складі та шлях до зображення. Для забезпечення зручності фільтрації реалізовано таблицю categories, яка дозволяє групувати товари за типами, наприклад, "Ноутбуки", "Монітори", "Процесори" тощо. Таблиця orders фіксує загальні відомості про замовлення: користувача, дату, статус, а таблиця order_items реалізує зв’язок між замовленням та товарами, дозволяючи зберігати кількість кожної позиції у межах одного замовлення.

На рисунку 3.3 наведено схему бази даних, що демонструє структуру таблиць та основні зв’язки між ними. Така організація дає змогу ефективно обробляти замовлення, виводити історію покупок, керувати каталогом і формувати динамічний вміст для сторінок магазину.

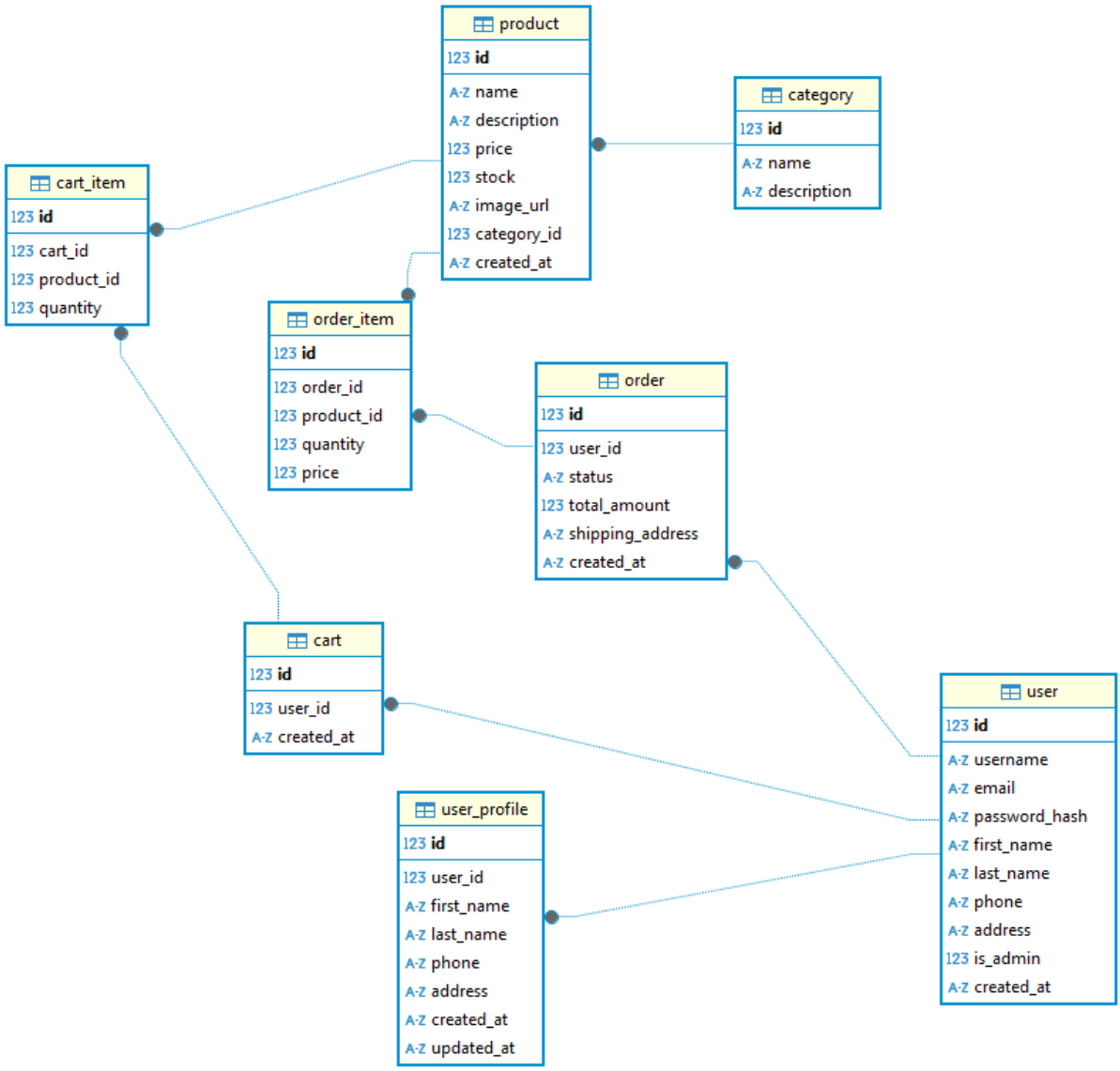

Рисунок 3.3 - Схема бази даних інтернет-магазину на основі SQLite

Структура бази побудована з урахуванням можливого розширення: у подальшому система дозволяє додати таблиці зі способами доставки, модулями оплати, відгуками користувачів або адміністративною аналітикою. Усі таблиці підтримують первинні та зовнішні ключі, що дозволяє забезпечити зв’язність і цілісність збережених даних при мінімальному обсязі запитів до бази.

Оскільки застосунок використовує Flask як backend-фреймворк, взаємодія з базою даних реалізована за допомогою бібліотеки sqlite3, що інтегрується у

Python стандартно. SQL-запити структуровано безпосередньо в коді обробників, а створення таблиць виконується окремими скриптами ініціалізації під час першого запуску. Задля безпеки обробки даних застосовано параметризовані запити, що запобігає ін’єкціям та несанкціонованому втручанню у сховище.

Усі функціональні елементи, пов’язані з базою даних, реалізовано таким чином, щоб забезпечити стабільність роботи під час використання одночасно декількома користувачами. Це дозволяє отримати повноцінну модель інформаційного обміну в системі інтернет-торгівлі, придатну як для навчального, так і для прикладного використання.

3.2. Побудова UML-діаграми класів інтернет-магазину

Під час проєктування програмного забезпечення особливу увагу приділяють формалізації об'єктної структури системи. Одним із найпоширеніших методів візуалізації взаємозв’язків між сутностями застосунку є побудова UML-діаграми класів. У межах цього підпункту така діаграма використовується для зображення основних компонентів інтернет-магазину комп’ютерної техніки, реалізованого з використанням фреймворку Flask і реляційної бази даних SQLite.

UML-діаграма дозволяє представити логічну структуру застосунку, а саме — класи, їх атрибути та методи, а також зв’язки між об’єктами. У контексті веб-застосунку інтернет-магазину ключовими сутностями є користувач, товар, замовлення, категорія та позиція замовлення. Кожен із цих об'єктів має свій набір полів і визначену роль у бізнес-логіці системи.

Клас User включає поля, які відповідають збереженим у базі даних: унікальний ідентифікатор, логін, email, хешований пароль і роль (наприклад, користувач або адміністратор). Клас Product містить атрибути, що описують товар: назву, опис, ціну, категорію, наявність та зображення. Для групування товарів створено клас Category, який дозволяє об'єднувати продукти у логічні розділи каталогу. Клас Order відповідає за загальну інформацію про замовлення:

дату, користувача, статус і суму. Клас OrderItem використовується для збереження складу кожного замовлення, пов’язуючи товар із кількістю в межах одного замовлення.

На рисунку 3.4 представлено UML-діаграму класів, що описує логіку взаємозв’язків між основними компонентами системи.

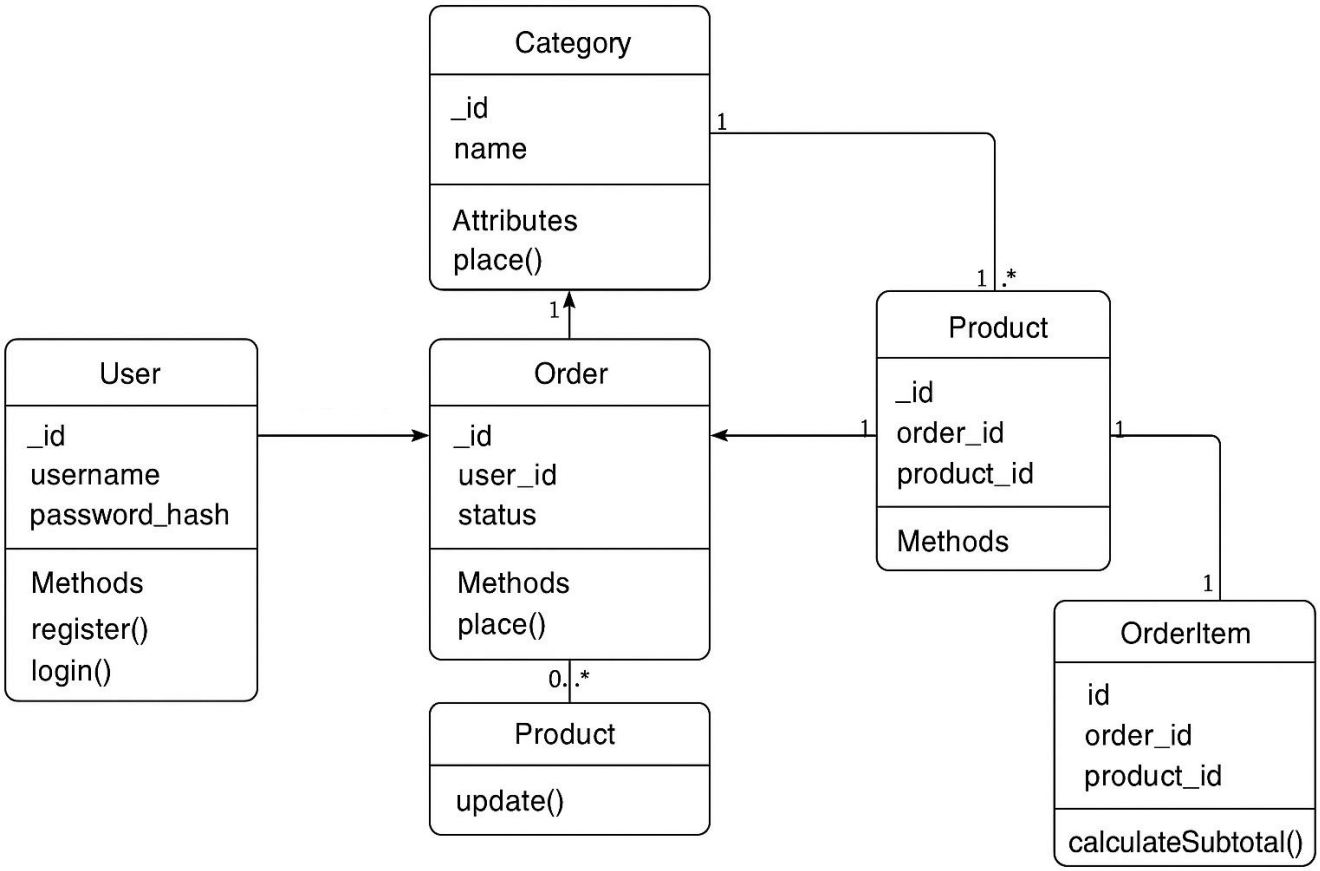

Рисунок 3.4 - UML-діаграма класів веб-застосунку інтернет-магазину

Класи мають чіткі зв’язки: зокрема, між User та Order встановлено співвідношення «один до багатьох», тобто кожен користувач може створити декілька замовлень. Зв’язок між Order та OrderItem також є «один до багатьох», адже замовлення може містити кілька товарних позицій. Кожна позиція у свою чергу пов’язана з конкретним екземпляром класу Product. Зв’язок між Product і Category забезпечує групування товарів, що дозволяє реалізувати структуру каталогу з поділом на підрозділи.

UML-діаграма класів дає змогу не лише структурувати систему, а й спростити процес розробки, тестування та подальшого масштабування. Вона служить зрозумілою формою для технічної документації, дозволяє швидко орієнтуватися в логіці даних та забезпечує відповідність між функціональними вимогами й програмною реалізацією.

3.3. Розробка інтернет-магазину для продажу комп'ютерної техніки

Процес розробки веб-застосунку інтернет-магазину складався з кількох ключових етапів: створення структури сайту, реалізації логіки взаємодії з базою даних, організації маршрутизації, підключення шаблонів і стилів, а також розробки інтерфейсів для користувача та адміністратора. Для реалізації застосунку було використано фреймворк Flask, який забезпечив обробку HTTP-запитів, генерацію HTML-сторінок, керування сесіями та зручну інтеграцію з SQLite. Розробка виконувалась у середовищі Python без використання ORM, що забезпечило повний контроль над SQL-запитами та спростило структуру проєкту.

Інтерфейс сайту реалізовано за допомогою HTML та CSS, з використанням базових можливостей JavaScript для обробки взаємодії з формами та елементами сторінки. Головна сторінка містить список товарів, згрупованих за категоріями, з можливістю фільтрації за назвою, ціною та іншими атрибутами.(рис. 3.5)

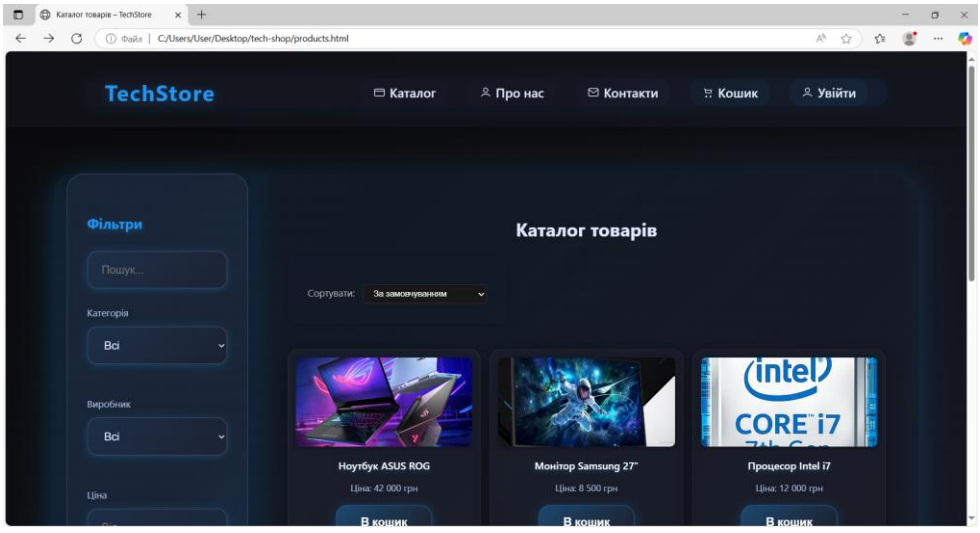

Рисунок 3.5 - Головна сторінка з вітриною товарів

Користувачі мають змогу переглядати повну інформацію про товар, включаючи розширений опис і технічні характеристики. Відображення відбувається у вигляді окремої сторінки товару, що генерується динамічно відповідно до вибраного ID продукту.(рис. 3.6)

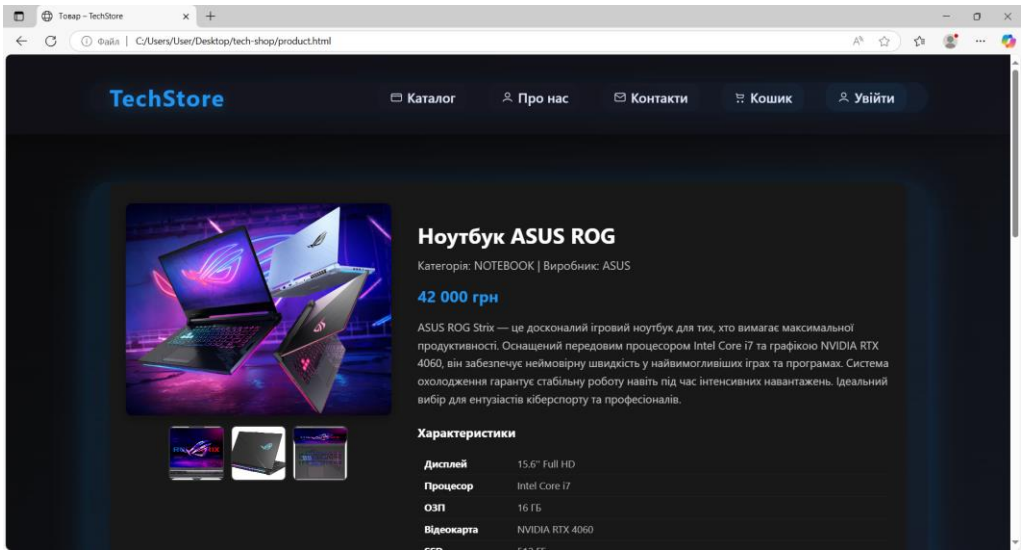

Рисунок 3.6 - Сторінка перегляду окремого товару

Функція кошика реалізована на основі сесійного сховища користувача. При додаванні товару до кошика інформація записується у словник, який зберігається в межах поточної сесії. Перехід до кошика дозволяє переглянути обрані товари, змінити кількість, видалити позиції або перейти до оформлення замовлення.(рис. 3.7)

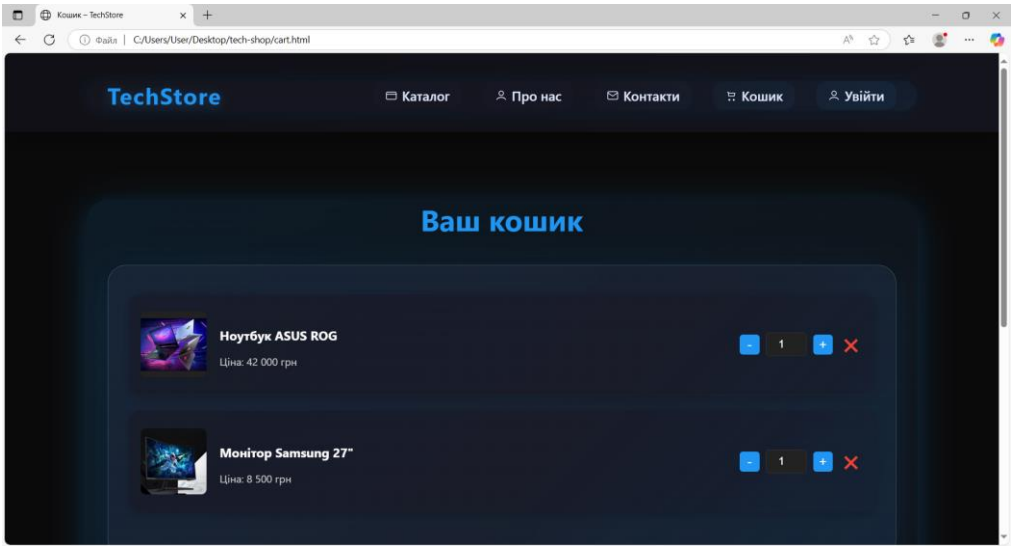

Рисунок 3.7 - Вміст кошика користувача

Форма оформлення замовлення дає змогу зареєстрованому користувачеві підтвердити покупку. На цьому етапі дані передаються серверу, де формується запис замовлення в базі даних із поточним станом і пов’язаними товарами. Реалізація враховує обробку помилок та виведення повідомлення про успішне оформлення.(рис. 3.8)

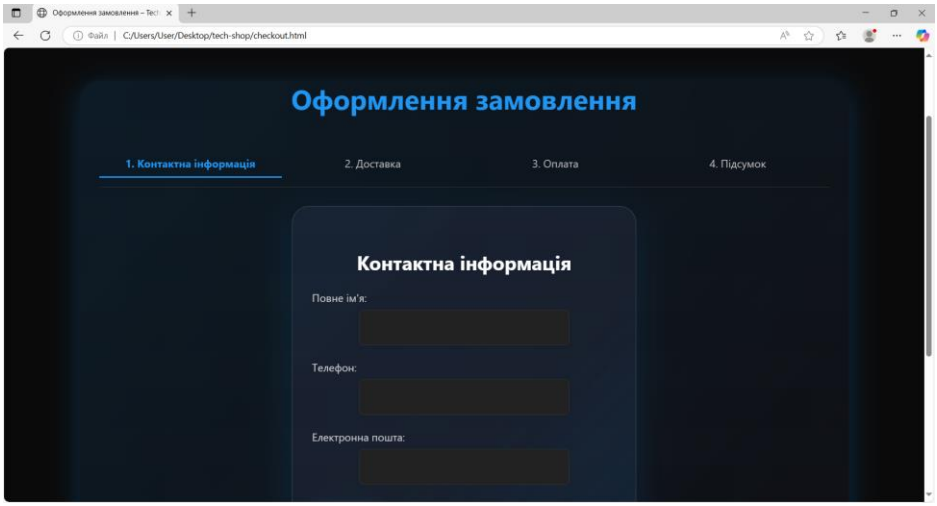

Рисунок 3.8 - Форма оформлення замовлення

Усі паролі зберігаються у хешованому вигляді з використанням бібліотеки bcrypt. Реалізовано повноцінну систему реєстрації, авторизації та виходу з акаунту. Зареєстровані користувачі мають змогу переглядати свою історію покупок, змінювати персональні дані та повторно переглядати оформлені замовлення.(рис. 3.9)

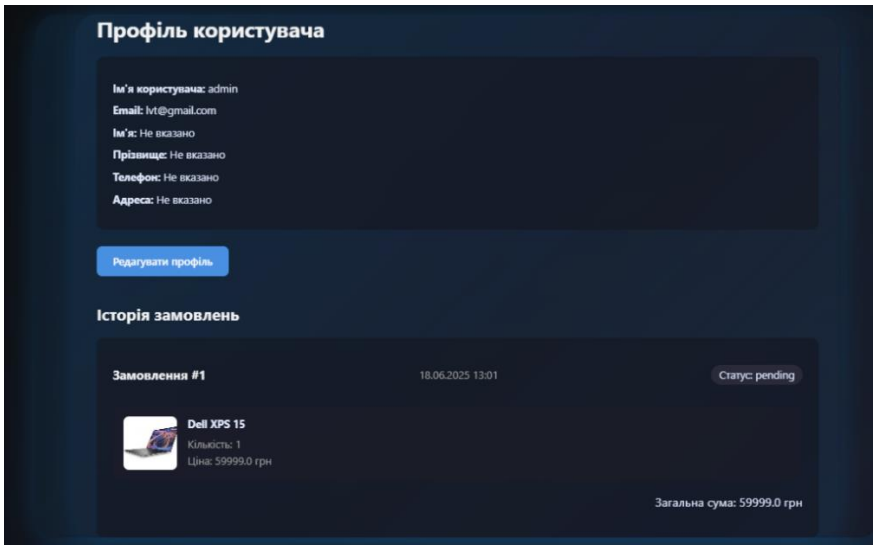

Рисунок 3.9 - Сторінка особистого кабінету користувача

Адміністративна частина реалізована у вигляді окремих сторінок, доступ до яких має лише користувач із роллю "admin". Інтерфейс адміністратора дозволяє додавати нові товари, редагувати наявні, видаляти позиції, а також переглядати список усіх замовлень і змінювати їх статус.(рис. 3.10)

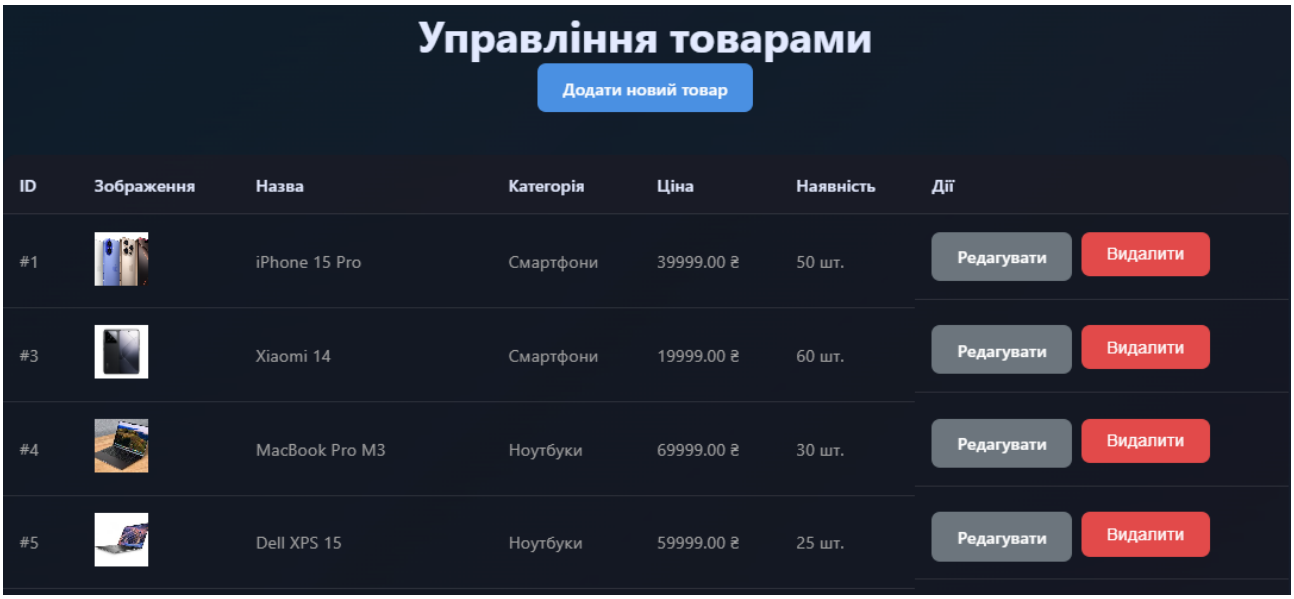

Рисунок 3.10 - Інтерфейс адміністратора для керування товарами

Загальна структура застосунку побудована з чітким розподілом на модулі, шаблони, статичні ресурси та базу даних. Оскільки код не вставляється безпосередньо в основну частину дипломної роботи, його повна реалізація представлена в додатку А. Зазначена логіка дозволила створити стабільний і гнучкий веб-застосунок, який може масштабуватись за рахунок додавання нових функцій або інтеграції зовнішніх сервісів.

3.3.1. Розробка інтерфейсу та функціоналу адміністратора для керування товарним каталогом

Ефективне функціонування інтернет-магазину неможливе без зручної та безпечної системи адміністрування, яка дозволяє керувати товарним асортиментом, актуальністю інформації, зображеннями, категоріями та замовленнями. У межах цього дипломного проєкту було реалізовано окрему адміністративну панель, яка надає обмежений доступ лише уповноваженим

особам. Така ізоляція дозволяє уникнути втручання з боку користувачів і забезпечує цілісність даних.

Адміністративна панель реалізована як окремий розділ сайту, доступ до якого відкривається лише після автентифікації адміністратора. Система автентифікації використовує зашифроване збереження паролів за допомогою bcrypt, що відповідає сучасним вимогам безпеки. Інтерфейс панелі побудований за принципами максимальної простоти й ефективності: адмін бачить перелік наявних товарів, може відфільтрувати їх за категоріями або ключовими словами, а також переходити до редагування, додавання або видалення записів.

На головній сторінці панелі відображається зведена таблиця з усіма товарами, включаючи назву, категорію, ціну, короткий опис, кількість на складі та статус видимості. Кожен запис супроводжується кнопками Редагувати та Видалити, що дозволяє миттєво вносити зміни до товарного ряду або вилучати застарілі позиції. Зовнішній вигляд загального списку товарів адміністратора було розглянуто раніше (рис. 3.10), тому нижче наведено приклади реалізації інших ключових функцій системи керування контентом.

Однією з ключових функцій є додавання нового товару. Для цього передбачена окрема форма, яка дозволяє адміністратору вводити всі необхідні параметри: назву товару, короткий та повний опис, вибрати категорію зі списку, вказати кількість у наявності, ціну, а також завантажити зображення товару. Завантажені фото автоматично зберігаються в окрему директорію, а їх назви зберігаються у базі даних у вигляді шляху до файлу.(рис. 3.12).

Після заповнення форми та натискання кнопки Додати товар, відбувається валідація даних: перевірка порожніх полів, типів значень, правильності завантаження зображення. У разі успішного проходження перевірок, новий товар додається до бази даних і відображається в загальному списку. Для зручності користувача після додавання виводиться повідомлення про успішне завершення операції.

Окремо реалізовано функцію редагування наявного товару. Адміністратор може змінити будь-яке поле товару, включаючи фото. Якщо фото не завантажується заново, залишається попереднє. Це дає можливість швидко оновлювати цінову політику, доступність або опис без потреби повторного додавання всього запису.(рис. 3.13)

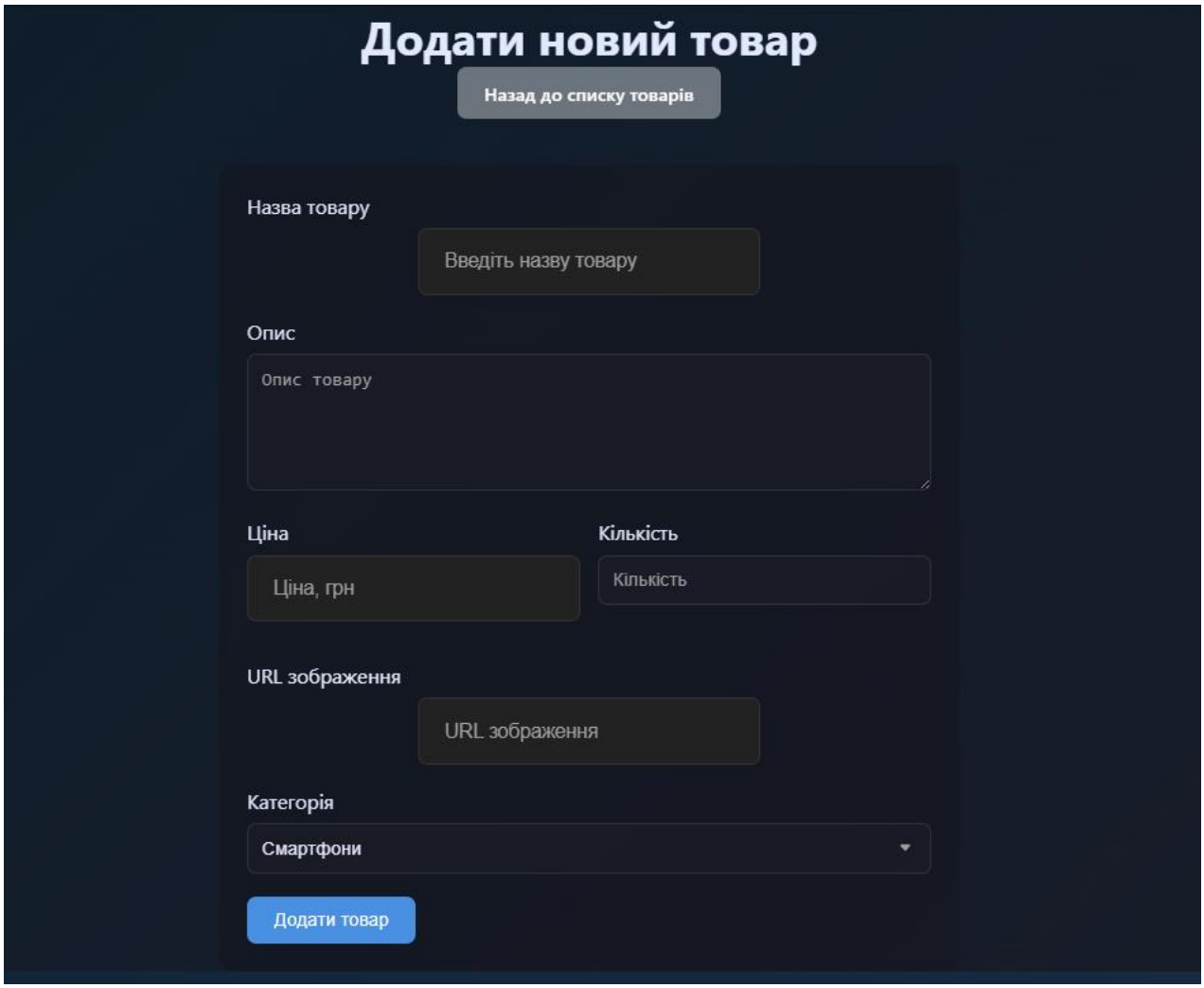

Рисунок 3.12 - Форма додавання нового товару в адмінпанелі

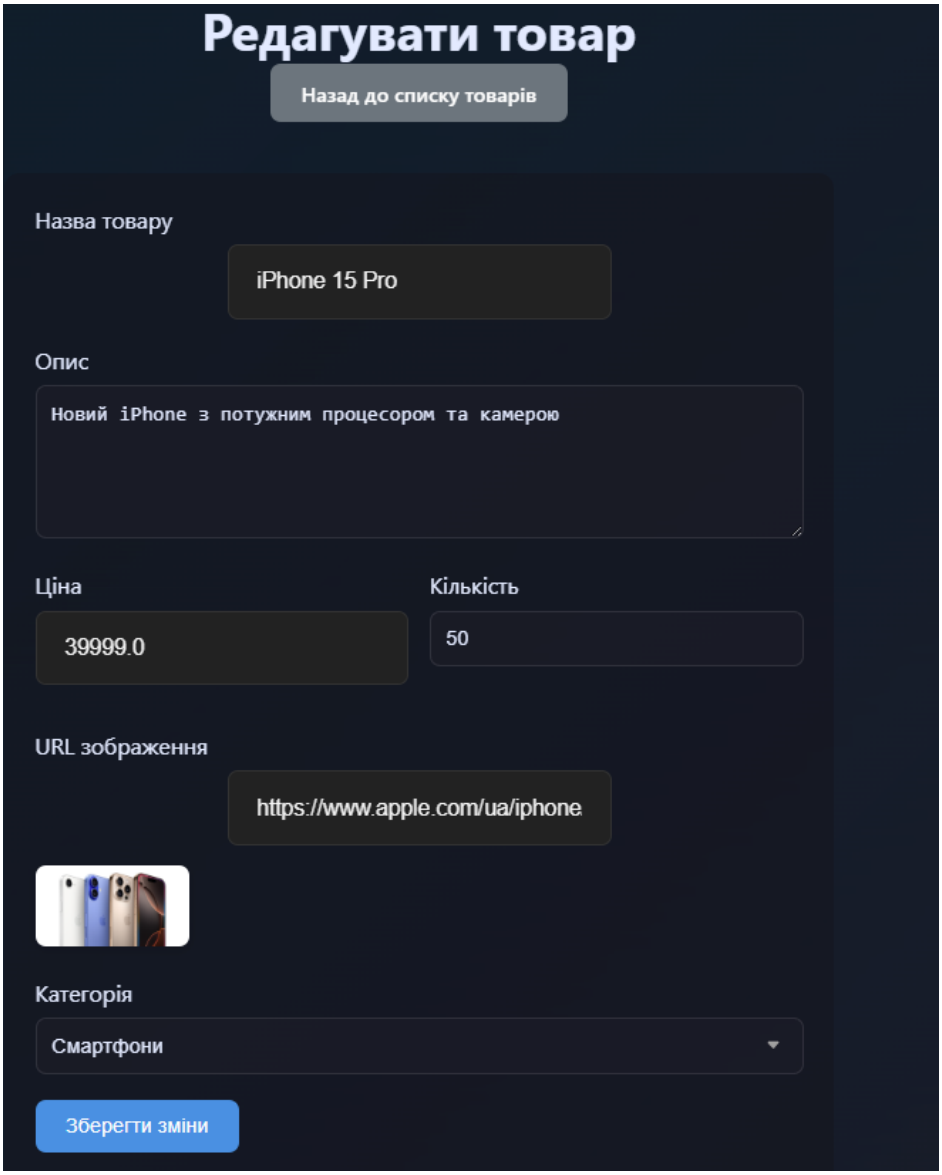

Рисунок 3.13 - Інтерфейс редагування товару

Ще однією зручністю є наявність пошуку по товарах. Адміністратор може за кілька секунд знайти потрібну позицію за частиною назви, що особливо важливо при великому асортименті. Це реалізовано через AJAX-запити до бази даних із подальшим оновленням таблиці без перезавантаження сторінки.

У структурі бази даних передбачено поле для статусу публікації, що дозволяє тимчасово приховувати товари, які більше не продаються або знаходяться на оновленні. Це зручно у випадках сезонних пропозицій, або коли товар тимчасово відсутній у продажу, але його не бажано остаточно вилучати.

Завдяки впровадженню повноцінної адмінпанелі, управління контентом магазину стало централізованим і контрольованим. Такий підхід спрощує подальше розширення функціональності, наприклад, додавання модулів управління замовленнями, користувачами або аналітикою продажів.

Функціональність адміністрування в даній системі є одним із ключових компонентів, що дозволяє підтримувати актуальність та цілісність товарного каталогу. Її реалізація стала можливою завдяки використанню Flask як серверного фреймворку, HTML/CSS для верстки, а також SQLite для зберігання даних.

3.3.2. Розробка та опис коду над створенням персональної інформації користувача

Система обліку персональної інформації користувача є одним із ключових функціональних компонентів інтернет-магазину. Вона відповідає за зберігання та безпечну обробку таких даних, як ім’я користувача, електронна пошта, пароль, історія замовлень, контактна інформація для доставки та інші параметри, які використовуються для персоналізації взаємодії з системою.

Для забезпечення захищеної обробки інформації про користувача реалізовано механізм реєстрації з обов’язковим введенням унікальної електронної пошти та пароля. На етапі реєстрації пароль шифрується за допомогою бібліотеки bcrypt, що забезпечує його надійне хешування перед збереженням у базу даних. Усі введені користувачем дані перевіряються на коректність та унікальність — наприклад, система не дозволяє зареєструвати два акаунти з однією й тією ж адресою електронної пошти.(рис. 3.14)

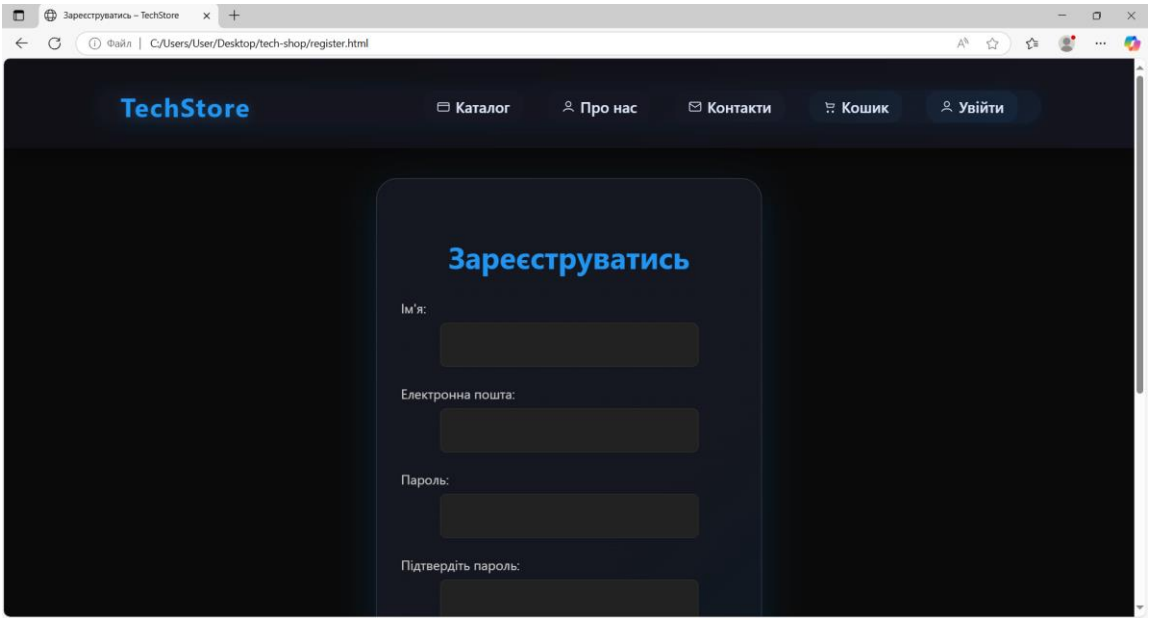

Рисунок 3.14 - Сторінка реєстрації користувача

Після успішної реєстрації користувач отримує можливість увійти до особистого кабінету. Під час авторизації система виконує перевірку введеного пароля, порівнюючи його хеш із тим, що збережений у базі. У разі збігу генерується сесія, яка зберігає ID користувача, його ім’я та роль, що дає змогу реалізувати гнучке керування доступом до функціональних елементів сайту.(рис. 3.15)

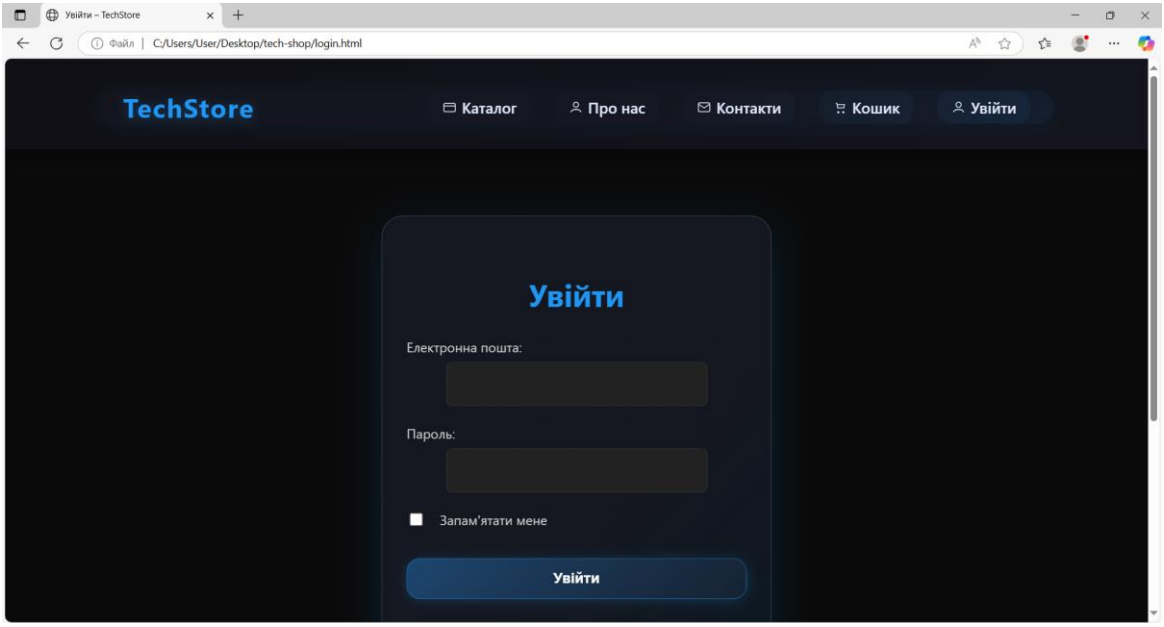

Рисунок 3.15 - Форма входу до акаунту користувача

Особистий кабінет користувача дозволяє змінювати деякі персональні дані (наприклад, ім’я, адресу доставки), а також переглядати історію оформлених замовлень. Для цього передбачено окремі маршрути, які взаємодіють із базою даних та повертають відповідні HTML-сторінки. Всі зміни відбуваються лише після повторного введення пароля, що забезпечує базовий рівень захисту навіть у випадку компрометації сесії.

З технічної точки зору, реалізація персонального профілю базується на окремій таблиці users у базі даних, де зберігаються атрибути користувача. Усі операції з оновленням інформації виконуються через серверну логіку Flask, із ретельною перевіркою вхідних даних, щоб уникнути SQL-ін’єкцій чи інших атак.

Повна реалізація логіки створення, збереження, оновлення та відображення персональних даних користувача представлена у додатку А. Завдяки простій, але надійній структурі реалізації система забезпечує базову безпеку та зручність для користувача, а також є готовою до масштабування.

3.4. Результат ручного тестування розробленого інтернет-магазину

Після завершення основного етапу розробки було проведено ручне тестування інтернет-магазину з метою перевірки коректності реалізації функціоналу, стійкості до помилок та відповідності очікуваній поведінці системи. Тестування відбувалося в локальному середовищі за допомогою браузерів Google Chrome та Mozilla Firefox. Особлива увага приділялась найважливішим сценаріям взаємодії користувача із системою, таким як реєстрація, авторизація, додавання товарів у кошик, оформлення замовлення, навігація сторінками, відображення контенту й обробка помилок.

Процес перевірки охоплював увесь шлях користувача — від створення облікового запису до завершення покупки. При реєстрації з валідними даними система створювала новий обліковий запис і переспрямовувала на сторінку входу. У разі помилкового введення email або занадто короткого пароля з’являлось інформативне повідомлення про помилку, що дозволяло виправити

введені дані. Авторизація працювала стабільно, і після входу користувач отримував доступ до функцій персонального кабінету.

Сторінка з товарами коректно відображала всі позиції бази даних, включно з фільтрацією за категоріями. Після відкриття сторінки окремого товару користувач міг додати його до кошика. У кошику підтримувалось збільшення або зменшення кількості товарів, а також обраховувалась підсумкова вартість. Оформлення замовлення супроводжувалось формою для введення контактної інформації, після чого відображалось повідомлення про успішне завершення операції. Усі замовлення зберігались у базі даних і надалі відображались у відповідному модулі.

Сайт коректно працював при переході між сторінками, не втрачав сесію після авторизації, а інтерфейс залишався зрозумілим та логічно побудованим. Завдяки проведенню детального тестування вдалося виявити та усунути декілька дрібних помилок, пов’язаних із стилізацією форм та відсутністю валідації в деяких полях. Це суттєво підвищило загальну якість проєкту.

Для наочного відображення результатів тестування в таблиці нижче наведено опис основних перевірених сценаріїв:

Таблиця 3.1 - Результати ручного тестування основних функцій

№	**Сценарій**	**Очікуваний результат**	**Фактичний результат**	**Статус**
1	2	3	4	5
1	Реєстрація з валідними даними	Обліковий запис створено	Обліковий запис створено	Успішно
2	Некоректний email	Повідомлення про помилку	Повідомлення відображається	Успішно
3	Авторизація з правильним логіном і паролем	Вхід у систему	Вхід виконано	Успішно

Продовження таблиці 3.1

1	2	3	4	5
4	Додавання товару в кошик	Товар з’являється у кошику	Товар додано	Успішно
5	Зміна кількості товарів	Сума змінюється відповідно	Оновлення працює	Успішно
6	Оформлення замовлення	Повідомлення про успішне замовлення	Повідомлення з’являється	Успішно
7	Видалення товару з кошика	Товар зникає з кошика	Видалення працює	Успішно
8	Збереження замовлення в базі	Дані зберігаються	Дані присутні в базі	Успішно
9	Переходи між сторінками	Усі посилання працюють	Переходи коректні	Успішно
10	Збереження сесії після авторизації	Користувач залишається авторизованим	Сесія зберігається	Успішно

Тестування проводилося в середовищі розробника на базі операційної системи Windows 10 із використанням локального сервера, розгорнутого за допомогою стандартного засобу запуску Flask. Версія Python становила 3.11, а для збереження даних використовувалась вбудована база даних SQLite, що дозволяло оперативно перевіряти зміни без потреби у складному конфігуруванні зовнішнього сервера. Після завершення кожної сесії тестування база даних очищувалася вручну, щоб уникнути накладень та забезпечити чистоту результатів.

Особливу увагу було приділено стійкості системи до некоректних дій користувача. Було змодельовано ситуації, коли вводились некоректні дані: пусті

поля, неправильні email-адреси, короткі паролі, спроби авторизації з неіснуючим логіном тощо. Система коректно реагувала на подібні виклики, повертаючи зрозумілі повідомлення про помилки та не допускаючи небажаних дій.

Також було здійснено перевірку функціонування інтерфейсу на різних пристроях. Зокрема, сайт відкривався на смартфоні з операційною системою Android та планшеті з iOS. У всіх випадках верстка залишалася адаптивною, навігація зручною, а всі інтерактивні елементи працювали належним чином. Було підтверджено, що навіть за умови змінення розмірів екрана основні функції (реєстрація, авторизація, додавання товарів до кошика) залишаються доступними без втрати юзабіліті.

Адміністративна частина веб-застосунку також була піддана ретельному тестуванню. Було перевірено сценарії додавання нового товару, його редагування, видалення та перевірка відображення змін на основній сторінці користувача. Усі дії з боку адміністратора миттєво оновлювались у базі даних і відображались без потреби перезавантаження сервера. У випадках, коли деякі поля залишались порожніми або заповнювались некоректно, система повідомляла про помилку і не зберігала дані, що свідчить про ефективну реалізацію базової валідації.

Додатково було проаналізовано швидкість завантаження сторінок. Час відкриття головної сторінки та картки товару становив у середньому менше 1 секунди при тестуванні в Google Chrome, що є позитивним показником для локального середовища. Це стало можливим завдяки використанню оптимізованих запитів до бази даних та мінімізації обсягу переданих зображень.

У результаті ручного тестування не було виявлено критичних помилок або збоїв у логіці роботи інтернет-магазину. Усі основні сценарії взаємодії були успішно перевірені, а функціональність системи визнано стабільною, що дозволяє в подальшому впроваджувати продукт на серверну інфраструктуру для повноцінного використання.

Таким чином, тестування показало повну функціональну готовність проєкту до використання. Всі основні компоненти працюють стабільно, система адекватно реагує на типові сценарії поведінки користувача, а виявлені недоліки були вчасно усунені.

3.5. Розгортання інтернет-магазину в Docker-середовищі

Після завершення розробки веб-застосунку важливо забезпечити можливість його стабільного, швидкого та передбачуваного запуску незалежно від середовища. У цьому контексті технологія Docker стала ключовим інструментом, що дозволяє створити ізольоване середовище з усіма необхідними компонентами — веб-сервером, інтерпретатором Python, бібліотеками та залежностями. Такий підхід значно спрощує розгортання, оскільки забезпечує однакову поведінку застосунку в будь-якій операційній системі. Docker також вирішує проблему конфліктів між версіями програмного забезпечення, що виникає при локальній установці залежностей, адже всі вони інкапсульовані в контейнері.

Проєкт інтернет-магазину реалізований як серверний застосунок на Flask, що взаємодіє з базою даних SQLite. Для запуску було створено файл Dockerfile, який визначає інструкції для побудови контейнерного образу, а також конфігураційний файл docker-compose.yml, який описує, як саме має запускатися контейнер. У Dockerfile передбачено встановлення усіх необхідних бібліотек, налаштування порту, на якому запускається Flask-додаток, і копіювання коду в контейнер. У docker-compose.yml прописано, що потрібно запускати саме цей образ, а також зазначено порт, що перенаправляється на локальний — зазвичай це порт 5000.

Завдяки docker-compose, розгортання системи стало максимально простим і зводиться до виконання лише однієї команди. Це значно знижує поріг входу для нових учасників команди, дозволяє безпечно проводити тести, а також пришвидшує доставку нових версій продукту. Особливо зручним є те, що після

розгортання веб-сервер стає доступним одразу через браузер — без необхідності додаткових налаштувань системи.(рис.3.16).

Після запуску контейнера веб-застосунок запускається автоматично, і його можна переглянути в браузері, звернувшись до localhost:5000. Після завершення роботи можна легко зупинити контейнер командою docker-compose down, що робить цей підхід не тільки ефективним, але й безпечним для системи розробника.

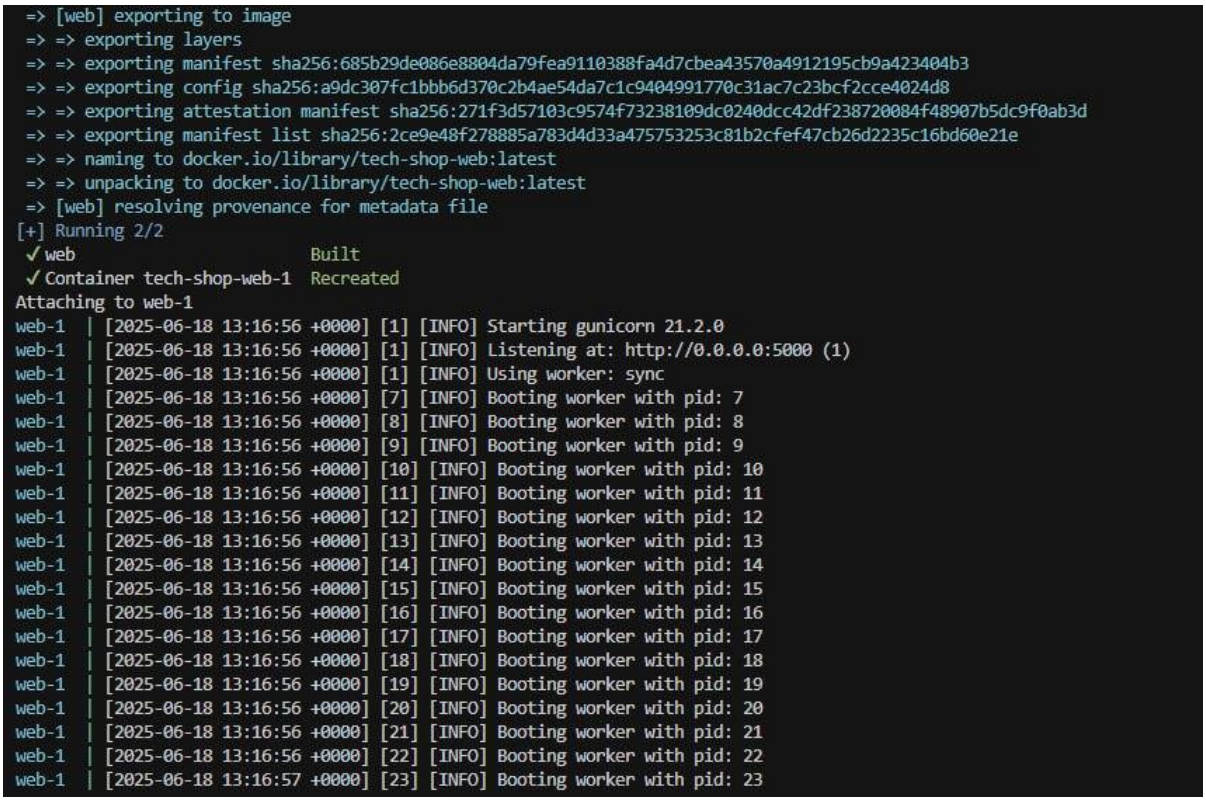

Рисунок 3.16 - Консольне вікно під час запуску інтернет-магазину

Для моніторингу стану контейнера використовувався зручний інструмент Docker Desktop. У ньому наочно відображається активний стан запущеного контейнера, зокрема його назва, статус, час безперервної роботи, споживання ресурсів та відкриті порти. Завдяки цьому інструменту можна оперативно перезапускати чи зупиняти застосунок, переглядати журнали логів та отримувати технічну інформацію, необхідну для супроводу.(рис. 3.17).

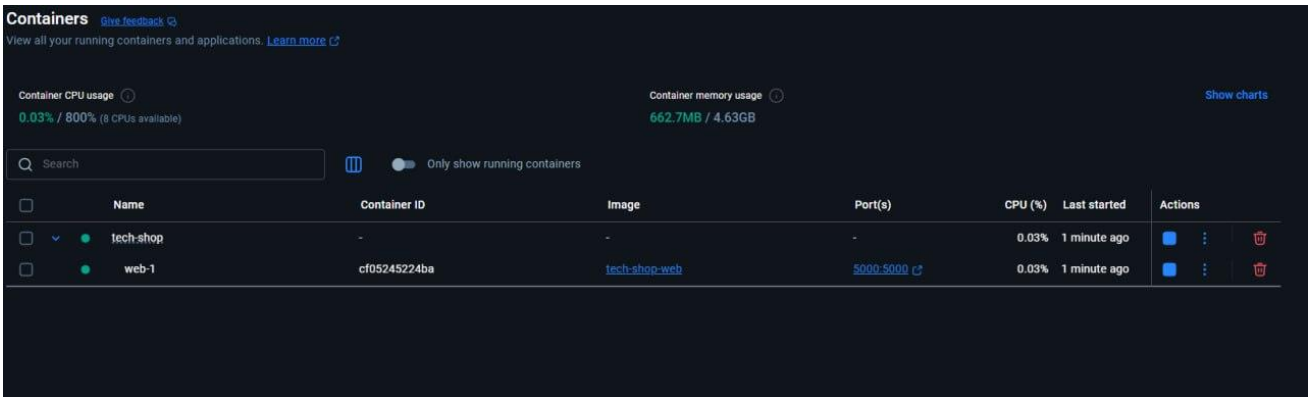

Рисунок 3.17 - Вигляд активного контейнера інтернет-магазину в Docker Desktop

Застосування Docker значно спрощує процес перенесення застосунку на інші машини або сервери, що робить його особливо зручним для розробки, тестування та впровадження. Крім того, контейнеризація дозволяє мінімізувати вплив зовнішніх чинників на роботу програми та швидко відновлювати працездатність у разі збоїв. У перспективі цей підхід відкриває можливість інтеграції з CI/CD-системами для автоматичного оновлення версій застосунку, що покращує гнучкість та надійність розгортання.

РОЗДІЛ 4.
ОХОРОНА ПРАЦІ

4.1. Поняття й завдання техніки безпеки

У сучасних умовах інтенсивного використання комп’ютерних технологій питання техніки безпеки набуває особливої актуальності. Техніка безпеки — це сукупність організаційних і технічних заходів, що мають на меті запобігання професійним захворюванням, виробничим травмам і створення комфортних умов праці. Попри уявну безпеку роботи з комп’ютером, фахівці галузі інформаційних технологій систематично піддаються впливу низки шкідливих чинників. Це зокрема тривала статична поза, зорове навантаження, психоемоційне напруження, а також вплив електромагнітного випромінювання й шуму.

Техніка безпеки є важливою складовою охорони праці, спрямованою на запобігання травмам, аваріям та іншим небажаним наслідкам під час виконання виробничих завдань. У сфері інформаційних технологій, де працівники здебільшого проводять багато часу за комп’ютерами, питання безпеки стосується не лише фізичної цілісності обладнання, а й захисту здоров’я користувача та створення сприятливого середовища праці.

Робоче місце спеціаліста ІТ-профілю характеризується високою інтенсивністю розумового навантаження, статичним положенням тіла, постійним впливом штучного освітлення та електромагнітного випромінювання. У зв’язку з цим особливого значення набувають дотримання правил ергономіки, організація режиму праці й відпочинку, а також дотримання відповідних санітарних вимог. Згідно з чинними нормами, «для профілактики впливу шкідливих факторів, що виникають при роботі з відео-дисплейними терміналами, необхідно дотримуватися відповідних санітарних норм» [3].

Техніка безпеки у сфері ІТ включає дотримання вимог до освітлення, температурного режиму, вологості, розміщення обладнання та засобів індивідуального захисту. До завдань техніки безпеки також належить своєчасне інформування працівників про потенційні небезпеки, проведення навчання та перевірки знань з охорони праці, наявність засобів первинного пожежогасіння, правильно організовані евакуаційні виходи тощо.

Окрему увагу слід приділити електробезпеці, адже комп’ютерна техніка працює від електромережі, і будь-які порушення в ізоляції, заземленні або навантаженні мережі можуть спричинити ураження струмом. У межах комп’ютерних кабінетів обов’язковою є наявність справного автоматичного вимикача, захисного заземлення та системи аварійного відключення.

Техніка безпеки — це не разова дія, а систематичний процес. Регулярний контроль, модернізація обладнання, аудит ризиків та залучення фахівців з охорони праці дозволяють знизити рівень потенційної небезпеки до мінімуму. Це особливо актуально для закладів освіти, де використовуються численні комп’ютерні робочі місця, а також для підприємств, що займаються розробкою програмного забезпечення чи обслуговуванням інформаційних систем.

4.2. Комп'ютерна техніка та її вплив на організм людини

Тривала робота за комп’ютером викликає комплексний вплив на організм людини, що охоплює фізіологічні, психоемоційні та ергономічні аспекти. Найбільше навантаження припадає на зір, опорно-руховий апарат та нервову систему, а в умовах недостатньо оптимізованого робочого місця — також і на дихальну та серцево-судинну системи.

Зорове напруження є найбільш поширеним негативним наслідком тривалої роботи з монітором. Часто спостерігаються симптоми, які в офіційній документації класифікуються як "комп’ютерний зоровий синдром": сухість очей, розмитість зображення, головний біль, почервоніння та біль у ділянці очей. У документі «ДСанПіН 3.3.2.007-98» зазначено: «Тривалість безперервної роботи з

відеотерміналами має бути обмежена та супроводжуватися регламентованими перервами з метою профілактики зорової втоми» [3].

Надмірне сидіння призводить до статичних навантажень на хребет, особливо в ділянці попереку та шиї, що в довготривалій перспективі може спричинити остеохондроз, викривлення постави, затиск нервових закінчень і болі в м’язах. Робота в незручному положенні, з відсутністю підтримки зап’ясть або спини, лише погіршує ситуацію. Невдале розміщення монітора (занадто високо, низько чи під кутом) додатково сприяє напруженню м’язів шиї та плечового пояса.

До психоемоційних наслідків надмірного використання комп’ютера відносяться дратівливість, зниження концентрації уваги, емоційне виснаження, особливо в умовах надмірного шуму чи візуального перевантаження на екрані. Важливим є не лише фізичне, але й психологічне здоров’я працівника, яке вимагає грамотного підходу до організації робочого процесу.

Значного значення набуває питання мікроклімату. Якщо температура повітря в приміщенні є надто високою або низькою, це викликає втому, сонливість, зниження працездатності. Згідно з чинними нормами, «температура в приміщенні, де використовуються персональні комп’ютери, повинна бути в межах 20–24 °С» [3]. Також потрібно контролювати рівень вологості, швидкість руху повітря, наявність вуглекислого газу.

Шкідливим є і вплив електромагнітного випромінювання, яке хоч і значно знижене в сучасних РК-дисплеях, однак не усунене повністю. Тривалий вплив навіть незначних рівнів ЕМП може накопичуватись, особливо якщо робоче місце не розташовано відповідно до норм.

Таким чином, вплив комп’ютерної техніки на організм людини є багатофакторним і вимагає комплексного підходу до організації безпечних умов праці. Всі ці чинники повинні враховуватись як при розробці, так і при використанні програмного забезпечення, зокрема інтернет-магазинів, що функціонують у середовищі з підвищеним навантаженням на користувача.

4.3. Шляхи оптимізації технічних, середовищних та ергономічних факторів

Удосконалення умов праці при роботі з комп’ютерною технікою є важливим елементом загальної системи охорони праці. Раціональна організація робочого місця, оптимізація технічного середовища та впровадження ергономічних рішень дозволяють не лише підвищити продуктивність працівника, а й суттєво знизити ризики розвитку професійних захворювань.

Першочерговим завданням є правильне облаштування робочого місця. Висота робочого столу, крісла, розміщення монітора й клавіатури повинні відповідати анатомічним особливостям користувача. Відстань від очей до екрана має становити 50–70 см, а центр монітора має бути нижче рівня очей приблизно на 10–15 см. Кут огляду повинен бути зручним для уникнення нахилу голови вперед, що зменшує напруження шийних м’язів.

Важливо використовувати офісні крісла з можливістю регулювання висоти сидіння, спинки та кута нахилу. Клавіатура повинна бути розташована на одному рівні з ліктями користувача або трохи нижче. Встановлення підставок під зап’ястя та опор для ніг дозволяє уникати зайвих статичних навантажень і покращити кровообіг під час тривалої роботи.

Оптимізація середовища включає належне освітлення — найкращим є комбіноване: природне світло з коригуванням за допомогою штучного. Необхідно уникати відблисків на екрані, для чого монітор слід розташовувати перпендикулярно до вікна, а освітлення має бути рівномірним. Також варто впроваджувати системи штучного освітлення з рівнем освітленості не менше ніж 300 лк, як передбачено нормативами.

Особливу увагу слід приділити мікроклімату. Температурний режим в офісних приміщеннях має бути стабільним і перебувати в межах 20–24 °C, відносна вологість — 40–60 %, а швидкість руху повітря — не більше 0,1 м/с [3]. Для підтримання цих параметрів рекомендовано використовувати кондиціонери з функцією зволоження, системи вентиляції з фільтрами тонкого очищення, а також регулярне провітрювання.

Щодо технічного боку, важливим аспектом є оновлення обладнання. Сучасні комп’ютери мають вищу енергоефективність, менший рівень шуму, поліпшені дисплеї з фільтрацією синього світла та антибліковим покриттям. Також програмне забезпечення має підтримувати масштабування інтерфейсу, регулювання шрифтів, кольорових схем і яскравості, що знижує візуальне навантаження.

Важливим напрямом є впровадження режимів праці та відпочинку. Згідно з вимогами «ДСанПіН 3.3.2.007-98», при роботі з ПЕОМ слід передбачати регламентовані перерви: 10–15 хвилин щогодини або спеціальні вправи для очей і м’язів. Такі практики сприяють зменшенню втоми, підвищенню продуктивності та профілактиці захворювань опорно-рухового апарату [3].

Таким чином, оптимізація умов праці — це комплексне завдання, що поєднує архітектурне, інженерне, санітарне й психологічне проєктування. В контексті дипломної роботи, яка передбачає створення інтернет-магазину, особливо важливо враховувати вплив інтерфейсу на користувача, мінімізувати візуальні перевантаження та забезпечити інтуїтивну логіку навігації.

ВИСНОВКИ

У результаті виконання дипломної роботи було розроблено та реалізовано повнофункціональний web-застосунок інтернет-магазину, орієнтований на продаж комп’ютерної техніки. Робота охопила повний цикл створення програмного продукту — від аналізу предметної області та постановки задачі до проєктування архітектури, реалізації інтерфейсу, тестування й обґрунтування заходів з охорони праці.

У першому розділі проведено огляд поняття електронної комерції, її переваг та технічних аспектів. Було визначено, що інтернет-магазин як форма електронної торгівлі має широкий потенціал для автоматизації бізнес-процесів, масштабування і підвищення зручності взаємодії з користувачем. Особливу увагу приділено функціональним характеристикам та структурі типової системи електронної торгівлі.

У другому розділі здійснено технічний аналіз основних компонентів програмної системи, включно з описом архітектури, функціональної моделі, реалізації бази даних, а також взаємодії між модулями. Було створено діаграми, які ілюструють функціональну та логічну побудову системи. Реалізація системи здійснена з використанням таких технологій, як Flask, HTML/CSS/JavaScript, SQLite і Docker. Окрема увага приділена безпеці даних, зокрема шифруванню паролів за допомогою bcrypt.

У третьому розділі виконано практичну реалізацію web-застосунку, представлено фрагменти інтерфейсу та результати ручного тестування. Функціонал забезпечує перегляд товарів, реєстрацію та авторизацію користувачів, додавання товарів у кошик та оформлення замовлення. Тестування підтвердило стабільність і працездатність ключових модулів.

У четвертому розділі висвітлено питання охорони праці при роботі з комп’ютерною технікою. Проаналізовано ризики, пов’язані з тривалою роботою за ПК, вплив електромагнітного випромінювання, недотримання ергономічних

норм, а також наведено шляхи зниження впливу негативних факторів згідно з чинним законодавством України.

У підсумку, розроблений інтернет-магазин є прикладом ефективного застосування сучасних інформаційних технологій у сфері електронної торгівлі. Він демонструє можливості створення масштабованих, безпечних і зручних web-рішень для реалізації продукції через інтернет. Отримані результати можуть бути використані як основа для подальшого розвитку — зокрема додавання платіжних систем, адміністративної панелі або інтеграції з CRM.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Бібліотека bcrypt для Python [Електронний ресурс]. – Режим доступу: https://pypi.org/project/bcrypt
2. Бутенко Д. С. Електронна комерція як запорука розвитку підприємництва в Україні // Ефективна економіка. – 2023. – № 3.
3. ДСанПіН 3.3.2.007-98. Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин. – Київ: МОЗ України, 1998.
4. Дражниця С. А. Електронна комерція: навчально-методичний посібник. – Київ: Новий світ 2000, 2025. – 184 с.
5. Іванов К. Р. Електронна комерція: сутність, функції, форми та сучасний стан // Науковий огляд. – 2024.
6. Кобиляцький Ю. О. Охорона праці в галузі: навчальний посібник. – Харків: ФОП Бровін О. В., 2021. – 336 с.
7. Kraus. Електронна комерція та Інтернет-торгівля: навчально-методичний посібник. – Київ: Аграр Медіа Груп, 2021. – 454 с.
8. MOYO.UA – інтернет-магазин цифрової та побутової техніки [Електронний ресурс]. – Режим доступу: https://www.moyo.ua
9. ОЛІЙНИК О. М. Охорона праці в ІТ: методичні аспекти. – Львів: ЛНУ імені Івана Франка, 2020. – 192 с.
10. Офіційна документація Docker [Електронний ресурс]. – Режим доступу: https://docs.docker.com
11. Офіційна документація Flask [Електронний ресурс]. – Режим доступу: https://flask.palletsprojects.com
12. Офіційна документація SQLite [Електронний ресурс]. – Режим доступу: https://sqlite.org
13. ROZETKA – інтернет-супермаркет електроніки [Електронний ресурс]. – Режим доступу: https://rozetka.com.ua

14. BRAIN – інтернет-магазин комп’ютерної техніки [Електронний ресурс]. – Режим доступу: https://brain.com.ua
15. COMFY – інтернет-магазин побутової техніки та електроніки [Електронний ресурс]. – Режим доступу: https://comfy.ua

ДОДАТОК А

ЛІСТИНГ ПРОГРАМНОГО КОДУ ВЕБ-ЗАСТОСУНКУ

```
from flask import Flask, render_template, request, redirect, url_for, flash, session
        from flask_sqlalchemy import SQLAlchemy
        from flask_login import LoginManager, UserMixin, login_user, login_required, logout_user,
current_user
        from datetime import datetime
        from sqlalchemy import text
        import os
        from flask_wtf import FlaskForm
        from wtforms import StringField, PasswordField, TextAreaField, FloatField, IntegerField,
SelectField
        from wtforms.validators import DataRequired, Email, Length, EqualTo, Optional
        import bcrypt
        from werkzeug.utils import secure_filename

        app = Flask(__name__)

        app.config['SECRET_KEY'] = os.environ.get('SECRET_KEY', 'your-super-secret-key-
123')  # Використовуємо змінну середовища
        app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///tech_shop.db'
        app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False

        db = SQLAlchemy(app)
        login_manager = LoginManager()
        login_manager.init_app(app)
        login_manager.login_view = 'login'

        class User(UserMixin, db.Model):
            id = db.Column(db.Integer, primary_key=True)
            username = db.Column(db.String(80), unique=True, nullable=False)
            email = db.Column(db.String(120), unique=True, nullable=False)
            password_hash = db.Column(db.String(128))
            is_admin = db.Column(db.Boolean, default=False)
            orders = db.relationship('Order', backref='user', lazy=True)
            cart = db.relationship('Cart', backref='user', uselist=False)

            def __init__(self, username, email, is_admin=False):
                self.username = username
                self.email = email
                self.is_admin = is_admin

            def set_password(self, password):

                salt = bcrypt.gensalt()
                self.password_hash = bcrypt.hashpw(password.encode('utf-8'), salt).decode('utf-8')

            def check_password(self, password):
```

```
            # Перевіряємо пароль
            return bcrypt.checkpw(password.encode('utf-8'), self.password_hash.encode('utf-8'))

        class Category(db.Model):
            id = db.Column(db.Integer, primary_key=True)
            name = db.Column(db.String(50), nullable=False)
            description = db.Column(db.Text)
            products = db.relationship('Product', backref='category', lazy=True)

            def __init__(self, name, description):
                self.name = name
                self.description = description

        class Product(db.Model):
            id = db.Column(db.Integer, primary_key=True)
            name = db.Column(db.String(100), nullable=False)
            description = db.Column(db.Text)
            price = db.Column(db.Float, nullable=False)
            stock = db.Column(db.Integer, default=0)
            image_url = db.Column(db.String(200))
            category_id = db.Column(db.Integer, db.ForeignKey('category.id'), nullable=False)
            created_at = db.Column(db.DateTime, default=datetime.utcnow)
              order_items = db.relationship('OrderItem', backref='product', lazy=True, cascade='all,
delete-orphan')
            cart_items = db.relationship('CartItem', backref='product', lazy=True, cascade='all, delete-
orphan')

            def __init__(self, name, description, price, stock, image_url, category_id):
                self.name = name
                self.description = description
                self.price = price
                self.stock = stock
                self.image_url = image_url
                self.category_id = category_id

        class Order(db.Model):
            id = db.Column(db.Integer, primary_key=True)
            user_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)
            status = db.Column(db.String(20), default='pending')
            total_amount = db.Column(db.Float, nullable=False)
            shipping_address = db.Column(db.String(200))
            created_at = db.Column(db.DateTime, default=datetime.utcnow)
            items = db.relationship('OrderItem', backref='order', lazy=True)

        class OrderItem(db.Model):
            id = db.Column(db.Integer, primary_key=True)
            order_id = db.Column(db.Integer, db.ForeignKey('order.id'), nullable=False)
            product_id = db.Column(db.Integer, db.ForeignKey('product.id'), nullable=False)
            quantity = db.Column(db.Integer, nullable=False)
            price = db.Column(db.Float, nullable=False)
```

```
class Cart(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)
    created_at = db.Column(db.DateTime, default=datetime.utcnow)
    items = db.relationship('CartItem', backref='cart', lazy=True)

class CartItem(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    cart_id = db.Column(db.Integer, db.ForeignKey('cart.id'), nullable=False)
    product_id = db.Column(db.Integer, db.ForeignKey('product.id'), nullable=False)
    quantity = db.Column(db.Integer, nullable=False, default=1)

class UserProfile(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)
    first_name = db.Column(db.String(50))
    last_name = db.Column(db.String(50))
    phone = db.Column(db.String(20))
    address = db.Column(db.String(200))
    created_at = db.Column(db.DateTime, default=datetime.utcnow)
        updated_at = db.Column(db.DateTime, default=datetime.utcnow,
onupdate=datetime.utcnow)

User.profile = db.relationship('UserProfile', backref='user', uselist=False)

with app.app_context():
    db.create_all()

@login_manager.user_loader
def load_user(user_id):
    return User.query.get(int(user_id))

@app.route('/')
def index():
    featured_products = Product.query.order_by(Product.created_at.desc()).limit(4).all()
    return render_template('index.html', featured_products=featured_products)

@app.route('/products')
def products():

    category_id = request.args.get('category', type=int)
    min_price = request.args.get('min_price', type=float)
    max_price = request.args.get('max_price', type=float)
    sort_by = request.args.get('sort', default='name')


    query = Product.query


    if category_id:
        query = query.filter_by(category_id=category_id)
```

```
    if min_price is not None:
        query = query.filter(text('price >= :min_price')).params(min_price=min_price)
    if max_price is not None:
        query = query.filter(text('price <= :max_price')).params(max_price=max_price)


    if sort_by == 'price_asc':
        query = query.order_by(text('price ASC'))
    elif sort_by == 'price_desc':
        query = query.order_by(text('price DESC'))
    elif sort_by == 'name':
        query = query.order_by(text('name ASC'))
    elif sort_by == 'newest':
        query = query.order_by(text('created_at DESC'))


    categories = Category.query.all()


    products = query.all()

    return render_template('products.html',
                           products=products,
                           categories=categories,
                           current_category=category_id,
                           current_min_price=min_price,
                           current_max_price=max_price,
                           current_sort=sort_by)

@app.route('/product/<int:product_id>')
def product(product_id):
    product = Product.query.get_or_404(product_id)
    return render_template('product.html', product=product)

@app.route('/cart')
@login_required
def cart():
    if not current_user.cart:
        current_user.cart = Cart()
        current_user.cart.user_id = current_user.id
        db.session.commit()
    return render_template('cart.html', cart=current_user.cart)

@app.route('/cart/add/<int:product_id>', methods=['POST'])
@login_required
def add_to_cart(product_id):
    product = Product.query.get_or_404(product_id)
    quantity = int(request.form.get('quantity', 1))

    if not current_user.cart:
        current_user.cart = Cart()
```

```
        current_user.cart.user_id = current_user.id
        db.session.commit()


    existing_item = CartItem.query.filter_by(
        cart_id=current_user.cart.id,
        product_id=product_id
    ).first()

    if existing_item:
        existing_item.quantity += quantity
    else:
        cart_item = CartItem()
        cart_item.cart_id = current_user.cart.id
        cart_item.product_id = product_id
        cart_item.quantity = quantity
        db.session.add(cart_item)

    db.session.commit()
    flash('Товар додано до кошика!', 'success')
    return redirect(url_for('cart'))

@app.route('/cart/update/<int:item_id>', methods=['POST'])
@login_required
def update_cart_item(item_id):
    cart_item = CartItem.query.get_or_404(item_id)
    if cart_item.cart.user_id != current_user.id:
        flash('У вас немає доступу до цього елемента', 'error')
        return redirect(url_for('cart'))

    change = int(request.form.get('change', 0))
    new_quantity = cart_item.quantity + change

    if new_quantity <= 0:
        db.session.delete(cart_item)
    else:
        cart_item.quantity = new_quantity

    db.session.commit()
    flash('Кошик оновлено!', 'success')
    return redirect(url_for('cart'))

@app.route('/cart/remove/<int:item_id>', methods=['POST'])
@login_required
def remove_from_cart(item_id):
    cart_item = CartItem.query.get_or_404(item_id)
    if cart_item.cart.user_id != current_user.id:
        flash('У вас немає доступу до цього елемента', 'error')
        return redirect(url_for('cart'))

    db.session.delete(cart_item)
```

```
    db.session.commit()
    flash('Товар видалено з кошика!', 'success')
    return redirect(url_for('cart'))

@app.route('/checkout', methods=['GET', 'POST'])
@login_required
def checkout():

    if not current_user.cart:
        current_user.cart = Cart()
        current_user.cart.user_id = current_user.id
        db.session.commit()


    cart = Cart.query.filter_by(user_id=current_user.id).first()
    if not cart:
        cart = Cart()
        cart.user_id = current_user.id
        db.session.add(cart)
        db.session.commit()

    if request.method == 'POST':

        full_name = request.form.get('fullName')
        phone = request.form.get('phone')
        email = request.form.get('email')
        city = request.form.get('city')
        street = request.form.get('street')
        house = request.form.get('house')
        apartment = request.form.get('apartment')
        payment_method = request.form.get('payment-method')

        shipping_address = f"{city}, {street}, {house}"
        if apartment:
            shipping_address += f", кв./офіс {apartment}"

        if not cart or not cart.items:
            flash('Ваш кошик порожній!', 'error')
            return redirect(url_for('cart'))
        total_amount = sum(item.product.price * item.quantity for item in cart.items)

        order = Order()
        order.user_id = current_user.id
        order.status = 'pending'
        order.total_amount = total_amount
        order.shipping_address = shipping_address
        db.session.add(order)
        db.session.commit()

        for item in cart.items:
            order_item = OrderItem()
```

```
            order_item.order_id = order.id
            order_item.product_id = item.product_id
            order_item.quantity = item.quantity
            order_item.price = item.product.price
            db.session.add(order_item)

        for item in cart.items:
            db.session.delete(item)
        db.session.commit()
        flash('Замовлення успішно оформлено!', 'success')
        return redirect(url_for('order_confirmation', order_id=order.id))
    return render_template('checkout.html', cart=cart)

@app.route('/order-confirmation/<int:order_id>')
@login_required
def order_confirmation(order_id):
    order = Order.query.get_or_404(order_id)
    return render_template('order-confirmation.html', order=order)

@app.route('/about')
def about():
    return render_template('about.html')

@app.route('/contact')
def contact():
    return render_template('contact.html')

@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        email = request.form.get('email')
        password = request.form.get('password')
        user = User.query.filter_by(email=email).first()

        if user and user.check_password(password):
            login_user(user)
            return redirect(url_for('index'))
        flash('Невірний email або пароль', 'error')
    return render_template('login.html')

@app.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        username = request.form.get('username')
        email = request.form.get('email')
        password = request.form.get('password')

        if User.query.filter_by(email=email).first():
            flash('Email вже зареєстрований', 'error')
            return redirect(url_for('register'))
```

```
            user = User(username=username, email=email)
            user.set_password(password)
            db.session.add(user)
            db.session.commit()
            login_user(user)
            return redirect(url_for('index'))
        return render_template('register.html')

    @app.route('/logout')
    @login_required
    def logout():
        logout_user()
        flash('Ви вийшли з системи', 'success')
        return redirect(url_for('index'))

    class EditProfileForm(FlaskForm):
        username = StringField('Ім\'я користувача', validators=[DataRequired(), Length(min=3, max=64)])
        email = StringField('Email', validators=[DataRequired(), Email()])
        address = TextAreaField('Адреса', validators=[Optional()])
        phone = StringField('Телефон', validators=[Optional()])
        current_password = PasswordField('Поточний пароль', validators=[Optional()])
        new_password = PasswordField('Новий пароль', validators=[Optional(), Length(min=6)])
        confirm_password = PasswordField('Підтвердження паролю', validators=[EqualTo('new_password')])

    @app.route('/profile')
    @login_required
    def profile():
        orders = Order.query.filter_by(user_id=current_user.id).order_by(Order.created_at.desc()).all()
        return render_template('profile.html', orders=orders)

    @app.route('/edit_profile', methods=['GET', 'POST'])
    @login_required
    def edit_profile():
        form = EditProfileForm()

        if not current_user.profile:
            profile = UserProfile()
            profile.user_id = current_user.id
            db.session.add(profile)
            db.session.commit()
        profile = current_user.profile
        if form.validate_on_submit():
            if form.current_password.data:
                if not current_user.check_password(form.current_password.data):
                    flash('Невірний поточний пароль', 'error')
                    return render_template('edit_profile.html', form=form)
                if form.new_password.data:
                    current_user.set_password(form.new_password.data)
```

```
        if form.username.data != current_user.username:
            if User.query.filter_by(username=form.username.data).first():
                flash('Це ім\'я користувача вже зайняте', 'error')
                return render_template('edit_profile.html', form=form)
            current_user.username = form.username.data
        if form.email.data != current_user.email:
            if User.query.filter_by(email=form.email.data).first():
                flash('Цей email вже зареєстрований', 'error')
                return render_template('edit_profile.html', form=form)
            current_user.email = form.email.data
        profile.address = form.address.data
        profile.phone = form.phone.data
        db.session.commit()
        flash('Профіль успішно оновлено', 'success')
        return redirect(url_for('profile'))

    if request.method == 'GET':
        form.username.data = current_user.username
        form.email.data = current_user.email
        form.address.data = profile.address
        form.phone.data = profile.phone
    return render_template('edit_profile.html', form=form)

@app.route('/admin')
@login_required
def admin():
    if not current_user.is_admin:
        flash('У вас немає доступу до цієї сторінки', 'error')
        return redirect(url_for('index'))
    return render_template('admin/index.html')

@app.route('/admin/orders')
@login_required
def admin_orders():
    if not current_user.is_admin:
        flash('У вас немає доступу до цієї сторінки', 'error')
        return redirect(url_for('index'))
    orders = Order.query.order_by(Order.created_at.desc()).all()
    return render_template('admin/orders.html', orders=orders)

@app.route('/admin/products')
@login_required
def admin_products():
    if not current_user.is_admin:
        flash('У вас немає доступу до цієї сторінки', 'error')
        return redirect(url_for('index'))
    products = Product.query.all()
    form = ProductForm()
    return render_template('admin/products.html', products=products, form=form)
```

```
@app.route('/admin/order/<int:order_id>', methods=['GET', 'POST'])
@login_required
def admin_order(order_id):
    if not current_user.is_admin:
        flash('У вас немає доступу до цієї сторінки', 'error')
        return redirect(url_for('index'))

    order = Order.query.get_or_404(order_id)

    if request.method == 'POST':
        new_status = request.form.get('status')
        if new_status in ['pending', 'processing', 'shipped', 'delivered', 'cancelled']:
            order.status = new_status
            db.session.commit()
            flash('Статус замовлення оновлено', 'success')
        return redirect(url_for('admin_order', order_id=order.id))

    return render_template('admin/order.html', order=order)

@app.route('/admin/products/edit/<int:product_id>', methods=['GET', 'POST'])
@login_required
def admin_edit_product(product_id):
    if not current_user.is_admin:
        flash('У вас немає доступу до цієї сторінки', 'error')
        return redirect(url_for('index'))
    product = Product.query.get_or_404(product_id)
    form = ProductForm(obj=product)
    categories = Category.query.all()
    form.category_id.choices = [(c.id, c.name) for c in categories]
    if form.validate_on_submit():
        product.name = form.name.data
        product.description = form.description.data
        product.price = form.price.data
        product.stock = form.stock.data
        product.image_url = form.image_url.data
        product.category_id = form.category_id.data
        db.session.commit()
        flash('Товар успішно оновлено', 'success')
        return redirect(url_for('admin_products'))
      return render_template('admin/product_form.html', form=form, categories=categories,
edit=True, product=product)

@app.route('/admin/products/delete/<int:product_id>', methods=['POST'])
@login_required
def admin_delete_product(product_id):
    if not current_user.is_admin:
        flash('У вас немає доступу до цієї сторінки', 'error')
        return redirect(url_for('index'))
    product = Product.query.get_or_404(product_id)
    db.session.delete(product)
    db.session.commit()
```

```
        flash('Товар видалено', 'success')
        return redirect(url_for('admin_products'))

    @app.errorhandler(404)
    def page_not_found(e):
        return render_template('404.html'), 404

    def create_admin():
        admin = User.query.filter_by(username='admin').first()
        if not admin:
            admin = User(
                username='admin',
                email='lvt@gmail.com',
                is_admin=True
            )
        else:
            admin.email = 'lvt@gmail.com'
            admin.is_admin = True

        admin.set_password('admin123')
        db.session.add(admin)
        db.session.commit()
        print('Адміністратор створено/оновлено!')
        print('Логін: admin')
        print('Email: lvt@gmail.com')
        print('Пароль: admin123')

    class ProductForm(FlaskForm):
        name = StringField('Назва товару', validators=[DataRequired(), Length(min=3,
max=100)])
        description = TextAreaField('Опис', validators=[DataRequired()])
        price = FloatField('Ціна', validators=[DataRequired()])
        stock = IntegerField('Кількість', validators=[DataRequired()])
        image_url = StringField('URL зображення', validators=[DataRequired()])
        category_id = SelectField('Категорія', coerce=int, validators=[DataRequired()])

    @app.route('/admin/products/new', methods=['GET', 'POST'])
    @login_required
    def admin_new_product():
        if not current_user.is_admin:
            flash('У вас немає доступу до цієї сторінки', 'error')
            return redirect(url_for('index'))

        form = ProductForm()
        categories = Category.query.all()
        form.category_id.choices = [(c.id, c.name) for c in categories]

        if form.validate_on_submit():
            try:
                product = Product(
                    name=form.name.data,
```

```
                description=form.description.data,
                price=form.price.data,
                stock=form.stock.data,
                image_url=form.image_url.data,
                category_id=form.category_id.data
            )
            db.session.add(product)
            db.session.commit()
            flash('Товар успішно додано', 'success')
            return redirect(url_for('admin_products'))
        except Exception as e:
            flash(f'Помилка при додаванні товару: {str(e)}', 'error')

    return render_template('admin/product_form.html', form=form, categories=categories)

if __name__ == '__main__':
    with app.app_context():
        db.create_all()
        create_admin()
    app.run(debug=True)
```