МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«РОЗРОБКА ІНТЕРНЕТ-МАГАЗИНУ ПОБУТОВОЇ ТЕХНІКИ З ВИКОРИСТАННЯМ FASTAPI»**

Виконала: студентка 2 курсу, групи 8.1213-іпз-1
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми інженерія програмного забезпечення
(назва освітньої програми)
О.Р. Смотрицька
(ініціали та прізвище)

Керівник завідувач кафедри програмної інженерії, доцент, к.ф.-м.н. Лісняк А.О.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент завідувач кафедри фундаментальної та прикладної математики, професор, д.т.н. Гребенюк С.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти магістр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2024 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТЦІ

Смотрицькій Олександрі Романівні
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка інтернет-магазину побутової техніки з використанням FastAPI

керівник роботи Лісняк Андрій Олександрович, к.ф.-м.н., доцент
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 01 » березня 2024 року № 385-с

2. Строк подання студентом роботи 25.11.2024 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.
3. Практична реалізація коду.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.
3. Реалізація проєкту.
4. Тестування вебсайту.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація за темою доповіді

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 07.03.2024 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	21.03.2024	
2.	Збір вихідних даних.	01.04.2024	
3.	Обробка методичних та теоретичних	07.05.2024	
	джерел.		
4.	Розробка першого та другого розділів.	28.06.2024	
5.	Розробка третього розділу.	29.08.2024	
6.	Розробка четвертого розділу.	05.11.2024	
7.	Оформлення та нормоконтроль	20.11.2024	
	кваліфікаційної роботи магістра.		
8.	Захист кваліфікаційної роботи.	13.12.2024	

Студент _____ О.Р. Смотрицька
(підпис) (ініціали та прізвище)

Керівник роботи _____ А.О. Лісняк
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота магістра «Розробка інтернет-магазину побутової техніки з використанням FastAPI»: 61 с., 24 рис., 8 табл., 20 джерел.

БАЗА ДАНИХ, ВЕБСАЙТ, ІНТЕРНЕТ МАГАЗИН, ПОБУТОВА ТЕХНІКА, CSS, FASTAPI, HTML, POSTGRESQL, PYTHON.

Об’єкт дослідження – процес розробки інтернет-магазину побутової техніки.

Мета роботи – розробка інтернет-магазину побутової техніки з використанням фреймворку FastAPI.

Предмет дослідження – Pycharm, FastAPI, draw.io.

Методи дослідження – методи об’єктно-орієнтованого програмування, методи програмної інженерії, системний аналіз.

У кваліфікаційній роботі проаналізовано фреймворки Python для створення інтернет магазинів. Для створення інтернет-магазину обрано FastAPI, оскільки цей фреймворк забезпечує високу продуктивність і швидкість обробки запитів завдяки асинхронній архітектурі. FastAPI підтримує сучасні стандарти OpenAPI і JSON Schema, що полегшує інтеграцію з іншими сервісами та генерацію документації. Крім того, FastAPI пропонує зручний синтаксис для опису маршрутів та обробки даних, що сприяє швидкому розвитку та підтримці проєктів.

В результаті роботи спроєктовано та створено інтернет-магазин побутової техніки з використанням FastAPI.

SUMMARY

Master’s qualifying paper “Development of an Online Store for Household Appliances Using FastAPI”: 61 pages, 24 figures, 8 tables, 20 references.

CSS, DATABASE, FASTAPI, HOUSEHOLD APPLIANCES, HTML, ONLINE STORE, POSTGRESQL, PYTHON, WEBSITE.

The object of the study is the process of developing an online household appliances store.

The aim of the study is to develop an online household appliances store using the FastAPI framework.

The subject of the study is the FastAPI web framework (Python).

The methods of research are object-oriented programming methods, software engineering methods, system analysis.

The paper analyzes Python frameworks for creating online stores. FastAPI was chosen for creating the online store because this framework ensures high performance and fast request processing due to its asynchronous architecture. FastAPI supports modern OpenAPI and JSON Schema standards, facilitating integration with other services and documentation generation. Additionally, FastAPI offers a convenient syntax for defining routes and data handling, promoting rapid development and maintenance of projects.

As a result of the work, a home appliances online store was designed and created using FastAPI.

ЗМІСТ

Завдання на кваліфікаційну роботу……..2
Реферат 4
Summary 5
Вступ..... 8
1 Теоритичні відомості 10
1.1 Мова програмування Python 10
1.2 Огляд фреймворків для розробки інтернет-магазинів 11
1.3 Аналіз та порівняння вебфреймворків..... 12
1.4 Технічне завдання 14
1.4.1 Опис предметної області..... 14
1.4.2 Функціональні вимоги..... 14
1.4.3 Нефункціональні вимоги..... 15
1.5 Висновки до розділу 1 16
2 Моделювання та проєктування системи..... 17
2.1 Діаграма прецедентів..... 17
2.2 Діаграма послідовності..... 21
2.3 Моделювання структури даних 25
2.4 Діаграма розгортання 31
2.5 Висновки до розділу 2 34
3 Реалізація системи..... 36
3.1 Реалізація бази даних..... 36
3.2 Реалізація обробки замовлень та управління товарами 41
3.3 Реалізація аутентифікації та авторизації 48
3.4 Реалізація інтерфейсу для перегляду товарів..... 51
3.5 Висновки до розділу 3 53
4 Опис інтерфейсу системи 55
4.1 Демонстрація роботи онлайн-магазину..... 55

4.2 Висновки до розділу 4 58
Висновки 59
Перелік посилань..... 60

ВСТУП

У сучасному світі інтернет-магазини відіграють важливу роль у забезпеченні споживачів товарами та послугами. Розвиток технологій та поширення доступу до інтернету зробили онлайн-шопінг невід'ємною частиною повсякденного життя. Інтернет-магазини надають можливість купувати товари з будь-якого куточка світу, заощаджуючи час та зусилля покупців. Однією з ключових категорій товарів, які користуються великим попитом в інтернет-магазинах, є побутова техніка.

Розробка інтернет-магазину потребує використання ефективних та надійних технологій, які забезпечують високу продуктивність, безпеку та зручність використання. Python, як одна з найпопулярніших мов програмування, пропонує широкий спектр фреймворків для створення вебдодатків. Серед них особливе місце займає FastAPI – сучасний та продуктивний фреймворк для створення веб-API з підтримкою асинхронної обробки запитів.

Мета даної роботи полягає у розробці інтернет-магазину побутової техніки з використанням фреймворку FastAPI. Для досягнення цієї мети необхідно проаналізувати особливості FastAPI, визначити його переваги та недоліки у порівнянні з іншими фреймворками, а також реалізувати функціонал, необхідний для повноцінного функціонування інтернет-магазину.

Задачі, які необхідно розв'язати для досягнення поставленої мети:

- проаналізувати сучасні рішення для розробки інтернет-магазинів та їх порівняння;
- проаналізувати предметну область;
- спроєктувати та реалізувати базу даних;
- розробити функціонал інтернет-магазину;
- реалізувати інтерфейс;
- протестувати роботу сайту.

Об’єкт дослідження – процес розробки інтернет-магазину побутової техніки з використанням.

Предмет дослідження – Pycharm, FastAPI, draw.io.

Методи дослідження – методи об’єктно-орієнтованого програмування, методи програмної інженерії, системний аналіз.

1 ТЕОРИТИЧНІ ВІДОМОСТІ

1.1 Мова програмування Python

Python – мова програмування, яка за свою історію стала уособленням простоти та потужності в одному рішенні. Народжений у 1991 році зусиллями Гвідо ван Россума, Python вирізняється прозорим і легким для розуміння синтаксисом, що перетворив його на справжнього улюбленця програмістів у всьому світі. Ця мова універсальна: вона однаково впевнено почувається як у світі науки й аналітики, так і у творчості веброзробників чи інженерів штучного інтелекту [1].

У своїх перших версіях Python використовувався переважно для скриптових задач, таких як автоматизація рутинних завдань, що включали роботу з файлами та теками, а також обробку текстових даних. З розвитком мови, люди почали використовувати Python для більш складних задач, включаючи розробку вебзастосунків, наукові обчислення, штучний інтелект та машинне навчання. Основна мета розробників Python – зробити мову якнайбільш доступною та читабельною. Це дозволило створити мову з ясною та лаконічною семантикою, яка значно спрощує процес розуміння коду [2].

До головних особливостей Python відносяться [3]:

– простота: його код не потрібно «зламувати», щоб зрозуміти, замість складних конструкцій – чистота й лаконічність;
– багатогранність: Python відкриває шлях до різних підходів у програмуванні від класичної об’єктно-орієнтованої архітектури до гнучкої функціональної логіки;
– величезна бібліотека можливостей: у стандартній бібліотеці знайдеться все необхідне для швидкого старту в будь-якому проєкті;
– потужна спільнота: світ Python – це мільйони розробників, які створюють бібліотеки, фреймворки й діляться своїм досвідом, полегшуючи життя іншим.

Python вже давно не просто інструмент, а ключ до найсучасніших досягнень. У веброзробці він дає змогу втілювати ідеї в життя за допомогою таких фреймворків, як Django чи Flask. У наукових дослідженнях Python перетворює складні набори даних на зрозумілі графіки й аналітичні моделі. У машинному навчанні ця мова допомагає розробляти інтелектуальні системи, які змінюють наше життя, використовуючи TensorFlow та PyTorch. Навіть у буденності Python не залишається осторонь: автоматизація, скрипти, тестування – усе це йому під силу.

Порівнюючи Python із іншими мовами, як-от Java чи C++, ми бачимо, що він – це баланс простоти, гнучкості й потужності. Його обирають не лише за ефективність, а й за відчуття, що з ним навіть складне завдання стає простим і захопливим.

1.2 Огляд фреймворків для розробки інтернет-магазинів

Фреймворки стали незамінним інструментом для розробників, забезпечуючи фундамент і структуру, які необхідні для створення програмного забезпечення. Ці платформи пропонують зручний набір готових рішень, завдяки чому програмісти можуть зосередитися на творчій частині процесу – розробці унікальної логіки та функціональності, замість того щоб витрачати час на налаштування інфраструктури. Вони прискорюють розробку, надаючи стандартизовані підходи до вирішення рутинних задач, і водночас створюють середовище, яке сприяє оптимізації робочого процесу. У світі Python існує чимало фреймворків, кожен із яких є унікальним, серед них найбільш популярні – це Flask, Django, Pyramid та FastAPI. Кожен з них має свої особливості, переваги та недоліки. Розглянемо детальніше кожен з вище зазначених фреймворків [4].

Flask – це мікрофреймворк, який, мов легкий інструмент, дозволяє швидко й ефективно створювати вебдодатки. Його простота й мінімалістичність роблять його чудовим вибором для невеликих проєктів,

адже він пропонує розробникам свободу у виборі бібліотек і компонентів. Навчання Flask не вимагає значних зусиль, що є ще однією причиною його популярності. Однак у великих проєктах розробникам доведеться приділяти більше уваги налаштуванням, адже фреймворк не має вбудованої ORM [4 – 6].

Django, навпаки, створений для тих, хто прагне отримати все й одразу. Цей фреймворк пропонує широкий набір інструментів: від ORM і системи аутентифікації до готової адмін-панелі. Високий рівень безпеки, інтегрований у його архітектуру, робить Django ідеальним вибором для складних вебдодатків. Але з великою кількістю функцій приходить і більша вага – для невеликих проєктів цей фреймворк може виглядати громіздким, а його структура обмежує гнучкість [7, 8].

Pyramid – це вибір для тих, хто прагне контролю над кожною деталлю свого проєкту. Він надає розробникам гнучкість, дозволяючи використовувати різні бази даних, шаблони та інструменти. Pyramid добре підходить для масштабних і складних додатків, проте вимагає більше часу на навчання та освоєння. Через меншу популярність порівняно з іншими фреймворками, доступних ресурсів для вивчення може бути недостатньо для початківців [9].

FastAPI є сучасним інструментом, який відчиняє двері до світу високопродуктивних API-додатків. Його основною перевагою є продуктивність, забезпечена завдяки підтримці асинхронного виконання та інструментів, як-от ASGI і Uvicorn. Автоматична генерація документації API та аннотації типів значно спрощують процес розробки й роблять код зрозумілим. Водночас як відносно новий проєкт, FastAPI має меншу спільноту й вимагає знання сучасних підходів у Python, що може стати викликом для розробників-початківців [4, 10 – 12].

1.3 Аналіз та порівняння вебфреймворків

Оглянувши фреймворки Python, можна скласти порівняльну таблицю (табл. 1.1).

Таблиця 1.1 – Порівняння фреймворків

Особливість	**Django**	**Flask**	**FastAPI**	**Pyramid**
Швидкодія	Середня	Середня	Висока	Середня
Простота використання	Середня	Висока	Висока	Середня
Спільнота	Велика	Велика	Росте	Середня
Документація	Відмінна	Хороша	Відмінна	Хороша
Екосистема	Велика	Середня	Росте	Середня
Асинхронність	Так	Так	Так	Так (за допомогою додаткових бібліотек)
Типізація	Так (опціонально)	Ні	Так	Так (опціонально)
Гнучкість	Середня	Висока	Середня	Висока
Розширюваність	Висока	Середня	Середня	Висока

Після ретельного аналізу технічних рішень для створення інтернет-магазину побутової техніки вибір упав на FastAPI. Цей фреймворк став очевидним фаворитом завдяки поєднанню продуктивності та простоти. Для інтернет-магазину, де високий трафік є звичним явищем, критично важливою є швидкість обробки запитів, і FastAPI ідеально відповідає цій вимозі. Його легкість у використанні дозволяє не лише швидко створити робочу версію продукту, а й підготуватися до майбутнього масштабування без значних витрат.

Ще однією зручністю є автоматична генерація документації API, яка допомагає налагоджувати процес командної розробки, роблячи взаємодію між програмістами плавною та зрозумілою. Асинхронна природа FastAPI додає йому ефективності в умовах великої кількості одночасних запитів, що забезпечує стабільну роботу магазину навіть у пікові моменти. А впровадження типізації дає змогу уникати більшості поширених помилок і

робить код чітким та доступним для розуміння, створюючи основу для його подальшого вдосконалення.

1.4 Технічне завдання

1.4.1 Опис предметної області

Розробляється вебсайт інтернет-магазину побутової техніки. На головній сторінці сайту будуть розташовані зображення товарів та їхній опис. Користувач зможе додавати товари списку бажань, до кошика, змінювати їхню кількість та видаляти товари з кошика. Під час оформлення онлайн-замовлення покупець має заповнити форму, вказавши інформацію про себе та про доставку. Після оформлення замовлення користувач зможе оплатити своє замовлення.

1.4.2 Функціональні вимоги

Інтернет-магазин побутової техніки створено з метою зробити процес купівлі максимально зручним та ефективним як для користувачів, так і для менеджерів. Для покупців це справжній віртуальний шоурум, де вони можуть переглядати багатий каталог товарів із детальними описами, яскравими фотографіями та актуальними цінами. Одним натиском миші можна додати товар до кошика, змінити кількість обраних позицій чи навіть видалити щось, що стало непотрібним. Після цього користувачі легко оформлюють замовлення, заповнюючи просту форму із даними для доставки, та здійснюють оплату через безпечні інтегровані платіжні системи.

Менеджерам магазин відкриває двері до широкого спектра можливостей для управління бізнесом. Вони мають повний контроль над каталогом товарів:

додають нові позиції, редагують інформацію чи видаляють застарілі пропозиції. Водночас обробка замовлень відбувається легко, з можливістю переглядати, змінювати або уточнювати їх статуси. Окрему увагу приділено взаємодії з обліковими записами користувачів – від технічної підтримки до вирішення складних питань. А для стратегічного розвитку менеджери отримують інструменти аналізу продажів і генерації статистичних звітів, які стають основою для обґрунтованих бізнес-рішень.

Цей інтернет-магазин не лише спрощує покупки, але й перетворює їх на зручний і безпечний процес, створюючи новий рівень обслуговування для своїх клієнтів і надійний інструмент для управління бізнесом.

1.4.3 Нефункціональні вимоги

Для створення якісного інтернет-магазину побутової техніки важливо врахувати низку нефункціональних вимог, що формують його загальну зручність і надійність. Продуктивність – це фундамент, на якому будується користувацький досвід. Система повинна реагувати блискавично: сторінки мають відкриватися швидко, а запити оброблятися миттєво, аби кожен візит до магазину ставав приємною подорожжю.

Надійність є запорукою довіри: жодних збоїв, максимальна стабільність і надійне збереження даних користувачів. Безпека стоїть на варті приватності: від захисту особистої інформації до гарантії безпечних транзакцій – жоден аспект не залишається поза увагою. Масштабованість же відкриває двері до зростання, дозволяючи платформі легко адаптуватися до збільшення кількості товарів і користувачів.

Сумісність із будь-якими пристроями, браузерами та операційними системами робить магазин доступним для кожного, незалежно від технічних уподобань. Дотримання стандартів і правил електронної торгівлі гарантує зручність користування, а продуманий дизайн із привабливим оформленням і

зрозумілою навігацією робить взаємодію із сайтом приємною та інтуїтивно зрозумілою. Завершальним штрихом є бездоганна технічна підтримка, яка завжди готова вирішити будь-яке питання, забезпечуючи безперебійну роботу платформи.

1.5 Висновки до розділу 1

Перший розділ роботи став основою для визначення оптимального інструментарію розробки інтернет-магазину побутової техніки. Було досліджено і проаналізовано провідні фреймворки – Django, Flask, Pyramid та FastAPI. Кожен із них мав свої переваги й недоліки, проте вибір зупинили на FastAPI. Його висока продуктивність, легкість у використанні та відповідність сучасним стандартам зробили цей інструмент найбільш відповідним для реалізації проєкту.

Крім вибору технологічної основи, у цьому розділі було детально описано технічне завдання для майбутнього інтернет-магазину. Воно включає не лише аналіз предметної області, але й ретельно сформульовані функціональні та нефункціональні вимоги. Таким чином, створено чіткий орієнтир для побудови системи, яка буде водночас зручною для користувачів і технологічно досконалою.

Цей розділ закладає ґрунт для подальшої роботи, де кожен технічний аспект отримає свою практичну реалізацію.

2 МОДЕЛЮВАННЯ ТА ПРОЄКТУВАННЯ СИСТЕМИ

2.1 Діаграма прецедентів

Діаграма прецедентів, яку також називають діаграмою варіантів використання, розробляється в рамках дипломного проєкту та показує, як користувачі (або актори) взаємодіють із системою. Ця діаграма допомагає зрозуміти функції, які надає система, і показати їх через призму потреб користувача [13 – 15].

Діаграма варіантів використання – це графічний спосіб представлення того, як користувачі (або актори) взаємодіють з системою, розробленою в рамках дипломного проєкту. Ця діаграма допомагає зрозуміти функції, які пропонує система, і показує її через призму потреб користувачів. Такі діаграми мають багато корисних застосувань. Вони дозволяють чітко структурувати бачення системи, допомагають визначити ключові вимоги до її розробки та побудувати ефективну взаємодію між вами, рецензентами та замовниками, навіть якщо вони беруть участь у проєкті [13 – 15].

Основні елементи діаграми – це актори (ті, хто користується системою), прецеденти (те, що система виконує для них), і зв’язки (лінії, що показують взаємодію) [13].

Для інтернет-магазину побутової техніки ця діаграма стане ключовим інструментом. Вона дозволить уявити всі важливі сценарії роботи, конкретизувати функціональні можливості та показати, як користувачі зможуть отримувати доступ до основних функцій магазину, від пошуку товару до його замовлення.

Діаграму прецедентів предметної області «Інтернет магазин побутової техніки» зображено на рисунку 2.1.

Наведена діаграма відображає взаємодію різних ролей (адміністратор, користувач, модератор) з системою, яка, пов’язана з управлінням деякими об’єктами (товари, користувачі, замовлення).

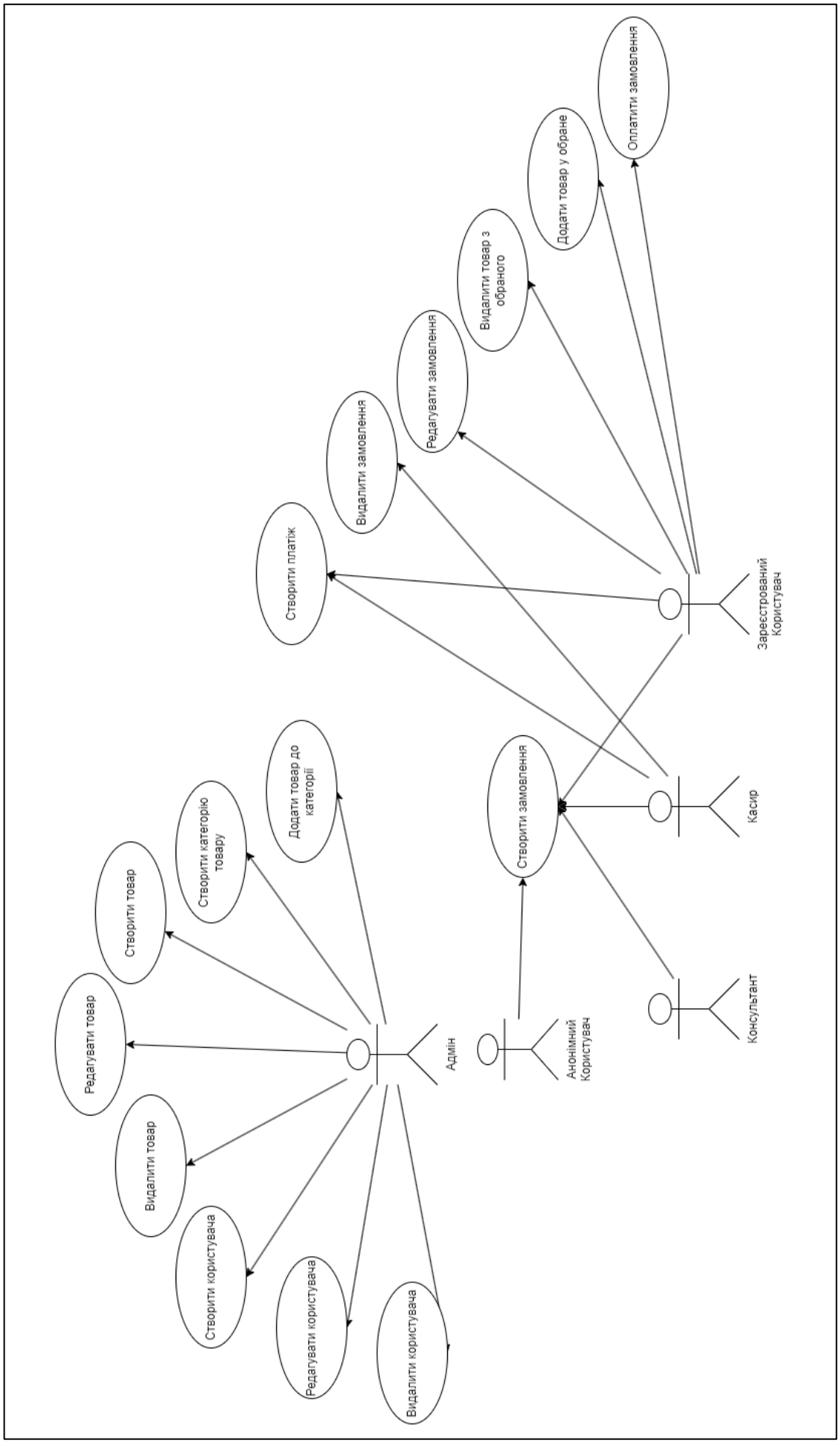


Рисунок 2.1 – Діаграма прецедентів

Бізнес-прецедент (Business Use Case) – це формалізований опис взаємодії між актором (користувачем, системою або зовнішньою подією) та системою, що відображає певний бізнес-процес або завдання. Він описує, як система реагує на запити актора і які результати при цьому отримує.

Розглянемо основні сценарії інтернет-магазину побутової техніки.

Створенння товару.

Короткий опис – прецедент реалізує процес створення нового товару для інтернет-магазину, який адміністратор додає в базу даних.

Цілі і результати – додати новий товар у систему, щоб він був доступний для перегляду та купівлі користувачами.

Основний сценарій:

– адміністратор входить у систему;

– адміністратор переходить до розділу управління товарами;

– адміністратор вводить дані про товар (назва, опис, ціна тощо);

– адміністратор зберігає інформацію про товар.

Спеціальні вимоги – адміністратор має бути авторизованим користувачем з відповідними правами доступу.

Зв’язок з іншими Business Use Case – прецедент пов’язаний з «Редагувати товар» та «Видалити товар», які також є частиною управління товарами.

Створення користувача.

Короткий опис – адміністратор створює новий обліковий запис користувача.

Цілі і результати – додати нового користувача для доступу до системи.

Основний сценарій:

– адміністратор входить у систему;

– адміністратор заповнює дані користувача (ім’я, email тощо);

– адміністратор зберігає обліковий запис.

Спеціальні вимоги – доступ до цієї функції має тільки адміністратор..

Зв’язок з іншими Business Use Case – пов’язаний із редагуванням і видаленням користувачів.

Створення замовлення.

Короткий опис – цей прецедент описує процес створення замовлення користувачем на платформі інтернет-магазину.

Цілі і результати – дозволити користувачу сформувати замовлення з вибраних товарів і надіслати його на обробку.

Основний сценарій:

– користувач заходить на головну сторінку сайту;
– користувач додає товари до кошика;
– користувач переходить до оформлення замовлення;
– користувач заповнює необхідні дані (адресу доставки, контактну інформацію тощо);
– користувач підтверджує замовлення.

Спеціальні вимоги – користувач повинен бути зареєстрованим для завершення замовлення.

Зв’язок з іншими Business Use Case – пов’язаний з «Оплатити замовлення», оскільки оплата виконується після створення замовлення.

Редагування замовлення.

Короткий опис – прецедент дозволяє користувачу вносити зміни в уже створене замовлення.

Цілі і результати – змінити деталі замовлення (наприклад, адресу доставки або склад товарів), якщо це потрібно.

Основний сценарій:

– користувач відкриває існуюче замовлення;
– користувач вносить необхідні зміни в замовлення;
– користувач підтверджує збереження змін.

Спеціальні вимоги – зміни можливі до моменту відправлення замовлення на доставку.

Зв’язок з іншими Business Use Case – може бути пов’язаний з «Створити замовлення» або «Видалити замовлення».

Додавання товару у обране.

Короткий опис – прецедент дозволяє додати товар у список обраних для

легкого доступу в майбутньому.

Цілі і результати – додати товар у список обраних, щоб зберегти його для подальшого перегляду або покупки.

Основний сценарій:

– користувач переглядає товар;

– користувач обирає опцію додати товар у список обраних;

– товар з’являється у списку обраних у профілі користувача.

Спеціальні вимоги – користувач повинен бути зареєстрованим для використання цієї функції.

Зв’язок з іншими Business Use Case – пов’язаний з «Видалити товар з обраного».

Оплата замовлення.

Короткий опис – прецедент реалізує процес оплати замовлення користувачем після його створення.

Цілі і результати – дозволити користувачу завершити покупку, сплативши за вибрані товари.

Основний сценарій:

– користувач переходить до розділу оплати;

– користувач обирає метод оплати та вводить платіжну інформацію;

– система обробляє платіж і підтверджує успішне завершення оплати.

Спеціальні вимоги – необхідний надійний та безпечний платіжний сервіс.

Зв’язок з іншими Business Use Case – пов’язаний з «Створити замовлення», оскільки оплата виконується після оформлення замовлення.

2.2 Діаграма послідовності

Діаграма послідовності (Sequence Diagram) – це один із типів діаграм у мові моделювання UML (Unified Modeling Language). Це своєрідний місток

між аналітикою та візуалізацією, який допомагає зрозуміти, як система функціонує в межах конкретного сценарію. Її графічний вигляд нагадує театральну постановку, де кожен актор виконує свою роль: користувач ініціює дію, система відповідає, а повідомлення передаються від одного об’єкта до іншого [13, 14, 16].

На діаграмах послідовностей буде показано обмін повідомленнями (тобто виклик методів) між декількома об’єктами у окремій обмеженій часом ситуації. Об’єкти є екземплярами класів. Основний наголос на діаграмах послідовностей робиться на порядок і моментах часу, у які повідомлення надсилаються об’єктам.

На діаграмах послідовностей об’єкти буде показано вертикальними штриховими лініями з назвою об’єкта над ними. Вісь часу також має вертикальний напрямок, її спрямовано вниз, повідомлення, які надсилаються від одного об’єкта до іншого, буде позначено стрілками з назвами операції і параметрів [13, 16].

Основні переваги цієї діаграми простягаються від зрозумілої візуалізації до чіткого формулювання логіки функціонування системи. Наприклад, у випадку «Інтернет-магазину побутової техніки» вона ілюструє, як користувач оформляє замовлення, вебсайт обробляє інформацію, а менеджер завершує процес, забезпечуючи покупця необхідним товаром. Цей інструмент – ідеальне поєднання науки та мистецтва в аналізі систем.

Діаграма послідовності складається з кількох ключових елементів. Актори – це учасники, які ініціюють події: люди, процеси або інші системи. Об’єкти – це внутрішні компоненти, що виконують свої функції у сценарії. Повідомлення зв’язують цих учасників у послідовному потоці комунікації [13, 16].

Лінії життя демонструють тривалість участі об’єктів у процесі, а активність відображає періоди, коли вони відіграють найважливіші ролі. Такий структурований підхід не лише полегшує аналіз, а й допомагає побачити кожну деталь як частину єдиного механізму, що працює злагоджено.

Діаграма послідовності предметної області «Інтернет-магазин побутової техніки» (див. рис. 2.2) відображає взаємодію між користувачем, вебсайтом та менеджером під час процесу покупки товарів на онлайн-платформі.

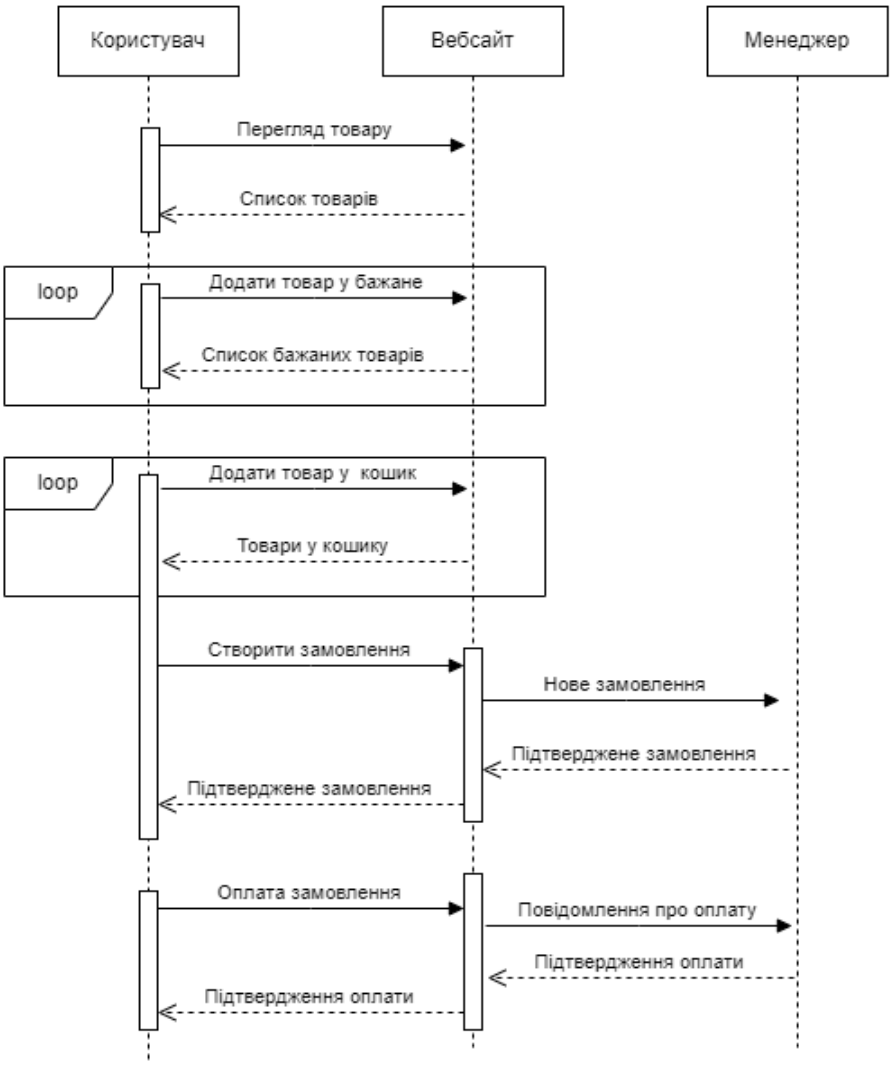


Рисунок 2.2 – Діаграма послідовності оформлення замовлення

Основні кроки процесу виглядають наступним чином:

а) перегляд товарів:
   1) користувач надсилає запит на перегляд товару,
   2) вебсайт відповідає списком доступних товарів;

б) додавання товару до списку бажань:
   1) користувач додає певний товар до списку бажань,
   2) вебсайт оновлює список бажаних товарів і відображає його користувачу,

3) процес додавання товарів до списку бажань може повторюватися (loop), якщо користувач хоче додати декілька товарів;

в) додавання товару до кошика:

1) користувач обирає товар зі списку бажань або безпосередньо зі списку товарів і додає його до кошика,
2) вебсайт оновлює кошик користувача, відображаючи додані товари,
3) ця дія також може повторюватися (loop), якщо користувач додає більше одного товару до кошика;

г) створення замовлення:

1) після того, як користувач додав усі потрібні товари в кошик, він ініціює процес створення замовлення,
2) вебсайт реєструє нове замовлення та передає його менеджеру;

д) підтвердження замовлення:

1) менеджер надсилає підтвердження отримання замовлення на вебсайт,
2) вебсайт повідомляє користувача про підтвердження замовлення;

е) оплата замовлення:

1) користувач переходить до оплати замовлення,
2) вебсайт отримує підтвердження оплати і передає інформацію менеджеру;

ж) підтвердження оплати:

1) менеджер отримує повідомлення про оплату та надсилає підтвердження вебсайту,
2) вебсайт відображає користувачу підтвердження успішної оплати.

Основні елементи діаграми:

– користувач – ініціює всі основні дії (перегляд, додавання в список бажань і кошик, створення замовлення, оплата);
– вебсайт – обробляє запити користувача, відображає необхідну інформацію, взаємодіє з менеджером;

– менеджер – обробляє нові замовлення та підтверджує оплату, що завершує процес покупки.

Діаграма послідовності корисна для демонстрації покрокової взаємодії між користувачем, вебсайтом та менеджером, що спрощує розуміння процесу виконання замовлення в онлайн-магазині.

2.3 Моделювання структури даних

ER-діаграма (Entity-Relationship Diagram, ERD) є графічною моделлю, що використовується для зображення структури даних та зв’язків між ними у базі даних. Вона є важливим інструментом під час проєктування баз даних, дозволяючи відобразити логічну організацію інформації та визначити взаємодію між сутностями [17 – 19].

Основними компонентами ER-діаграми є [17 – 19]:

– сутності (Entities) – це об’єкти або елементи, що зберігають дані, наприклад, таблиці в базі даних;
– атрибути (Attributes) – цевластивості або характеристики сутностей, що описують їхню природу;
– зв’язки (Relationships) – це взаємодії між сутностями, що визначають спосіб їхнього зв’язування;
– кардинальність – вказує на кількісні обмеження зв’язків між сутностями.

ER-діаграма – це ніби путівник у світі даних, що водночас поєднує простоту і потужність. Її графічна структура дозволяє легко розгледіти основні взаємозв’язки між сутностями, мов би візуалізуючи складний задум у чіткій і зрозумілій формі.

Зрозумілість і доступність. Зображення даних у формі ER-діаграми допомагає аналітикам, розробникам і навіть замовникам швидко осягнути логіку структури бази. Це значно спрощує комунікацію та спільне прийняття рішень.

Забезпечення правильного планування. На етапі створення моделі ER-діаграми дозволяють уникнути зайвих або дубльованих даних, що економить ресурси системи та знижує ризики помилок.

Пошук найкращих рішень. Завдяки ER-діаграмі можливо знайти оптимальні способи зберігання інформації, що впливає не лише на якість роботи бази даних, але й на її швидкодію.

Гнучкість для майбутнього. Добре опрацьована діаграма забезпечує легкість внесення змін. Ви ніби отримуєте карту, яка дозволяє швидко орієнтуватися в будь-яких оновленнях чи розширеннях бази.

ER-діаграма – це не лише технічний інструмент, але й спосіб зробити складне зрозумілим, забезпечуючи гармонійний розвиток системи на всіх етапах її життєвого циклу.

ER-діаграма виконує роль своєрідного фундаменту для побудови баз даних, що дозволяє [17 – 19]:

– візуалізувати складні взаємозв’язки та структуру даних;
– ефективно планувати архітектуру бази даних;
– забезпечувати оптимальну обробку даних з максимальним результатом;
– гарантувати цілісність та узгодженість даних на всіх етапах їх обробки.

Цей інструмент є незамінним для проєктів, що вимагають збереження й обробки великого обсягу структурованої інформації, надаючи їм потужну, логічну та надійну основу для створення ефективних баз даних.

Виходячи з аналізу вимог для інтернет-магазину побутової техніки, можна виділити наступні концептуальні сутності.

Сутність «Товар (Product)» зберігає інформацію про товари, доступні для продажу, а саме: назву товару, опис, ціну, стан (новий чи вживаний), кількість на складі, а також категорію, до якої він належить.

Сутність «Категорія (Category)» зберігає інформацію про категорії товарів, включаючи назву, опис, ідентифікатор батьківської категорії (для

ієрархічної структури категорій), а також роль, яка може слугувати для додаткової класифікації категорій.

Сутність «Користувач (User)» зберігає інформацію про зареєстрованих користувачів системи, а саме: ім’я користувача, електронну пошту, пароль, роль користувача (наприклад, клієнт, адміністратор), статус активності облікового запису та дату створення.

Сутність «Замовлення (Order)» зберігає інформацію про замовлення, що здійснюються користувачами, а саме: дату створення замовлення, користувача, який зробив замовлення, загальну суму замовлення та його статус (наприклад, обробляється, доставлено).

Сутність «Елемент замовлення (OrderItem)» зберігає деталі окремих товарів у складі замовлення, а саме: товар, кількість цього товару у замовленні та його ціну за одиницю на момент замовлення.

Сутність «Платіж (Payment)» зберігає інформацію про платежі, здійснені за замовлення, а саме: суму платежу, статус (наприклад, успішний, неуспішний), метод оплати (наприклад, картка, готівка), ідентифікатор транзакції, а також дату проведення платежу.

Сутність «Обраний товар (LikedProduct)» представляє список обраних товарів для користувача і зберігає зв’язок між користувачем та товарами, які він додав у вибране.

Ці сутності забезпечують логічну структуру для зберігання та управління інформацією в інтернет-магазині, покриваючи основні бізнес-процеси, такі як управління товарами, користувачами, замовленнями, платежами та обраними товарами.

У наступних таблицях (2.1 – 2.7) будуть розглянуті необхідні для побудови ER-діаграми сутності, атрибути, типи даних та ключі.

Таблиця 2.1 – Таблиця для сутності «Продукт (Product)»

Атрибут	**Тип даних**	**Ключ**
id	Числовий	Первинний

Продовження табл. 2.1

Атрибут	Тип даних	Ключ
name	Текстовий	
description	Текстовий	
price	Числовий	
condition	Текстовий	
stock_quantity	Числовий	
category_id	Числовий	Зовнішній

Таблиця 2.2 – Таблиця для сутності «Категорія (Category)»

Атрибут	Тип даних	Ключ
id	Числовий	Первинний
name	Текстовий	
description	Текстовий	
parent_id	Числовий	Зовнішній (FK)
role	Текстовий	

Таблиця 2.3 – Таблиця для сутності «Order (Замовлення)»

Атрибут	Тип даних	Ключ
id	Числовий	Первинний
user_id	Числовий	Зовнішній
created_at	Дата	
status	Текстовий	
total_price	Числовий	

Таблиця 2.4 – Таблиця для сутності «LikedProduct (Вподобаний продукт)»

Атрибут	Тип даних	Ключ
user_id	Числовий	Первинний, Зовнішній
product_id	Числовий	Первинний, Зовнішній

Таблиця 2.5 – Таблиця для сутності «OrderItem (Елемент замовлення)»

Атрибут	Тип даних	Ключ
id	Числовий	Первинний
order_id	Числовий	Зовнішній
product_id	Числовий	Зовнішній
quantity	Числовий	
price_per_unit	Числовий	

Таблиця 2.6 – Таблиця для сутності «User (Користувач)»

Атрибут	Тип даних	Ключ
id	Числовий	Первинний
Атрибут	Тип даних	Ключ
username	Текстовий	
email	Текстовий	Унікальний
password	Текстовий	
role	Текстовий	
is_active	Логічний	
created_at	Дата	

Таблиця 2.7 – Таблиця для сутності «Payment (Платіж)»

Атрибут	Тип даних	Ключ
id	Числовий	Первинний
order_id	Числовий	Зовнішній
amount	Числовий	
status	Текстовий	
payment_method	Текстовий	
transaction_id	Текстовий	
created_at	Дата	

Далі розглянемо зв’язки між сутностями.

Між сутностями Category та Product зв’язок 1:Б (один-до-багатьох), оскільки одна категорія може містити декілька продуктів, але кожен продукт належить лише до однієї категорії.

Між сутностями Product та OrderItem зв’язок 1:Б (один-до-багатьох), оскільки один продукт може бути включений у декілька позицій замовлень (OrderItem), але кожна позиція замовлення відповідає одному продукту.

Між сутностями Order та OrderItem зв’язок 1:Б (один-до-багатьох), тому що одне замовлення може включати декілька позицій замовлень (OrderItem), але кожна позиція замовлення належить лише до одного замовлення.

Між сутностями User та Order зв’язок 1:Б (один-до-багатьох), оскільки один користувач (User) може створити декілька замовлень (Order), але кожне замовлення прив’язане до одного користувача.

Між сутностями Order та Payment зв’язок 1:1, оскільки для кожного замовлення існує один платіж, і кожен платіж відповідає одному замовленню.

Між сутностями User та LikedProduct зв’язок 1:Б (один-до-багатьох), тому що один користувач може додати до улюблених кілька продуктів, але кожен запис про улюблений продукт стосується лише одного користувача.

Між сутностями Product та LikedProduct зв’язок Б:1 (багато-до-одного), оскільки один продукт може бути улюбленим для багатьох користувачів, але кожен запис в таблиці LikedProduct відноситься лише до одного продукту.

ER-діаграму предметної області «Інтернет-магазин побутової техніки» зображено на рисунку 2.3.

Представлена ER-діаграма є концептуальною моделлю даних, яка відображає структуру інформації в інтернет-магазині побутової техніки. Вона деталізує основні сутності та їхні взаємозв’язки. Діаграма слугує основою для подальшого проєктування фізичної моделі бази даних та розробки програмного забезпечення.

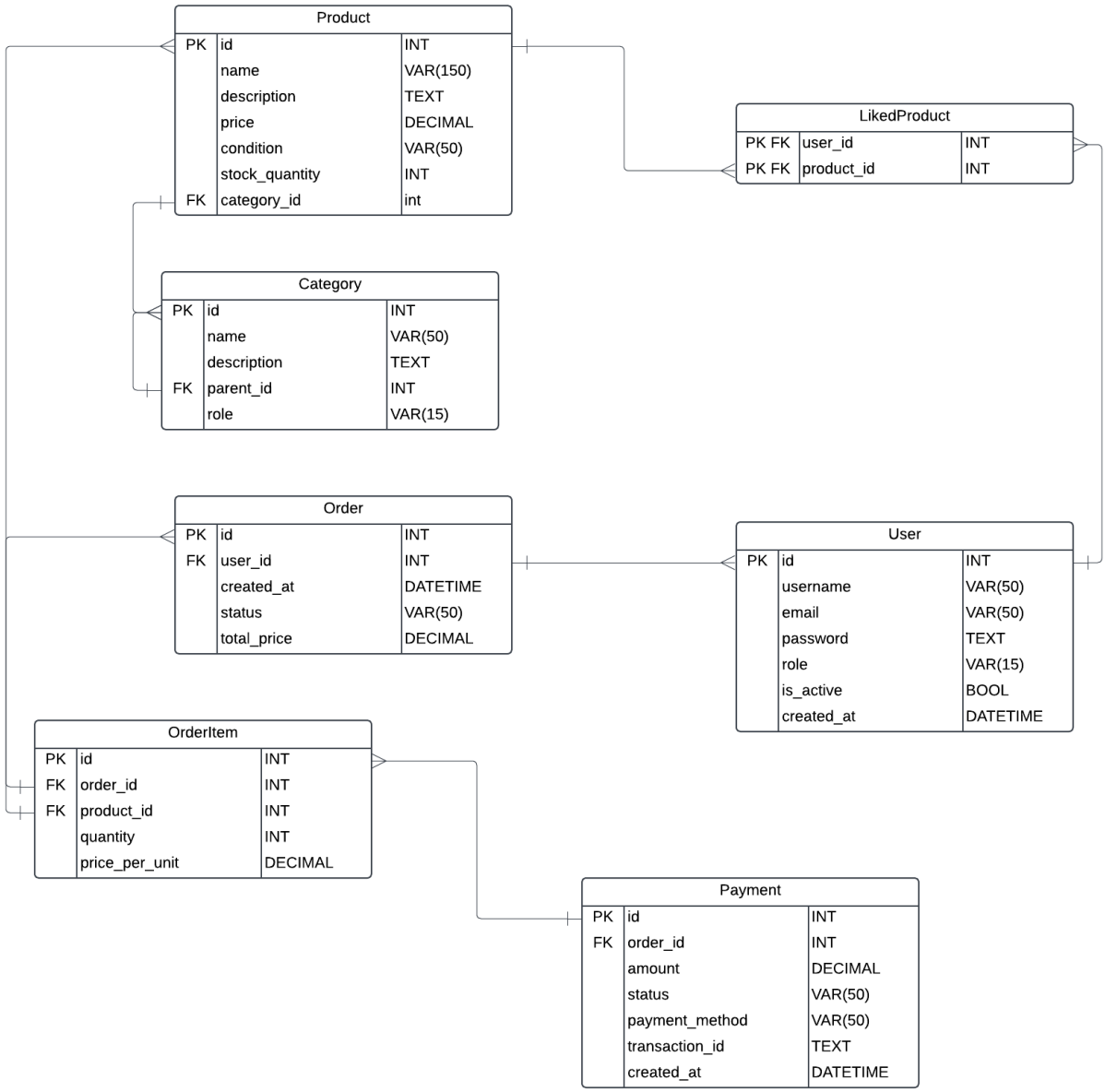


Рисунок 2.3 – ER-діаграма

2.4 Діаграма розгортання

Діаграма розгортання (Deployment Diagram) – це тип діаграми в моделюванні UML (Unified Modeling Language), який використовується для відображення фізичної архітектури системи. Вона показує, як програмне забезпечення буде розгортатися на апаратному обладнанні, які компоненти будуть встановлені на яких серверах чи пристроях і як ці елементи взаємодіють між собою. Зазвичай діаграми розгортання включають в себе вузли (nodes), компоненти та зв’язки між ними [20].

Основними елементами діаграми розгортання є [20]:

– вузли (Nodes) – це фізичні або віртуальні пристрої, такі як сервери, комп’ютери, мобільні пристрої тощо;
– компоненти (Components) – це модулі програмного забезпечення, які встановлюються на вузли;
– зв’язки (Links) – це комунікаційні канали між вузлами, які можуть представляти мережеві з’єднання (наприклад, HTTP, TCP/IP) або інші типи зв’язків.

Діаграма розгортання необхідна для розуміння фізичного розміщення програмного забезпечення та обладнання у системі. Вона допомагає інженерам і розробникам зрозуміти, як програма працюватиме в реальних умовах, на яких серверах будуть виконуватись певні процеси, і як ці вузли взаємодіятимуть між собою. Це важливо на етапі проєктування для планування архітектури системи і забезпечення надійності, масштабованості та безпеки.

Діаграму розгортання предметної області «Інтернет-магазин побутової техніки» зображено на рисунку 2.4.

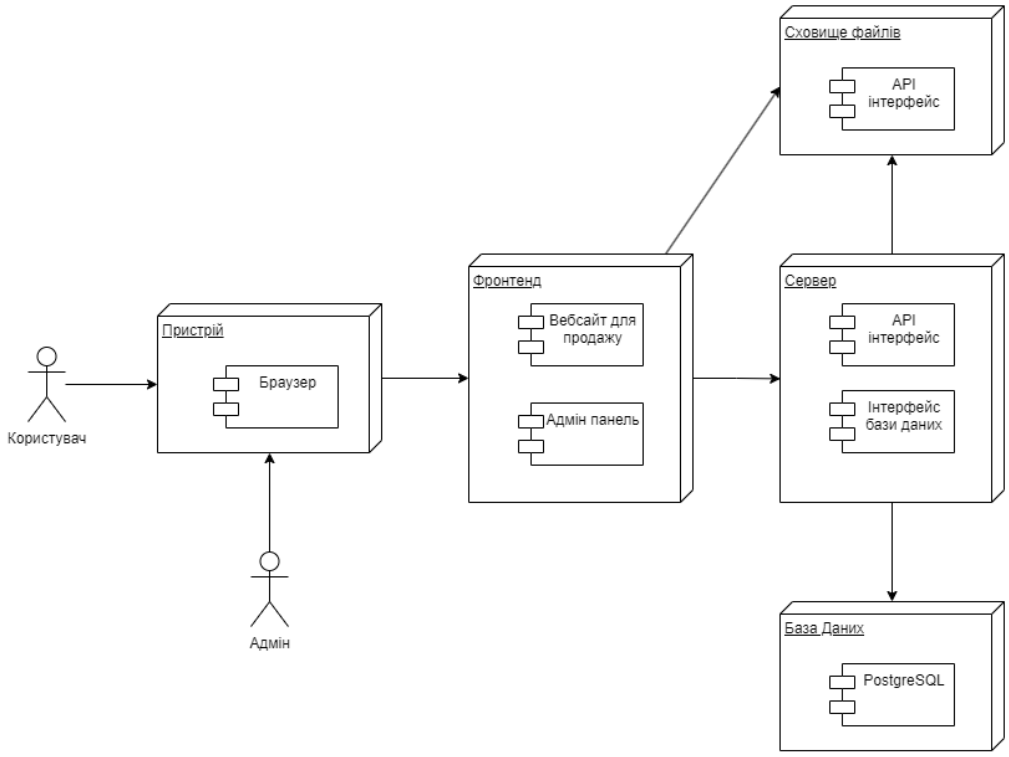


Рисунок 2.4 – Діаграма розгортання

Діаграма розгортання являє собою картину архітектури системи, в якій кожен елемент відіграє свою роль у загальному процесі взаємодії. Детально опишемо її складові. Діаграма містить два основних типи користувачів: Користувач та Адмін. Користувачі застосовують свої пристрої для взаємодії з системою через браузер. Адміністратор, у свою чергу, також здійснює доступ до системи через браузер, але вже для керування адмін-панеллю.

Далі ми маємо Пристрій. Це будь-який пристрій – комп'ютер, планшет чи смартфон, через який здійснюється взаємодія з системою. Пристрій оснащений браузером, який дає змогу заходити на вебсайт і в адмін-панель.

Фронтенд представлений у вигляді вебсайту та адмін-панелі. Вебсайт є зручним інтерфейсом для користувачів, що дозволяє їм переглядати асортимент товарів, додавати їх у кошик і здійснювати покупки. Адмін-панель – це спеціалізований інтерфейс для адміністратора, який відкриває доступ до управління товарами, перегляду замовлень та виконання інших адміністративних завдань.

Сервер виконує роль обробки запитів, які надходять від фронтенду.Він має API інтерфейс, що забезпечує взаємодію фронтенду із системою, дозволяючи отримувати необхідні дані та передавати запити. Сервер включає інтерфейс бази даних, який відкриває доступ до збереженої інформації.

Сховище файлів – зовнішній компонент для зберігання файлів, таких як зображення товарів або інші медіа-файли. API інтерфейс надає доступ до цих файлів з інших компонентів системи.

PostgreSQL служить основною базою даних для зберігання структурованої інформації про користувачів, замовлення, товари та інші важливі дані. Сервер здійснює взаємодію з базою даних через спеціалізований інтерфейс для запису та отримання інформації.

Опишемо зв'язки між компонентами. Користувачі та адміністратори взаємодіють із системою через браузери на своїх пристроях, відправляючи запити до фронтенду. Фронтенд отримує ці запити та передає їх на сервер для обробки. Сервер здійснює зв'язок із базою даних для отримання і збереження

даних, що необхідні для виконання запитів. Крім того, сервер взаємодіє з сховищем файлів, де зберігаються медіафайли, що можуть бути асоційовані з товарами або іншими елементами системи.

Ця система має декілька важливих переваг, які забезпечують її ефективність та адаптивність. Її модульна структура дає змогу чітко розподіляти функції між окремими компонентами, що спрощує масштабування та обслуговування. Окрім цього, завдяки використанню API-зв'язків між компонентами, досягається високий рівень безпеки, адже це дозволяє зручніше контролювати доступ до чутливих даних. І нарешті, система характеризується гнучкістю: завдяки можливості легко оновлювати або замінювати окремі компоненти, такі як базу даних чи сховище файлів, без значних змін в основній архітектурі, вона залишається відкритою до подальших змін та вдосконалень. Діаграма розгортання є наочним відображенням взаємодії основних елементів системи, що дозволяє забезпечити безперешкодний доступ до функцій вебсайту як для кінцевих користувачів, так і для адміністраторів.

2.5 Висновки до розділу 2

У другому розділі було здійснено проєктування ключових структур і процесів для онлайн-магазину побутової техніки. В рамках цього дослідження були розроблені й проаналізовані діаграми прецедентів, послідовності, ER-діаграма та діаграма розгортання.

Діаграма прецедентів дозволила виявити основні ролі та функції системи, що дає змогу побачити, як користувачі взаємодіють із системою через конкретні сценарії.

Діаграма послідовності, у свою чергу, відобразила динамічні моменти роботи системи, демонструючи послідовність кроків при виконанні ключових операцій, таких як оформлення замовлення та оплата.

ER-діаграма, надаючи розуміння структури даних, показала сутності та зв’язки між ними, що забезпечує точну основу для проєктування бази даних.

Діаграма розгортання, в свою чергу, дозволила розглянути фізичне розміщення компонентів системи на різних серверах, а також визначити, як клієнтська і серверна частини взаємодіють між собою.

Завдяки проведеному моделюванню вдалося чітко визначити всі основні елементи системи, їх зв’язки та послідовність дій, що є надзвичайно важливим для подальшого розвитку системи. Ці результати становлять надійний фундамент для ефективного проєктування і реалізації онлайн-магазину, який відповідатиме всім функціональним і нефункціональним вимогам.

3 РЕАЛІЗАЦІЯ СИСТЕМИ

3.1 Реалізація бази даних

Ініціалізуємо базовий клас від якого успадковуються всі моделі, що дозволяє легко створювати таблиці у базі даних. Перерахунки (Enums) для ролей користувачів, стану продуктів та статусу платежів допомагають визначити обмежені значення для конкретних атрибутів. Наприклад, UserRole вказує роль користувача (адміністратор, касир, звичайний користувач), а PaymentStatus відображає статус оплати (очікування, завершено, невдача тощо). Таблиця асоціацій для обраних продуктів реалізує зв’язок «багато-до-багатьох» між користувачами та продуктами, які їм сподобалися.

Модель User містить інформацію про користувачів: ідентифікатор, ім’я користувача, електронну пошту, роль, стан активності. Встановлює зв’язок з таблицею замовлень (Order) та обраними продуктами (Product).

Модель Category описує категорії товарів та їхні взаємозв’язки. Містить атрибут parent_id, який реалізує ієрархію категорій (наприклад, підкатегорії). Встановлює зв’язок з продуктами.

Модель Product визначає основні характеристики продуктів, такі як назва, опис, ціна, кількість на складі та стан продукту. Ця модель також зв’язується з категоріями та замовленнями через OrderItem.

Модель Order описує замовлення, містить атрибути user_id, created_at, status, total_price. Встановлює зв’язки з користувачем (User), елементами замовлення (OrderItem) та платіжною інформацією (Payment).

Модель OrderItem відповідає за товари, що входять у кожне замовлення. Кожен запис вказує на певний продукт та його кількість у замовленні.

Модель Review зберігає відгуки користувачів про продукти. Містить інформацію про рейтинг (від 1 до 5), коментар користувача, час створення, а також зв’язки з користувачем і продуктом.

Модель Payment відповідає за інформацію про оплату, включаючи суму, метод оплати, статус та ідентифікатор транзакції. Модель містить зв’язок із замовленням, якому належить оплата.

Ці моделі забезпечують цілісну структуру для зберігання та обробки даних в онлайн-магазині, з чіткими взаємозв’язками між користувачами, продуктами, категоріями, замовленнями, відгуками та платежами.

Фрагмент коду, який відображає реалізацію бази даних, послідовно наведено на рисунках 3.1 – 3.4 відповідно.

```
    from sqlalchemy import (
        Column, Integer, String, ForeignKey, Float, Enum,
Text, DateTime, Boolean, Table
    )
    from sqlalchemy.orm import relationship
    from sqlalchemy.ext.declarative import declarative_base
    from sqlalchemy.types import Enum as SQLAEnum
    import enum
    from datetime import datetime

    Base = declarative_base()

    # Enums for user roles, product condition, and payment
status
    class UserRole(enum.Enum):
        ADMIN = "admin"
        STAFF = "staff"
        CASHIER = "cashier"
        USER = "user"

    class ProductCondition(enum.Enum):
        NEW = "new"
        USED = "used"
        REFURBISHED = "refurbished"

    class PaymentStatus(enum.Enum):
        PENDING = "pending"
        COMPLETED = "completed"
        FAILED = "failed"
        REFUNDED = "refunded"

    # Association table for liked products
    liked_products_table = Table(
        'liked_products', Base.metadata,
        Column('user_id', Integer, ForeignKey('users.id'),
primary_key=True),
```

Рисунок 3.1

```
        Column('product_id', Integer,
ForeignKey('products.id'), primary_key=True)
    )

    # User table with roles
    class User(Base):
        __tablename__ = 'users'

        id = Column(Integer, primary_key=True)
        username = Column(String(50), nullable=False,
unique=True)
        email = Column(String(100), nullable=False,
unique=True)
        password_hash = Column(String(255), nullable=False)
        role = Column(SQLAEnum(UserRole),
default=UserRole.USER)
        is_active = Column(Boolean, default=True)
        created_at = Column(DateTime, default=datetime.utcnow)

        # Relationships
        orders = relationship("Order", back_populates="user")
        liked_products = relationship("Product",
secondary=liked_products_table,
back_populates="liked_by_users")

    # Product Category with self-referencing FK for
breadcrumbs
    class Category(Base):
        __tablename__ = 'categories'

        id = Column(Integer, primary_key=True)
        name = Column(String(50), nullable=False, unique=True)
        description = Column(Text)
        parent_id = Column(Integer,
ForeignKey('categories.id'))

        # Relationships
        parent = relationship("Category", remote_side=[id],
backref="subcategories")
        products = relationship("Product",
back_populates="category")

    # Products table
    class Product(Base):
        __tablename__ = 'products'

        id = Column(Integer, primary_key=True)
        name = Column(String(100), nullable=False)
        description = Column(Text, nullable=True)
        price = Column(Float, nullable=False)
        condition = Column(SQLAEnum(ProductCondition),
default=ProductCondition.NEW)
```

Рисунок 3.2

```
        stock_quantity = Column(Integer, default=0)
        category_id = Column(Integer,
ForeignKey('categories.id'), nullable=False)

        # Relationships
        category = relationship("Category",
back_populates="products")
        order_items = relationship("OrderItem",
back_populates="product")
        liked_by_users = relationship("User",
secondary=liked_products_table,
back_populates="liked_products")

    # Orders table
    class Order(Base):
        __tablename__ = 'orders'

        id = Column(Integer, primary_key=True)
        user_id = Column(Integer, ForeignKey('users.id'),
nullable=False)
        created_at = Column(DateTime, default=datetime.utcnow)
        status = Column(String(50), default="pending")
        total_price = Column(Float, nullable=False)

        # Relationships
        user = relationship("User", back_populates="orders")
        items = relationship("OrderItem",
back_populates="order")
        payment = relationship("Payment", uselist=False,
back_populates="order")

    # Order Items table to handle products in each order
    class OrderItem(Base):
        __tablename__ = 'order_items'

        id = Column(Integer, primary_key=True)
        order_id = Column(Integer, ForeignKey('orders.id'),
nullable=False)
        product_id = Column(Integer,
ForeignKey('products.id'), nullable=False)
        quantity = Column(Integer, nullable=False)
        price_per_unit = Column(Float, nullable=False)

        # Relationships
        order = relationship("Order", back_populates="items")
        product = relationship("Product",
back_populates="order_items")

    # Reviews table for product reviews
    class Review(Base):
        __tablename__ = 'reviews'
```

Рисунок 3.3

```
        id = Column(Integer, primary_key=True)
        product_id = Column(Integer,
ForeignKey('products.id'), nullable=False)
        user_id = Column(Integer, ForeignKey('users.id'),
nullable=False)
        rating = Column(Integer, nullable=False)  # Assume 1-5
scale
        comment = Column(Text, nullable=True)
        created_at = Column(DateTime, default=datetime.utcnow)

        # Relationships
        user = relationship("User")
        product = relationship("Product")

    # Payment table
    class Payment(Base):
        __tablename__ = 'payments'

        id = Column(Integer, primary_key=True)
        order_id = Column(Integer, ForeignKey('orders.id'),
nullable=False)
        amount = Column(Float, nullable=False)
        status = Column(SQLAEnum(PaymentStatus),
default=PaymentStatus.PENDING)
        payment_method = Column(String(50), nullable=False)  #
e.g., "credit_card", "paypal", "bank_transfer"
        transaction_id = Column(String(100), nullable=True,
unique=True)  # External payment system transaction ID
        created_at = Column(DateTime, default=datetime.utcnow)

        # Relationships
        order = relationship("Order",
back_populates="payment")
```

Рисунок 3.4

Ці фрагменти коду представляють основні моделі бази даних для онлайн-магазину побутової техніки, створені за допомогою SQLAlchemy. Кожна модель відповідає окремій сутності в базі даних та описує структуру даних, яка використовується в проєкті.

Ці фрагменти містять визначення моделей бази даних (User, Category, Product, Order, OrderItem, Review, Payment) як класів SQLAlchemy, опис енумів для категорій, ролей користувачів, стану продуктів і статусу оплати, створення асоціативної таблиці liked_products для реалізації багатозначних відносин.

3.2 Реалізація обробки замовлень та управління товарами

Реалізуємо функціонал для створення, перегляду та видалення замовлень користувачів. Він буде включати серверний код для роботи з даними замовлень, що дозволяє взаємодіяти з базою даних.

Імпортуємо бібліотеки та залежності. Бібліотеки FastAPI, SQLAlchemy, і Pydantic забезпечують інструменти для створення серверної частини додатку, управління сесіями бази даних, валідацію та створення моделей для обробки запитів і відповідей.

Моделі OrderItemResponse та OrderResponse визначають структуру даних для валідації і забезпечують зручність роботи з об’єктами SQLAlchemy. Фрагмент orm_mode = True дозволяє FastAPI автоматично перетворювати об’єкти SQLAlchemy у формати JSON для відповідей.

Моделі OrderCreateItem та OrderCreate описують структуру даних для запитів на створення замовлень. Вони містять інформацію про продукти та їх кількість.

Додаємо маршрут для створення нового замовлення. Цей маршрут дозволяє створити нове замовлення для певного користувача. Перевіряється наявність користувача в базі даних за допомогою db.query. Якщо користувач не знайдений, сервер повертає помилку 404 Not Found.

Кожен продукт у замовленні перевіряється на наявність у базі даних, а також обчислюється загальна вартість замовлення.

Створюється новий об’єкт замовлення, який додається до сесії бази даних і зберігається за допомогою db.commit. Оновлений об’єкт повертається як відповідь.

Створюємо маршрут, який повертає всі замовлення для конкретного користувача, якщо вони існують та маршрут, який дозволяє отримати конкретне замовлення за ID користувача та ID замовлення. Також додаємо маршрут, який дозволяє видаляти замовлення за ID користувача та замовлення. Якщо замовлення знайдено, воно видаляється з бази даних, і користувач отримує повідомлення про успішне видалення.

Цей серверний код забезпечує API для роботи з замовленнями в системі електронної комерції, дозволяючи створювати, переглядати та видаляти замовлення, що робить його важливим компонентом для управління замовленнями користувачів.

Фрагмент коду, який відображає управління замовленнями, послідовно наведено на рисунках 3.5 – 3.7 відповідно.

```
    from fastapi import FastAPI, Depends, HTTPException,
status
    from sqlalchemy.orm import Session
    from typing import List
    from pydantic import BaseModel
    from database import get_db  # Database dependency
(session management)
    from models import Order, OrderItem, User, Product  #
Import your SQLAlchemy models

    app = FastAPI()

    # Pydantic models for request/response validation
    class OrderItemResponse(BaseModel):
        product_id: int
        quantity: int
        price_per_unit: float

        class Config:
            orm_mode = True

    class OrderResponse(BaseModel):
        id: int
        user_id: int
        status: str
        total_price: float
        created_at: str
        items: List[OrderItemResponse]

        class Config:
            orm_mode = True

    # Create Order
    class OrderCreateItem(BaseModel):
        product_id: int
        quantity: int
    class OrderCreate(BaseModel):
        items: List[OrderCreateItem]
```

Рисунок 3.5

```
    @app.post("/users/{user_id}/orders/",
response_model=OrderResponse,
status_code=status.HTTP_201_CREATED)
    def create_order(user_id: int, order: OrderCreate, db:
Session = Depends(get_db)):
        db_user = db.query(User).filter(User.id ==
user_id).first()
        if not db_user:
            raise
HTTPException(status_code=status.HTTP_404_NOT_FOUND,
detail="User not found")

        # Calculate total price
        total_price = 0
        order_items = []
        for item in order.items:
            db_product = db.query(Product).filter(Product.id
== item.product_id).first()
            if not db_product:
                raise
HTTPException(status_code=status.HTTP_404_NOT_FOUND,
detail=f"Product with ID {item.product_id} not found")
            total_price += db_product.price * item.quantity

order_items.append(OrderItem(product_id=item.product_id,
quantity=item.quantity, price_per_unit=db_product.price))

        # Create order
        db_order = Order(user_id=user_id, status="pending",
total_price=total_price, items=order_items)
        db.add(db_order)
        db.commit()
        db.refresh(db_order)
        return db_order

    # Read all orders for a user
    @app.get("/users/{user_id}/orders/",
response_model=List[OrderResponse])
    def read_user_orders(user_id: int, db: Session =
Depends(get_db)):
        orders = db.query(Order).filter(Order.user_id ==
user_id).all()
        if not orders:
            raise
HTTPException(status_code=status.HTTP_404_NOT_FOUND, detail="No
orders found for this user")
        return orders
    # Read single order by order ID
    @app.get("/users/{user_id}/orders/{order_id}",
response_model=OrderResponse)
    def read_user_order(user_id: int, order_id: int, db:
Session = Depends(get_db)):
```

Рисунок 3.6

```
        order = db.query(Order).filter(Order.user_id ==
user_id, Order.id == order_id).first()
        if not order:
            raise
HTTPException(status_code=status.HTTP_404_NOT_FOUND,
detail="Order not found")
        return order

    # Delete order by order ID
    @app.delete("/users/{user_id}/orders/{order_id}",
status_code=status.HTTP_204_NO_CONTENT)
    def delete_order(user_id: int, order_id: int, db: Session
= Depends(get_db)):
        order = db.query(Order).filter(Order.user_id ==
user_id, Order.id == order_id).first()
        if not order:
            raise
HTTPException(status_code=status.HTTP_404_NOT_FOUND,
detail="Order not found")

        db.delete(order)
        db.commit()
        return {"message": "Order deleted successfully"}
```

Рисунок 3.7

Ці фрагменти коду на сервері реалізує API для обробки замовлень користувачів. Тут представлено створення, читання та видалення замовлень для конкретного користувача. Кожне замовлення містить інформацію про товари, які були додані до нього, включаючи кількість і ціну. При створенні замовлення, API обробляє вхідні дані, перевіряє наявність відповідних продуктів і обчислює загальну вартість замовлення. Код також обробляє можливі помилки, наприклад, якщо користувач або продукт не знайдені в базі даних. Такий підхід дозволяє забезпечити надійне управління замовленнями та зручний API для взаємодії з фронтендом.

Наступним кроком буде розробка серверної реалізації API для управління продуктами.

Імпортуємо необхідні модулі, включаючи FastAPI для створення API, SQLAlchemy для роботи з базою даних та Pydantic для валідації вхідних даних. get_db забезпечує управління сесіями бази даних, а Product – модель продукту з SQLAlchemy.

ProductBase визначає основні поля, що описують продукт: назва, опис, ціна, стан, кількість на складі та категорія. Ця модель є базовою для створення, оновлення і відповіді про продукт. ProductCreate наслідує від ProductBase, і є моделлю для створення нового продукту. ProductUpdate наслідує ProductBase, але всі її поля є необов'язковими, що дозволяє оновлювати тільки вказані атрибути продукту. ProductResponse додає поле id для відповіді з продуктом і містить orm_mode = True, що дозволяє перетворювати об'єкти SQLAlchemy в JSON для відповіді.

Додаємо маршрути для створення нового продукту, для перегляду продукту за його ID, для перегляду всіх продуктів, для оновлення продукту за його ID та для видалення продукту за його ID:

Таким чином, цей ми реалізуємо основні CRUD-операції для управління продуктами, що є важливою складовою функціональності онлайн-магазину

Фрагмент коду, який відображає створення і управління продуктами, послідовно наведено на рисунках 3.8 – 3. 10 відповідно.

```
    from fastapi import FastAPI, Depends, HTTPException,
status
    from sqlalchemy.orm import Session
    from typing import List
    from pydantic import BaseModel
    from database import get_db  # Database dependency
(session management)
    from models import Product  # Product model from the
SQLAlchemy schema

    app = FastAPI()

    # Pydantic model for request/response validation
    class ProductBase(BaseModel):
        name: str
        description: str = None
        price: float
        condition: str
        stock_quantity: int
        category_id: int

    class ProductCreate(ProductBase):
        pass
```

Рисунок 3.8

```
    class ProductUpdate(ProductBase):
        name: str = None
        description: str = None
        price: float = None
        condition: str = None
        stock_quantity: int = None
        category_id: int = None

    class ProductResponse(ProductBase):
        id: int

        class Config:
            orm_mode = True

    # Create Product
    @app.post("/products/", response_model=ProductResponse,
status_code=status.HTTP_201_CREATED)
    def create_product(product: ProductCreate, db: Session =
Depends(get_db)):
        db_product = Product(**product.dict())
        db.add(db_product)
        db.commit()
        db.refresh(db_product)
        return db_product

    # Read Product by ID
    @app.get("/products/{product_id}",
response_model=ProductResponse)
    def read_product(product_id: int, db: Session =
Depends(get_db)):
        product = db.query(Product).filter(Product.id ==
product_id).first()
        if product is None:
            raise
HTTPException(status_code=status.HTTP_404_NOT_FOUND,
detail="Product not found")
        return product

    # Read All Products
    @app.get("/products/",
response_model=List[ProductResponse])
    def read_products(skip: int = 0, limit: int = 10, db:
Session = Depends(get_db)):
        products =
db.query(Product).offset(skip).limit(limit).all()
        return products

    # Update Product by ID
    @app.put("/products/{product_id}",
response_model=ProductResponse)
    def update_product(product_id: int, product:
ProductUpdate, db: Session = Depends(get_db)):
```

Рисунок 3.9

```
        db_product = db.query(Product).filter(Product.id ==
product_id).first()
        if db_product is None:
            raise
HTTPException(status_code=status.HTTP_404_NOT_FOUND,
detail="Product not found")

        for key, value in
product.dict(exclude_unset=True).items():
            setattr(db_product, key, value)

        db.commit()
        db.refresh(db_product)
        return db_product

    # Delete Product by ID
    @app.delete("/products/{product_id}",
status_code=status.HTTP_204_NO_CONTENT)
    def delete_product(product_id: int, db: Session =
Depends(get_db)):
        db_product = db.query(Product).filter(Product.id ==
product_id).first()
        if db_product is None:
            raise
HTTPException(status_code=status.HTTP_404_NOT_FOUND,
detail="Product not found")

        db.delete(db_product)
        db.commit()
        return {"message": "Product deleted successfully"}
```

Рисунок 3.10

Ці фрагменти коду відповідають за серверну реалізацію API для управління продуктами. Вони дозволяють створювати, переглядати, оновлювати та видаляти інформацію про продукти в системі. Для кожного продукту визначені ключові атрибути, такі як назва, опис, ціна, стан, кількість на складі та категорія. Код включає валідацію вхідних даних, що гарантує коректність інформації перед її збереженням у базі даних. У випадку помилок (наприклад, якщо продукт не знайдено), API повертає відповідне повідомлення. Це API надає фронтенду всі необхідні функції для повноцінного управління продуктами.

3.3 Реалізація аутентифікації та авторизації

Реалізуємо систему аутентифікації користувачів, яка дозволяє зберігати і обробляти маркери доступу (JWT) для забезпечення захищеного доступу до сторінок та ресурсів.

Імпортуємо бібліотеки і створюємо контексту аутентифікації. Створюється контекст AuthContext, який забезпечує централізоване управління станом аутентифікації в додатку. Використовується бібліотека axios для здійснення HTTP-запитів до API сервера.

Компонент AuthProvider зберігає стан токену аутентифікації (token) і дані користувача (user). Стан token ініціалізується з localStorage, що дозволяє зберігати токен навіть після перезавантаження сторінки.

Використовуємо ефекту useEffect для обробки зміни токену. Він викликається щоразу, коли token змінюється. Якщо токен встановлено, він зберігається в localStorage, і він додається як заголовок Authorization до axios для всіх подальших запитів. Якщо токен видалено, то з localStorage видаляється збережене значення, і заголовок Authorization очищується.

Функція fetchUserProfile здійснює запит до API для отримання даних профілю користувача. Якщо запит успішний, інформація про користувача зберігається в стані user. Якщо виникає помилка, викликається функція logout, яка очищає дані користувача та токен.

Функція login виконує запит до API для входу користувача, надсилаючи ім’я користувача та пароль. У разі успіху, з відповіді API зберігаються token та дані користувача, що дозволяє користувачеві отримати доступ до захищених ресурсів.

Функція logout очищає стан token та user, що означає, що користувач більше не має доступу до захищених ресурсів.

AuthContext.Provider надає інші компоненти значення контексту (token, user, login, logout, isAuthenticated). Ці значення можна отримати в будь-якому компоненті, що знаходиться всередині AuthProvider, за допомогою useAuth.

Користувацький хук useAuth повертає значення AuthContext, що полегшує доступ до стану аутентифікації в різних компонентах додатка.

Отже, ми створили код, що реалізує централізовану систему керування аутентифікацією на фронтенді, що спрощує процес входу/виходу та забезпечує безпечний доступ до API для авторизованих користувачів.

Фрагмент коду, який відображає централізовану систему керування аутентифікацією на фронтенді, послідовно наведено на рисунках 3.11 – 3.12 відповідно.

```
     import React, { createContext, useContext, useState,
useEffect } from 'react';
     import axios from 'axios';

     const AuthContext = createContext();

     export const AuthProvider = ({ children }) => {
       const [token, setToken] =
useState(localStorage.getItem('token') || null);
       const [user, setUser] = useState(null);

       useEffect(() => {
         if (token) {
           localStorage.setItem('token', token);
           axios.defaults.headers.common['Authorization'] =
`Bearer ${token}`;
           fetchUserProfile();
         } else {
           localStorage.removeItem('token');
           delete
axios.defaults.headers.common['Authorization'];
         }
       }, [token]);

       const fetchUserProfile = async () => {
         try {
           const response = await
axios.get('/api/user/profile'); // Replace with your profile
endpoint
           setUser(response.data);
         } catch (error) {
           console.error('Failed to fetch user profile',
error);
           logout();
         }
       };
```

Рисунок 3.11

```
      const login = async (username, password) => {
        try {
          const response = await axios.post('/api/auth/login',
{ username, password });
          setToken(response.data.token);
          setUser(response.data.user);
        } catch (error) {
          console.error('Login failed', error);
          throw new Error('Login failed');
        }
      };

      const logout = () => {
        setToken(null);
        setUser(null);
      };

      return (
        <AuthContext.Provider value={{ token, user, login,
logout, isAuthenticated: !!token }}>
          {children}
        </AuthContext.Provider>
      );
    };

    export const useAuth = () => {
      return useContext(AuthContext);
    };
```

Рисунок 3.12

Ці фрагменти коду відповідають за реалізацію контексту для управління аутентифікацією користувачів на фронтенді, використовуючи JWT (JSON Web Token).

Компонент AuthProvider надає глобальний контекст аутентифікації, що дозволяє зберігати токен користувача, отримувати інформацію про користувача та виконувати дії для входу і виходу з системи.

При успішній аутентифікації токен зберігається в localStorage, а також передається у заголовки HTTP-запитів для авторизації на сервері. Цей контекст можна використовувати в різних компонентах, забезпечуючи уніфіковане управління аутентифікацією на рівні застосунку.

3.4 Реалізація інтерфейсу для перегляду товарів

Реалізуємо фронтенд-код для відображення списку продуктів.

Починаємо з імпорту необхідних бібліотек. React, useEffect, та useState забезпечують функціональність компонента, а axios використовується для відправки HTTP-запитів на сервер.

Компонент ProductView отримує productId як пропс, який використовується для отримання даних про конкретний продукт. Для зберігання стану використовуються хуки useState: product для збереження інформації про продукт, loading для відстеження процесу завантаження, і error для обробки помилок.

Функція fetchProduct для отримання даних про продукт. useEffect викликає функцію fetchProduct при кожній зміні productId. Функція fetchProduct надсилає GET-запит на сервер для отримання інформації про продукт за допомогою axios. Якщо запит успішний, результат зберігається у product. У разі помилки встановлюється повідомлення про помилку у error. loading встановлюється на false після завершення запиту, незалежно від його результату.

Рендеринг стану завантаження і помилки. Поки триває завантаження, компонент показує повідомлення “Loading...”. Якщо виникає помилка, відображається повідомлення про помилку червоним кольором.

У разі успішного отримання даних, компонент відображає інформацію про продукт, включно з зображенням, назвою, описом, ціною, станом та кількістю на складі. Вся інформація відображається в структурованому і стильному вигляді. Кнопка “Add to Cart” дозволяє додати продукт до кошика (функціонал додавання може бути реалізований окремо).

Експортуємо компонент ProductView. Він експортується як стандартний компонент для подальшого використання у додатку.

Компонент ProductView є частиною інтерфейсу для перегляду інформації про товари у вебдодатку. Він дозволяє отримати детальну

інформацію про конкретний продукт за допомогою API-запиту до серверу і відобразити цю інформацію користувачеві у зручному вигляді. Завдяки використанню useEffect для управління запитами і обробки стану, компонент підтримує асинхронну взаємодію з сервером, що підвищує зручність користувацького досвіду.

Фрагмент коду, який відображає реалізацію інтерфейсу для перегляду товарів, послідовно наведено на рисунках 3.13 – 3.14 відповідно.

```
     import React, { useEffect, useState } from 'react';
     import axios from 'axios';

     const ProductView = ({ productId }) => {
       const [product, setProduct] = useState(null);
       const [loading, setLoading] = useState(true);
       const [error, setError] = useState(null);

       useEffect(() => {
         const fetchProduct = async () => {
           try {
             const response = await
axios.get(`/api/products/${productId}`);
             setProduct(response.data);
           } catch (err) {
             setError("Failed to load product.");
           } finally {
             setLoading(false);
           }
         };

         fetchProduct();
       }, [productId]);

       if (loading) return <p>Loading...</p>;
       if (error) return <p className="text-red-
500">{error}</p>;

       return (
         <div className="max-w-md mx-auto p-6 bg-white shadow-
md rounded-lg">
           <img
             src={product.imageUrl}
             alt={product.name}
             className="w-full h-64 object-cover mb-4 rounded"
           />
           <h2 className="text-2xl font-bold mb-
2">{product.name}</h2>
```

Рисунок 3.13

```
            <p className="text-gray-600 mb-
4">{product.description}</p>
            <p className="text-lg font-semibold mb-2">Price:
${product.price}</p>
            <p className="text-gray-600">Condition:
{product.condition}</p>
            <p className="text-gray-600">Stock:
{product.stock_quantity}</p>
            <button className="mt-4 px-4 py-2 bg-blue-500 text-
white rounded hover:bg-blue-600">
              Add to Cart
            </button>
          </div>
        );
      };

      export default ProductView;
```

Рисунок 3.14

Ці фрагменти коду реалізує компонент на React під назвою ProductView, який відповідає за відображення інформації про продукт на стороні користувача. Код взаємодіє з бекендом через API-запит, отримуючи інформацію про конкретний продукт та відображаючи її у зручному форматі. Це дозволяє користувачам переглядати детальну інформацію про товар, включно з ціною, станом та доступністю на складі, а також додає кнопку для додавання товару до кошика.

3.5 Висновки до розділу 3

Цей розділ зосереджений на практичній реалізації ключових компонентів системи для онлайн-магазину побутової техніки. У ньому ретельно розглянуто, як створювалася база даних, налаштовувалася взаємодія з API, оброблялися замовлення та розроблялися інтерфейси для управління продуктами й виведення даних на клієнтську сторону.

Структура бази даних стала фундаментом для надійного збереження всіх важливих даних – від інформації про користувачів до записів про замовлення,

платежі та відгуки. Серверна частина системи реалізує основні бізнес-функції, такі як оформлення замовлень, управління каталогом товарів і реагування на запити користувачів. Надійність і безпеку забезпечує інтеграція механізму аутентифікації через JWT, що відповідає сучасним вимогам кібербезпеки.

Клієнтська сторона системи, створена з урахуванням передових принципів UX/UI-дизайну, пропонує інтуїтивно зрозумілий і привабливий інтерфейс. Відображення товарів і управління замовленнями зроблено максимально зручним для користувачів, сприяючи створенню позитивного досвіду взаємодії з платформою.

Завдяки інтеграції всіх компонентів вдалося створити цілісну систему, яка відповідає технічним і функціональним вимогам. Злагоджений зв'язок між бекендом і фронтендом не лише гарантує стабільність і ефективність роботи магазину, а й закладає міцний фундамент для його майбутнього розвитку та масштабування.

4 ОПИС ІНТЕРФЕЙСУ СИСТЕМИ

4.1 Демонстрація роботи онлайн-магазину

Результатом роботи є вебсайт «Інтернет-магазин побутової техніки» з використанням фреймворку FastAPI.

Заходячи на сайт користувач бачить головну сторінку сайту (рис. 4.1). На ній він бачить перелік товарів, їхні фото та назву.

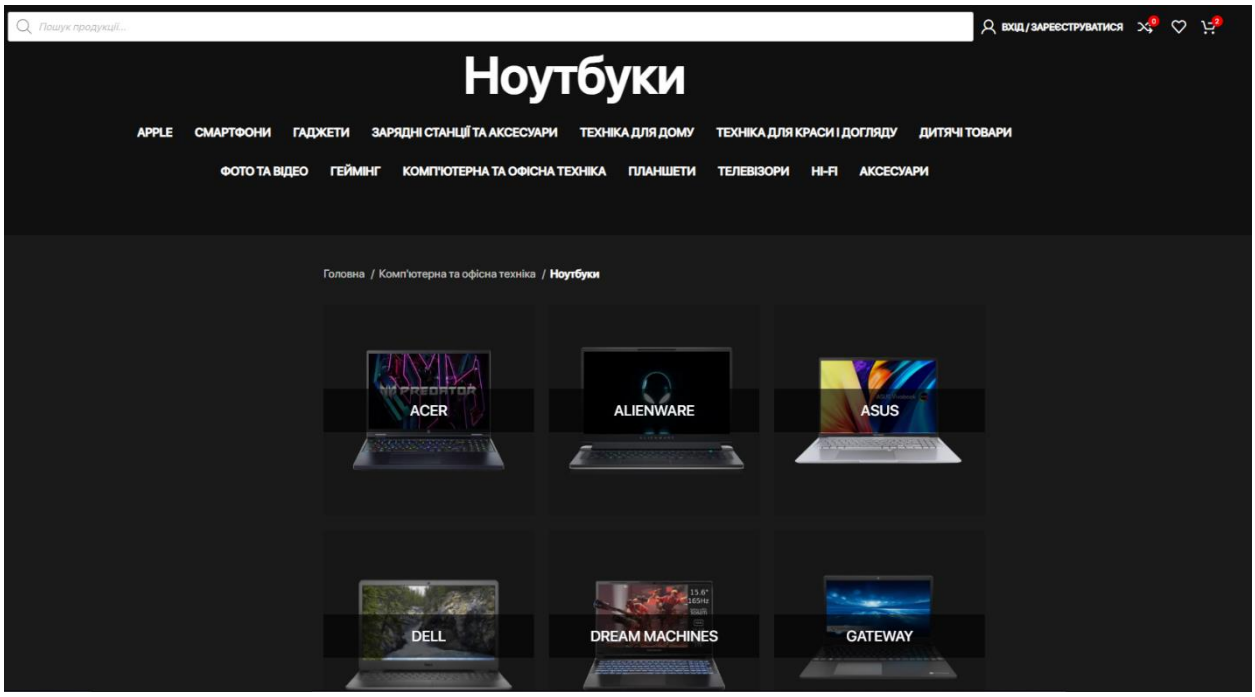


Рисунок 4.1 – Головна сторінка

Натиснувши на товар, користувач переходить на сторінку з детальною інформацією про товар, має можливість додати його до списку бажань та до кошику (див. рис. 4.2).

Користувач може переглянути свій список бажань (див. рис. 4.3).

Користувач має можливість додати товар до кошику, натиснувши кнопку «Додати в кошик». Після цього на сторінці кошика з’являються додані товари, які можна прибрати або додати кількість товарів (див. рис. 4.4).

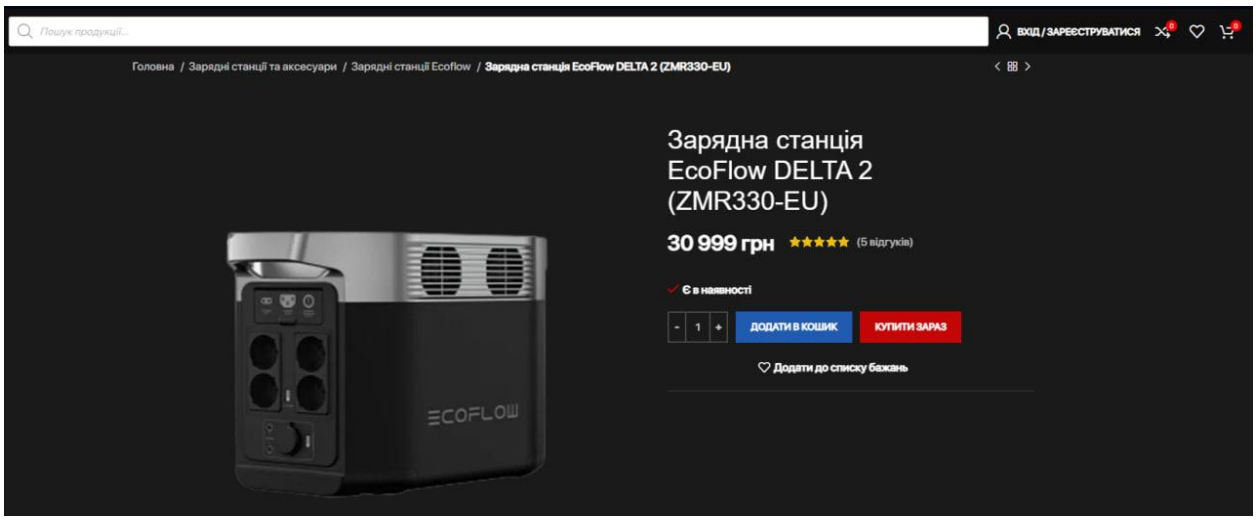


Рисунок 4.2 – Сторінка товару

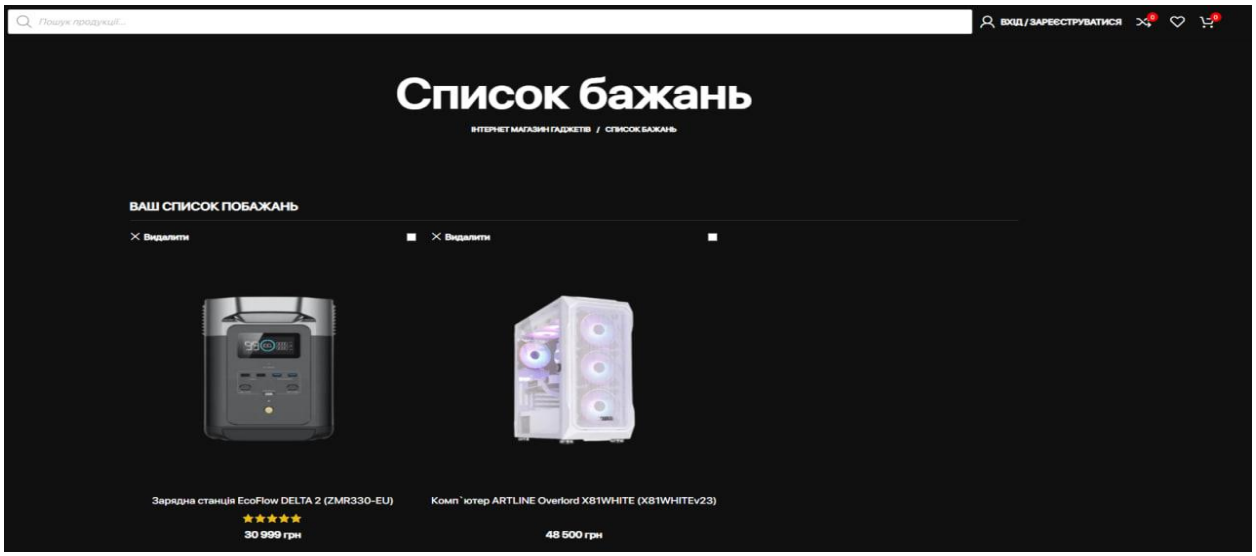


Рисунок 4.3 – Список бажань

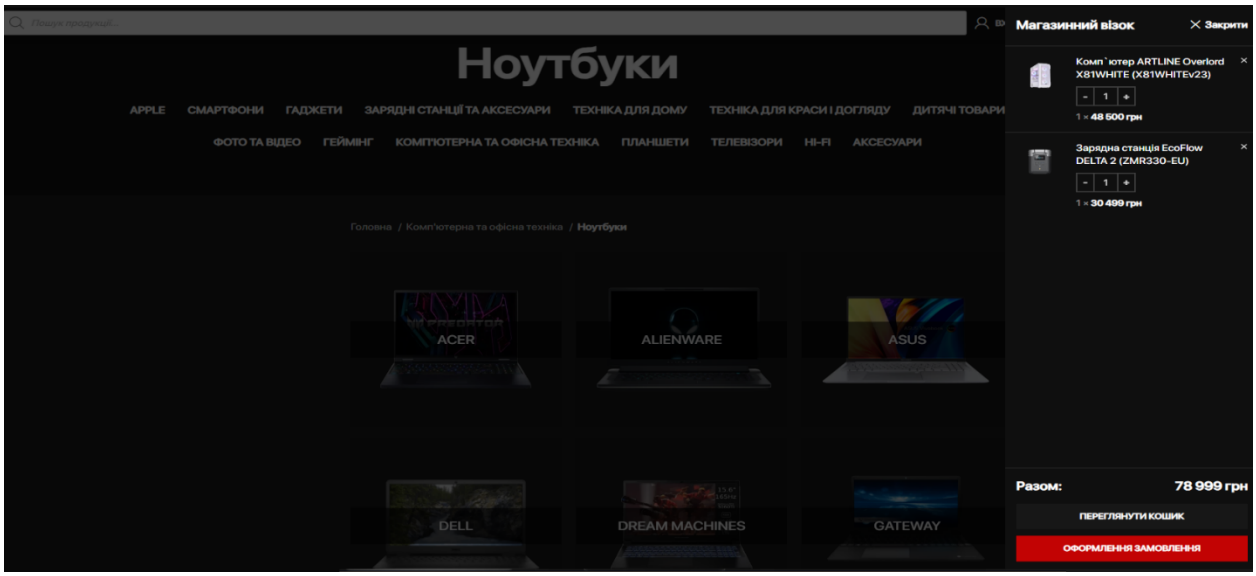


Рисунок 4.4 – Кошик

Далі користувач натискає кнопку «Оформити замовлення» і переходить до сторінки оформлення замовлення, де бачить форму, в яку треба внести особисті дані та дані про доставку (рис. 4.5).

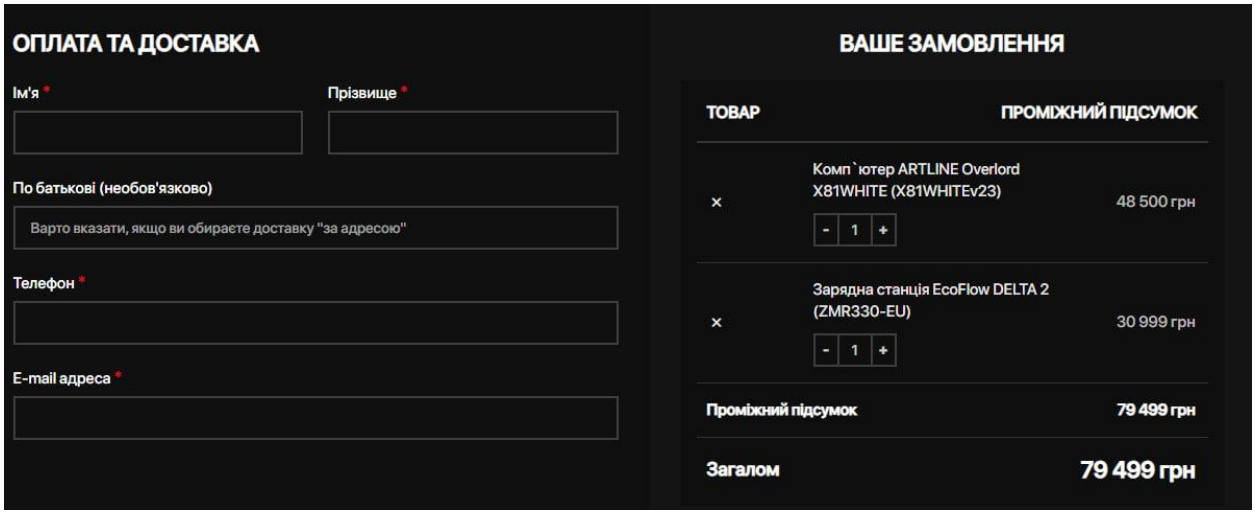


Рисунок 4.5 – Оформлення замовлення

Після того як користувач вніс необхідні дані, то він переходить до оплати. Його перенаправляє на платіжний сервіс Stripe, де користувач може здійснити оплату замовлення (рис. 4.6).

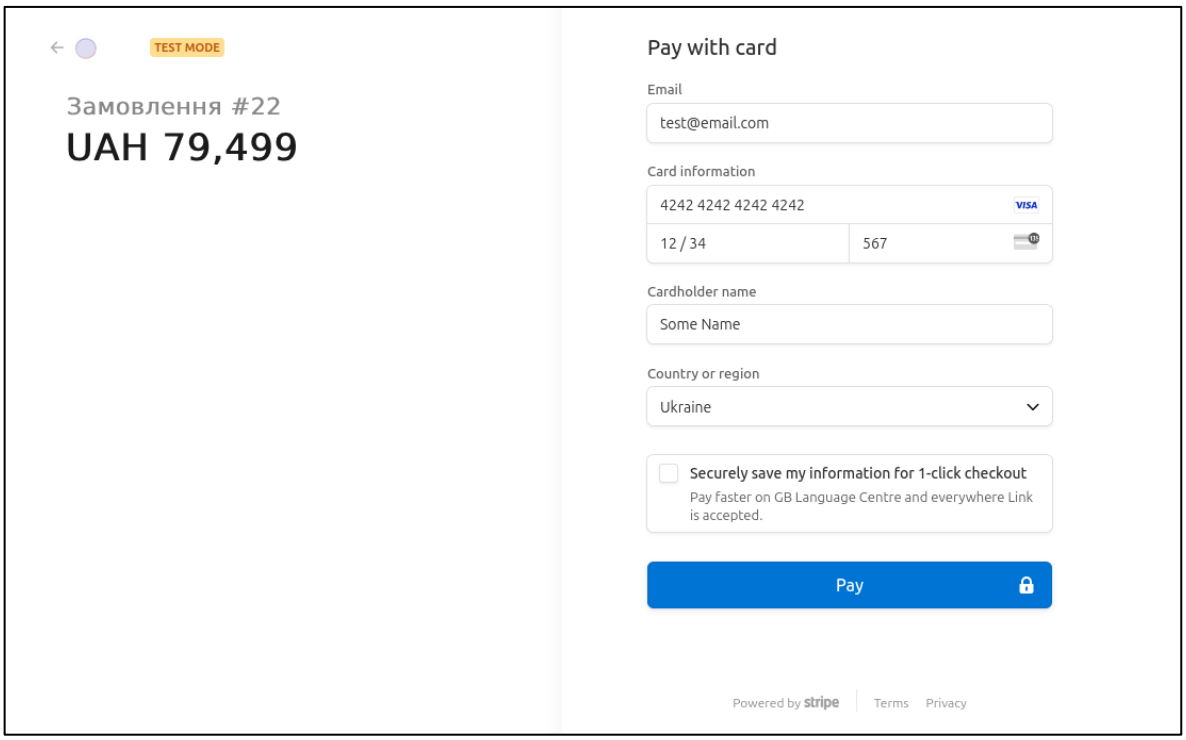


Рисунок 4.6 – Оплата замовлення

4.2 Висновки до розділу 4

У цьому розділі було розглянуто ключові сторінки вебсайту «Інтернет-магазин побутової техніки», а саме головна сторінка, список бажань, кошик, сторінка оформлення замовлення, сторінка з оплатою та сторінка з підтвердженним замовленням. Кожна із цих сторінок має важливе значення для забезпечення зручності користування та реалізації основних функцій онлайн-магазину. Продемонстровані скріншоти показують, як розроблені сторінки сприяють досягненню цілей проєкту, зокрема інтуїтивно зрозумілої навігації та ефективності процесу здійснення покупок.

ВИСНОВКИ

Отже, в результаті виконання кваліфікаційної роботи був розроблений вебсайт «Інтернет-магазин побутової техніки» з використанням FastAPI та React.js. Проведене дослідження сучасних технологій розробки вебзастосунків дозволило визначити, що FastAPI є оптимальним вибором для створення API. Для зберігання інформації було обрано PostgreSQL. Вибір бібліотек React і Tailwind CSS для фронтенд-розробки забезпечив ефективну реалізацію адаптивного і зручного інтерфейсу користувача. Для інтеграції платіжних систем було обрано Stripe, що забезпечує просту та безпечну реалізацію онлайн-платежів.

Згідно поставлених завдань було проаналізовано функціональні та технічні вимоги інтернет-магазину побутової техніки. В результаті чого було побудовано Use-case діаграму та описано її прецеденти. Було побудовано діаграму розгортання та діаграму послідовності основного бізнес-процесу системи, а саме оформлення замовлення.

Було проаналізовано предметну область та спроєктовано ER-діаграму для зберігання інформації в системі. Розроблено API для виконання основних функцій інтернет-магазину, таких як управління товарами, оформлення і оплата замовлень, робота зі списком бажань.

Було виконано ручне тестування та складено опис інтерфейсу системи.

Вебсайт «Інтернет магазин побутової техніки» досить функціональний прототип, але для реального використання, може бути вдосконалений.

ПЕРЕЛІК ПОСИЛАНЬ

1. Harper A. Python Essentials For Beginners: Absolute Beginner’s Guide to Python Programming and the Fundamentals of Python Programming with Step-by-Step Guide. Independently Published, 2024. 138 p.

2. Чому варто вивчати Python в 2023 році. URL: https://mate.academy/blog/python/learning-python-2023/ (дата звернення: 15.04.2024).

3. Welcome to Python.org. URL: https://www.python.org (дата звернення: 17.04.2024).

4. Peralta J.H. Microservice APIs: Using Python, Flask, FastAPI, OpenAPI and more. New York : Manning Publications, 2023. 440 p.

5. Global E. Python Flask for Web Development: Build Web Applications in Python Using Flask Framework. Independently Published, 2022. 237 p.

6. Welcome to Flask – Flask Documentation (3.1.x). URL: https://flask.palletsprojects.com/en/stable/ (дата звернення: 22.04.2024).

7. Carroll T. Django Framework for Web Developers: Create powerful Python web applications from scratch. Independently Published, 2024. 126 p.

8. The web framework for perfectionists with deadlines | Django. URL: https://www.djangoproject.com (дата звернення: 24.04.2024).

9. Welcome to Pyramid, a Python Web Framework. URL: https://trypyramid.com (дата звернення: 24.04.2024).

10. Lathkar M. High-Performance Web Apps with FastAPI: The Asynchronous Web Framework Based on Modern Python. New York : Apress, 2023. 332 p.

11. Voron F. Building Data Science Applications with FastAPI: Develop, manage, and deploy efficient machine learning applications with Python, 2nd Edition. Birmingham, United Kingdom: Packt Publishing, 2023. 422 p.

12. FastAPI. URL: https://fastapi.tiangolo.com (дата звернення: 30.04.2024).

13. Підручник з Umbrello UML Modeller. URL: https://docs.kde.org/trunk5/uk/umbrello/umbrello/uml-basics.html (дата звернення: 15.05.2024).

14. Regis M. UML Diagrams – A Simple Guide for Beginners: A basic guide to systems analysis and design. Learn how to structure software projects using UML diagrams. Independently Published, 2024. 73 p.

15. What is Use Case Diagram? URL: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-use-case-diagram/ (дата звернення: 17.05.2024).

16. What is Sequence Diagram? URL: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-sequence-diagram/ (дата звернення: 25.05.2024).

17. Korth H., Silberschatz A., Sudarshan S. Database System Concepts | 7th Edition. London : Generic, 2021. 555 p.

18. Elmasri R., Navathe S. B. Fundamentals of Database Systems (7th ed.). London : Pearson, 2010. 1280 p.

19. What is an Entity Relationship Diagram (ERD)? URL: https://www.lucidchart.com/pages/er-diagrams (дата звернення: 03.06.2024).

20. What is Deployment Diagram? URL: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-deployment-diagram/ (дата звернення: 05.06.2024).