НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ
Завідувач кафедри
Комп'ютерних наук
_____ Голуб Б.Л.

"___" _____ 2025 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА
на тему

«Web-орієнтована інформаційна система для магазину електроніки»
Спеціальність 122 – «Комп'ютерні науки»

Гарант освітньої програми
Д.е.н., професор _____ Руденський Р.А

Керівник бакалаврської кваліфікаційної роботи

_____ _____ __Міловідов Ю.О__
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Виконав _____ _____Ягодка Роман Сергійович_____
(підпис) (ПІБ студента)

КИЇВ – 2025

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
БІОРЕСУРСІВ І
ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри
Комп'ютерних наук
_____ Голуб Б.Л.
"___" _____ 2025 р.

З А В Д А Н Н Я
на виконання бакалаврської кваліфікаційної роботи студенту
Ягодки Роману Сергійовичу
(прізвище, ім'я, по батькові)

Спеціальність 122 – «Комп'ютерні науки»

Тема бакалаврської кваліфікаційної роботи «Web-орієнтована інформаційна система для магазину електроніки»

затверджена наказом ректора НУБіП України від "16" 12 2024р № 2246С

Термін подання завершеної роботи на кафедру_____
(рік, місяць, число)

Вихідні дані до бакалаврської кваліфікаційної роботи:
створення інтернет магазину електроніки.

Перелік питань, що розглядаються: Аналіз проблемної області. Моделювання предметної області. Проєктування програмної системи. Впровадження та експлуатація системи

Дата видачі завдання "___"_____ 2024 р.
Керівник бакалаврської кваліфікаційної роботи

_____	_____	_____
(науковий ступінь та вчене звання)	(підпис)	(ПІБ)

Завдання прийняв до виконання _____ _____
(підпис) (ПІБ студента)

ЗМІСТ

ВСТУП

Сучасна автоматизація бізнес-процесів перетворилася на один з ключових чинників ефективності та конкурентоспроможності бізнесу в сучасному світі. Однією з галузей, яка найбільше потребує використання сучасних інформаційних технологій, є роздрібна торгівля електронікою, яка характеризується великою різноманітністю товарів, динамікою продажів та великою кількістю інформації про клієнтів.

Більшість підприємств роздрібної торгівлі електронікою досі використовують застарілі інструменти для управління товарами, виконання замовлень та роботи з клієнтами. Це призводить до дублювання інформації, обтяжує персонал щоденними завданнями, підвищує ймовірність помилок і знижує якість обслуговування. Усе це сповільнює зростання бізнесу та ускладнює прийняття рішень для керівництва.

Існує нагальна потреба у створенні інформаційної системи на основі веб-технологій. Система допоможе безпечно зберігати товарну продукцію в єдиному місці, автоматизувати обробку замовлень, здійснювати ефективну роботу з клієнтами та надавати точні дані про продажі. Рішення покращить роботу магазину, покращить обслуговування клієнтів та сприятиме зростанню та розширенню електронної комерції.

Метою цієї дипломної роботи є розробка покращених способів роботи з клієнтами, замовленнями та товарами в магазині електроніки за допомогою простої у використанні веб-інформаційної системи. Система буде включати в себе всі найважливіші компоненти для менеджерів та працівників магазину, а саме: роботу з товарами, обробку замовлень, відстеження клієнтів та створення звітів про продажі.

Система реалізована у вигляді веб-додатку на мові C# на фреймворку ASP.NET Core. Для ефективного та безпечного зберігання даних використовується Microsoft SQL Server [26]. Інтерфейс програми зручний для

користувачів: він простий у розумінні, корисний і відповідає реальним потребам магазину електроніки, дозволяючи працівникам швидко виконувати завдання з меншими витратами часу та ресурсів.

Актуальність даної роботи полягає в практичному застосуванні сучасних технологій веб-розробки для оптимізації ключових бізнес-процесів у сфері роздрібної торгівлі електронікою. Автоматизація рутинних операцій, таких як управління товарними залишками, обробка замовлень, облік клієнтів і формування звітності, дозволяє значно підвищити ефективність роботи магазину, зменшити кількість помилок і прискорити обслуговування клієнтів. Запропонована web-орієнтована інформаційна система покликана підвищити продуктивність персоналу, забезпечити зручне керування даними та сприяти цифровій трансформації торгового процесу.

Об’єктом дослідження є електронний магазин, який розглядається як комерційний суб'єкт, що здійснює фізичний або онлайн-продаж товарів. Предметом дослідження є автоматизація основних операцій магазину за допомогою веб-інформаційної системи. Бакалаврська робота присвячена розробці сучасного програмного рішення, яке допоможе магазину електроніки оптимізувати управління запасами, обробку замовлень, обробку даних про клієнтів та інші ключові бізнес-операції.

Система повинна бути розроблена з сильним акцентом на масштабованість, надійність і безпеку, і в той же час відповідати асиметричним потребам працівників магазину та його клієнтів.

Для досягнення цих цілей були визначені наступні завдання:

- проаналізувати існуючі бізнес-процеси в електронній комерції;
- визначити функціональні та нефункціональні вимоги до системи;
- розробити схему бази даних для зберігання інформації про товари, замовлення та користувачів;
- розробити архітектуру, користувацький інтерфейс та логіку обробки даних веб-додатку;

- провести тестування системи для оцінки функціональних і нефункціональних аспектів, включаючи продуктивність і зручність користування

Результатом цього проекту є повністю функціональний прототип веб-інформаційної системи, який може стати основою для подальшого розвитку та можливого впровадження. Система сприятиме безперебійному функціонуванню роздрібної торгівлі електронікою завдяки підвищенню операційної ефективності, зменшенню ручної роботи та впровадженню параметрів, які підвищать якість обслуговування.

1 АНАЛІЗ ПРОБЛЕМНОЇ ОБЛАСТІ

1.1 Опис предметної області

Торгівля електронікою — це складна багатоетапна бізнес-процедура, яка включає управління асортиментом товарів, облік клієнтів, обробку замовлень, логістику та взаємодію з постачальниками. Ці процеси реалізуються як у фізичних магазинах, так і через онлайн-платформи, що вимагає від бізнесу точності, оперативності та гнучкості в управлінні всіма даними.

Незважаючи на стрімкий технологічний прогрес, незліченна кількість магазинів електроніки продовжує використовувати традиційні методи обліку або ручні процеси. Цей факт сам по собі може створювати безліч перешкод. Паперова документація та хаотично заповнені електронні таблиці підвищують ймовірність втрати цінної інформації. Кількість дублюючих записів і помилок при введенні даних продовжує зростати; автоматизація ручних процесів зупиняється до того, як ви витратите час і навантажите своїх співробітників повторюваними завданнями, такими як оновлення запасів, замовлення на продаж і створення звітів.

Окрім цих адміністративних перешкод, магазини також стикаються з проблемами в обслуговуванні клієнтів. Без централізованої системи важко перевірити історію замовлень, визначити статус відправлення замовлення або дізнатися про наявність товару на різних складах чи в різних місцях. Такий операційний спад призводить до зниження якості обслуговування, відтоку клієнтів і втрати конкурентоспроможності.

Тому необхідним є створення веб-інформаційної системи для централізованого управління всією інформацією в магазині електроніки, від обліку запасів і обробки замовлень до аналізу продажів.

Більшість програмних додатків, доступних сьогодні, або застарілі, або пропонують обмежені функції, або просто несумісні з новими вимогами електронної комерції для продавців електроніки. Більшість із цих систем мають

не інтуїтивно зрозумілий веб-інтерфейс, не мають зв'язку з оплатою, доставкою та CRM, а аналітика та звітність у них неможливі. Крім того, вони не можуть мати єдиного місця зберігання даних і, як наслідок, не мають контролю доступу для захисту даних про клієнтів та транзакції, що є фундаментальним кроком у забезпеченні безпеки та конфіденційності даних.

Отже, існує постійна і постійно зростаюча потреба в оновленні процесу управління електронною комерцією за допомогою частково веб-орієнтованої системи початкової інформації, яка стане сховищем для всіх основних функціональних операцій. Менеджери магазинів зможуть виконувати ряд обов'язків, включаючи ефективну роботу з товарами, клієнтами, замовленнями, звітність та аналітику, що відображає ситуацію в режимі реального часу. Реалізація цього проекту призведе до зменшення навантаження на персонал, а також підвищить точність обліку та забезпечить клієнтам найкращу якість обслуговування, щоб клієнти могли ефективно змусити магазин працювати краще, ніж будь-коли.1

Прикладом проекту електронної комерції є інтернет-магазин, який має кілька функцій, таких як прийняття, обробка та доставка замовлень, адміністрування товарів і клієнтів, а також оплата доставки. Основними видами діяльності таких магазинів є періодичне оновлення каталогу, надання зручного для клієнтів веб-інтерфейсу, контроль запасів, управління логістичними послугами та обслуговування клієнтів.

Звичайна компанія електронної комерції складається з різних працівників, таких як адміністратори сайту, які відповідають за оновлення товарів, контроль запасів, а також ціноутворення на товари. Обробкою замовлень, координацією доставки та періодами відновлення товару займається менеджер. Клієнти можуть взаємодіяти з системою через веб-інтерфейс, переглядати, замовляти, оплачувати, а також відстежувати посилки.

Питання конфіденційності охоплює той факт, що можуть існувати різні види інформації, наприклад, особисті дані користувачів (ім'я, контактна інформація, поштові адреси), на додаток до них дані про товар (бренд, ціна,

наявність, опис), історія покупок, стадія замовлення та дані про оплату. Саме з цієї необхідності випливає вимога дотримання законодавства про захист даних, що передбачає використання надійних і безпечних схем управління даними.

Наразі велика кількість магазинів електроніки мають ручні процеси або використовують застарілі системи, які все ще залишаються основними джерелами інформації. Таким чином, вони мають доступ до даних лише час від часу, і, як правило, обмежені оновленням одного файлу або, скажімо, кількох, які знаходяться в одному місці. Звідси і низька якість обслуговування, і запізніла поява нових продуктів, і помилкові логістичні рішення. Це також ситуація, яка цілком обґрунтовано здатна створювати такі проблеми, як помилки в здійснених транзакціях, багаторазовий облік одного і того ж замовлення, розбіжність з даними про кількість товарів або навіть втрата безцінної інформації.

Для впровадження автоматизації всієї роздрібної торгівлі через Інтернет, система категорії електронної комерції була б більш привабливою, оскільки це призвело б до реального вирішення обговорюваних проблем. Програмне забезпечення буде виконувати ряд необхідних операцій, наприклад, зберігати каталог товарів у базі даних, автоматично обробляти замовлення, готувати звіти та здійснювати комунікацію з клієнтами. Очікується, що це зробить бухгалтерський облік більш точним, пришвидшить процес обробки інформації, додасть впевненості та захистить дані під час збереження та передачі файлів.

Детальний опис класів та атрибутів предметної області наведено у таблиці 1.1.

Таблиця 1.1

Опис атрибутів класів предметної області

Клас предметної області	**Атрибут**	**Опис**
Користувач	Прізвище, ім’я, по батькові	Повне ім’я особи
	Дата народження	Число, місяць і рік народження
	Емейл	Електронний адрес
	Контактний телефон	Номер телефону для зв’язку
Товар	Назва	Назва товару
	Опис	Детальний опис товару
	Стара ціна	Для відображення при знижці
	Нова ціна	Офіційна ціна
	Кількість	Кількість товарів на складі
Замовлення	Дата замовлення	День, місяць рік замовлення
	Загальна ціна	Підсумована ціна з всіх товарів
	Статус	Відображення статусу
Елемент в замовленні	Товар	Товар з магазину
	Кількість	Вибрана кількість в замовленні
	Ціна	Ціна за товар
Корзина	Товар	Товар з магазину
	Кількість	Кількість товару в корзині
Список бажань	Товар	Товар з магазину
Постачальник	Назва	Назва постачальника
	Контактні дані	Дані для звязку, номер телефону, пошта
Категорія	Назва	Назва категорії
Доставка	Трек номер	Номер доставки

Клас предметної області	Атрибут	Опис
	Дата доставки	День, місяць, рік доставки
	Адреса	Адреса доставки
Оплата	Замовлення	Замовлення товарів
	Сума	Сума оплати
	Метод оплати	Накладений платіж, на разрахунковий рахунок
	Дата оплати	День, місяць, рік оплати

1.2 Огляд існуючих рішень

Сьогодні в Україні та світі існує велика кількість програм для автоматизації торгівлі, які загальні системи електронної комерції не пристосовані до вимог управління магазинами електроніки. Загальні CRM- або ERP-платформи є найпоширенішим рішенням для подібних бізнес-операцій, але вони потребують значної кастомізації для ведення складського обліку електронних товарів, відстеження гарантійних зобов'язань, моніторингу специфікацій товарів та інтеграції з сервісними центрами.

OpenCart [18], PrestaShop [19] і Shopify [20] - це три найпоширеніші інструменти для малого та середнього бізнесу, оскільки вони надають фундаментальні функції, які включають каталогізацію товарів, обробку замовлень, управління кошиками та платежами. Ці рішення виявляються недостатніми для точного відстеження гарантійного терміну та управління технічними параметрами, а також інформацією про доставку та обслуговування, що є критично важливими для ведення бізнесу в сфері електроніки.

Впровадження веб-інформаційної системи, що підтримує специфічні для галузі функції, дозволить здійснювати повне управління товарними позиціями, клієнтами, замовленнями, гарантійними зобов'язаннями та аналітикою на єдиній платформі [3].

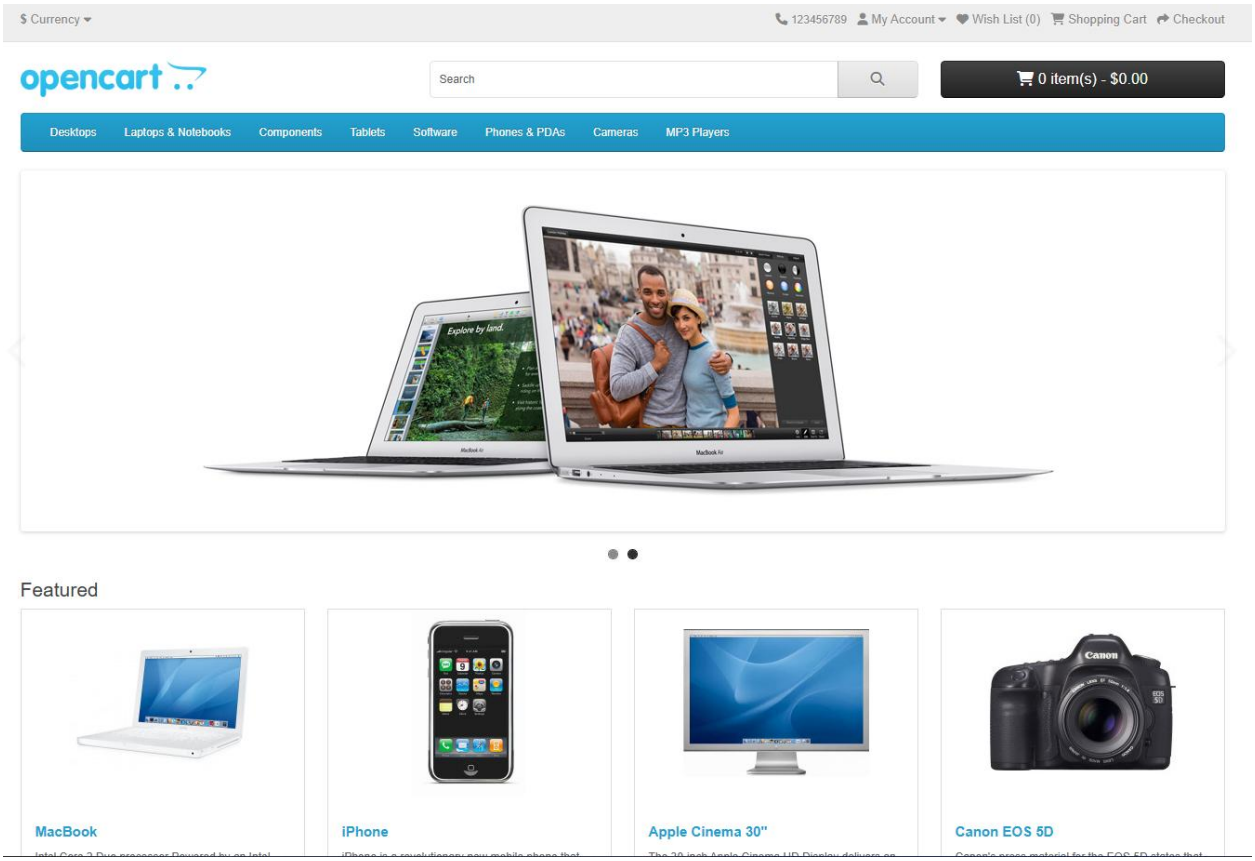

Рис. 1 Інтернет магазин “ OpenCart [18]”

Одним із популярних програмних рішень для створення онлайн-магазинів є система OpenCart [18] — безкоштовна, з відкритим вихідним кодом, CMS-платформа, орієнтована на електронну комерцію. Вона широко використовується малим та середнім бізнесом завдяки своїй простоті налаштування, зручному інтерфейсу та широкому вибору модулів і шаблонів. OpenCart [18] дозволяє ефективно керувати каталогом товарів, обробляти замовлення, керувати кошиком, здійснювати оплату онлайн і налаштовувати доставку.

Серед переваг OpenCart [18] — підтримка багатомовності, багатовалютності, гнучка система знижок та купонів, а також можливість розширення функціоналу за допомогою тисяч модулів. Це робить платформу

привабливою для реалізації інтернет-магазину електроніки, де важливу роль відіграє детальна класифікація товарів, фільтрація за технічними параметрами, відстеження гарантій, управління залишками на складі тощо.

Однак, незважаючи на переваги, OpenCart [18] має і певні недоліки. Його базова версія надає обмежений функціонал, і для реалізації специфічних функцій магазину електроніки (наприклад, управління гарантійним терміном, інтеграції з сервісними центрами, автоматизованого оновлення залишків від постачальників) часто потрібна стороння розробка або використання платних модулів. Крім того, система не є «з коробки» оптимізованою для високих навантажень, тому для масштабування можуть знадобитися додаткові технічні рішення — кешування, CDN, оптимізація запитів до БД.

Ще одним обмеженням є безпека: OpenCart [18] потребує ретельного налаштування, регулярних оновлень і перевірки встановлених модулів, щоб уникнути вразливостей. Без цього система може бути вразливою до атак.

У підсумку, OpenCart [18] є зручним стартовим рішенням для побудови інформаційної системи інтернет-магазину електроніки, але для відповідності сучасним вимогам та специфіці предметної області необхідно передбачити додаткові модифікації та інтеграції. Додаток наступного покоління також має відповідати мінливим юридичним і технічним вимогам до обробки даних у державних установах, пропонуючи більш безпечне та ефективне середовище для роботи призовної комісії .

Розглянемо інше програмне рішення на ринку.

PrestaShop [19] — це одна з популярних систем для створення інтернет-магазинів, яка широко використовується в різних країнах, включаючи Україну. Вона створена для оптимізації процесів управління товарами, обробки замовлень та взаємодії з клієнтами в межах підприємства або установи. Система дозволяє керувати каталогом електроніки, обробляти замовлення, а також забезпечує зручний доступ до інформації для всіх користувачів.

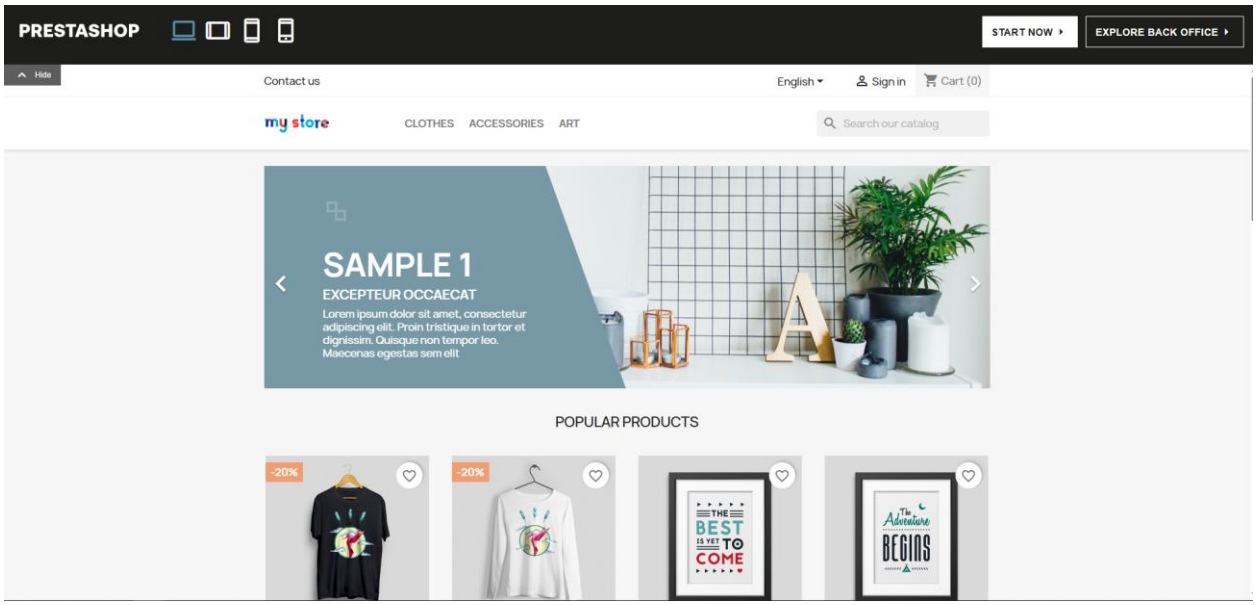

Рис. 2 Інтернет магазин “PrestaShop [19]”

Основною особливістю PrestaShop [19] є її здатність централізовано керувати всіма аспектами діяльності інтернет-магазину електроніки через єдину зручну веб-платформу. Система дозволяє адміністратору додавати, редагувати та категоризувати товари, оновлювати ціни та залишки на складі, а також налаштовувати процеси оплати й доставки. Інтерфейс орієнтований на зручність користувача як з боку адміністрації магазину, так і з боку клієнтів, що дозволяє скоротити час на обробку замовлень та покращити якість обслуговування.

PrestaShop [19] також включає модулі для аналітики продажів, управління клієнтською базою та реалізації маркетингових стратегій, як-от знижки, промокоди, розсилки та акції. Завдяки широкій екосистемі плагінів, система може бути інтегрована з платіжними шлюзами, службами доставки, CRM-системами тощо, що забезпечує гнучкість та масштабованість під потреби конкретного магазину.

Проте, незважаючи на численні переваги, PrestaShop [19] має і певні обмеження. Основним з них є необхідність додаткової технічної підтримки та розширення функціональності через модулі, багато з яких є платними. Також при значному зростанні обсягу товарів чи трафіку, система потребує оптимізації продуктивності для уникнення навантаження на сервер.

Крім того, базова версія PrestaShop [19] не містить вбудованих рішень для складського обліку на рівні великого рітейлу або повноцінної багатоскладської логістики, що може потребувати розробки або впровадження сторонніх рішень. Інтеграція з ERP-системами чи бухгалтерськими програмами також потребує індивідуального налаштування.

Таким чином, хоча PrestaShop [19] є сучасним і функціональним інструментом для створення онлайн-магазину електроніки, для повного задоволення специфічних потреб такого бізнесу доцільним є впровадження додаткових модулів або розробка індивідуального функціоналу, що дозволить адаптувати систему під конкретні бізнес-процеси.

1.3 Постановка завдання

Основною метою розробки програмного забезпечення для магазину електроніки є спрощення та автоматизація найбільш важливих бізнес-процесів, які включають управління асортиментом, замовленнями, базою клієнтів, аналітикою продажів тощо. Система допоможе в ефективному обліку товарів, контролі їх готовності до складу, обробці замовлень і наданні високого рівня клієнтського сервісу через найшвидший доступ до необхідних для цього даних.

Для досягнення вище згаданих цілей платформа має містити ретельно структуровану базу даних, котра зберігатиме повну інформацію про товари (назва, опис, характеристики, ціна, фактична наявність), клієнтів (персональні дані, історія транзакцій) та фінансові операції. Ефективність цієї бази даних зумовлюється можливостями своєчасного оновлення інформації, гнучкого пошуку та сортування даних, а також підтримки системи персоналізованих рекомендацій для клієнтів на основі їхніх минулих покупок або переваг.

Крім того, система повинна мати модуль автоматичних сповіщень, таких як попередження про необхідність поповнення запасів на складі, отримання нового замовлення чи завершення оформлення замовлення від

покупця. Ці механізми дозволять оперативно реагувати на потреби бізнесу та покращити показники конверсії продажів. Програмне забезпечення буде зосереджено на таких ключових функціях:

- система дозволить адміністраторам магазину зберігати та оновлювати детальну інформацію про товари, включаючи назву, опис, категорію, вартість, доступність на складі та характеристики продукції;
- додаток надаватиме можливість вести облік клієнтів: реєструвати нових користувачів, зберігати контактні дані, історію замовлень і статуси оплат;
- система оброблятиме замовлення в реальному часі, фіксуватиме їх статуси (наприклад, “нове”, “оплачено”, “відправлено”, “доставлено”), а також забезпечуватиме сповіщення для адміністратора про зміну статусу або нестачу товару на складі;
- програмне забезпечення автоматично генеруватиме звіти щодо продажів, популярності товарів, активності користувачів, що допоможе власнику магазину приймати обґрунтовані рішення щодо закупівель, маркетингу й оптимізації асортименту.

Інтерфейс буде організований у звичний і зручний спосіб: адміністратор зможе легко додавати нові товари, змінювати наявні, перевіряти стан замовлень, складати звіти на основі даних, а в разі потреби - зупиняти або змінювати якісь дії.

Очевидно, що побудована система стане великою підмогою в управлінні магазином. Крім того, вона також дозволить реорганізувати сфери обліку, обробки замовлень та комунікації з клієнтами таким чином, щоб операції здійснювалися ефективно. Новостворений веб-додаток стане місцем, де бізнес зможе реалізувати всі ці функції - зменшити кількість помилок, підвищити рівень задоволеності клієнтів, а також допоможе у дотриманні правил електронної комерції.

1.4 Функціональні та нефункціональні вимоги

Функціональні вимоги:

1. Аутентифікація користувачів: система повинна забезпечувати безпечний вхід до облікового запису за допомогою імені користувача та пароля (із підтримкою JWT токенів або ASP.NET Identity).
2. Рольовий доступ: підтримка різних ролей користувачів (наприклад, адміністратор, менеджер, працівник складу, клієнт) з різним рівнем доступу до функціоналу системи.
3. Управління товарами: можливість створення, редагування, видалення та перегляду інформації про електронні товари (назва, опис, ціна, категорія, характеристики, наявність на складі).
4. Управління клієнтами: зберігання персональної інформації клієнтів: ім’я, електронна пошта, адреса доставки, контактні дані.
5. Обробка замовлень: система повинна зберігати історію замовлень клієнтів, їхній поточний статус (нове, оплачено, обробляється, доставлено) та пов’язані платіжні дані.
6. Додавання відгуків: клієнти можуть залишати відгуки про товари, які вони придбали, включаючи оцінку та коментар.
7. Категоризація товарів: товари мають бути згруповані за категоріями, брендами або іншими властивостями, щоб полегшити навігацію й пошук.
8. Система сповіщень: повідомлення для адміністраторів про зменшення кількості товарів на складі, нові замовлення або технічні помилки.
9. Формування звітів: автоматичне створення звітів про обсяги продажів, активність користувачів, найпопулярніші товари тощо.

Нефункціональні вимоги:

1. Висока продуктивність: система має ефективно працювати з великим обсягом даних (тисячі товарів, замовлень, клієнтів) без помітного зниження швидкодії.
2. Реальний час: обробка змін у замовленнях, оновлення статусу доставки чи зміна цін мають відбуватися без затримок.
3. Зручний інтерфейс: UI повинен бути інтуїтивним і зручним для користувачів із різним рівнем технічної підготовки (адміністраторів, клієнтів).
4. Мінімальне навчання: інтерфейс повинен бути настільки зрозумілим, щоб користувач міг виконувати базові дії (додавання товару, створення замовлення, перегляд звітів) без попереднього навчання.
5. Масштабованість: система має легко розширюватись — додавання нових категорій, ролей, інтеграція з платіжними або логістичними API.
6. Легка підтримка: можливість оновлення функцій або додавання нових модулів без зупинки роботи системи.
7. Надійність: система має забезпечувати стабільну роботу з мінімальними простоями, включаючи пікові навантаження (акції, розпродажі).
8. Відновлення після збоїв: обов’язкова наявність механізмів резервного копіювання, логування та відновлення системи після критичних помилок.

1.5 Вимоги до інтерфейсу користувача

Інтерфейс користувача (UI) веб-орієнтованої інформаційної системи для магазину електроніки має бути орієнтований на зручність використання, швидкість взаємодії та привабливий сучасний вигляд. Оскільки система призначена для широкого кола користувачів — від адміністратора до

звичайного покупця — інтерфейс повинен забезпечувати зрозумілу, послідовну взаємодію незалежно від рівня технічної підготовки. Ключовим завданням UI є забезпечення максимальної доступності функцій із мінімальними зусиллями з боку користувача. Єдина мова дизайну, адаптивна верстка, зрозуміла навігація та швидкий доступ до основних дій створюють позитивний користувацький досвід.

Система відкривається екраном входу, що містить чисту форму авторизації з полями для введення електронної пошти та пароля, а також посиланнями на відновлення пароля та реєстрацію нового користувача. Після входу користувач потрапляє на головну інформаційну панель. Для адміністратора це може бути огляд замовлень, кількості товарів у наявності, найпопулярніших категорій і швидкого доступу до модулів керування (товари, клієнти, замовлення, відгуки, звіти). Для клієнта — це персоналізоване вітання з коротким оглядом останніх переглядів, поточних замовлень і рекомендованих товарів. Верхня панель забезпечує швидкий пошук по товарах, доступ до кошика, профілю та меню з категоріями.

Кожен розділ інтерфейсу — керування товарами, оформлення замовлень, перегляд звітів або редагування профілю — структурований за принципом мінімалізму: ключова інформація подається у вигляді таблиць, карток або списків, із можливістю швидкого сортування, фільтрації та редагування. Кнопки мають чітке маркування й логічне розташування, а візуальні індикатори (наприклад, кольори статусу замовлень або повідомлення про низький рівень товару) полегшують розуміння ситуації без потреби в додатковому натисканні.

Інтерфейс пристосований до мобільних пристроїв, забезпечуючи однакову ефективність і на смартфонах, і на десктопах. Реалізовані спливаючі повідомлення, модальні вікна підтвердження дій та інтуїтивні піктограми дозволяють користувачам діяти швидко, зменшуючи кількість помилок. Загалом, система створена для того, щоб користувачі могли виконувати

повсякденні завдання — від перегляду товару до оформлення або обробки замовлення — максимально просто, швидко й ефективно.

Модуль для керування товарами в магазині електроніки надає користувачам зручний і ефективний інтерфейс для керування асортиментом продукції. Усі товари представлені у вигляді списку з можливістю пошуку та сортування, що дозволяє швидко фільтрувати за різними критеріями, такими як назва, категорія, ціна або наявність на складі. Користувачі можуть вибрати конкретний товар для перегляду детальної інформації, що включає зображення продукту, його технічні характеристики, ціну та доступність на складі. Інтерфейс також підтримує редагування існуючих товарів, даючи можливість змінювати їхню інформацію або додавати нові варіанти (кольори, розміри, модифікації).

Для полегшення введення та редагування даних інтерфейс містить логічно структуровані форми, де кожен параметр товару, такий як назва, ціна, опис, категорія або виробник, має окремі поля введення.

Для управління замовленнями система пропонує зручний інтерфейс для перегляду та обробки замовлень клієнтів. Кожне замовлення можна переглядати в детальному форматі, що включає особисті дані клієнта, перелік замовлених товарів, статус обробки, оплату та доставку. Користувачі можуть швидко змінювати статус замовлення, додавати нові товари або редагувати існуючі записи. Інтерфейс дозволяє зручно відстежувати прогрес виконання замовлення та приймати необхідні дії.

Створення звітів у модулі управління товарами також є важливою функцією. Користувачі можуть створювати звіти з попередньо визначеними шаблонами для типових запитів або використовувати гнучкі фільтри для створення кастомізованих звітів. Звіти можна експортувати в популярні формати, такі як PDF або Excel, що дозволяє зручно працювати з даними поза системою. Сповіщення в реальному часі допомагають підтримувати ефективність роботи, сповіщаючи користувачів про необхідність обробки

замовлень або виконання інших важливих дій, таких як оновлення статусу товарів або залишків на складі.

Налаштування веб-орієнтованої інформаційної системи для магазину електроніки забезпечило зручність контролю інформації про клієнтів, управління обліковими записами адміністраторів, що відповідають за рівні доступу користувачів, і загальну безпеку всієї системи. Обов'язок і право адміністратора - контролювати всі дані користувачів, перевіряючи, редагуючи або видаляючи їхні облікові записи та призначаючи їм певні ролі, будь то менеджер з продажу, оператор складу або системний адміністратор.

За допомогою ролей система може контролювати доступ співробітників до окремих модулів, що є одним із джерел конфіденційності та запобігання несанкціонованим змінам. Крім того, для забезпечення надійності даних, система має можливість автоматичного створення резервних копій, а також швидкого відновлення системи у випадку збою.

Щоб зберегти індивідуальність системи, користувачі можуть не тільки змінювати мову інтерфейсу, але й змінювати вигляд, вписуючи темну тему, масштабуючи текст або постійно переміщаючись по клавіатурі, щоб увімкнути ці функції доступності. Завдяки цим функціям ви можете зробити систему максимально зручною для різних працівників, які відвідують її щодня протягом більшої частини часу.

2 МОДЕЛЮВАННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

2.1 Загальні відомості

У контексті створення Web-орієнтованої інформаційної системи для магазину електроніки, уніфікована мова моделювання UML [4] відіграє не менш ключову роль, ніж у розробці будь-якої іншої складної програмної системи. Вона забезпечує формалізований підхід до аналізу, проєктування та документування архітектури програмного продукту, дозволяючи команді розробників, бізнес-аналітиків і замовників ефективно спілкуватися, ґрунтуючись на єдиному візуальному представленні системи.

Завдяки UML [4] можна структуровано описати основні сутності магазину електроніки — такі як товари, категорії, користувачі, замовлення, кошик, постачальники, платежі тощо — а також показати, як вони пов’язані між собою та взаємодіють у межах платформи. Наприклад, діаграми класів [10] дозволяють візуалізувати структуру бази даних, включно з атрибутами та зв’язками між об'єктами. Це забезпечує чітке розуміння внутрішньої моделі даних ще до реалізації.

Діаграми варіантів використання (use case diagrams [6]) демонструють, як основні ролі — адміністратор, покупець, менеджер або оператор складу — взаємодіють із системою: переглядають каталог, оформлюють замовлення, керують запасами, обробляють повернення тощо. Це допомагає виявити всі сценарії, які система має підтримувати.

Діаграми діяльності (activity diagrams [9]) і послідовності (sequence diagrams [8]) корисні для опису ключових бізнес-процесів, таких як оформлення замовлення, обробка платежу чи доставка товару. Вони дозволяють розробникам чітко бачити потоки даних і логіку виконання, що сприяє зниженню помилок під час реалізації.

Застосування UML [4] на етапах планування та проєктування не лише сприяє побудові системи, що відповідає бізнес-потребам, але й забезпечує

якісну документацію, яка залишатиметься корисною у процесі підтримки та масштабування платформи. UML [4] значно знижує ризик виникнення критичних помилок, покращує розуміння системи між учасниками проєкту та слугує надійним орієнтиром для реалізації функціоналу Web-орієнтованого магазину електроніки.

У розробці будь-якої складної програмної системи діаграми відіграють вирішальну роль у візуалізації архітектури, бізнес-процесів і взаємодії між її компонентами. Вони слугують ефективним інструментом комунікації між розробниками, аналітиками, дизайнерами та замовниками, надаючи всім учасникам єдине уявлення про структуру та поведінку системи в різних сценаріях.

Для Web-орієнтованої інформаційної системи магазину електроніки особливо актуальними є кілька типів діаграм UML [4], кожна з яких виконує свою специфічну функцію, охоплюючи як структурні, так і поведінкові аспекти.

Діаграми варіантів використання (Use Case Diagrams [6]) демонструють взаємодію між користувачами системи (покупцями, адміністраторами, менеджерами складу) та основними функціями — такими як перегляд товарів, оформлення замовлення, керування асортиментом, обробка платежів і доставка. Це дозволяє сформулювати функціональні вимоги з точки зору кінцевих користувачів.

Діаграми класів (Class Diagrams) [10] моделюють структуру бази даних і логіку предметної області: класи Продукт, Категорія, Користувач, Замовлення, Постачальник тощо. Вони відображають атрибути об’єктів і зв’язки між ними, що є основою для реалізації як бізнес-логіки, так і схеми збереження даних.

Діаграми діяльності (Activity Diagrams [9]) описують бізнес-процеси, наприклад, процедуру оформлення замовлення: від додавання товарів у кошик до підтвердження оплати. Це дозволяє виявити критичні точки, паралельні дії та моменти прийняття рішень.

Діаграми послідовності (Sequence Diagrams [8]) ілюструють послідовність взаємодії між об'єктами або модулями в межах певного сценарію — наприклад, обробка повернення товару або підтвердження транзакції

Діаграми компонентів (Component Diagrams) і діаграми розгортання (Deployment Diagrams) застосовуються для опису архітектури програмного забезпечення та розміщення компонентів на серверному обладнанні, що особливо актуально для Web-систем, розгорнутих у хмарному або локальному середовищі.

Загалом, UML-діаграми [4] не лише покращують розуміння та взаємодію між учасниками розробки, але й відіграють важливу роль у створенні якісної документації для подальшої підтримки та розвитку Web-орієнтованої системи магазину електроніки. Вони забезпечують наочне уявлення як про технічну, так і про функціональну структуру системи, сприяючи створенню добре організованого, масштабованого та ефективного програмного рішення, що відповідає потребам як бізнесу, так і кінцевих користувачів.

2.2 Об’єктне та функціональне моделювання

2.2.1 Діаграма прецедентів. Діаграма варіантів використання - це один з ключових інструментів моделювання в UML [4], який фокусується на фіксації функціональних вимог до системи з точки зору її користувачів. Замість детального опису реалізації, вона демонструє, що робить система і хто з нею взаємодіє. Це робить діаграми варіантів використання особливо корисними на ранніх стадіях розробки, коли важливо сформувати чітке уявлення про потреби користувачів і функціональність системи.

У контексті веб-інформаційної системи для магазину електроніки діаграма варіантів використання надає огляд основних функцій і ролей користувачів, які з ними працюють. Вона візуалізує акторів (наприклад, покупця, адміністратора, завідувача складу) і прецеденти (функції, які

можуть виконувати користувачі - реєстрація, перегляд товарів, оформлення замовлення, обробка доставки, управління товарами тощо).

Завдяки цій діаграмі можна швидко і наочно визначити, які ролі мають доступ до яких функцій, що дозволяє ефективно планувати архітектуру системи, розмежовувати права доступу, орієнтуватися на зручність взаємодії кінцевого користувача з системою.

Кожен актор належить до варіанту використання, за який він відповідає, і таким чином показує, як він взаємодіє з системою. Наприклад, клієнт може бути пов'язаний з такими варіантами використання, як «Переглянути каталог товарів», «Додати товар до кошика», «Оформити замовлення» або «Переглянути статус доставки». Тоді як адміністратор має повноваження виконувати «Керування продуктами», «Керування користувачами» або «Створення звітів про продажі».

Діаграма варіантів використання не тільки слугує документацією функціональних вимог - вона дозволяє всім зацікавленим сторонам (розробникам, тестувальникам, менеджерам проектів) однозначно розуміти, як користувачі будуть взаємодіяти з системою. Це гарантує повне покриття всіх необхідних функцій, знижує ризик нереалізації основних сценаріїв і стає відправною точкою для більш конкретного технічного моделювання та розробки, орієнтованої на практичні потреби кінцевих користувачів.

Розроблена діаграма прецедентів використання представлена на рис. 3.

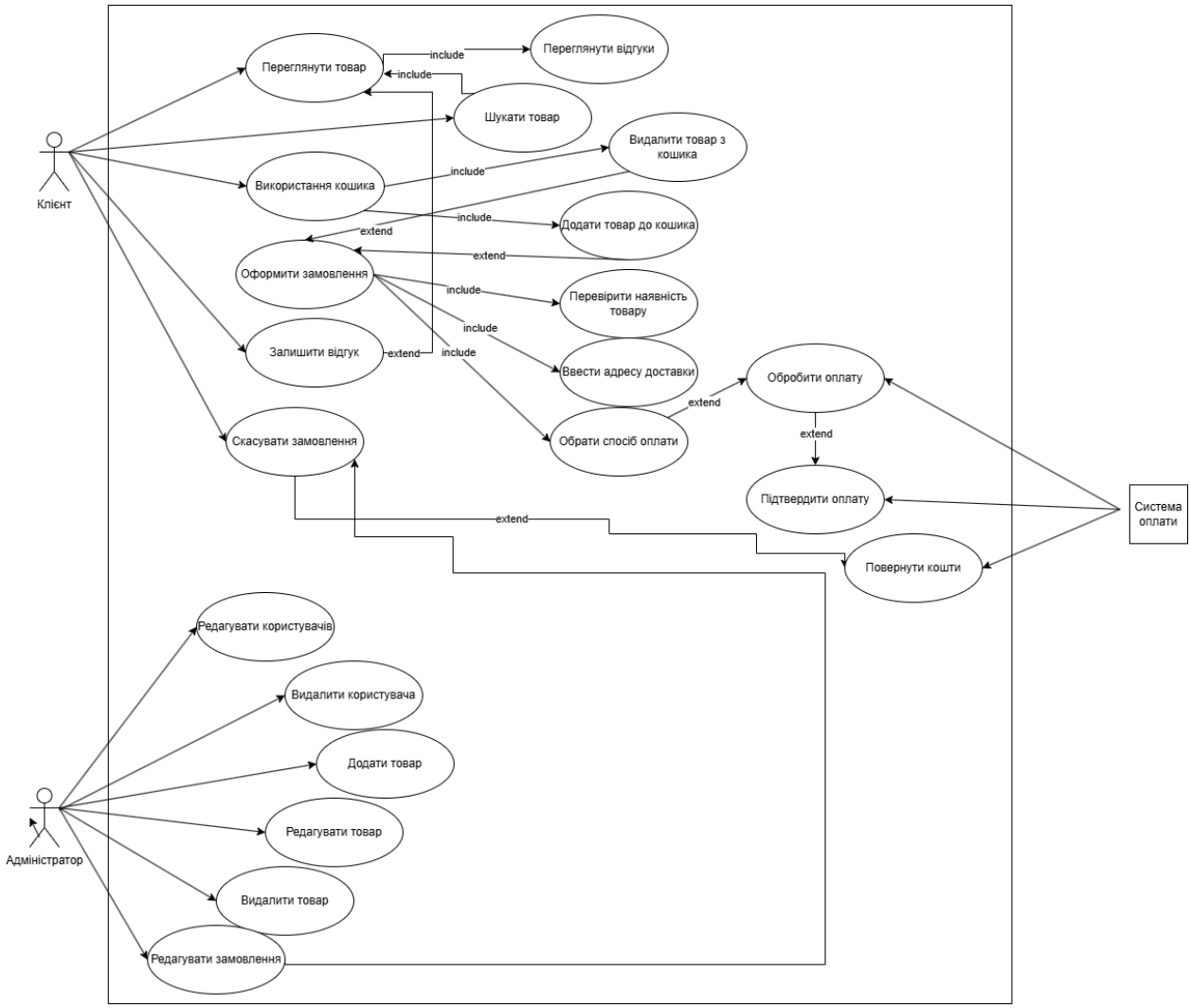

Рис. 3 Діаграма прецедентів [6]

Створена діаграма прецедентів містить акторів:

- “Клієнт”;
- “Адміністратор”;
- “Система оплати”.

Актор «Клієнт» включає такі прецеденти:

- переглянути товар;
- переглянути відгуки;
- шукати товар;
- використання кошика;
- додати товар до кошика;
- видалити товар з кошика;
- оформити замовлення;

- залишити відгук;
- скасувати замовлення;

Актор «Адміністратор» включає такі прецеденти:

- редагувати користувачів;
- видалити користувача;
- додати товар;
- редагувати товар;
- видалити товар;
- редагувати замовлення;

Актор «Система оплати» включає такі прецеденти:

- обробити оплату;
- підтвердити оплату;
- повернути кошти;

Прецеденти певним чином залежать одне від одного.

Розглянемо детальніше вищеописані прецеденти.

Сценарій використання: оформлення замовлення

Актор: клієнт

Мета: Придбати товари через онлайн-магазин, завершивши процес оформлення замовлення.

Передумови: Клієнт переглянув товари, додав бажані до кошика та має обліковий запис (за потреби).

Опис сценарію:

1. Клієнт заходить до онлайн-магазину та переглядає доступні товари, використовуючи функцію пошуку або категорії.
2. Переглядаючи товар, клієнт читає відгуки інших покупців, щоб прийняти рішення.
3. Обрані товари додаються до кошика. При потребі клієнт може перевірити наявність товару.

4. У кошику клієнт може змінити кількість товару або видалити його.
5. При оформленні замовлення клієнт вводить адресу доставки та вибирає спосіб оплати.
6. Система обробляє платіж через зовнішню систему оплати. Після цього підтверджує оплату.
7. Клієнт отримує повідомлення про успішне оформлення замовлення.

Альтернативні потоки:

1. Якщо клієнт вирішує скасувати замовлення до завершення оплати, система дозволяє це зробити.
2. Якщо оплата не проходить, клієнт може вибрати інший спосіб оплати або повторити спробу.
3. При помилці в адресі доставка система просить виправити дані перед продовженням.

Цей сценарій ілюструє типову реальну дію покупця в онлайн-магазині та відображає практичний процес здійснення покупки. Він забезпечує послідовну взаємодію користувача із системою – від вибору товару до завершення оплати – і гарантує, що усі дані про замовлення, включаючи доставку та платіж, фіксуються для подальшої обробки, відстеження статусу, аналітики та обслуговування клієнтів. Це ключовий процес для забезпечення клієнтського сервісу та комерційного функціонування магазину.

Розглянемо ще один сценарій використання.

Сценарій використання: редагування товару

Актор: адміністратор

Мета: Оновити інформацію про товар у системі.

Умови: Адміністратор аутентифікований у системі та має права доступу на редагування.

Опис сценарію:

1. Адміністратор заходить у панель управління товарним каталогом.

2. Зі списку товарів він обирає той, який потребує змін.
3. У відповідній формі редагує поля: назву, опис, ціну, категорію, доступність, фото тощо.
4. Після внесення змін натискає кнопку «Зберегти».
5. Система перевіряє коректність введених даних та оновлює інформацію в базі.

Альтернативні потоки:

1. Якщо товар більше не актуальний, адміністратор може видалити його замість редагування.
2. Якщо товар ще не існує, адміністратор може скористатися функцією додавання нового товару.
3. При введенні некоректних даних система не дозволяє зберегти зміни до виправлення помилок.

Цей сценарій ілюструє типову дію адміністратора онлайн-магазину щодо оновлення інформації в каталозі товарів. Він забезпечує своєчасну актуалізацію даних про продукцію, що є критично важливою для точного інформування клієнтів, запобігання помилкам у замовленнях та ефективного управління складськими залишками. Такий сценарій підтримує внутрішній контроль, аналітику продажів і репутацію магазину як надійного джерела товарів.

2.2.2 Діаграма послідовності. Діаграма послідовності [8] - це інструмент, який використовується при розробці програмного забезпечення для візуального представлення зв'язку між різними частинами системи в часі. Вона дає уявлення про те, як користувачі або об'єкти в системі використовують обмін повідомленнями для спілкування один з одним, і таким чином розбиває логіку певного процесу або сценарію. На відміну від статичних діаграм, діаграми послідовності [8] підкреслюють часову послідовність дій, що не тільки допомагає розробникам краще зрозуміти динамічну поведінку системи, але й робить систему більш ефективною.

Діаграми послідовності [8] мають велике значення при проектуванні веб-інформаційної системи для магазину електроніки, оскільки вони показують нам ключові процеси, що відбуваються в системі, такі як розміщення замовлення, авторизація користувача, редагування інформації про товар або перегляд статусу доставки. Наприклад, коли клієнт знаходиться в процесі оформлення замовлення, блок-схема пояснює, як клієнт взаємодіє з веб-інтерфейсом, як дані надсилаються на сервер, як вони потім обробляються серверною частиною, звіряються з базою даних і повертаються у вигляді підтвердження або повідомлення про помилку. Таким чином, він візуально представляє весь життєвий цикл запиту від отримання до виконання.

Процес показаний у вигляді користувачів, веб-інтерфейсу тощо, які зображені вертикальною лінією, що називається «лінією життя». Обміни повідомленнями між учасниками позначені стрілками з відповідними мітками, які є викликами функцій, запитами або відповідями. Фрагменти над лініями життя містять інформацію про смуги активації, які показують періоди, коли об'єкт є активним або перебуває в стані низької активності, очікуючи на відповідь.

Наприклад, під час оформлення замовлення в інтернет-магазині електроніки, коли клієнт кладе товар до кошика, система спочатку перевіряє його сесію або автентифікацію. Потім користувацький інтерфейс надсилає

запит до серверної частини для оновлення вмісту кошика. Підключення до бази даних сервера використовується сервером для перевірки наявності товару на складі, його ціни та поточних знижок. Потім кошик оновлюється і повертається на фронтенд, де користувач може його побачити. Якщо клієнт вирішує завершити покупку, система збирає інформацію про доставку, обробляє оплату за допомогою зовнішнього сервісу, зберігає замовлення в базі даних і надсилає користувачеві підтвердження.

Всі ці етапи вичерпно описані логічно послідовною схемою, яка наочно показує, як відбувається взаємодія між фронтендом, сервером, базою даних і сторонніми сервісами.

Ці візуальні ілюстрації - набагато більше, ніж просто картинки, вони є фундаментальною частиною технічної документації. Розробникам, бізнес-аналітикам та стейкхолдерам ці діаграми допомагають в ефективній комунікації. Також передбачено діаграми послідовності [8] для відстеження реакції окремих компонентів системи на дії користувача в режимі реального часу, які допомагають переконатися, що бізнес-логіка не тільки реалізована правильно, але й відповідає потребам замовника, а також є надійною.

Діаграма послідовності [8], зображена на рис. 4, включає наступні об'єкти:

- “Клієнт”;
- “Веб сайт”;
- “Сервер”;
- “БД”.
- “Система оплати”.

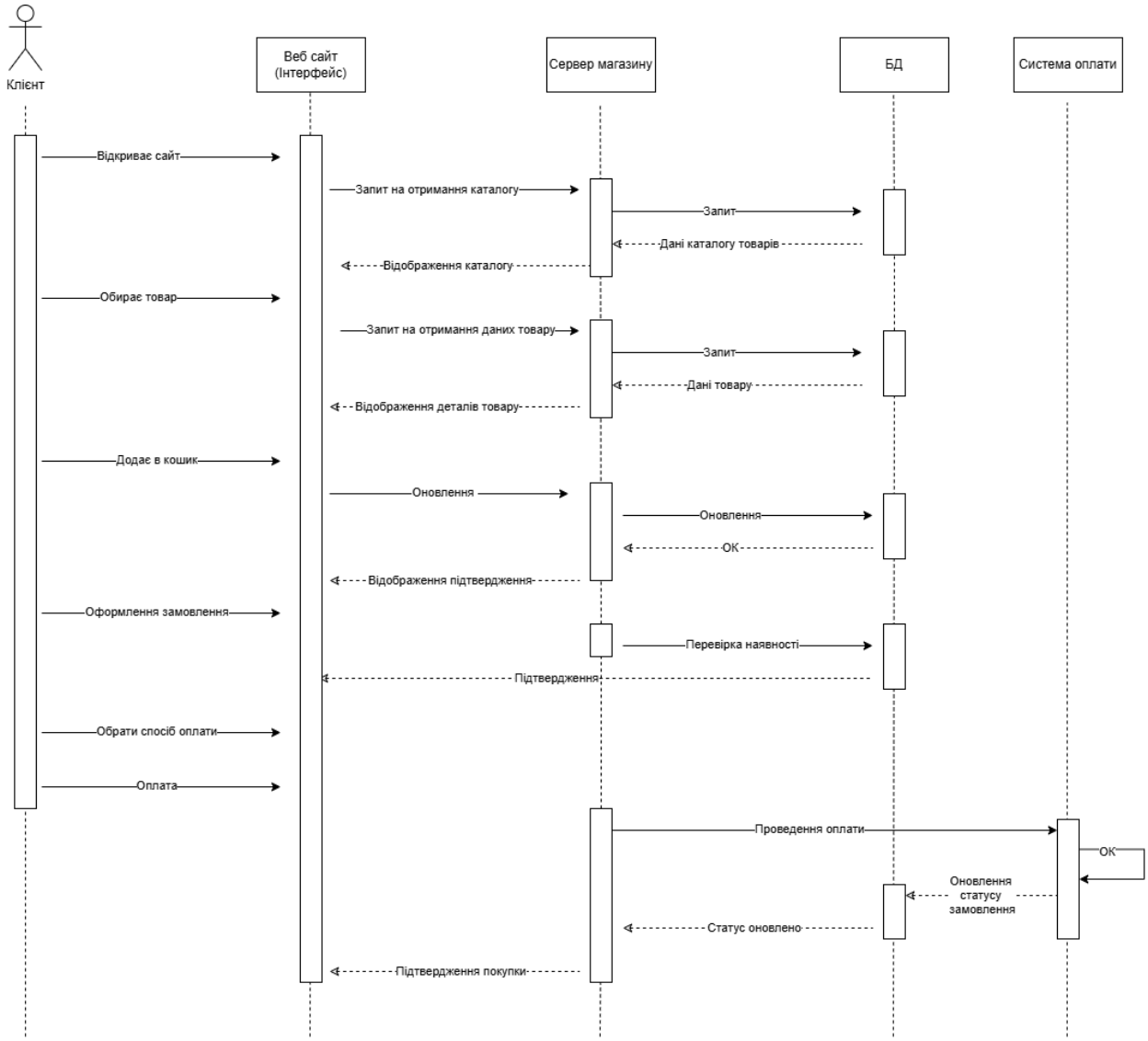

Рис. 4 Діаграма послідовності [8]

Діаграма ілюструє спільну роботу основних учасників - клієнта, веб-інтерфейсу магазину, серверної частини, бази даних і платіжної системи - таким чином, чітко показуючи послідовність дій в процесі онлайн-покупок.

Клієнт - це той, хто починає процес. Саме клієнт відкриває веб-сайт магазину, а інтерфейс магазину надсилає запит, який використовується для отримання каталогу товарів. Зареєстровані дані надаються сервером, а потім інформація безпосередньо відображається клієнту.

Після цього клієнт обирає товар. Веб-інтерфейс надсилає запит на деталі конкретного продукту, який потім обробляється сервером через базу даних. Таким чином, покупець отримує повну інформацію про товар.

Після того, як покупець кладе товар у кошик, система перевіряє сервер на наявність актуальних даних, і в базу даних вносяться зміни, що означає, що вибір підтверджено. На етапі оформлення замовлення відбувається перевірка транзакції купівлі. Сервер сканує складські запаси в базі даних, надсилає підтвердження, якщо товар є в наявності, тощо.

Після цього настає час для клієнта здійснити оплату. Покупець обирає спосіб оплати і після цього підтверджує покупку. Система запитує платіж за допомогою дистанційного способу оплати. Потім сервер оновлює базу даних зі статусом платежу, і остаточне підтвердження покупки передається покупцеві.

Діаграма ефективно передає синхронізовану взаємодію між усіма учасниками. Вона забезпечує розмежування між різними гравцями з їхніми відповідними обов'язками: інтерфейс реалізує взаємодію з користувачем, сервер перевіряє логіку і валідацію, завдання бази даних - зберігати інформацію, а платіжна система забезпечує фінансові питання. Такий підхід забезпечує безпечну, надійну та зручну роботу інтернет-магазину.

2.2.3 Діаграма активності. Діаграма діяльності [9] — це тип діаграми UML (Unified Modeling Language) [4], який використовується для моделювання динамічного потоку керування в системі. Він візуально представляє послідовність дій або кроків у процесі, показуючи, як дії виконуються від початку до кінця, і як керуються різними рішеннями, циклами та паралельними процесами.

Діаграми діяльності [9] зазвичай використовуються для моделювання бізнес-процесів, робочих процесів або будь-якої ситуації, коли потік контролю проходить через ряд дій або дій. На відміну від діаграм послідовності [8], які зосереджені на взаємодії між об’єктами, діаграми діяльності [9] висвітлюють потік контролю та те, як здійснюються різні процеси, часто допомагаючи проілюструвати складну поведінку більш зрозумілим способом.

На діаграмі діяльності [9] дії або завдання представлені у вигляді заокруглених прямокутників, а потік керування позначається стрілками, що

з’єднують ці завдання. Вузли прийняття рішень використовуються для представлення точок розгалуження, де процес може слідувати одним із кількох можливих шляхів залежно від умови. Вузли злиття об’єднують різні потоки, тоді як розгалуження та об’єднання використовуються для демонстрації паралельної обробки. Початкова точка (позначена зафарбованим колом) і кінцева точка (показана у вигляді кола з рамкою) допомагають позначити початок і завершення дії.

Розроблена діаграма активності представлена на рис. 5

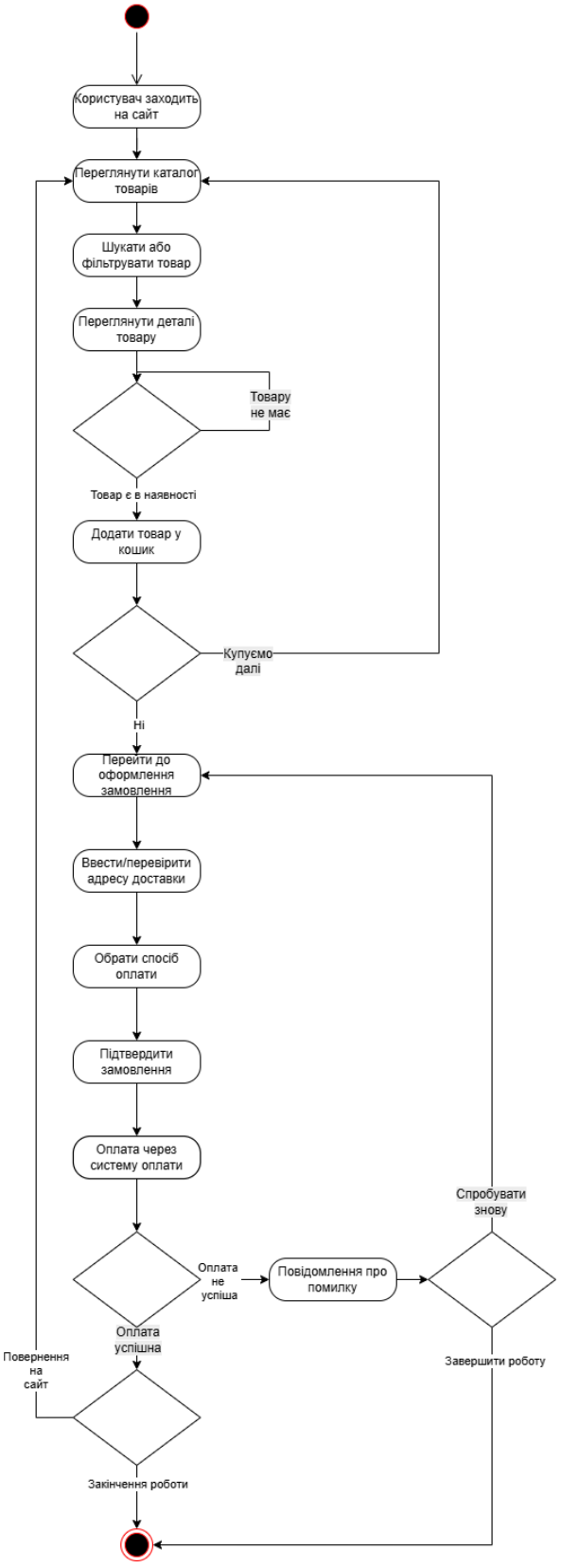

Рис. 5 Діаграма активності [9]

Ця діаграма діяльності [9] представляє ключові елементи взаємодії користувача з сайтом, які є основою інформаційної системи електронного магазину. Вона детально описує кроки транзакції - від входу на сайт до завершення покупки або виходу з системи.

Перший крок - це момент, коли користувач заходить на сайт, щоб переглянути товари, які пропонує каталог. Користувач може здійснювати пошук, фільтрувати товари або переглядати інформацію про конкретний товар. Якщо товару немає в наявності, користувач повертається до інтернет-магазину. Якщо товар є в наявності, його можна додати до кошика.

Після цього користувач може або продовжити покупки, або перейти до наступного етапу - оформлення замовлення. Для оформлення замовлення користувач заповнює адресу доставки, обирає спосіб оплати, підтверджує замовлення та здійснює оплату через платіжну систему.

Після успішного завершення платежу система підтверджує замовлення. Однак, якщо платіж не пройшов успішно, користувач отримує відповідне повідомлення і має можливість або спробувати ще раз, або покинути сайт.

Ця діаграма показує різні шляхи користувача та випадки помилок, пов'язані з недоступністю товару або помилкою оплати під час взаємодії з інформаційною системою через головний екран. Це може допомогти вам чіткіше зрозуміти логіку користувацького досвіду і буде корисним для подальшого аналізу та оптимізації системних процесів.

2.3 Абстракції предметної області

У сфері розробки програмного забезпечення термін "абстрагування" означає спрощення складних систем шляхом визначення основних функціональних можливостей, а побічні деталі залишаються поза увагою. У сфері програмного забезпечення для інтернет-магазинів електроніки абстракції поступаються місцем реальному відображенню взаємодії процесів користувача, адміністрації магазину та складу. Ці абстракції стосуються сутності об'єкта та

його зв'язків. Перевага цього полягає в тому, що він здатний представити функціональність системи таким чином, що дає чітке уявлення про неї з висоти пташиного польоту без необхідності вдаватися в подробиці реалізації.

Безумовно, у випадку програмного забезпечення для інтернет-магазину електроніки первинними об'єктами зазвичай є абстракції предметної області, такі як користувачі (профілі), товари, категорії, кошик, замовлення, оплата та логістичні модулі. Крім того, агрегуються основні залучені процеси: вибір товару, пошук, додавання в кошик, покупка, оплата і логістика; крім того, адміністрування асортименту і обробка замовлень. Слід чітко зазначити, що ця особливість узагальнення дозволяє розробляти функціональні модулі, які охоплюють найбільш часто виконувані операції, без необхідності управління дрібними завданнями

Мета використання абстракцій у цій предметній області - зробити програмну систему менш складною, гарантувати легкість запуску програми та її розширення, а також забезпечити чіткий метод взаємодії користувачів. Замість того, щоб виконувати окремі запити для кожного товару в базі даних, система працюватиме з "каталогом товарів" або "сторінкою товару" і, таким чином, автоматично надаватиме необхідну інформацію. Аналогічно, процес оплати виглядає як одна дія, хоча насправді він передбачає комунікацію з кількома сервісами.

Таким чином, абстракції в предметній області програмного забезпечення онлайн-магазину електроніки дозволяють зосередитись на основних функціональних вимогах, полегшують реалізацію бізнес-логіки та забезпечують зрозумілий інтерфейс для користувача. Це сприяє ефективному проєктуванню, зручному супроводу та розширенню системи в майбутньому.

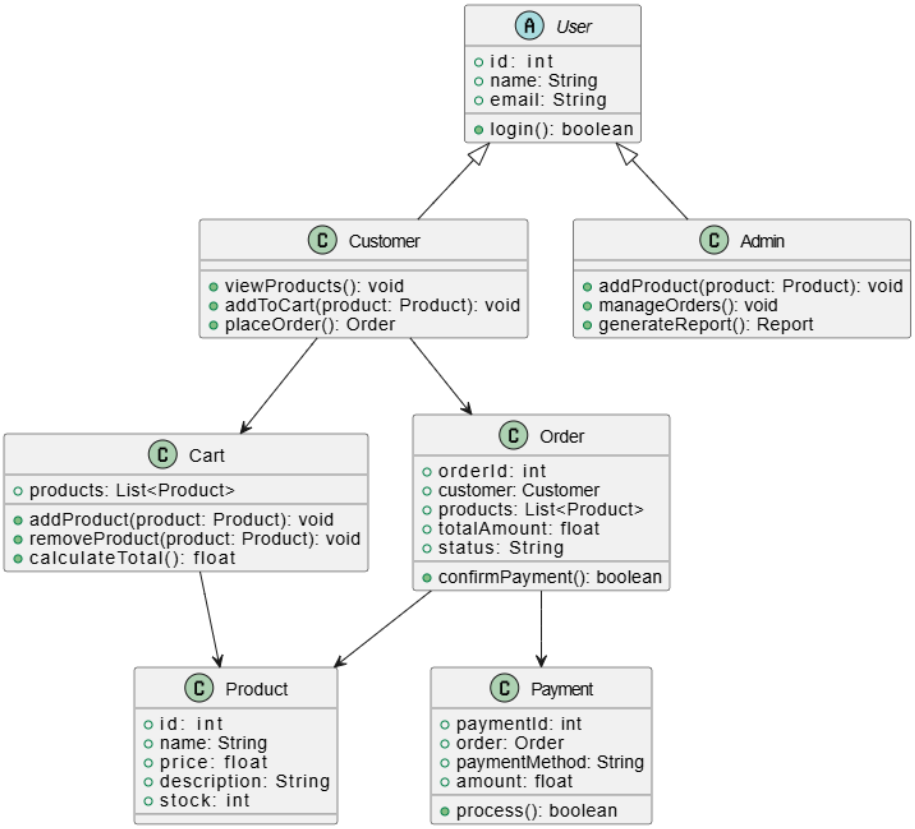

Рис. 6 Абстракції предметної області

2.4 Діаграма класів

Діаграма класів [10] - це дуже важливий документ об'єктно-орієнтованого проектування, який візуально показує статичну структуру інформаційної системи інтернет-магазину електроніки. Вона не тільки представляє взаємозв'язки між класами, але й виділяє атрибути та функціональні методи класів. Така діаграма допомагає зрозуміти організацію системи, логічний дизайн і взаємодію основних компонентів ще до етапу реалізації системи.

Розпізнати різні класи, їхні атрибути та методи, які вони містять, досить легко через їхнє представлення на діаграмі класів [10]. На діаграмі кожен клас показаний у вигляді прямокутника, який має три відсіки: верхній шар містить назву класу (з репрезентативною назвою - User, Order, Product), список атрибутів, а останній - методи (наприклад, placeOrder(), addToCart()). Це дає дуже чітке розуміння того, які дані зберігає кожна з сутностей і які дії виконує.

Крім того, діаграма пояснює зв'язки між класами, які ще більше прояснюють спосіб, у який об'єкти пов'язані між собою. Якщо ми візьмемо зв'язок між класами User і Order, то тим самим ми декларуємо той факт, що користувач, який має роль однієї сутності, ймовірно, матиме кілька замовлень. Наприклад, концепція успадкування позначає ієрархію, тобто класи Адміністратор і Замовник є підкласами загального класу Користувач.

Агрегація (як показано порожнистим ромбом) - це рішення, яке зробить класи Замовлення і Товар взаємопов'язаними, і це показує, що замовлення - це об'єкт, що складається з одного або декількох чисел іншого класу, товарів. Композиція (яка позначається заповненим ромбом) використовується для реалізації тісно пов'язаних об'єктів, наприклад, класу Cart і класу Cart Item. Залежність, зображена пунктирною стрілкою, використовується для позначення випадку, коли клас Payment Interface дійсно повністю контролюється зовнішнім сервісом, але не має до нього доступу.

Таким чином, в інтернет-магазині електроніки діаграма класів [10] - це блок-схема, яка документує дизайн системи, ідентифікує класи, розкриває їх атрибути, методи, а також форми взаємозв'язків, що значно полегшить роботу з реалізації та супроводу програмного забезпечення.

Діаграми класів [10] є життєво важливими для об'єктно-орієнтованого проєктування, оскільки вони надають чітке та структуроване уявлення про те, як побудована система. Визначаючи класи, їхні атрибути, методи та зв’язки між ними, діаграми класів [10] полегшують розуміння взаємодії компонентів системи та способів обробки даних. Наприклад, у системі онлайн-магазину електроніки діаграма класів [10] може включати такі сутності, як Користувач, Товар, Кошик, Замовлення та Оплата. Кожен із цих класів має відповідні атрибути — наприклад, ім’я, email, адреса доставки у класі Користувач, або назва, ціна, категорія у класі Товар. Методи визначають поведінку об’єктів, наприклад, додатиДоКошика(), оформитиЗамовлення() або здійснитиОплату().

Завдяки таким діаграмам розробники можуть ефективно моделювати логіку системи, визначати ролі класів і правильно організовувати їхню взаємодію. Це значно спрощує реалізацію, тестування та подальшу підтримку системи.

Для цього проєкту було створено спеціальну діаграму класів [10] з використанням об’єктно-орієнтованого підходу (рис. 7).

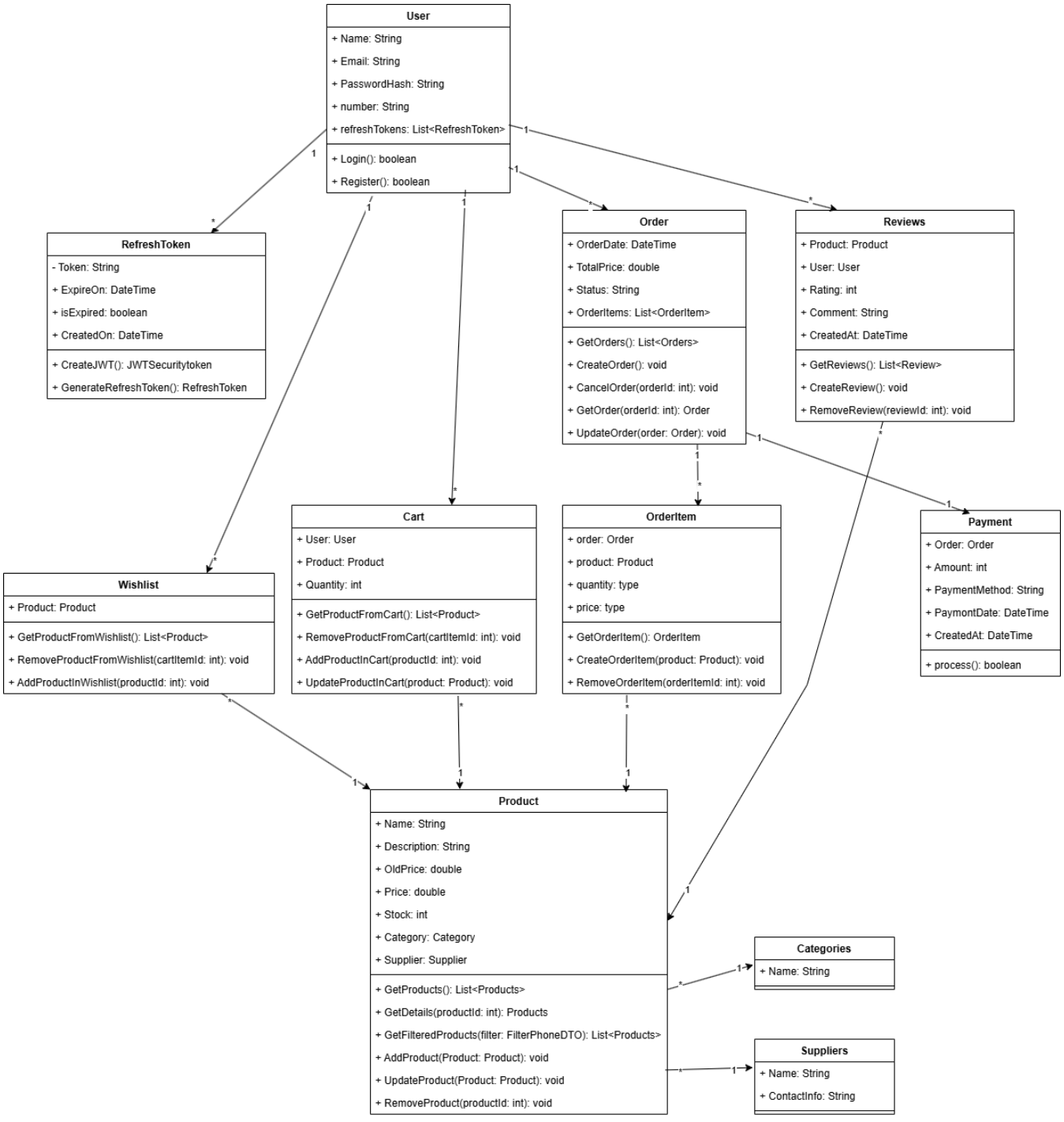

Рис. 7 Діаграма класів [10]

Представлена діаграма класів [10] моделює архітектуру програмного забезпечення для інтернет-магазину електроніки PixelPulse. Вона демонструє ключові сутності системи, їх атрибути та взаємозв'язки, що забезпечують ефективне управління товарами, користувачами, замовленнями, кошиком, оплатою та іншими бізнес-процесами.

Центральною сутністю системи є клас User, який містить основну інформацію про користувача і пов'язаний з сутностями Order, Cart, Wishlist, Reviews та RefreshToken для реалізації автентифікації, персоналізації та взаємодії з товарами.

Клас Product представляє продукт і містить такі атрибути, як назва, опис, ціна, залишок на складі, а також посилання на категорії та постачальників. Продукти можуть бути додані до Кошика або списку бажань, а також оцінені через систему відгуків.

Замовлення формується з декількох позицій (OrderItem) і має статус, дату, загальну вартість і посилання на оплату. Система дозволяє створювати, переглядати, оновлювати та скасовувати замовлення.

Для обробки платежів використовується клас Payment, який фіксує спосіб оплати, дату транзакції та асоціюється з конкретним замовленням. Крім того, через клас RefreshToken реалізована підтримка токенів поновлення для безпечної авторизації користувачів.

Таким чином, діаграма класів [10] відображає повну модель предметної області інтернет-магазину та підтримує ключові бізнес-функції системи.

2.5 Функціональна модель

Діаграма методу структурного аналізу та проектування (SADT) - це наочний приклад, який зображує системи в структурованому та ієрархічному вигляді. Метод особливо підходить для сприйняття складних систем і представлення систем та документації, оскільки він забезпечує чітке розуміння взаємозв'язків між різними частинами системи і способу, в який

інформація проходить через неї. SADT-діаграми зазвичай застосовуються в галузі системної інженерії та розробки програмного забезпечення для спрощення системи шляхом поділу складних частин на простіші компоненти, які можна легко зрозуміти і спроектувати.

Основна ідея SADT полягає в поділі системи на різні менші частини, де певну частину можна розглядати ізольовано, і, звичайно, аналізувати і проектувати далі. Процес роботи системи зазвичай пояснюється набором даних або функціональними діаграмами потоків, які описують, як частини взаємодіють з іншими частинами, виконуючи таким чином завдання, і як система отримує вигоду від цього. На діаграмах також можна побачити, з якими об'єктами система взаємодіє і які типи даних вона обробляє в процесі роботи, що ілюструють діаграми.

Перевага SADT-діаграми полягає в тому, що вона дозволяє отримати широкий огляд через її верхній рівень, а також детальне уявлення про систему. Отже, цей метод можна використовувати як для аналізу старих систем, так і для розробки нових. В принципі, SADT-діаграма може бути розпочата в загальному вигляді, щоб закінчитись на конкретній операції, що виконується в системі. Ієрархічна структура полегшує розуміння деталей складних систем і гарантує правильну ідентифікацію та облік усіх компонентів.

На практиці діаграми SADT (Structured Analysis and Design Technique) дуже добре підходять для моделювання бізнес-процесів в інформаційній системі інтернет-магазину електроніки. Вони допомагають розділити загальні функції системи на логічні частини, а отже, програмний продукт можна краще зрозуміти, спроектувати та підтримувати.

Одним з найпростіших способів виконання такого завдання є SADT-діаграми, в рамках яких реалізація процесу замовлення може бути наведена як приклад. На кожному кроці процес оформлення замовлення конкретизується серією рухів, таких як: вибір товару, додавання товару до кошика, підтвердження замовлення, здійснення оплати і, нарешті, генерація

квитанції. Кожен підпроцес зображений окремим блоком, а стрілки між ними ілюструють потік даних, управління, механізми або результати.

Ці діаграми можуть не тільки описувати формальну роботу управління користувачами, адміністрування продукту, обробку відгуків та повернень, управління постачальниками тощо, але й моделювати їх формально.

Говорячи більш конкретно, SADT-діаграми є чудовим засобом графічного представлення структури, що формує веб, отже, вони дозволяють чітко визначити обов'язки кожного компонента, пов'язати процеси і найбільш ефективно здійснювати систематичне проектування, що має особливе значення для масштабованості інформаційних систем.

Рис. 8 Функціональна модель

3 ПРОЄКТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ

3.1 Логічна модель даних

Логічна модель даних для Web-орієнтованої інформаційної системи магазину електроніки слугує концептуальним планом організації, обробки та взаємозв'язку даних у межах системи. Вона не залежить від конкретної реалізації чи технологій зберігання, а зосереджується на структурі даних і бізнес-правилах, які ці дані повинні відображати. Основна мета логічної моделі — забезпечити узгодженість структури даних із функціональними та бізнес-вимогами магазину.

У центрі логічної моделі розташовані сутності, що представляють ключові об'єкти бізнес-домену. Наприклад, такими сутностями можуть бути: Користувач, Товар, Замовлення, Кошик, Постачальник, Оплата, Відгук тощо. Кожна з цих сутностей має набір атрибутів, які описують її властивості: наприклад, для сутності Товар — це назва, ціна, опис, категорія, кількість на складі; для Користувача — ім'я, електронна пошта, адреса доставки тощо.

Важливу роль відіграють відносини між сутностями, які відображають логічні зв'язки між бізнес-об'єктами. Наприклад, один користувач може мати багато замовлень (зв'язок один до багатьох), а кожне замовлення містить кілька товарів через проміжну сутність OrderItem (зв'язок багато до багатьох). Сутність Відгук може бути пов'язана як з користувачем, що залишив відгук, так і з товаром, якого він стосується.

Такі логічні зв'язки зазвичай містять інформацію про кардинальність (1:1, 1:N, M:N) та дозволяють правильно проєктувати майбутню фізичну модель бази даних. Логічна модель даних є основою для побудови ER-діаграм [5] і подальшої реалізації ефективної, масштабованої та узгодженої структури даних магазину електроніки.

Іншим ключовим аспектом логічної моделі даних є використання первинних ключів, які виступають унікальними ідентифікаторами для

кожного екземпляра сутності. Наприклад, кожен товар у системі може мати унікальний ID товару, що дозволяє однозначно ідентифікувати його серед інших. Аналогічно, кожен користувач, замовлення чи постачальник мають власні унікальні ідентифікатори.

Для встановлення зв’язків між сутностями використовуються зовнішні ключі, які посилаються на первинні ключі інших сутностей. Наприклад, сутність Замовлення може містити зовнішній ключ, що вказує на конкретного користувача, який здійснив це замовлення, або зв’язок між замовленням і товарами реалізується через зовнішні ключі в проміжній сутності OrderItem.

Логічна модель даних створюється на ранньому етапі розробки системи, оскільки вона формує основу для подальшого проєктування фізичної бази даних. Вона забезпечує спільне розуміння структури даних для всіх учасників проєкту: як для розробників, так і для аналітиків і бізнес-стейкхолдерів.

На відміну від фізичної моделі, логічна модель не деталізує технічні аспекти зберігання, індексів чи типів даних. Її завдання — визначити сутності, атрибути, зв’язки та правила узгодженості, які повинні дотримуватись у системі.

У підсумку, логічна модель даних є стратегічним інструментом для ефективного проєктування Web-орієнтованої інформаційної системи магазину електроніки. Вона дозволяє забезпечити гнучкість, масштабованість і підтримуваність системи, водночас гарантуючи, що її структура відповідає бізнес-потребам і може адаптуватися до змін у майбутньому.

Логічна модель системи представлена на рисунку 9.

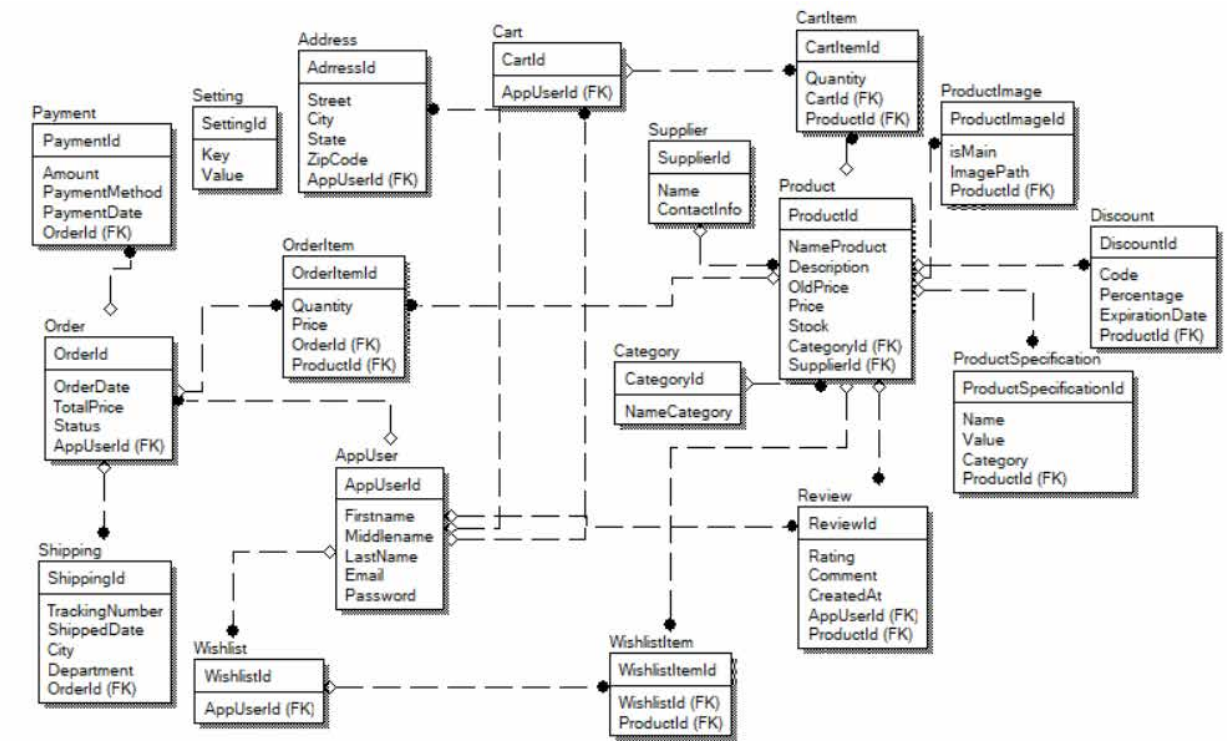

Рис. 9 ER-діаграма [5]

Ця ER-діаграма [5] представляє структуру взаємозв'язків між різними сутностями у базі даних веб-орієнтованої інформаційної системи магазину електроніки. Вона демонструє атрибути сутностей, зовнішні ключі та зв'язки між таблицями, забезпечуючи логічну модель для управління користувачами, замовленнями, товарами, оплатами, доставкою та іншими аспектами системи.

Сутності та їх атрибути:

1. AppUser — містить персональні дані користувача: ім'я, прізвище, по батькові, email та пароль.
2. Address — адреса доставки користувача (вулиця, місто, область, поштовий індекс).
3. Order — інформація про замовлення: дата, сума, статус та зв'язок із користувачем
4. OrderItem — деталі замовлення: товар, кількість і ціна.
5. Cart — кошик користувача.
6. CartItem — елементи у кошику (товар, кількість).

7. Product — опис товару: назва, опис, ціна, кількість на складі, категорія та постачальник.
8. ProductImage — зображення товару (шлях до зображення, основне чи ні).
9. ProductSpecification — технічні характеристики товару (назва, значення, категорія).
10. Discount — знижки: код, дата завершення дії, знижка на товар.
11. Category — категорія товару (назва категорії)
12. Supplier — постачальник товару (назва, контактна інформація).
13. Review — відгуки користувачів про товари (оцінка, коментар, дата).
14. Wishlist — список бажань користувача.
15. WishlistItem — товари в списку бажань.
16. Payment — оплата: метод, сума, дата, замовлення.
17. Shipping — доставка: номер відстеження, дата відправлення, область, відділення, зв’язок із замовленням.
18. Setting — налаштування системи: ключ і значення.

Зв’язки між сутностями:

1. AppUser пов’язаний з Order, Address, Cart, Review, Wishlist.
2. Order складається з кількох OrderItem, має зв’язок з Payment та Shipping.
3. Cart має багато CartItem, кожен з яких пов'язаний із певним Product.
4. Product зв’язаний із Category, Supplier, ProductImage, ProductSpecification, Discount, Review.
5. Wishlist пов’язаний з користувачем і включає WishlistItem, кожен із яких вказує на товар.
6. Address пов’язаний із користувачем.

7. Shipping та Payment пов’язані з замовленням через зовнішній ключ.

Ця ER-діаграма [5] створює логічну, гнучку і масштабовану структуру даних, що дозволяє системі ефективно керувати товарами, замовленнями, обслуговуванням клієнтів і підтримувати бізнес-процеси інтернет-магазину.

3.2 Вибір системи управління базою даних та її реалізація

Вибір правильної системи керування базами даних (СУБД) має вирішальне значення для будь-якої web-орієнтованої інформаційної системи, оскільки це безпосередньо впливає на продуктивність, масштабованість, безпеку та зручність обслуговування всієї інфраструктури. У контексті магазину електроніки, де обробляється велика кількість структурованих даних — таких як інформація про товари, замовлення, користувачів, оплати, постачальників та відгуки — необхідно обрати СУБД, здатну ефективно підтримувати ці операції.

Реляційна система керування базами даних (RDBMS) є найкращим вибором для такого проєкту, оскільки дозволяє організувати дані у вигляді взаємопов’язаних таблиць. Така структура ідеально підходить для зберігання і опрацювання даних про товари, клієнтів, замовлення та їх зв’язків між собою. Завдяки використанню SQL (Structured Query Language), розробники мають змогу легко створювати запити, фільтрувати, сортувати й агрегувати дані, що є критично важливим для роботи онлайн-магазину.

Серед доступних варіантів Microsoft SQL Server [26] вирізняється як надійна та масштабована СУБД корпоративного рівня. Вона забезпечує високу швидкодію, підтримку транзакцій, механізми резервного копіювання, шифрування та безпеки, що особливо актуально для комерційних web-систем. Крім того, SQL Server інтегрується з .NET-платформою, яка може використовуватися для реалізації самої системи, що робить цей вибір ще більш доцільним.

Під час впровадження бази даних першим кроком є розробка логічної моделі даних, яка ґрунтується на функціональних і бізнес-вимогах магазину електроніки. Ця модель описує ключові сутності системи (такі як «Користувач», «Продукт», «Замовлення», «Оплата», «Постачальник»), їхні атрибути (наприклад, назва товару, ціна, email клієнта, статус замовлення) та зв’язки між ними. Після визначення структури даних формується схема бази

даних, яка включає створення таблиць, визначення первинних ключів і встановлення зовнішніх ключів для забезпечення зв’язків між таблицями.

Наприклад, таблиця Order матиме зовнішній ключ, який пов’язує її з таблицею User, що дозволяє однозначно встановити, яке замовлення належить якому користувачу. Подібно, таблиця OrderItem міститиме посилання на Product, вказуючи, які саме товари включені в замовлення. Така реляційна структура сприяє цілісності даних і забезпечує зручну роботу з пов’язаною інформацією через SQL-запити.

Під час реалізації бази даних важливим етапом є створення індексів для стовпців, які часто використовуються в пошукових запитах (наприклад, Product.Name, Order.Status), що значно підвищує продуктивність системи. Також застосовується нормалізація для усунення надмірності та дублювання даних, зберігаючи логічну чистоту бази. Для забезпечення надійності слід впровадити регулярне резервне копіювання, а також стратегії аварійного відновлення, що дозволить системі безперервно функціонувати навіть у разі збоїв.

Після створення схеми бази даних її буде інтегровано з програмною частиною системи. Веб-застосунок для магазину електроніки взаємодіє з базою даних через SQL-запити, що дозволяє користувачам виконувати типові операції — перегляд товарів, оформлення замовлень, оновлення інформації про товари, оплату тощо. Наприклад, коли адміністратор додає новий товар або змінює ціну, застосунок генерує відповідний SQL-запит для вставлення або оновлення запису в таблиці Product.

Рішення використовувати MS SQL Server для реалізації бази даних інформаційної системи магазину було ухвалено з огляду на низку переваг, які забезпечують високу продуктивність, безпеку та масштабованість.

По-перше, MS SQL Server — це потужна реляційна СУБД корпоративного рівня, яка чудово підходить для зберігання структурованих даних з чітко визначеними зв’язками, як-от інформація про товари, користувачів, замовлення, оплату, постачальників тощо. У системі магазину,

наприклад, один користувач може мати кілька замовлень, а кожне замовлення містить перелік товарів. Реляційна структура дозволяє легко проєктувати ці взаємозв’язки, забезпечуючи цілісність даних і зручність у подальшому обслуговуванні.

По-друге, надійні функції безпеки MS SQL Server мають вирішальне значення, оскільки система оперує конфіденційною інформацією — зокрема особистими даними користувачів, історією замовлень і платіжними відомостями. СУБД підтримує шифрування, автентифікацію, контроль доступу на основі ролей та інші інструменти захисту, що дозволяє відповідати сучасним вимогам до збереження та обробки персональних даних клієнтів.

Таким чином, вибір MS SQL Server є оптимальним рішенням для реалізації надійної та безпечної бази даних у рамках веб-орієнтованої системи електронної торгівлі.

На додаток до безпеки, MS SQL Server забезпечує високу цілісність даних завдяки дотриманню принципів ACID (атомність, узгодженість, ізоляція, довговічність). Це означає, що всі транзакції, пов’язані з покупками, оплатами, оновленнями інформації про товари чи користувачів, виконуються повністю або не виконуються зовсім, зберігаючи цілісність бази даних навіть у разі збою системи. Для веб-магазину, який обробляє велику кількість одночасних транзакцій, таких як оформлення замовлень, повернення товарів, оновлення наявності товарів на складі — дотримання принципів ACID гарантує стабільну та надійну роботу.

Ще однією важливою перевагою MS SQL Server є його масштабованість. Очікується, що з ростом бізнесу зростатиме кількість клієнтів, товарів, транзакцій і аналітичних запитів. MS SQL Server здатен ефективно обробляти великі обсяги даних і запитів без втрати продуктивності, що є критичним для онлайн-магазину, де важлива швидка відповідь системи, зокрема в періоди пікового навантаження (наприклад, під час акцій, розпродажів тощо). Це дозволяє інформаційній системі

масштабуватись відповідно до потреб бізнесу, не вимагаючи повної перебудови інфраструктури.

Програмне забезпечення отримує більше переваг від широкого спектру вбудованих інструментів MS SQL Server, таких як розширені можливості запитів, збережені процедури та тригери. Ці інструменти відіграють центральну роль у розробці новітньої функціональності інтернет-магазину, дозволяючи ефективно автоматизувати складну бізнес-логіку, наприклад, оновлення статусів замовлень, створення звітів про продажі або відстеження залишків на складі. Крім того, збережені процедури допомагають виконувати операції з базою даних з центру, що, в свою чергу, призводить до підвищення продуктивності та зменшення кількості помилок при виконанні повторюваних запитів.

Ще одним важливим моментом є те, що MS SQL Server є придатною платформою для об'єднання інших технологій Microsoft, таких як .NET та C#, що дає можливість створити веб-додаток інтернет-магазину. Таке об'єднання дозволяє побудувати і встановити добре скоординовану і високоякісну систему, в якій база даних і логіка додатку функціонують безперебійно, дозволяючи здійснювати пошук даних, надання інформації в режимі реального часу та інші важливі функції на високій швидкості.

Таким чином, Microsoft SQL Server [26] пропонує всі необхідні функціональні можливості та високу продуктивність, щоб гарантувати успішну роботу сучасного інтернет-магазину електроніки. Він швидкий, безпечний і здатний до зростання, що є ознаками придатності сервера баз даних для обробки як транзакційних, так і аналітичних даних, що стосуються продажів, клієнтів, товарів і доставки. Завдяки відповідному дизайну бази даних та впровадженню сучасних механізмів обробки запитів, система може надавати послуги великій кількості онлайн-користувачів у режимі реального часу та одночасно ефективно обробляти величезні обсяги даних.

3.3 Архітектура програмного забезпечення

Архітектура програмного забезпечення інтернет-магазину електроніки розроблена таким чином, щоб бути надійною, розширюваною і безпечною системою, яка легко і в найкращий спосіб задовольняє всі потреби в інформації про товари, клієнтів, замовлення, доставку та оплату. В основу архітектури було покладено принципи чистої архітектури (Clean Architecture [2]), системи в цій структурі добре відокремлені, що дозволяє добре знати будь-яку частину і легше тримати її в руках протягом життєвого циклу проекту, систему легше тестувати і розвивати.

Система структурована на кілька незалежних рівнів, кожен з яких відповідає за окрему зону відповідальності. Рівень презентації, це інтерфейс користувача побудований у вигляді окремого веб-застосунку, який реалізує взаємодію з користувачами через Web UI. Покупці, адміністратори та постачальники можуть переглядати товари, оформлювати замовлення, залишати відгуки, керувати постачанням тощо. Цей рівень не містить бізнес-логіки, а лише відправляє HTTP-запити до ASP.NET Core Web API, який виступає посередником між UI та прикладною логікою.

Рівень бізнес-логіки(Application) використання бізнес-функціоналу у вигляді сервісів, команд, обробників запитів (наприклад, з використанням CQRS), інтерфейсів репозиторіїв, сервісів кешування тощо. Він не знає, як саме зберігаються або отримуються дані — лише оголошує контракти, які реалізуються на рівні Infrastructure. Усі основні сценарії, як-от обробка замовлення, реєстрація користувача, генерація звітів або перевірка товару на складі, реалізуються саме тут.

Рівень доменної моделі (Domain) відповідає за сутності (Entities), перелічення (Enums), value-об’єкти, інтерфейси доменних сервісів, бізнес-правила та інваріанти. Цей шар є незалежним від будь-якої інфраструктури чи технологій. Наприклад, сутності Product, Order, Customer, Review тощо

реалізовано разом з їхньою поведінкою (наприклад, логікою обчислення загальної вартості замовлення або перевірки наявності товару).

Рівень інфраструктури (Infrastructure) реалізує всі залежності, визначені в Application. Це включає реалізацію інтерфейсів репозиторіїв через Entity Framework Core, доступ до MS SQL Server, зовнішні сервіси, логування, електронну пошту, кешування MemoryCache і Redis, файлове сховище тощо. Саме тут налаштовуються контексти бази даних (DbContext), міграції, транзакції тощо.

Зв'язок між цими шарами чітко визначений і структурований. Веб-інтерфейс (UI) відправляє запити до Web API, який діє як фасад до рівня Application. API-контролери викликають команди або запити, які реалізують логіку в Application. Ті у свою чергу викликають інтерфейси репозиторіїв, які реалізуються в Infrastructure. Таким чином, дані проходять знизу вгору, обробляються, і результат повертається назад до користувача.

Безпека є ключовим аспектом системи, оскільки система обробляє конфіденційну інформацію, таку як персональні дані користувачів, платіжні дані, історія замовлень, велика увага приділяється захисту. Архітектура включає численні заходи безпеки, включаючи аутентифікація та авторизація реалізовано через ASP.NET Core Identity та JWT для API. Рольовий доступ такий як адмін, клієнт дозволяє обмежити функціональність. Валідація даних виконується як на UI, так і в Application-рівні. Шифрування конфіденційних даних, захист від SQL-ін’єкцій, CSRF, XSS.

Щоб веб-орієнтована інформаційна система для магазину електроніки залишалася ефективною та швидкодійною під час обробки великої кількості запитів і даних про товари, користувачів, замовлення й постачальників, рівень доступу до даних (Infrastructure) був оптимізований для продуктивності. Запити до бази даних реалізовані з урахуванням продуктивності, зокрема використовуються індекси на часто використовуваних полях, таких як артикул товару, email користувача або статус замовлення. Це значно пришвидшує пошук, фільтрацію та обробку

даних. Для підвищення продуктивності та зменшення навантаження на базу даних у рівні бізнес-логіки (Application Layer) реалізовані механізми кешування, наприклад, для часто використовуваних списків товарів або категорій. Це дозволяє зменшити кількість повторних запитів до бази даних та пришвидшити завантаження сторінок у Web UI. Також було додано асинхронну обробку на рівні запитів і рівні бізнес логіки. Для оптимізації на рівні інфраструктури було створення діставання об’єктів із БД не як об’єкт класу, а як запити до БД типу IQueryable, що зменшує навантаження на систему і використовує менше оперативної пам’яті, так як в ній знаходиться тільки запит, а не весь об’єкт і тільки коли він знадобиться він створиться.

Технологічний стек системи побудовано на основі сучасного та надійного набору інструментів, ASP.NET Core Web API забезпечує реалізацію API для обробки запитів від користувацького інтерфейсу (UI) та мобільних клієнтів. Web UI побудований як окремий фронтенд-клієнт на Razor Pages, який взаємодіє з API, обробляючи запити користувача й відображаючи отримані результати. C# використовується для написання бізнес-логіки у шарах Domain та Application, де реалізовані правила обробки замовлень, керування товарами, обробка платежів тощо, та MS SQL Server використовується як система керування базами даних, що забезпечує надійне зберігання інформації про продукти, користувачів, замовлення, відгуки та постачальників.

Інтеграція цих компонентів у рамках чистої архітектури дозволяє легко масштабувати систему, забезпечити гнучкість при впровадженні нових функцій (наприклад, додавання оплати онлайн чи фільтрації за характеристиками товару) та підтримувати високу продуктивність і безпеку в процесі обробки конфіденційних даних користувачів і фінансових транзакцій.

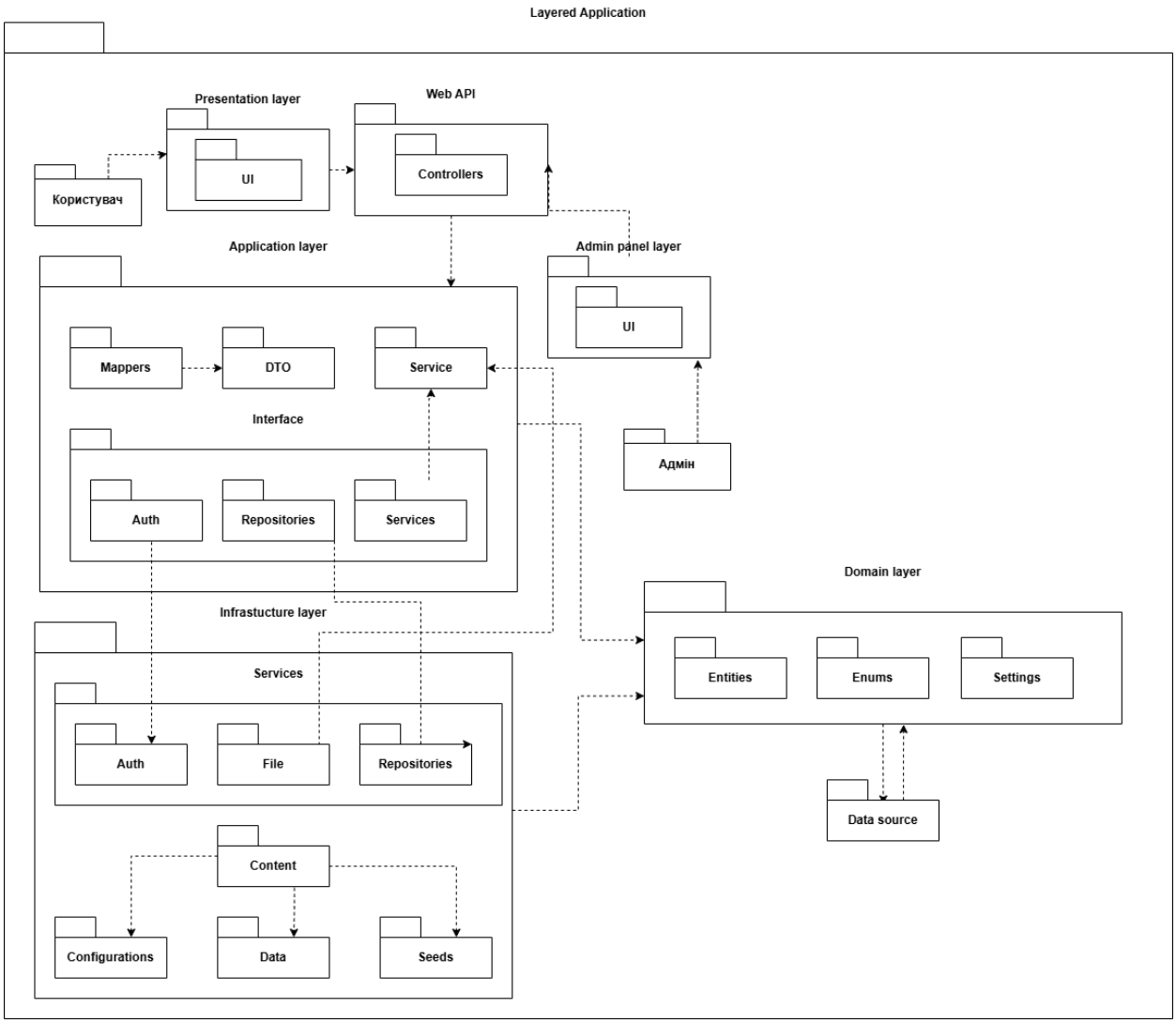

Рис. 10 Clean Architecture [2]

Підсумовуючи, програмна архітектура інформаційної системи інтернет-магазину електроніки була розроблена з урахуванням вимог безпеки, масштабованості та ремонтопридатності. Чіткий розподіл завдань на різних рівнях - від користувацького інтерфейсу до бізнес-логіки та інфраструктури - забезпечує зручність у розширенні та обслуговуванні системи. За допомогою MS SQL Server додаток може зберігати та отримувати дані у великих обсягах, зберігаючи при цьому продуктивність та надійність на високому рівні. Такий дизайн вирішує не тільки сучасні вимоги електронної комерції, але й створює можливості для інтеграції все нових і нових сервісів у майбутньому.

3.4 Організаційна структура програмного забезпечення

3.4.1 Діаграма пакетів. Діаграма пакетів [13] є важливою частиною процесу розробки веб-інформаційної системи, оскільки вона надає високорівневе уявлення про архітектуру магазину електроніки, групуючи пов'язані класи і компоненти в логічно організовані пакети. Це допомагає структурувати систему на зрозумілі функціональні блоки, що спрощує як обслуговування, так і масштабування програми.

У контексті інформаційної системи для магазину електроніки ця діаграма показує, як додаток розділений на кілька ключових модулів, кожен з яких відповідає за певні функції: управління товарами, обробка замовлень, управління користувачами, кошик, оплата, аналітика тощо. Кожен з цих модулів згрупований в окремі пакети, які взаємодіють між собою через чітко визначені інтерфейси.

Такий підхід забезпечує гнучкість, повторне використання коду та ізоляцію змін - наприклад, оновлення модуля обробки платежів не вплине на роботу модуля каталогу товарів. Це робить систему більш стійкою до змін, адаптованою до вимог бізнесу та зручною для подальшого розвитку.

Пакет інтерфейсу користувача (UI) відповідає за керування всіма елементами, пов’язаними з візуальним представленням системи магазину електроніки. Це включає веб-сторінки, кнопки, форми пошуку товарів, кошик, сторінки оформлення замовлення тощо. Пакет UI слугує точкою входу для користувачів – клієнтів, адміністраторів та менеджерів, дозволяючи їм переглядати каталог товарів, додавати продукти до кошика, оформлювати замовлення, залишати відгуки, а також керувати своїм профілем. Зв’язок між інтерфейсом користувача та рівнем бізнес-логіки має важливе значення, оскільки UI надсилає запити (через ASP.NET Web API) до відповідних сервісів і отримує оброблені результати для подальшого відображення.

Пакет бізнес-логіки (Application) є основою функціональної частини системи, відповідаючи за обробку запитів користувача та реалізацію ключових бізнес-процесів. Він виконує операції, пов’язані з оформленням замовлень, перевіркою доступності товарів на складі, застосуванням знижок, обчисленням загальної вартості кошика, реєстрацією користувачів, формуванням звітів тощо. Цей рівень реалізує логіку за допомогою сервісів і обробників запитів (use cases [6]), інкапсулюючи правила, які визначають поведінку системи. Пакет Application не взаємодіє безпосередньо з базою даних, а звертається до визначених інтерфейсів репозиторіїв, описаних у доменній моделі.

Пакет предметної області (Domain) містить основні сутності, що моделюють реальні об’єкти магазину електроніки: товари, замовлення, кошик, користувачі, відгуки, постачальники, категорії тощо. Кожна сутність має власну поведінку та інваріанти, які забезпечують правильність виконання дій у межах системи. Окрім сутностей, пакет також містить Value Objects (наприклад, ціна, адреса доставки) та доменні сервіси. Цей рівень є незалежним від реалізації інтерфейсу користувача, бази даних чи інших технічних деталей, що робить його стабільною основою архітектури та дозволяє легко адаптувати бізнес-логіку до нових вимог.

Пакет інфраструктури (Infrastructure) відповідає за реалізацію доступу до зовнішніх ресурсів системи. Це включає підключення до бази даних, реалізацію інтерфейсів репозиторіїв за допомогою Entity Framework Core, взаємодію з платіжними сервісами, сервісами доставки, API постачальників, електронною поштою та іншими зовнішніми компонентами. Цей рівень реалізує контракти, задані в Domain, і використовується в Application через ін’єкцію залежностей. Таким чином, Infrastructure забезпечує технічне підґрунтя для роботи всієї системи, не впливаючи безпосередньо на бізнес-логіку.

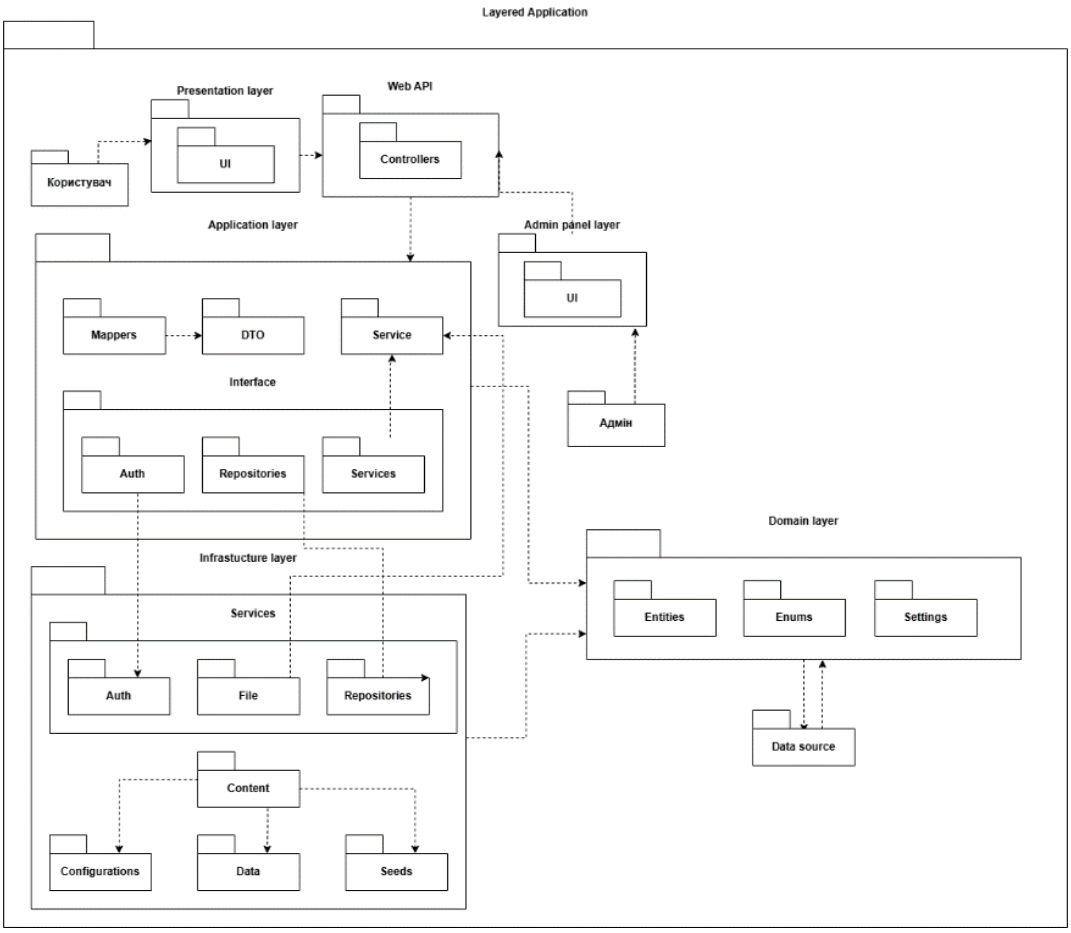

Рис. 11 Діаграма пакетів [13]

Ця діаграма пакетів [13] ілюструє багаторівневу архітектуру програмного забезпечення, розділену на логічні шари. Вона демонструє взаємозв’язки між компонентами системи, їх ролі та напрями обміну даними. Такий підхід сприяє модульності, масштабованості та спрощенню підтримки програмного коду.

Основні пакети включають:

- Presentation layer (UI), який відповідає за взаємодію з користувачем через UI та Web API. Тут розташовані елементи інтерфейсу та контролери, які приймають запити від користувача.;
- Admin panel layer – окрема частина для адміністративного інтерфейсу, яка має власний UI та апі для керування внутрішніми функціями.
- Application layer – містить сервіси, мапери, DTO та інтерфейси для авторизації, роботи з репозиторіями та іншими сервісами. Цей шар виступає посередником між UI та інфраструктурою, реалізуючи бізнес-логіку.
- Infrastructure layer – реалізує доступ до даних, файлових систем, конфігурацій та іншого зовнішнього середовища. Також містить реалізації сервісів, які описані в application layer.
- Domain layer – зберігає основну модель предметної області: сутності, енумерації та налаштування. Він забезпечує чисту бізнес-логіку, яка не залежить від технічних деталей реалізації.

Взаємозв’язки між шарами позначені стрілками, які показують напрямок викликів та залежностей. Це дає змогу чітко відокремлювати відповідальність кожного компонента, що полегшує тестування, зміну або заміну окремих частин системи без впливу на інші.

3.5 Вибір інструментарію для створення програмного забезпечення

Розробка веборієнтованої інформаційної системи для магазину електроніки вимагає ретельного підбору технологій, які забезпечують масштабованість, стабільну роботу, безпеку та зручний інтерфейс для користувачів і працівників магазину. Після аналізу вимог до системи, враховуючи потреби в управлінні товарами, замовленнями, клієнтами та адміністративними функціями, для реалізації проєкту були обрані такі інструменти: C#, ASP.NET Core, Entity Framework Core, MS SQL Server та Identity.

C# виступає основною мовою програмування завдяки своїй потужній підтримці веброзробки через ASP.NET Core. Його об'єктно-орієнтований підхід дозволяє ефективно організувати структуру коду, а також легко підтримувати та масштабувати систему в майбутньому.

ASP.NET Core [1] було обрано як основний вебфреймворк завдяки його високій продуктивності, кросплатформеності та гнучкості. Він дозволяє створювати сучасні багаторівневі вебдодатки, що добре масштабуються.

Entity Framework Core забезпечує зручну роботу з базою даних, дозволяючи автоматично зв'язувати об'єктну модель з реляційною структурою таблиць у MS SQL Server. Це спрощує реалізацію CRUD-операцій для управління товарами, категоріями, замовленнями, клієнтами тощо.

Для аутентифікації та авторизації використовується Identity, що забезпечує безпечну обробку даних користувачів, включаючи ролі адміністраторів, менеджерів і клієнтів магазину.

Додатково, для побудови зручного інтерфейсу адміністративної панелі та вітрини магазину, застосовується принцип розділення на шари (Presentation, Application, Infrastructure, Domain), як зображено на діаграмі. Така багаторівнева архітектура дозволяє ізолювати бізнес-логіку від

зовнішніх залежностей, полегшує тестування та подальшу підтримку системи.

Загалом, обрані інструменти створюють надійну основу для реалізації ефективної, безпечної та масштабованої системи управління магазином електроніки, орієнтованої як на внутрішній персонал, так і на кінцевих користувачів через вебінтерфейс.

ASP.NET Core [1] обрано як основну вебплатформу для розробки інформаційної системи магазину електроніки. ASP.NET Core [1] — це сучасний, високопродуктивний і кросплатформенний фреймворк, що дозволяє створювати масштабовані вебдодатки та Web API. Його модульна структура, підтримка залежностей та легка інтеграція з іншими технологіями Microsoft робить його ідеальним вибором для створення багаторівневої системи з чітким розділенням відповідальностей. У рамках цього проєкту ASP.NET Core [1] забезпечує надійну серверну частину, через яку здійснюється обробка запитів клієнтів, управління даними та бізнес-логікою магазину.

Для побудови API-інтерфейсів використовується ASP.NET Core Web API, що дозволяє реалізувати RESTful-сервіси для взаємодії з фронтендом (вітриною магазину) та адміністративною панеллю. Такий підхід забезпечує гнучкість, масштабованість та зручність підключення до різних клієнтів, включаючи мобільні додатки чи сторонні сервіси.

Entity Framework Core (EF Core) обрано як ORM-рішення для доступу до бази даних. EF Core дозволяє розробникам працювати з даними у вигляді об’єктів, автоматично перетворюючи їх на SQL-запити. Це значно спрощує реалізацію операцій з товарами, категоріями, замовленнями, користувачами та іншими елементами системи. Крім того, EF Core підтримує міграції, що дозволяє зручно керувати змінами структури бази даних у процесі розробки.

MS SQL Server виступає як основна система керування базами даних для проєкту. Це потужна, масштабована та безпечна платформа для зберігання даних. Вона забезпечує підтримку складних запитів, транзакцій і аналітичних операцій, що є критично важливим для обробки великої кількості замовлень і клієнтських записів. Завдяки вбудованим функціям безпеки, таким як автентифікація, авторизація та шифрування, SQL Server гарантує захист конфіденційної інформації про користувачів.

Для керування автентифікацією та авторизацією користувачів у систему інтегровано ASP.NET Core Identity. Ця бібліотека надає потужний набір функцій для обробки реєстрації, входу, керування паролями, скидання доступу та контролю прав на основі ролей. У контексті магазину електроніки Identity відповідає за забезпечення доступу до функціоналу залежно від ролі користувача: адміністратор, менеджер, або звичайний клієнт. Це дозволяє, наприклад, обмежити доступ до управління товарами, перегляду замовлень клієнтів або редагування інформації лише для авторизованих працівників.

ASP.NET Core Identity також забезпечує високий рівень безпеки завдяки вбудованим механізмам хешування паролів, підтримці багатофакторної автентифікації (2FA) та можливості реалізації гнучких політик контролю доступу. Це критично важливо для захисту облікових записів користувачів, персональних даних і історії покупок, що зберігаються в системі.

Використання C#, ASP.NET Core, Entity Framework Core, MS SQL Server та ASP.NET Core Identity забезпечує надійну технологічну основу для створення безпечної, масштабованої та зручної вебсистеми. Усі ці інструменти добре інтегруються між собою та відповідають сучасним стандартам розробки. Вони дозволяють ефективно реалізувати як клієнтську, так і адміністративну частину магазину, з дотриманням принципів багаторівневої архітектури та захисту даних.

Завдяки цьому стеку технологій, система здатна ефективно обслуговувати великий обсяг користувачів, витримувати зростання

функціональності та забезпечувати стабільну роботу в умовах реального навантаження. Такий підхід дозволяє легко масштабувати рішення, адаптуючи його до змін бізнес-вимог та зростання обсягу замовлень.

4 ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЯ СИСТЕМИ

4.1 Вимоги до апаратного та програмного забезпечення

Успішна робота веборієнтованої інформаційної системи для магазину електроніки значною мірою залежить від відповідності апаратного та програмного забезпечення певним вимогам. Ці вимоги гарантують стабільність, безпеку, швидкодію та зручність роботи як для клієнтів, так і для персоналу, який керує замовленнями, товарами та обліковими записами користувачів.

З апаратної точки зору система призначена для роботи на стандартному офісному обладнанні та серверах із сучасними характеристиками. Для клієнтської частини (браузера) достатньо комп’ютера або ноутбука з багатоядерним процесором, 8 ГБ оперативної пам’яті та стабільним підключенням до Інтернету. Вебінтерфейс не потребує потужного «заліза», але для адміністраторів і менеджерів, які працюють із великими обсягами даних або звітністю, рекомендується використовувати комп’ютери зі швидкодіючими SSD-накопичувачами та процесором не нижче Intel Core i5 або аналогічного.

Серверна частина, де розміщується ASP.NET Core Web API, потребує продуктивного середовища для обробки запитів, авторизації, взаємодії з базою даних і забезпечення швидкого відгуку користувачам. Оптимально — віртуальний або фізичний сервер з 4+ ядрами CPU, не менше 16 ГБ оперативної пам’яті та SSD-диском із достатнім обсягом для зберігання бази даних на основі MS SQL Server.

З боку периферії — для офісної роботи персоналу магазину необхідні монітор із роздільною здатністю не менше Full HD, стандартна клавіатура, миша, а також принтер для друку накладних, звітів та супровідної

документації. За потреби можна інтегрувати сканери штрих-кодів або POS-обладнання для підтримки торгівельного процесу.

Надійне підключення до Інтернету також є критично важливим для ефективної роботи веборієнтованої системи магазину електроніки, особливо в умовах багатокористувацького доступу або взаємодії з зовнішніми сервісами, такими як платіжні системи чи постачальники. Швидке дротове або стабільне бездротове підключення з достатньою пропускною здатністю забезпечує безперебійну передачу запитів, обробку замовлень у реальному часі та швидкий доступ до актуальної інформації, мінімізуючи затримки та збої.

Щодо програмних вимог, система розробляється з використанням мови програмування C# у середовищі Visual Studio. Основу становить ASP.NET Core Web API, який дозволяє реалізувати сучасну, масштабовану та кросплатформену архітектуру. Завдяки використанню чистої архітектури система поділена на логічні шари: Domain, Application, Infrastructure та UI, що дозволяє чітко розділити бізнес-логіку, інтерфейси та доступ до даних.

Інтерфейс користувача реалізовано у вигляді вебклієнта або окремого фронтенду, що взаємодіє з API, надаючи зручний доступ до функціоналу системи через браузер. Це дозволяє забезпечити як користувачів, так і працівників магазину адаптивним інтерфейсом без потреби встановлювати локальне ПЗ.

Компонент зберігання даних базується на MS SQL Server, що забезпечує надійність, безпеку та продуктивність при роботі з великою кількістю інформації — від товарів та категорій до замовлень і клієнтських профілів. Для взаємодії з базою використовується Entity Framework Core, який забезпечує зручну роботу з даними через об’єктно-реляційне відображення (ORM), що прискорює розробку і полегшує супровід.

Для автентифікації та керування ролями у системі використовується ASP.NET Core Identity, який дозволяє реалізувати безпечний контроль доступу з поділом прав між адміністраторами, менеджерами і звичайними

клієнтами. Identity підтримує сучасні засоби безпеки, такі як хешування паролів, авторизація на основі ролей, двофакторна автентифікація та інші механізми захисту.

Конфігурація локальної мережі або хмарної інфраструктури повинна забезпечувати захищений обмін файлами між користувачами системи, наприклад адміністраторами, менеджерами з продажу та складськими працівниками. Крім того, важливо впровадити надійні механізми резервного копіювання даних — як у хмарі, так і на локальних носіях — для запобігання втраті інформації у випадку технічних збоїв чи зловмисного втручання.

Ці апаратні та програмні компоненти обрано з метою формування стабільного, безпечного та зручного робочого середовища. Вони дозволяють працівникам магазину ефективно обслуговувати клієнтів, вести облік товарів, здійснювати замовлення та керувати базою даних у режимі реального часу. Такий підхід гарантує цілісність, конфіденційність та доступність критичних бізнес-даних, що є ключовими чинниками для безперервної та ефективної роботи електронного магазину.

4.2 Тестування системи

Запустивши систему, потрапляємо на головну сторінку, тут ми можемо подивитися наявні товари або перейти до авторизації/реєстрації (рис. 12).

Рис. 12 Головна сторінка

Після натискання на чоловічка на навігаційній панелі ви попадаєте на сторінку авторизації (Рис. 13).

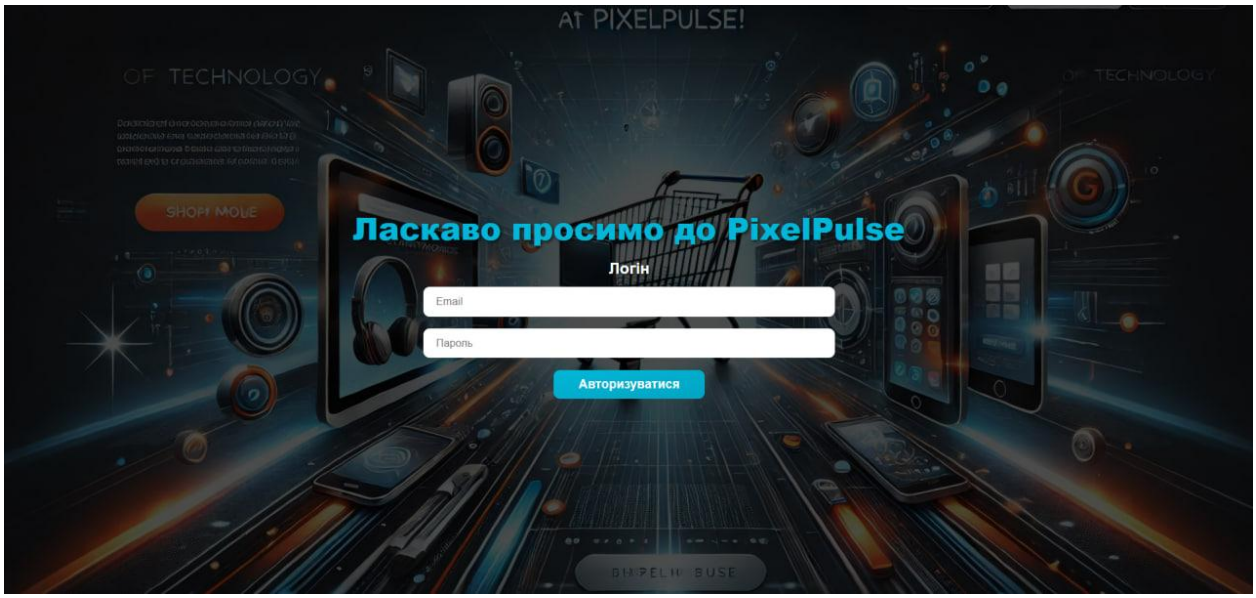

Рис. 13 Сторінка “Авторизація”

Якщо облікового запису не має, можете натиснути на чоловічка і вибрати реєстрація (Рис. 14).

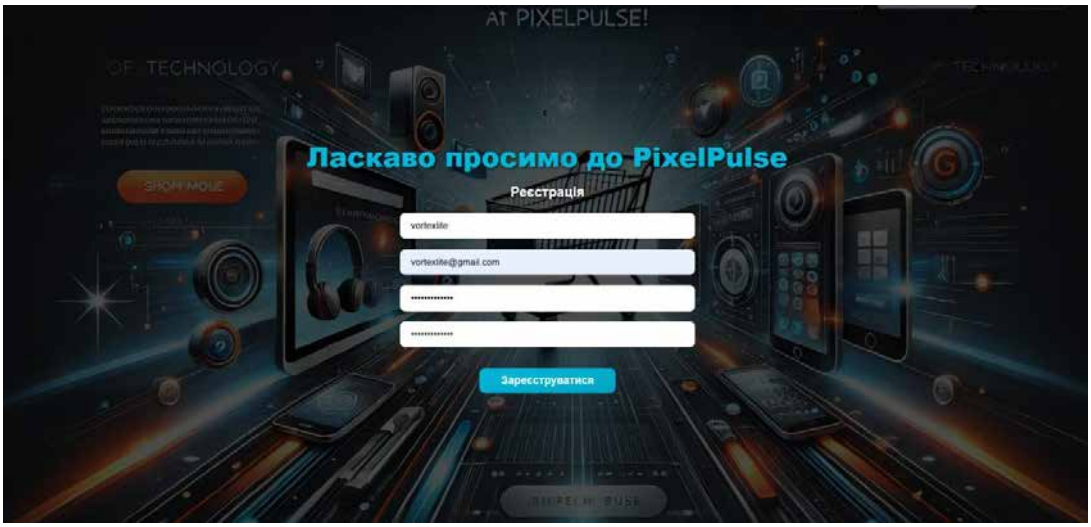

Рис. 14 Сторінка “Реєстрація”

Після цих дій, ми можемо також через чоловічка зайти особистий кабінет, де буде меню із основними сторінками для інтернет магазина електроніки. На ній в першу чергу перекидає на налаштування персональної інформації (Рис. 15).

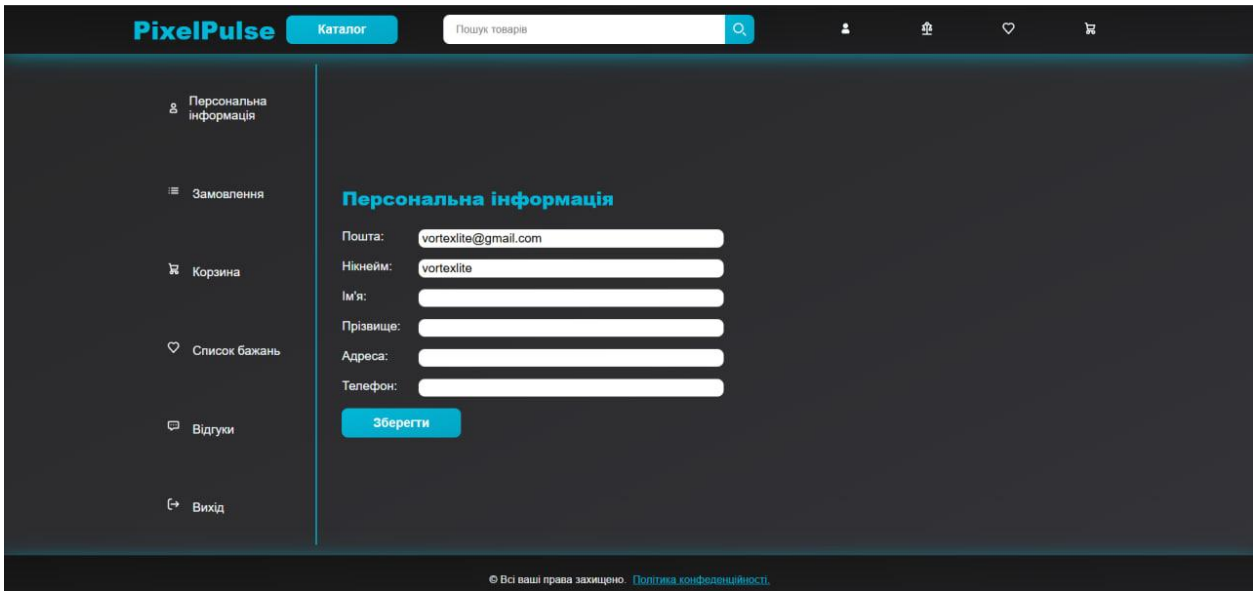

Рис. 15 Сторінка “Особистий кабінет”

Якщо ми залишимось на головній сторінці, то можемо подивитися товари в наявності, додати їх список улюбленого або корзину (Рис. 16).

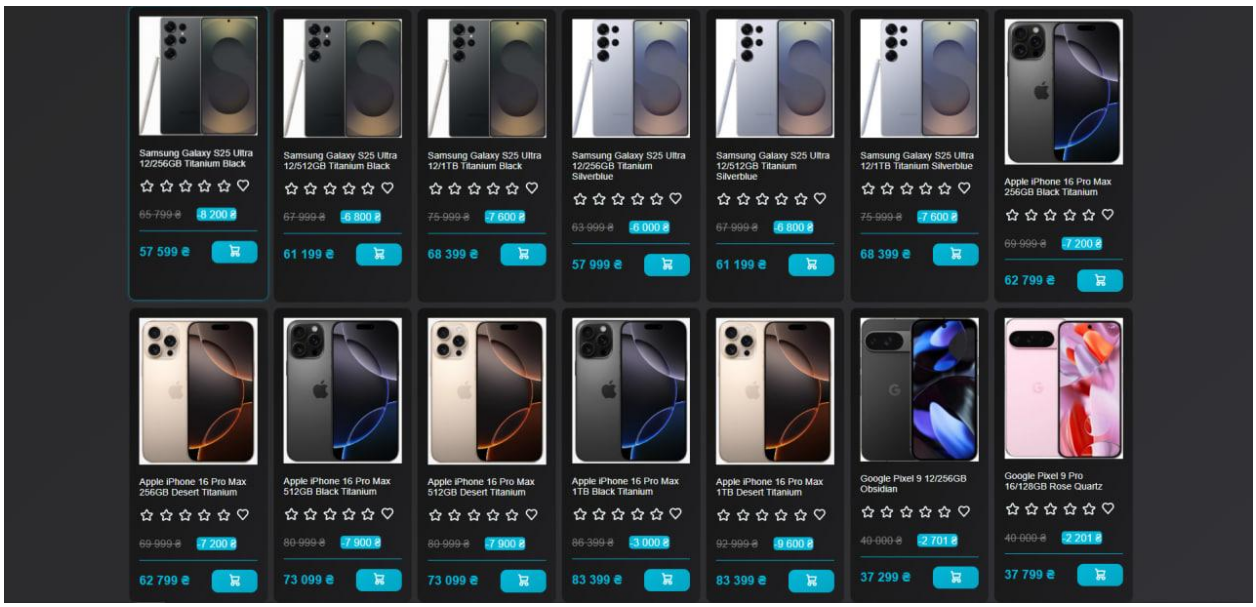

Рис. 16 Головна сторінка вибір товарів

Якщо натиснути на товар, відкриється його сторінка із детальним описом і слайдером фото. Також звідси можемо додати товар в корзину (Рис. 17).

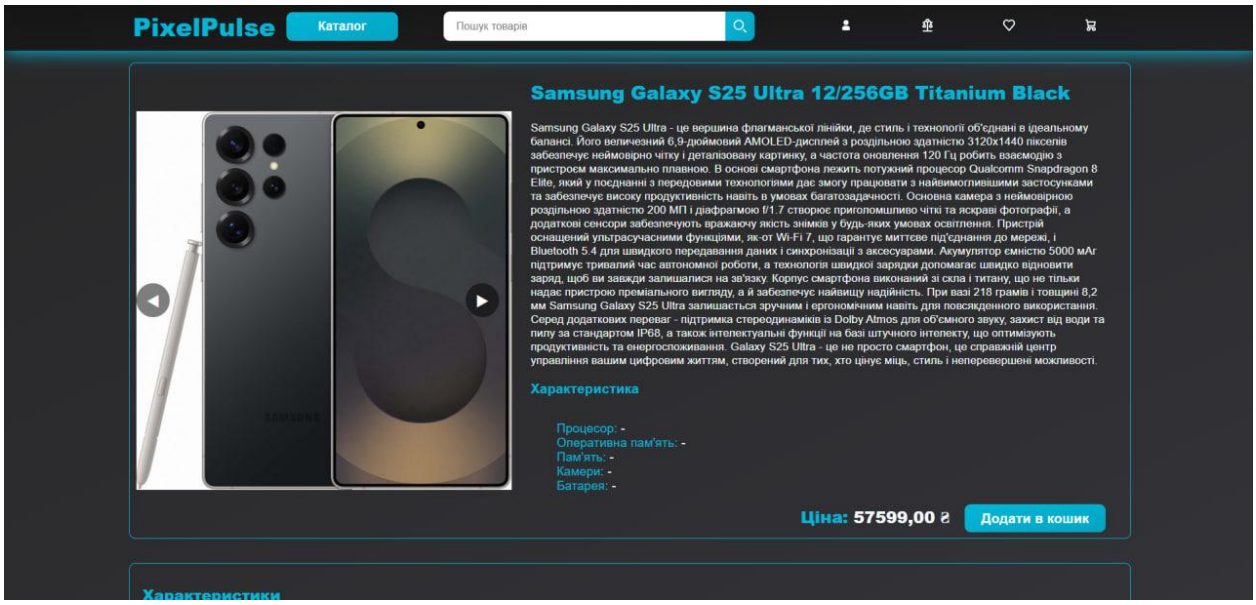

Рис. 17 Сторінка “Деталі товар”

Якщо натиснути на сердечко, товар попаде в список бажань, він потрібен для збереження товара щоб в майбутньому його замовити (Рис. 18).

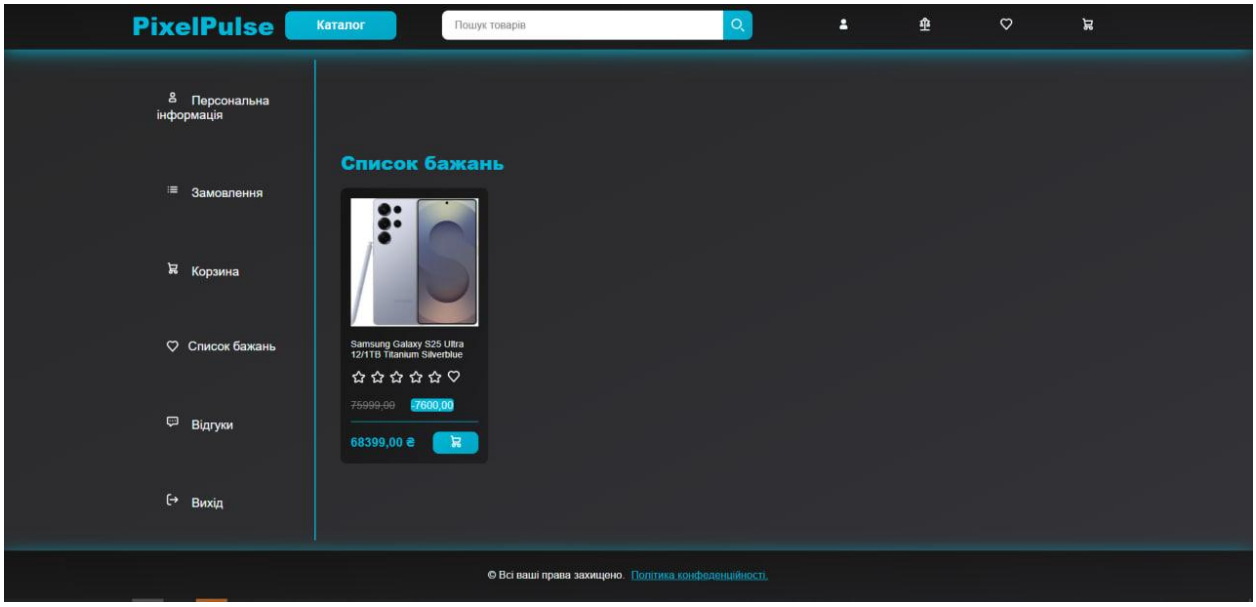

Рис. 18 Сторінка “Список бажань”

Якщо натиснути на товарі на корзину, товар попаде в неї, звідси можна уже робити замовлення. Щоб відкрити корзину, потрібно по ній натиснути на навігаційній панелі (Рис. 19).

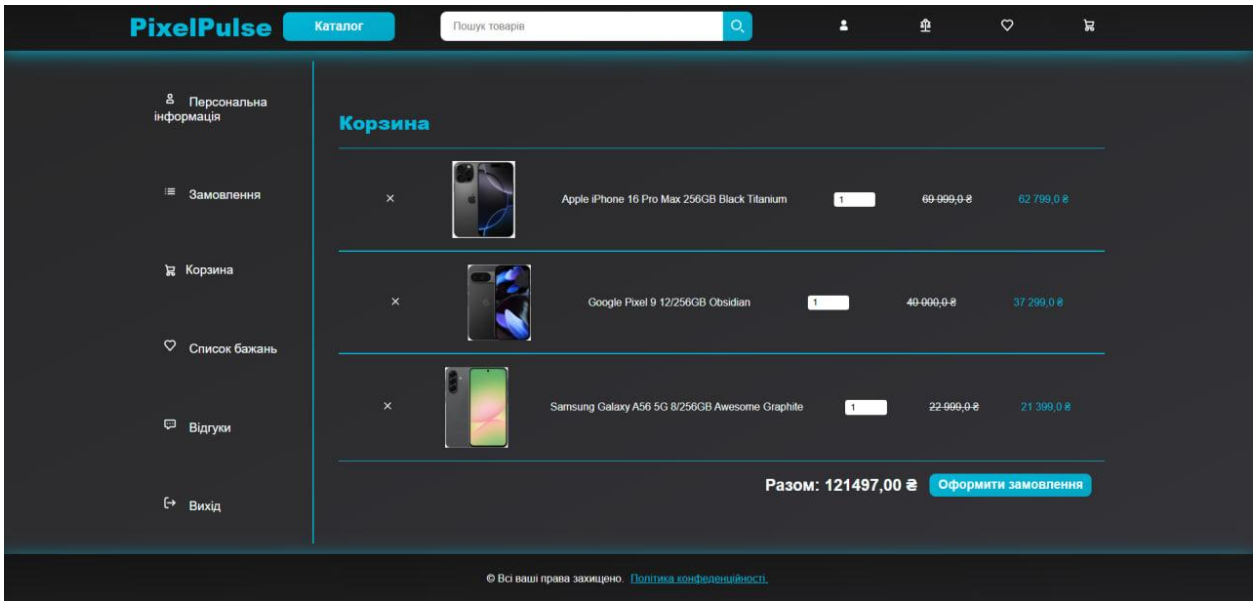

Рис. 19 Сторінка “Корзина”

При натисканні на кнопку оформити замовлення нас перекидає на сторінку оформлення, де ми можемо ввести дані отримувача, його адресу і відділеня новой пошти, а також вибір оплати.

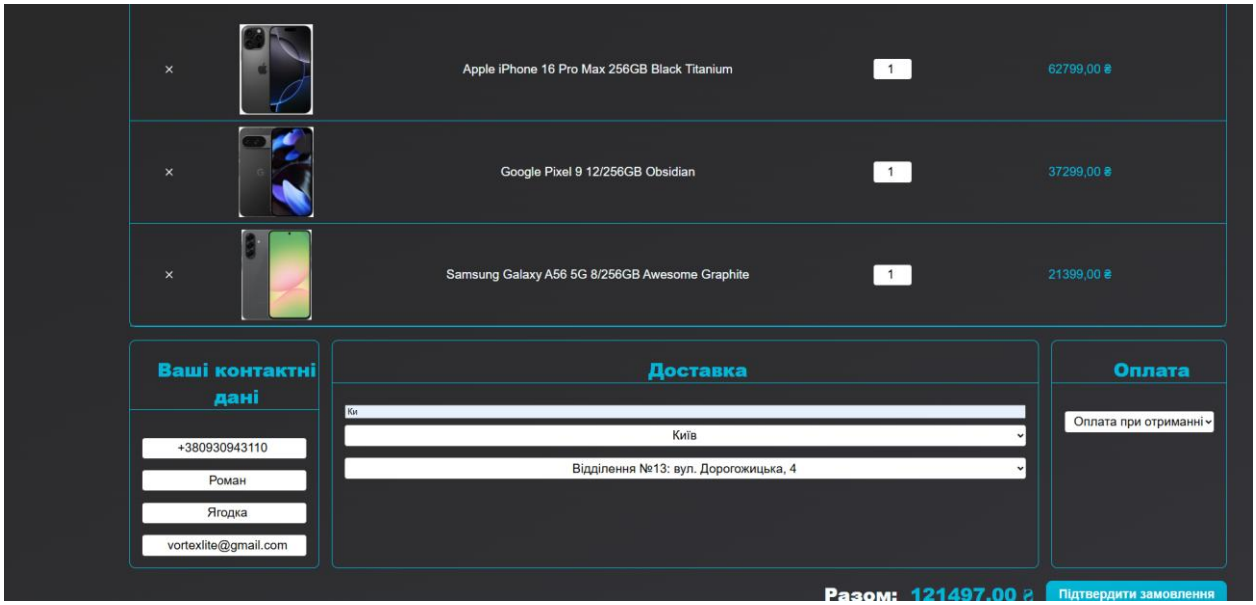

Рис. 20 Сторінка “Оформлення замовлення”

Після оформлення замовлення воно з’явиться на сторінці замовлення, буде показан номер замовлення, дату, суму, статус і під список товарів які входять в це замовлення.

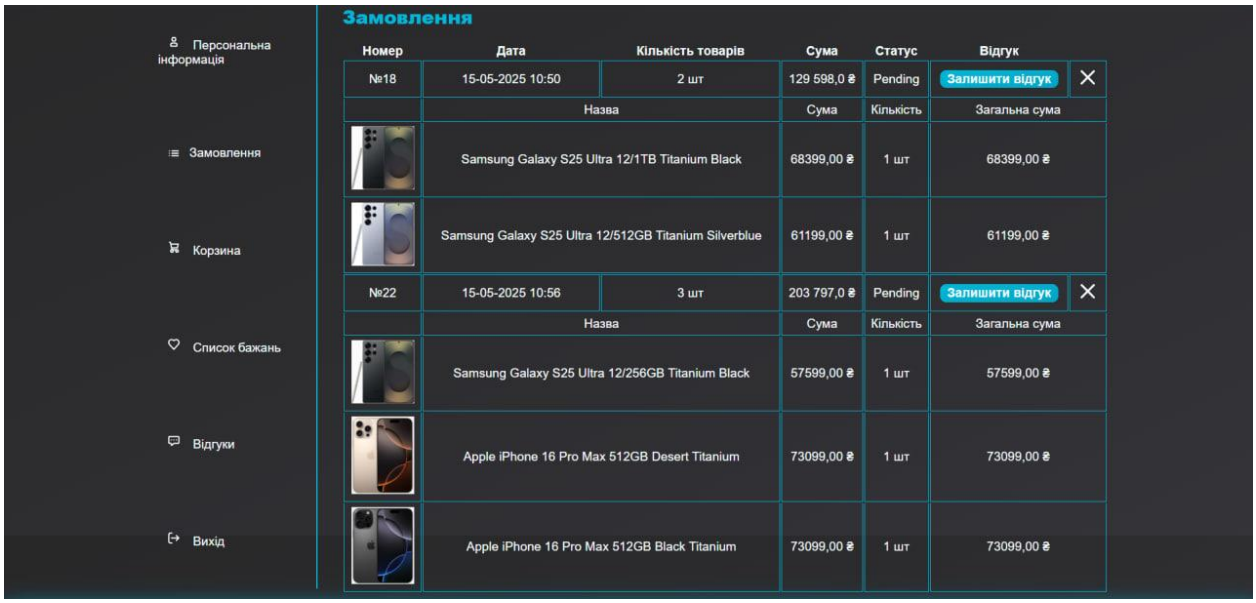

Рис. 21 Сторінка “Замовлення”

Також тут є простенька адмін панель, де можна керувати користувачами, товарами, замовленнями і звітами.

Рис. 22 Сторінка “Адмін панель”

Якщо вибрати управління користувачами, ми можемо редагувати кожним користувачем, вносити в нього зміни або видалити, також є варіант додати нового але це більше для тесту системи.

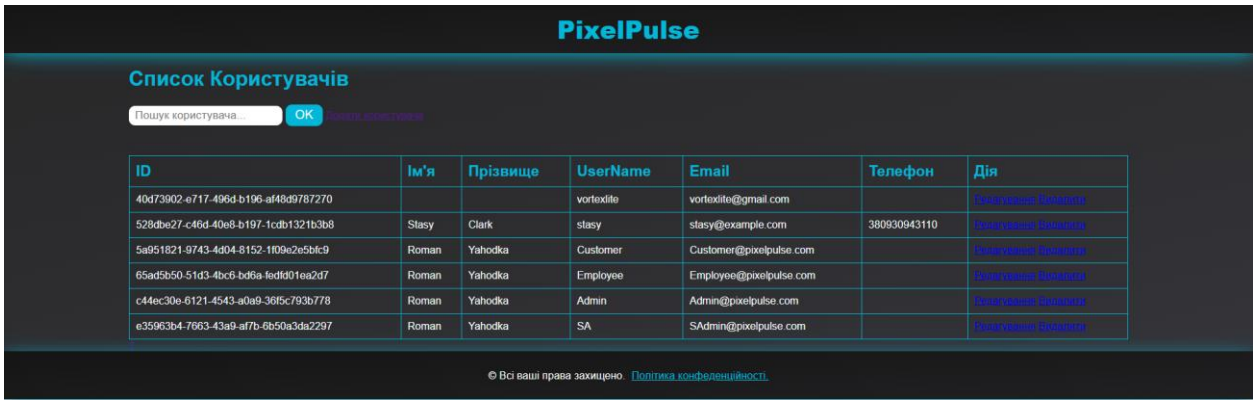

Рис. 23 Сторінка “Управління користувачами”

Якщо вибрати керування товарами, відкриється список товарів, де також я їх можна редагувати, додавати або видаляти. Також при редагування можна видалити фото товара, або поміняти фото яке буде головним.

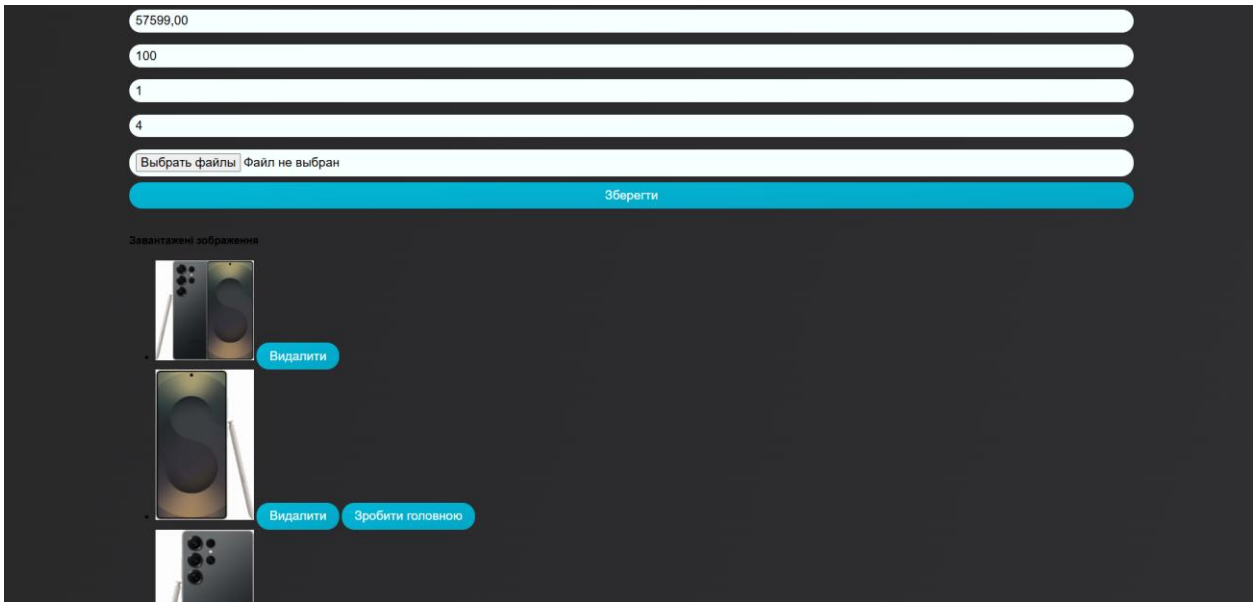

Рис. 24 Сторінка “Управління товарами”

Аналогічні дії присутні і у керування замовленнями, де адміністратор сайту їй налаштовує і змінює їхній статус.

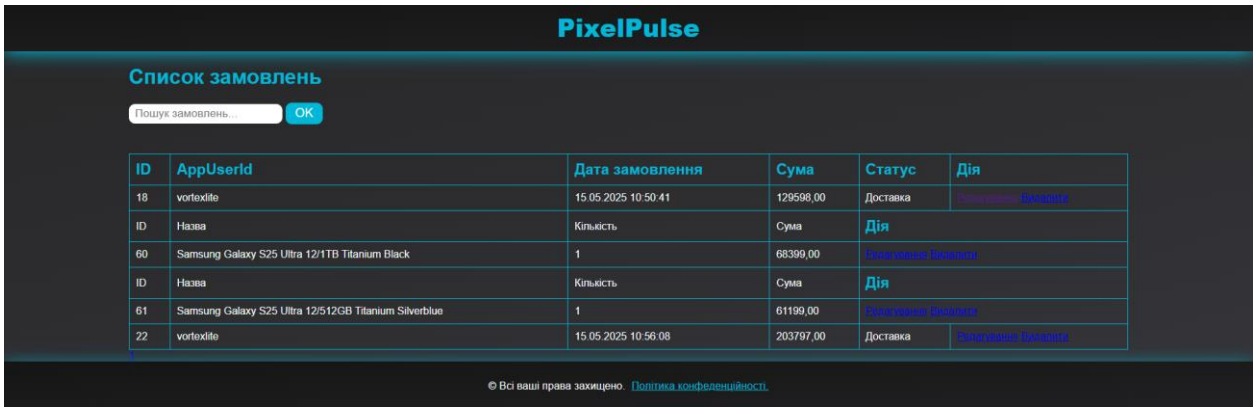

Рис. 25 Сторінка “Управління замовленнями”

Якщо вибрати управління звітами, ми попадемо на сторінку з ними, де буде вибір який звіт згенерувати.

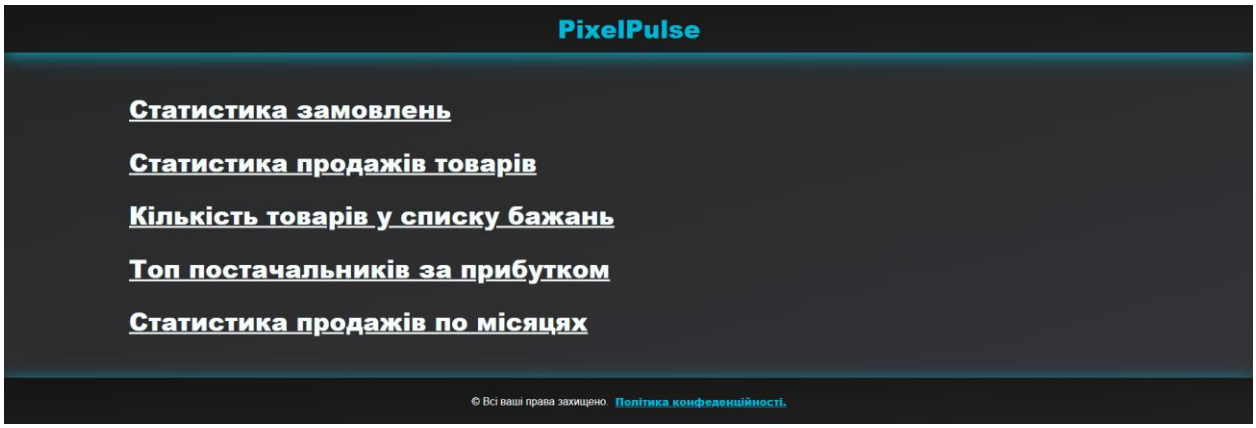

Рис. 26 Сторінка “Управління звітами”

Перший звіт це звіт по замовленням

Рис. 27 Сторінка “Статистика замовлень”

Другий звіт це звіт по продажам товарів

Рис. 28 Сторінка “Статистика продажів товарів”

Третій звіт це звіт по кількості товарів у списку бажань, для аналізу що більше подобається покупцям і можливість створення знижок або розіграшів по ним.

Рис. 29 Сторінка “Статистика кількості товарів у списку бажань”

Четвертий звіт це звіт по постачальникам за прибутком.

Рис. 30 Сторінка “Статистика постачальниках за прибутком”

П’ятий звіт це статистика продажів по місяцях.

Рис. 31 Сторінка “Статистика продажів по місяцях”

ВИСНОВКИ

Розробка веб-орієнтованої інформаційної системи для магазину електроніки вирішує актуальну потребу в цифровій трансформації торговельної діяльності, орієнтованої на споживача. Система покликана автоматизувати ключові бізнес-процеси, зокрема управління товарами, обробку замовлень, ведення бази клієнтів, керування постачальниками та аналітику продажів. Це дозволяє значно підвищити ефективність обслуговування клієнтів, зменшити кількість помилок при обробці замовлень і забезпечити зручну взаємодію з товарним асортиментом в онлайн-середовищі.

Запропоноване рішення, створене з використанням C#, ASP.NET Core [1] та сучасного підходу Clean Architecture [2], у поєднанні з базою даних Microsoft SQL Server [26], забезпечує стабільну й масштабовану роботу вебсистеми. Використання Entity Framework Core спрощує взаємодію з базою даних, а вбудована підтримка ASP.NET Core Identity гарантує безпечну автентифікацію та авторизацію користувачів. Інтуїтивний користувацький інтерфейс, реалізований з урахуванням сучасних вебтехнологій, забезпечує зручний доступ до функціоналу як для покупців, так і для адміністраторів магазину.

Проведений аналіз конкурентних рішень та сучасних вимог до електронної комерції засвідчив необхідність створення спеціалізованої системи, яка б об'єднала можливості онлайн-продажів, управління товарними залишками, системи лояльності та розширеної звітності. Розроблений вебдодаток відповідає цим вимогам, забезпечуючи централізоване зберігання інформації, інтерактивне керування товарами, підтримку онлайн-замовлень та інструменти для прийняття управлінських рішень.

Окрім виконання функціональних завдань, система сприяє модернізації торговельної інфраструктури вітчизняних підприємств шляхом переходу від

застарілих інструментів обліку до сучасного веб-інтерфейсу. Це сприяє підвищенню прозорості процесів, покращенню комунікації з клієнтами та оптимізації логістики. Зокрема, інтеграція зі сторонніми сервісами, такими як платіжні шлюзи або служби доставки, відкриває нові можливості для зростання бізнесу та формування клієнтської довіри.

У перспективі система пропонує можливості для подальшого розвитку – впровадження мобільної версії, інтеграції з системами CRM, впровадження елементів штучного інтелекту для рекомендацій товарів, а також використання модулів бізнес-аналітики для глибшого аналізу продажів і поведінки користувачів. Таким чином, розроблене рішення не тільки задовольняє поточні потреби магазину електроніки, але й формує основу для стратегічного розвитку підприємства в умовах цифрової економіки.

Підсумовуючи, система, створена в рамках цієї бакалаврської роботи, повністю відповідає поставленим цілям і демонструє високу ефективність у вирішенні завдань електронної комерції. Вона є вагомим кроком на шляху до повноцінної цифрової трансформації роздрібної торгівлі та може бути використана як основа для подальших розробок у сфері веб-орієнтованих бізнес-систем.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Microsoft. ASP.NET Core Documentation – [Електронний ресурс] – Режим доступу: https://learn.microsoft.com/aspnet/core
2. The Clean Architecture – [Електронний ресурс] – Режим доступу: https://medium.com/clean-code-channel/clean-architecture-the-solution-to-have-a-reusable-flexible-and-testable-code-ac7e296d1a75
3. Типи архітектури програмного забезпечення – [Електронний ресурс] – Режим доступу: https://medium.com/nuances-of-programming/4-типа-архитектуры-программного-обеспечения-917133174724
4. What is Unified Modeling Language (UML)? – [Електронний ресурс] – Режим доступу: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/
5. ПРОЄКТУВАННЯ ER-ДІАГРАМИ – [Електронний ресурс] – Режим доступу: https://nationalteam.worldskills.ru/skills/proektirovanie-er-diagrammy/
6. Діаграма варіантів використання (UseCase diagram) – [Електронний ресурс] – Режим доступу: https://flexberry.github.io/ru/fd_use-case-diagram.html
7. ПРОЄКТУВАННЯ USE CASE ДІАГРАМИ. ВИЗНАЧЕННЯ ФУНКЦІОНАЛЬНИХ МОЖЛИВОСТЕЙ СИСТЕМИ – [Електронний ресурс] – Режим доступу: https://nationalteam.worldskills.ru/skills/proektirovanie-use-case-diagrammy-opredelenie-funktsionalnykh-vozmozhnostey-sistemy/
8. What is Sequence Diagram? – [Електронний ресурс] – Режим доступу: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-sequence-diagram/
9. UML - Activity Diagrams – Tutorialspoint – [Електронний ресурс] – Режим доступу: https://www.tutorialspoint.com/uml/uml_activity_diagram.htm

10. Побудова діаграми класів – [Електронний ресурс] – Режим доступу: https://flexberry.github.io/ru/gpg_class-diagram.html
11. UML-діаграми класів – [Електронний ресурс] – Режим доступу: https://prog-cpp.ru/uml-classes/
12. Entity Relationship Diagram - Data Modeling – [Електронний ресурс] – Режим доступу: https://www.visualparadigm.com/VPGallery/datamodeling/EntityRelationshipDiagram.html
13. UML 2 Tutorial - Package Diagram - Sparx Systems – [Електронний ресурс] – Режим доступу: https://sparxsystems.com/resources/tutorials/uml2/package-diagram.html
14. Microsoft Docs. (2021). Layered architecture pattern – [Електронний ресурс] – Режим доступу: https://docs.microsoft.com/en-us/azure/architecture/patterns/layered
15. What is Component Diagram? – [Електронний ресурс] – Режим доступу: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-component-diagram/
16. Deployment Diagram in UML: Definition, Examples & Components – [Електронний ресурс] – Режим доступу: https://study.com/academy/lesson/deployment-diagram-in-uml-definition-examples-components.html
17. What is a data flow diagram? – [Електронний ресурс] – Режим доступу: https://www.lucidchart.com/pages/data-flow-diagram
18. OpenCart Documentation – [Електронний ресурс] – Режим доступу: https://docs.opencart.com
19. PrestaShop Developer Guide – [Електронний ресурс] – Режим доступу: https://devdocs.prestashop-project.org
20. Shopify Help Center – [Електронний ресурс] – Режим доступу: https://help.shopify.com

21. Соммервіль, І. (2016). Інженерія програмного забезпечення (10-е вид.). Pearson Education.
22. Прессман Р. С. та Максим Б. Р. (2015). Software Engineering: A Practitioner’s Approach (8-е видання). Освіта McGraw-Hill.
23. Пун, А., Лоу, К. (2017). «Приплив електронної комерції в індустрію гостинності: дослідження онлайн-туристичних агентств». Міжнародний журнал досліджень туризму, 19(6), 703–710.
24. Коннолі, Т., і Бегг, К. (2014). Системи баз даних: практичний підхід до проєктування, впровадження та управління (6-е видання). Pearson Education.
25. Фріман, Е., Робсон, Е. (2021). Head First Design Patterns (2-е видання). O'Reilly Media.
26. Корпорація Microsoft. (2023). Документація Microsoft SQL Server. [Електронний ресурс] – Режим доступу: https://docs.microsoft.com/sql
27. Ріхтер, Дж. (2012). CLR через C# (4-е видання). Microsoft Press.
28. Альбахарі, Дж., і Альбахарі, Б. (2021). C# 10 in a Nutshell (8-е видання). O'Reilly Media.
29. Буч, Г., Рамбо, Дж., і Якобсон, І. (2005). Посібник користувача з уніфікованої мови моделювання (2-е видання). Аддісон-Уеслі.
30. Якобсон І., Крістерсон М., Йонссон П. та Овергаард Г. (1992). Об'єктно-орієнтоване програмне забезпечення: підхід, орієнтований на використання. Аддісон-Уеслі.
31. Васильєв, А. Н. (2020). «Автоматизація кадрового діловодства в державних установах». Журнал інформаційних систем та державного управління, 7(2), 122–130.
32. Парсонс, Д. та Оджа, Д. (2016). Нові погляди на комп’ютерні концепції 2016. Cengage Learning.
33. Беннетт С., МакРобб С. та Фармер Р. (2010). Об'єктно-орієнтований системний аналіз і проєктування з використанням UML (4-е видання). Макгроу-Хілл

ДОДАТОК А

Фрагменти програмного коду. Код підключення до бази даних створення таблиць і їх конфігурування

```
var builder = WebApplication.CreateBuilder(args);

var connectionString =
builder.Configuration.GetConnectionString("DefaultConnection");
builder.Services.AddDbContext<AppIdentityContext>(options =>
    options.UseSqlServer(connectionString));

builder.Services.Configure<JWT>(builder.Configuration.GetSection("JWT"));
builder.Services.AddIdentity<AppUser, IdentityRole>(options =>
    {
        options.Password.RequiredLength = 8;
        options.Password.RequireNonAlphanumeric = false;
        options.Password.RequireLowercase = false;
        options.Password.RequireUppercase = false;
    })
    .AddEntityFrameworkStores<AppIdentityContext>()
    .AddDefaultTokenProviders();

using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;
using PixelPulse.Application.DTOs.ViewSQL;
using PixelPulse.Domain.Entities;
using PixelPulse.Infrastructure.Content.Configurations;

namespace PixelPulse.Infrastructure.Content.Data;

public class AppIdentityContext(DbContextOptions<AppIdentityContext> options) :
IdentityDbContext(options)
{
    public DbSet<AppUser> AppUsers { get; set; }
    public DbSet<Category> Categories { get; set; }
    public DbSet<Product> Products { get; set; }
    public DbSet<Address> Addresses { get; set; }
    public DbSet<Cart> Carts { get; set; }
    public DbSet<CartItem> CartItems { get; set; }
    public DbSet<Discount> Discounts { get; set; }
    public DbSet<Order> Orders { get; set; }
    public DbSet<OrderItem> OrderItems { get; set; }
    public DbSet<Payment> Payments { get; set; }
    public DbSet<ProductImage> ProductImages { get; set; }
    public DbSet<Review> Reviews { get; set; }
    public DbSet<Setting> Settings { get; set; }
    public DbSet<Shipping> Shippings { get; set; }
    public DbSet<Supplier> Suppliers { get; set; }
    public DbSet<Wishlist> Wishlists { get; set; }
    public DbSet<WishlistItem> WishlistItems { get; set; }
    public DbSet<ProductSpecification> ProductSpecifications { get; set; }


    public DbSet<OrderStatisticsView> OrderStatisticsView { get; set; }
    public DbSet<MonthlySalesSummaryView> MonthlySalesSummaryViews { get; set; }
    public DbSet<ProductSalesSummaryView> ProductSalesSummaryViews { get; set; }
    public DbSet<TopSuppliersByRevenueView> TopSuppliersByRevenueViews { get;
set; }
    public DbSet<UserWishlistStatsView> UserWishlistStatsViews { get; set; }
```

```
    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        base.OnModelCreating(modelBuilder);

modelBuilder.ApplyConfigurationsFromAssembly(typeof(AppIdentityContext).Assembly
);

        modelBuilder.ApplyConfiguration(new AddressConfiguration());
        modelBuilder.ApplyConfiguration(new CartConfiguration());
        modelBuilder.ApplyConfiguration(new CartItemConfiguration());
        modelBuilder.ApplyConfiguration(new CategoryConfiguration());
        modelBuilder.ApplyConfiguration(new DiscountConfiguration());
        modelBuilder.ApplyConfiguration(new OrderConfiguration());
        modelBuilder.ApplyConfiguration(new OrderItemConfiguration());
        modelBuilder.ApplyConfiguration(new PaymentConfiguration());
        modelBuilder.ApplyConfiguration(new ProductConfiguration());
        modelBuilder.ApplyConfiguration(new ProductImageConfiguration());
        modelBuilder.ApplyConfiguration(new ReviewConfiguration());
        modelBuilder.ApplyConfiguration(new SettingConfiguration());
        modelBuilder.ApplyConfiguration(new ShippingConfiguration());
        modelBuilder.ApplyConfiguration(new SupplierConfiguration());
        modelBuilder.ApplyConfiguration(new WishlistConfiguration());
        modelBuilder.ApplyConfiguration(new WishlistItemConfiguration());
        modelBuilder.ApplyConfiguration(new
ProductSpecificationConfiguration());

        modelBuilder
                .Entity<OrderStatisticsView>()
                .HasNoKey()
                .ToView("vw_OrderStatistics");

        modelBuilder.Entity<ProductSalesSummaryView>()
            .HasNoKey()
            .ToView("vw_ProductSalesSummary");

        modelBuilder.Entity<UserWishlistStatsView>()
            .HasNoKey()
            .ToView("vw_UserWishlistStats");

        modelBuilder.Entity<TopSuppliersByRevenueView>()
            .HasNoKey()
            .ToView("vw_TopSuppliersByRevenue");

        modelBuilder.Entity<MonthlySalesSummaryView>()
            .HasNoKey()
            .ToView("vw_MonthlySalesSummary");
    }
}
```

ДОДАТОК Б

Фрагменти програмного коду. Сервіс роботи з товарами

```
using Mapster;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Caching.Memory;
using Microsoft.Extensions.Logging;
using PixelPulse.Application.DTOs.Filter;
using PixelPulse.Application.DTOs.Product;
using PixelPulse.Application.Interfaces.Repositories;
using PixelPulse.Application.Interfaces.Service;
using PixelPulse.Domain.Entities;

namespace PixelPulse.Application.Service.Product;

public class ProductService : IProductService
{
    private readonly ILogger<ProductService> _logger;
    private readonly IProductRepository _productRepository;
    private readonly IProductImageRepository _productImageRepository;
    private readonly IFileService _fileService;
    private readonly IMemoryCache _cache;

    private const string _productCacheKey = "productItem_";

    public ProductService(ILogger<ProductService> logger, IProductRepository
productRepository, IMemoryCache cache, IProductImageRepository
productImageRepository, IFileService fileService)
    {
        _logger = logger;
        _productRepository = productRepository;
        _cache = cache;
        _productImageRepository = productImageRepository;
        _fileService = fileService;
    }

    public async Task<List<ProductMainDTO>?> GetProductsWithImageTake20Async()
    {
        if (!_cache.TryGetValue(_productCacheKey + "products", out
List<ProductMainDTO> cachedProducts))
        {
            var products = _productRepository.GetProductsWithImageTake20();
            cachedProducts = await products
                .ProjectToType<ProductMainDTO>()
                .ToListAsync();

            _cache.Set(_productCacheKey + "products", cachedProducts,
TimeSpan.FromMinutes(30));
            foreach (var product in cachedProducts)
                _cache.Set(_productCacheKey + $"{product.Id}", product,
TimeSpan.FromMinutes(30));
        }

        return cachedProducts;
    }

    public async Task<ProductCategoryDTO>? GetDetailsProduct(int productId)
    {
        if (!_cache.TryGetValue(_productCacheKey + "details" + $"{productId}",
out ProductCategoryDTO cachedProducts))
        {
```

```
            var product =
_productRepository.GetProductsWithImageAndCategoriesById(productId);
            cachedProducts = await product
                .ProjectToType<ProductCategoryDTO>()
                .FirstOrDefaultAsync();

            _cache.Set(_productCacheKey + "details" + $"{productId}",
cachedProducts, TimeSpan.FromMinutes(30));
        }

        return cachedProducts;
    }

    public async Task<List<ProductMainDTO>?> GetProductsTake100()
    {
        if (!_cache.TryGetValue(_productCacheKey + "take100", out
List<ProductMainDTO> cachedProducts))
        {
            var product = _productRepository.GetAll();
            cachedProducts = await product
                .ProjectToType<ProductMainDTO>()
                .ToListAsync();

            _cache.Set(_productCacheKey + "take100", cachedProducts,
TimeSpan.FromMinutes(30));
        }

        return cachedProducts;
    }

    public async Task<List<ProductMainDTO>?> GetFilteredProduct(FilterPhoneDTO
filter)
    {
        var productQuery = _productRepository.GetFilteredProduct(filter);
        return await productQuery
            .ProjectToType<ProductMainDTO>()
            .ToListAsync();
    }

    public async Task<bool> UpdateProductWithImage(ProductUpdateRequest request)
    {
        var product = await
_productRepository.GetById(Convert.ToInt32(request.Id)).FirstOrDefaultAsync();
        if (product == null)
            return false;

        product.Name = request.Name;
        product.Description = request.Description;
        product.OldPrice = request.OldPrice;
        product.Price = request.Price;
        product.Stock = request.Stock;
        product.CategoryId = request.CategoryId;
        product.SupplierId = request.SupplierId;

        if (request.ProductImages != null && request.ProductImages.Count > 0)
        {
            var imageList = new List<ProductImage>();
            foreach (var file in request.ProductImages)
            {
                var imagePath = await _fileService.SaveFileAsync(file,
"images");
                imageList.Add(new ProductImage
                {
```

```
                    ProductId = product.Id,
                    isMain = false,
                    ImagePath = imagePath
                });
            }

            product.ProductImages = imageList;
        }

        return await _productRepository.UpdateAsync(product);
    }

    public async Task<Domain.Entities.Product> GetProductById(int id)
    {
        return await _productRepository.GetById(id).FirstOrDefaultAsync();
    }

    public async Task<Domain.Entities.Product>? GetProducWithImageById(int
productId)
    {
        var product = await
_productRepository.GetById(productId).FirstOrDefaultAsync();
        var images = await
_productImageRepository.GetImagesByProductId(productId).ToListAsync();
        product.ProductImages = images;

        return product;
    }

    public async Task<bool> DeleteProduct(int productId)
    {
        var product = await
_productRepository.GetById(productId).FirstOrDefaultAsync();
        return await _productRepository.DeleteAsync(product);
    }

    public async Task<bool> MainProductImage(ProductImageIdDTO dto)
    {
        var image = await
_productImageRepository.GetImagesByProductId(dto.ProductId)
            .Where(pi => pi.isMain == true).FirstOrDefaultAsync();
        if (image != null)
        {
            image.isMain = false;
            await _productImageRepository.UpdateAsync(image);
        }

        image = await
_productImageRepository.GetById(dto.ImageId).FirstOrDefaultAsync();
        if (image != null)
        {
            image.isMain = true;
            await _productImageRepository.UpdateAsync(image);
        }

        return true;
    }

    public async Task<bool> DeleteProductImage(int imageId)
    {
        var productImage = await
_productImageRepository.GetById(imageId).FirstOrDefaultAsync();
        return await _productImageRepository.DeleteAsync(productImage);
```

```
    }

    public async Task<bool> CreateProductWithImage(ProductUpdateRequest request)
    {
        var product = new Domain.Entities.Product
        {
            Name = request.Name,
            Description = request.Description,
            OldPrice = request.OldPrice,
            Price = request.Price,
            Stock = request.Stock,
            CategoryId = request.CategoryId,
            SupplierId = request.SupplierId
        };

        if (request.ProductImages != null && request.ProductImages.Count > 0)
        {
            var imageList = new List<ProductImage>();
            foreach (var file in request.ProductImages)
            {
                var imagePath = await _fileService.SaveFileAsync(file,
"images");
                imageList.Add(new ProductImage
                {
                    ProductId = product.Id,
                    isMain = false,
                    ImagePath = imagePath
                });
            }

            product.ProductImages = imageList;
        }

        return await _productRepository.CreateAsync(product);
    }
}
```