Перший європейський університет
ННІ «Європейська школа бізнесу»
Кафедра інформаційних технологій

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри інформаційних технологій
ННІ «Європейська школа бізнесу», професор

О. В. Нестеренко

«___ » _____ 2026 р.

КВАЛІФІКАЦІЙНА БАКАЛАВРСЬКА РОБОТА
зі спеціальності 121 «Інженерія програмного забезпечення»
ТЕМА: **«Розробка програмного засобу для моделювання бізнес-процесу оформлення замовлення на основі об'єктно-орієнтованого підходу мовою Java»**

Виконавець: Деркач Дарія Валеріївна
(ПІБ)

Науковий керівник: Шевчук Б. В., кандидат педагогічних наук, доцент.
(ПІБ)

Консультанти з окремих розділів пояснювальної записки:

_____
(ПІБ)

_____
(ПІБ)

Нормоконтролер:

_____
(ПІБ)

КИЇВ – 2026

Перший європейський університет
ННІ «Європейська школа бізнесу»
Кафедра інформаційних технологій

Ступінь вищої освіти – перший (бакалаврський) рівень

Спеціальність: 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма: «Інженерія програмного забезпечення»

Освітньо-кваліфікаційний рівень: бакалавр

ЗАТВЕРДЖУЮ:

Завідувач кафедри інформаційних технологій
ННІ «Європейська школа бізнесу», професор

О. В. Нестеренко

«____» _____2026 р.

З А В Д А Н Н Я

НА ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ БАКАЛАВРСЬКОЇ РОБОТИ

Деркач Дарія Валеріївна

1. Тема:

«Розробка програмного засобу для моделювання бізнес-процесу оформлення замовлення на основі об’єктно-орієнтованого підходу мовою Java»,

затверджена наказом ректора від «24» грудня 2025 р. № 207-к.

2. Термін виконання роботи:

з «26» травня 2026 р. по «22» червня 2026 р

3. Дата подання роботи на випускну кафедру:

«12» червня 2026 р.

4. Вихідні дані роботи:

Методології об’єктно-орієнтованого програмування, мова програмування Java, принципи моделювання бізнес-процесів, UML-діаграми, архітектурні підходи до розробки програмного забезпечення, технології створення користувацького інтерфейсу JavaFX/Swing, методи збереження даних, засоби тестування програмного забезпечення, сучасні підходи до автоматизації процесу оформлення замовлення в інформаційних системах електронної комерції.

5. Зміст пояснювальної записки:

Вступ

- Розділ 1. Аналіз предметної області та постановка задачі
- Розділ 2. Проєктування та архітектура програмного засобу
- Розділ 3. Програмна реалізація та тестування системи
- Висновки
- Список використаних джерел
- Додатки

6. Перелік обов’язкового графічного матеріалу:

UML-діаграма прецедентів використання системи. UML-діаграма класів. UML-діаграма послідовності. Схема архітектури програмного засобу. Схема структури бази даних. Інтерфейс користувача програмного засобу. Скріншоти реалізованої системи. Результати тестування програмного засобу.

7. Календарний план - графік виконання КБР

№ з/п	Завдання	Термін виконання	Відмітка про виконання
1.	Складання і затвердження індивідуальних завдань на виконання кваліфікаційної бакалаврської роботи (КБР)	28.02.26	
2.	Підготовка вступу і розділу 1 КБР	17.03.26	
3.	Підготовка розділу 2 КБР	10.04.26	
4.	Підготовка розділу 3 КБР, висновків і переліку використаних джерел	10.05.26	
5.	Подання студентом завершальної КБР науковому керівнику для перевірки на плагіат та оформлення відгуку	09.06.26	
6.	Попередній розгляд КБР на комісії від кафедри	12-13.06.26	
7.	Доопрацювання роботи, прийняття кафедрою рішення про допуск роботи до захисту в ЕК, оформлення та зовнішнє рецензування	16-20.06.26	
8.	Захист кваліфікаційної бакалаврської роботи в ЕК і присвоєння випускникам кваліфікації	23-24.06.26	

Студент_____

(підпис, ПІБ)

Керівник_____

(підпис, ПІБ)

РЕФЕРАТ

Тема: «Розробка програмного засобу для моделювання бізнес-процесу оформлення замовлення на основі об’єктно-орієнтованого підходу мовою Java»

Кваліфікаційна бакалаврська робота зі спеціальності 121 «Інженерія програмного забезпечення», ОПП «Інженерія програмного забезпечення», Перший європейський університет, 2026 р.

Загальний обсяг роботи становить **85 сторінок**. Робота складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків; містить таблиці, UML-діаграми, лістинги, ілюстрації та схеми.

Ключові слова: Java, об’єктно-орієнтоване програмування, бізнес-процес, оформлення замовлення, UML, програмний засіб, моделювання, JavaFX, автоматизація, інформаційна система.

Актуальність теми роботи зумовлена необхідністю автоматизації процесів оформлення замовлення в сучасних інформаційних системах електронної комерції. Зростання кількості онлайн-сервісів та цифрових платформ потребує створення програмних засобів, здатних забезпечувати ефективне моделювання та обробку бізнес-процесів із використанням сучасних підходів до розробки програмного забезпечення. Використання об’єктно-орієнтованого підходу дозволяє створювати гнучкі, масштабовані та зрозумілі програмні системи, придатні для подальшого розширення й підтримки.

Метою роботи є розробка програмного засобу для моделювання бізнес-процесу оформлення замовлення із застосуванням об’єктно-орієнтованого підходу мовою Java.

Для досягнення поставленої мети у роботі вирішено такі завдання:

1. дослідити особливості бізнес-процесу оформлення замовлення та сучасні підходи до його автоматизації;

2. провести аналіз існуючих програмних рішень у сфері обробки замовлень;

3. обґрунтувати доцільність використання об’єктно-орієнтованого підходу при розробці системи;

4. спроєктувати архітектуру програмного засобу та виконати UML-моделювання системи;

5. реалізувати програмний засіб мовою Java;

6. реалізувати механізми обробки та збереження даних замовлення;

7. провести тестування працездатності програмного засобу та проаналізувати результати його роботи.

Об’єктом дослідження є процес автоматизації оформлення замовлення в інформаційних системах.

Предметом дослідження є методи, моделі та програмні засоби реалізації бізнес-процесу оформлення замовлення на основі об’єктно-орієнтованого підходу.

Практичне значення одержаних результатів полягає у створенні програмного засобу, який може бути використаний для моделювання процесу оформлення замовлення в навчальних цілях, демонстраційних проєктах та системах електронної комерції. Розроблений програмний засіб демонструє практичне застосування принципів об’єктно-орієнтованого програмування, UML-моделювання та сучасних підходів до побудови програмних систем мовою Java.

ABSTRACT

Topic: “Development of a software tool for modeling the business process of order processing based on the object-oriented approach using Java”

Qualification bachelor thesis in specialty 121 “Software Engineering”, Educational and Professional Program “Software Engineering”, International European University, 2026.

The total volume of the work is **85 pages**. The thesis consists of an introduction, three chapters, conclusions, a list of references and appendices; contains tables, UML diagrams, illustrations and schemes.

Keywords: Java, object-oriented programming, business process, order processing, UML, software tool, modeling, JavaFX, automation, information system.

Relevance of the research topic:

The relevance of the work is determined by the need to automate order processing in modern e-commerce information systems. The rapid growth of online services and digital platforms requires the development of software tools capable of efficient business process modeling and data processing using modern software engineering approaches. The use of object-oriented programming makes it possible to create flexible, scalable and maintainable software systems suitable for further development and support.

The purpose of the work is to develop a software tool for modeling the business process of order processing using an object-oriented approach in the Java programming language.

To achieve the stated goal, the following tasks were completed:

1. investigate the features of the business process of order processing and modern approaches to its automation;
2. analyze existing software solutions in the field of order management systems;
3. justify the feasibility of using the object-oriented approach during software development;

4. design the software architecture and perform UML modeling of the system;
5. implement the software tool using Java;
6. develop mechanisms for processing and storing order data;
7. test the developed software system and analyze the results of its operation.

The object of the research is the process of automation of order processing in information systems.

The subject of the research is methods, models and software tools for implementing the business process of order processing based on the object-oriented approach.

The practical significance of the obtained results lies in the development of a software tool that can be used for modeling the order processing workflow in educational purposes, demonstration projects and e-commerce systems. The developed software demonstrates the practical application of object-oriented programming principles, UML modeling and modern approaches to software system development using Java.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

Скорочення / позначення	Розшифровка
API	Application Programming Interface – інтерфейс програмування застосунків
CRUD	Create, Read, Update, Delete – базові операції роботи з даними
CSS	Cascading Style Sheets – каскадні таблиці стилів
ER-діаграма	Entity Relationship Diagram – діаграма зв’язків між сутностями
GUI	Graphical User Interface – графічний користувацький інтерфейс
IDE	Integrated Development Environment – інтегроване середовище розробки
JSON	JavaScript Object Notation – текстовий формат обміну даними
JVM	Java Virtual Machine – віртуальна машина Java
JDK	Java Development Kit – набір засобів для розробки програм мовою Java
JavaFX	Платформа для створення графічних інтерфейсів мовою Java
MVC	Model–View–Controller – архітектурний шаблон побудови програмного забезпечення
ООР / ООП	Object-Oriented Programming – об’єктно-орієнтоване програмування
SQL	Structured Query Language – мова структурованих запитів
UML	Unified Modeling Language – уніфікована мова моделювання
UI	User Interface – користувацький інтерфейс
UX	User Experience – досвід взаємодії користувача із системою
XML	eXtensible Markup Language – розширювана мова розмітки
Git	Система контролю версій програмного коду
Maven	Інструмент автоматизації збірки Java-проєктів

ЗМІСТ

ВСТУП

Сучасний розвиток інформаційних технологій та електронної комерції суттєво вплинув на організацію процесів взаємодії між користувачами та цифровими сервісами. Одним із ключових елементів функціонування більшості інформаційних систем у сфері електронної комерції є процес оформлення замовлення, який забезпечує взаємодію між клієнтом, системою обробки даних та механізмами збереження інформації про придбані товари чи послуги. В умовах стрімкого розвитку електронної комерції особливого значення набуває автоматизація процесів обробки та оформлення замовлень [24, 25]. Від ефективності реалізації цього процесу залежить зручність користування системою, швидкість обробки даних, коректність формування замовлень та можливість подальшого супроводу програмного забезпечення.

У сучасних програмних системах процес оформлення замовлення є не лише технічною функцією, а повноцінним бізнес-процесом, який включає декілька взаємопов’язаних етапів: формування кошика товарів, перевірку даних користувача, розрахунок вартості замовлення, зміну статусів обробки, збереження інформації та подальше управління створеним замовленням. Реалізація таких процесів потребує використання підходів, які дозволяють забезпечити структурованість системи, повторне використання компонентів, масштабованість та простоту підтримки програмного коду. Сучасні інформаційні системи дозволяють значно підвищити ефективність обробки замовлень та мінімізувати вплив людського фактора [8, 27].

Одним із найбільш поширених підходів до розробки програмного забезпечення є об’єктно-орієнтоване програмування [1, 2]. Використання об’єктно-орієнтованого підходу дозволяє моделювати реальні бізнес-процеси за допомогою взаємодії класів та об’єктів, що особливо важливо під час створення систем, пов’язаних із обробкою замовлень, управлінням товарами та взаємодією користувачів із програмним середовищем. Застосування принципів інкапсуляції, наслідування та поліморфізму сприяє створенню

гнучких програмних рішень, придатних до подальшого розширення та модернізації.

Для реалізації програмного засобу у роботі використовується мова програмування Java, яка є однією з найбільш популярних мов розробки програмного забезпечення завдяки своїй платформонезалежності, підтримці об’єктно-орієнтованої моделі програмування та широкому набору інструментів для створення прикладних систем. Використання Java дозволяє реалізувати стабільну архітектуру програмного засобу та забезпечити ефективну організацію бізнес-логіки системи.

Актуальність теми полягає у необхідності створення програмних засобів, здатних забезпечувати моделювання та автоматизацію процесів оформлення замовлення з використанням сучасних підходів до розробки програмного забезпечення. У зв’язку зі зростанням кількості електронних сервісів та інформаційних систем особливого значення набуває розробка програмних рішень, які поєднують зручність використання, структуровану архітектуру та можливість подальшого вдосконалення.

Метою кваліфікаційної роботи є розробка програмного засобу для моделювання бізнес-процесу оформлення замовлення на основі об’єктно-орієнтованого підходу мовою Java.

Для досягнення поставленої мети необхідно виконати такі завдання:

1) дослідити особливості бізнес-процесу оформлення замовлення;

2) провести аналіз існуючих програмних рішень у сфері автоматизації оформлення замовлень;

3) обґрунтувати використання об’єктно-орієнтованого підходу при розробці програмного засобу;

4) виконати проєктування архітектури системи та UML-моделювання;

5) реалізувати програмний засіб мовою Java;

6) реалізувати механізми обробки та збереження даних замовлення;

7) провести тестування програмного засобу та аналіз результатів його роботи.

Об’єктом дослідження є процес автоматизації оформлення замовлення в інформаційних системах.

Предметом дослідження є методи, моделі та програмні засоби реалізації бізнес-процесу оформлення замовлення на основі об’єктно-орієнтованого підходу.

У роботі використано такі методи дослідження: аналіз наукових джерел, методи об’єктно-орієнтованого проєктування, UML-моделювання, методи структурного аналізу, порівняльний аналіз програмних рішень та методи тестування програмного забезпечення.

Практичне значення одержаних результатів полягає у створенні програмного засобу, який може бути використаний для моделювання та автоматизації процесу оформлення замовлення в інформаційних системах, а також у навчальних і демонстраційних цілях під час вивчення принципів об’єктно-орієнтованого програмування та проєктування програмного забезпечення.

Структура кваліфікаційної роботи складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків.

РОЗДІЛ 1.АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1. Сутність та особливості бізнес-процесу оформлення замовлення

Під час проєктування архітектури сучасних веб-ресурсів та платформ електронної комерції процедура оформлення замовлення (checkout-процес) є одним із ключових елементів системи. Цей модуль відповідає за фінальну стадію взаємодії з користувачем – підтвердження наміру придбання товару або замовлення послуги. На практиці неефективна організація логіки обробки даних на етапі оформлення замовлення призводить до збоїв у роботі системи, навіть за умови швидкого функціонування інтерфейсу та зручного каталогу. Наслідком цього є відмова користувача від завершення операції та зниження ефективності функціонування інформаційної системи загалом. Тому коректна побудова цього процесу є пріоритетним завданням під час розробки програмного забезпечення.

Фактично бізнес-процес оформлення замовлення являє собою послідовність взаємопов’язаних дій [27, 28]. Система повинна обробити запит користувача, перевірити необхідні параметри, виконати розрахунок вартості та зберегти інформацію про замовлення у базі даних. Окрему роль у цьому процесі відіграє валідація даних, оскільки система повинна перевіряти правильність введеної інформації та запобігати появі помилкових записів.

Якщо розглядати типовий алгоритм оформлення замовлення, то зазвичай можна виділити декілька основних етапів.

Наповнення та ініціалізація кошика. Користувач додає товари до кошика, після чого система звертається до бази даних та перевіряє наявність необхідної кількості товарів.

Збір персональних даних. На цьому етапі користувач вводить контактну інформацію, номер телефону, адресу доставки або обирає поштового оператора.

Розрахунок вартості замовлення. Система виконує обчислення загальної суми покупки з урахуванням кількості товарів, можливих знижок, промокодів та вартості доставки.

Фіналізація замовлення. Програма генерує унікальний номер замовлення, присвоює йому початковий статус та створює запис для подальшої обробки.

Однією з головних особливостей автоматизації оформлення замовлень є необхідність постійної взаємодії між різними компонентами системи. Процес оформлення замовлення не існує окремо від інших модулів. Для коректної роботи система повинна взаємодіяти з модулем користувачів, підсистемою складського обліку, сервісами обробки платежів та механізмами надсилання повідомлень [24, 25]. У разі виникнення помилки система повинна забезпечувати коректне відновлення даних та запобігати появі неузгоджених записів.

Якщо розглядати цей процес із точки зору об’єктно-орієнтованого програмування, то він є хорошим прикладом взаємодії окремих об’єктів предметної області. У межах програмної реалізації кожен елемент системи може бути представлений окремим класом. Наприклад, клас Order (Замовлення) може містити інформацію про покупця, список товарів та дані про доставку. Такий підхід дозволяє структурувати програмний код та спрощує подальшу підтримку системи.

Важливо також враховувати масштабованість програмного засобу. Навіть якщо початкова реалізація системи є відносно простою, у майбутньому може виникнути необхідність додавання нових функцій: підтримки різних способів оплати, системи знижок, бонусних програм або інтеграції із зовнішніми сервісами. Саме тому архітектура програмного забезпечення повинна залишатися достатньо гнучкою для подальшого розширення функціональності.

Таким чином, бізнес-процес оформлення замовлення вимагає комплексної організації бізнес-логіки та гнучкого проєктування архітектури.

Застосування об’єктно-орієнтованого підходу та інструментарію мови Java дозволяє створити структурований програмний засіб із високим потенціалом до розширення.

1.2. Аналіз існуючих програмних рішень для автоматизації оформлення замовлень

Автоматизація процесу оформлення замовлення на сьогодні є обов'язковим елементом будь-якої комерційної інформаційної системи. Конкуренція між цифровими платформами стимулює розробників до постійного вдосконалення checkout-процесу за трьома ключовими напрямками: швидкість обробки запитів, стабільність роботи під навантаженням та зручність взаємодії з кінцевим користувачем. З метою визначення підходів, які доцільно застосувати у власній розробці, було проведено огляд існуючих рішень різного масштабу.

Великі комерційні платформи. Серед вітчизняних прикладів варто виділити платформу Rozetka – один із найбільших українських маркетплейсів. Система оформлення замовлення на цьому ресурсі реалізована як багаторівневий конвеєр: авторизація користувача, формування кошика, автоматичний розрахунок вартості доставки залежно від обраного перевізника та відстеження статусів замовлення в особистому кабінеті. Перевагою такого рішення є стабільна робота з великою кількістю одночасних запитів та розвинена інтеграція з платіжними сервісами. Водночас складність подібної архітектури зумовлює суттєві витрати на підтримку та потребує значної команди розробників для розвитку системи.

Показовим прикладом на міжнародному рівні є платформа Amazon, де процес оформлення замовлення зведений до мінімальної кількості кроків. Система зберігає адреси доставки, платіжні дані та параметри попередніх замовлень, завдяки чому повторна покупка може бути завершена буквально за один клік. Крім того, Amazon інтегрує рекомендаційні алгоритми безпосередньо в процес checkout, що дозволяє підвищіти економічні

ефективності платформи. З архітектурної точки зору подібні рішення базуються на мікросервісному підході, де кожна функція – оплата, доставка, знижки – реалізована як окремий незалежний сервіс.

Платформи для малого та середнього бізнесу. У сегменті МСБ широко застосовуються готові рішення у сфері електронної комерції (e-commerce): Shopify, WooCommerce та OpenCart. Їхньою головною перевагою є швидке розгортання інтернет-магазину без необхідності розробки з нуля – система вже містить готові модулі кошика, оплати та доставки. Проте саме тут і проявляється основне обмеження таких платформ: внутрішня бізнес-логіка є закритою або жорстко фіксованою, що унеможливлює її глибоке налаштування під специфічні вимоги конкретного проєкту. Будь-яке відхилення від стандартного сценарію потребує написання додаткових плагінів або обходу обмежень платформи.

CRM та ERP-системи. На рівні середніх і великих підприємств для управління замовленнями використовуються комплексні системи класу CRM та ERP – зокрема SAP, Salesforce або Microsoft Dynamics. Вони охоплюють не лише безпосередньо оформлення замовлення, а й складський облік, фінансові розрахунки, аналітику продажів та управління взаємовідносинами з клієнтами. Однак впровадження таких систем є тривалим і дорогим процесом, який передбачає серйозну адаптацію під бізнес-процеси конкретного підприємства.

Порівняльний аналіз та висновки щодо розробки. Узагальнюючи розглянуті рішення, можна виділити спільну проблему: готові платформи або є надмірно складними для навчальних та дослідницьких цілей, або навпаки – надто обмеженими для демонстрації повноцінної бізнес-логіки. Жодне з наявних рішень не дозволяє в повній мірі дослідити внутрішні механізми побудови процесу оформлення замовлення з позицій об'єктно-орієнтованого програмування [21, 24].

Саме це обґрунтовує доцільність розробки власного програмного засобу в межах даної кваліфікаційної роботи. Авторська реалізація дає змогу не лише відтворити ключові елементи бізнес-логіки, але й продемонструвати

застосування принципів ООП, UML-моделювання та сучасних підходів до проєктування архітектури програмного забезпечення без обмежень, властивих готовим комерційним платформам [25].

1.3. Обґрунтування використання об’єктно-орієнтованого підходу при розробці програмного засобу

Вибір парадигми програмування є одним із ключових архітектурних рішень на етапі проєктування будь-якої інформаційної системи. Від нього залежить не лише структура коду, а й складність його супроводу, можливості розширення та загальна якість програмного продукту. У даній роботі як основний підхід до розробки обрано об'єктно-орієнтоване програмування (ООП) – і це рішення має конкретне технічне обґрунтування.

Відповідність предметній області. Бізнес-процес оформлення замовлення природно описується через сукупність взаємодіючих сутностей: покупець, товар, кошик, замовлення, спосіб доставки, платіжна операція. Кожна з цих сутностей має власний стан і поведінку, що є прямою проєкцією на поняття об'єкта в ООП. На відміну від процедурного підходу, де логіка системи зосереджена у функціях, що оперують глобальними даними, об'єктно-орієнтована модель дозволяє відобразити структуру предметної області безпосередньо у структурі програмного коду. Зокрема, клас Order інкапсулює стан замовлення та операції над ним, клас Product – характеристики товарної позиції, клас Customer – дані покупця, а клас DeliveryService – логіку розрахунку та вибору способу доставки.

Інкапсуляція як засіб захисту бізнес-логіки. Принцип інкапсуляції дозволяє приховати внутрішню реалізацію компонентів і надати зовнішньому коду доступ лише до визначеного публічного інтерфейсу. Практично це означає, що логіка розрахунку вартості замовлення, застосування знижок або зміни статусу є внутрішньою справою відповідного класу і не може бути порушена зовні. Це суттєво знижує ймовірність появи помилок при модифікації системи та полегшує локалізацію дефектів під час тестування.

Наслідування та поліморфізм для розширюваності системи. У реальних системах електронної комерції існує безліч варіантів схожих сутностей: різні типи користувачів (зареєстрований, гостьовий, корпоративний), різні способи доставки (кур'єрська, поштова, самовивіз), різні платіжні механізми. ООП вирішує цю проблему через наслідування та поліморфізм: базовий клас DeliveryMethod визначає загальний інтерфейс, а конкретні класи CourierDelivery, PostDelivery та SelfPickup реалізують його по-своєму. Такий підхід дозволяє додавати нові варіанти поведінки без зміни вже написаного і протестованого коду – що є прямою реалізацією принципу відкритості/закритості (Open/Closed Principle) з набору SOLID [2, 11].

Повторне використання коду та підтримуваність. У системі оформлення замовлення ряд операцій – валідація даних, зміна статусів, обчислення знижок – використовується в різних частинах системи. ООП дозволяє виділити такі операції в окремі класи або утилітарні методи і використовувати їх повторно без дублювання логіки. Це безпосередньо впливає на підтримуваність коду: при необхідності зміни алгоритму знижок достатньо модифікувати один клас, а не шукати відповідний код у кількох місцях програми.

Обґрунтування вибору Java як мови реалізації. Java є класичною об'єктно-орієнтованою мовою програмування, в якій усі конструкції – від простих утилітарних класів до складних ієрархій наслідування – підпорядковані принципам ООП. Мова забезпечує сувору типізацію, що зменшує кількість помилок на етапі компіляції, підтримує механізми інтерфейсів та абстрактних класів для гнучкого проєктування, а також надає розвинену стандартну бібліотеку для роботи з колекціями, введенням/виведенням даних та обробкою виключень. Платформонезалежність Java через JVM дозволяє розгортати розроблений програмний засіб без прив'язки до конкретної операційної системи. Додатково, Java має широку екосистему інструментів розробки – Maven,

IntelliJ IDEA, JUnit – що забезпечує зручність як написання коду, так і його тестування [1, 6].

Таким чином, вибір об'єктно-орієнтованого підходу та мови Java для розробки програмного засобу моделювання бізнес-процесу оформлення замовлення є технічно обґрунтованим рішенням. Цей підхід забезпечує відповідність архітектури структурі предметної області, підтримуваність і розширюваність системи, а також надає можливість продемонструвати практичне застосування ключових принципів ООП у контексті реального бізнес-сценарію.

1.4. Постановка задачі та формування вимог до програмного засобу

На підставі проведеного аналізу предметної області та огляду існуючих програмних рішень сформульовано основну задачу кваліфікаційної роботи: розробити програмний засіб мовою Java, який моделює бізнес-процес оформлення замовлення із застосуванням об'єктно-орієнтованого підходу. На відміну від готових комерційних платформ, розроблюваний засіб орієнтований не на промислове розгортання, а на демонстрацію повноцінної внутрішньої логіки процесу – від формування кошика до фіналізації замовлення – з акцентом на якість архітектурних рішень.

Мета та межі розробки розробки. Програмний засіб має відтворити ключові етапи бізнес-процесу оформлення замовлення в межах єдиної керованої системи. Архітектура повинна безпосередньо відображати структуру предметної області: кожна значуща сутність – користувач, товар, кошик, замовлення, доставка – реалізується окремим класом із чітко визначеною відповідальністю. Взаємодія між класами організовується через інтерфейси та методи, що відповідає принципам слабкого зв'язування компонентів.

Функціональні вимоги. Програмний засіб повинен забезпечувати повний цикл операцій із замовленням. В частині управління кошиком система реалізує додавання та видалення товарних позицій із автоматичним

перерахунком загальної вартості при кожній зміні. Обробка даних користувача передбачає введення контактної інформації та адреси доставки з обов'язковою валідацією коректності значень – формату телефонного номера та заповненості обов'язкових полів. Формування замовлення здійснюється на основі даних кошика та користувача: система генерує унікальний ідентифікатор і присвоює замовленню початковий статус. Розрахунок підсумкової вартості виконується з урахуванням кількості товарів, можливих знижок та тарифу обраного способу доставки. Окремо реалізується управління життєвим циклом замовлення через послідовну зміну статусів: CREATED → CONFIRMED → PROCESSING → SHIPPED → DELIVERED. Результати роботи системи відображаються через користувацький інтерфейс, який також інформує користувача про помилки та результати виконаних операцій.

Нефункціональні вимоги. Структурованість коду забезпечується дотриманням принципів SOLID та чітким розподілом відповідальності між класами без дублювання логіки. Розширюваність архітектури означає, що додавання нових типів доставки, способів оплати або категорій товарів не вимагатиме переписування наявного коду. Стійкість до некоректних даних реалізується через обробку виключних ситуацій – порожній кошик, відсутній товар, некоректне введення – без аварійного завершення роботи програми. Читабельність коду підтримується документуючими коментарями у форматі Javadoc для ключових класів та методів.

Технічні рішення. Для реалізації обрано мову Java версії не нижче 17, що забезпечує підтримку сучасних мовних конструкцій: лямбда-виразів, Stream API та записів. Середовищем розробки є IntelliJ IDEA, збірка проєкту організована через Maven. Графічний інтерфейс реалізовано засобами JavaFX, що надає більш сучасний інструментарій порівняно з Swing. Збереження даних здійснюється з використанням реляційної бази даних SQLite.

У результаті виконання сформульованих задач має бути створений програмний засіб, що демонструє повний цикл моделювання бізнес-процесу

оформлення замовлення, реалізований із дотриманням принципів ООП та придатний до подальшого розширення функціональності.

Висновки до розділу 1

У першому розділі кваліфікаційної роботи проведено комплексний аналіз предметної області, який охопив дослідження сутності бізнес-процесу оформлення замовлення, огляд існуючих програмних рішень та обґрунтування обраного підходу до розробки.

Встановлено, що процес оформлення замовлення є технічно складним багатоетапним об'єктом автоматизації, який включає формування кошика, валідацію персональних даних користувача, розрахунок вартості з урахуванням знижок і доставки, генерацію замовлення та управління його життєвим циклом через зміну статусів. Визначено, що коректна реалізація цього процесу вимагає не лише правильної бізнес-логіки, а й забезпечення цілісності даних та стабільної взаємодії між усіма компонентами системи.

Аналіз наявних програмних рішень – від масштабних комерційних маркетплейсів до спеціалізованих платформ для малого бізнесу та корпоративних CRM/ERP-систем – показав, що готові екосистеми мають суттєві обмеження для дослідницьких цілей. Вони або є надмірно складними для аналізу внутрішньої логіки, або мають закриту архітектуру, яка унеможливлює гнучке налаштування процесів на рівні програмного коду. Це підтвердило доцільність розробки власного програмного засобу, орієнтованого на демонстрацію архітектурних рішень.

Обґрунтовано вибір об'єктно-орієнтованого підходу як основної парадигми розробки. Показано, що ООП забезпечує природне відображення сутностей предметної області у структуру програмного коду, а застосування принципів SOLID – зокрема принципу єдиної відповідальності та принципу відкритості/закритості – гарантує розширюваність системи без модифікації вже реалізованих компонентів. Мова програмування Java обрана як платформа

реалізації з огляду на повноцінну підтримку ООП, розвинену екосистему інструментів та платформонезалежність.

Сформульовано функціональні та нефункціональні вимоги до програмного засобу, визначено технічний стек розробки та окреслено межі розробки системи. Отримані результати формують повноцінну основу для переходу до етапу проєктування архітектури програмного засобу, що є предметом другого розділу кваліфікаційної роботи.

РОЗДІЛ 2.ПРОЄКТУВАННЯ ТА АРХІТЕКТУРА ПРОГРАМНОГО ЗАСОБУ

2.1. Обґрунтування вибору технологій та середовища розробки

Вибір технологічного стеку є визначальним етапом проєктування інформаційної системи, від якого залежить ефективність її подальшого супроводу, масштабованість та якість програмного продукту. Обґрунтоване визначення інструментів розробки дозволяє реалізувати архітектурні рішення без необхідності створення додаткових програмних обмежень. Тому вибір технологій у даній роботі здійснювався з урахуванням специфіки предметної області та технічних вимог до системи.

Мова програмування Java. Обрання мови програмування Java зумовлене тим, що її об'єктна модель повноцінно відповідає структурі досліджуваної предметної області. Система оформлення замовлення складається з чітко відокремлених сутностей: покупець, товар, кошик, замовлення, доставка. Кожна з них має власну поведінку і стан – і Java дозволяє відобразити цю структуру напряму, через класи, інтерфейси та ієрархії наслідування, без жодних архітектурних компромісів. Ще один аргумент – сувора статична типізація: вона виловлює цілий клас помилок ще на етапі компіляції, що при розгалуженій логіці обробки даних є дуже відчутною перевагою. А платформонезалежність через JVM означає, що готовий застосунок запускається на будь-якій операційній системі без перекомпіляції – це спрощує і тестування, і передачу проєкту [2, 13].

Середовище розробки IntelliJ IDEA. Серед доступних IDE для Java – Eclipse, NetBeans, IntelliJ IDEA – остання була обрана як основне середовище.

Доцільність використання IntelliJ IDEA зумовлена наявністю потужних засобів статичного аналізу вихідного коду. Вона не просто підсвічує синтаксичні помилки – вона виявляє потенційні логічні проблеми, пропонує варіанти рефакторингу і попереджає про порушення патернів ООП ще до запуску програми. Вбудований дебагер із підтримкою точок зупинки,

покрокового виконання та інспекції стану об'єктів суттєво скорочує час на пошук помилок у логіці обробки замовлень [19, 36].

Порівняльна характеристика найбільш поширених середовищ розробки Java наведена у таблиці 2.1.

Таблиця 2.1 - Порівняння середовищ розробки Java

Середовище	**Переваги**	**Недоліки**
IntelliJ IDEA	Потужний аналіз коду, зручний інтерфейс, підтримка Maven та Git	Частина функцій доступна лише у Ultimate
Eclipse	Безкоштовне, велика кількість плагінів	Складніший інтерфейс
NetBeans	Простота використання	Менша популярність та повільніший розвиток

Як видно з таблиці, IntelliJ IDEA забезпечує найбільш зручні засоби аналізу коду та підтримки сучасної архітектури проєкту.

JavaFX для побудови інтерфейсу. Графічний інтерфейс реалізовано засобами JavaFX. Порівняно зі Swing, який існує з 1997 року і тягне за собою чималий архітектурний багаж, JavaFX будувалась із нуля з урахуванням сучасних підходів до проєктування UI. Ключова відмінність – чітке розділення представлення і логіки: розмітка інтерфейсу описується у файлах FXML, стилізація – через CSS, а обробка подій залишається в Java-коді. Завдяки цьому зовнішній вигляд форм замовлення можна змінювати, не торкаючись бізнес-логіки, – що є прямим втіленням принципу розділення відповідальностей на рівні UI [14, 19].

Збереження даних. Питання вибору механізму збереження даних виявилось нетривіальним. Збереження даних здійснюється з використанням реляційної бази даних SQLite, що не потребує розгортання окремого серверного середовища та є зручним рішенням для навчального програмного засобу. Такий підхід дозволяє зберігати інформацію про товари, користувачів, замовлення та позиції замовлення у структурованому вигляді, а також забезпечує коректну роботу з пов’язаними записами.

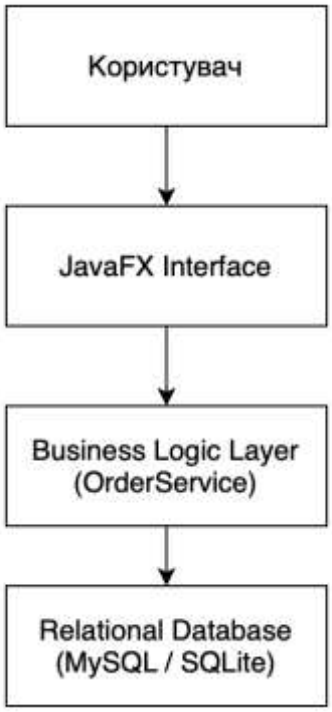


Рисунок 2.1 – Схема взаємодії програмного засобу з базою даних

Як показано на рисунку 2.1, взаємодія користувача із системою здійснюється через графічний інтерфейс JavaFX, після чого запити передаються до шару бізнес-логіки та модуля роботи з даними.

Maven та Git. Збірка проєкту організована через Maven: усі залежності декларативно описані у файлі pom.xml, що дозволяє відтворити ідентичне середовище розробки на будь-якій машині без ручного завантаження бібліотек. Контроль версій ведеться через Git – кожна значуща зміна в архітектурі або логіці системи фіксується окремим комітом, що дає змогу відстежити еволюцію проєкту і за потреби повернутись до попереднього стану [17, 39].

Основні технології, використані під час розробки програмного засобу, наведено у таблиці 2.2.

Таблиця 2.2 - Використані технології та їх призначення

Технологія	Призначення
Java	Реалізація бізнес-логіки системи
JavaFX	Побудова графічного інтерфейсу
SQLite	Збереження інформації про замовлення
Maven	Керування залежностями та збіркою проєкту
Git	Контроль версій програмного коду
IntelliJ IDEA	Середовище розробки

Використання зазначених технологій дозволило організувати структурований процес розробки та забезпечити стабільну роботу програмного засобу.

Сформований технологічний стек – Java 17+, IntelliJ IDEA, JavaFX, реляційна БД, Maven, Git – забезпечує всі необхідні умови для реалізації поставлених задач. Кожен інструмент підбирався під конкретну технічну потребу, а не за принципом популярності – саме це і визначає обґрунтованість зробленого вибору.

2.2. Проєктування архітектури програмного засобу

Архітектура – це те, що вирішує наперед, чи буде система зручною у підтримці через рік або складеться у безладний клубок залежностей після перших же змін. Якщо на початку не продумати, як компоненти взаємодіють між собою і де проходять межі відповідальності, навіть відносно невеликий проєкт швидко стає важким для будь-яких модифікацій.

Для системи моделювання оформлення замовлення це питання стоїть особливо гостро: програма одночасно взаємодіє з користувачем, обробляє товари, формує замовлення, змінює статуси та зберігає інформацію про виконані операції. Тримати всю цю логіку в одному місці – значить приректи проєкт на постійні проблеми при змінах.

Саме тому в основу покладено багаторівневу архітектуру з поділом системи на окремі логічні компоненти. Кожен із них відповідає за чітко визначену ділянку роботи і не втручається у справи інших. У структурі системи виділено шість основних модулів:

– модуль користувацького інтерфейсу;

– модуль бізнес-логіки;

– модуль роботи з даними;

– модуль керування замовленнями;

– модуль кошика;

– модуль роботи з товарами.

Модуль користувацького інтерфейсу – це єдина точка контакту між людиною і системою. Саме через нього користувач додає товари до кошика, вводить свої дані та підтверджує замовлення. Принципово важливо, щоб цей модуль залишався простим і не перекладав на користувача зайву роботу.

Для того щоб наочно показати взаємодію між основними шарами системи та розподіл функціональності між компонентами, було сформовано загальну архітектурну схему програмного засобу. Вона демонструє послідовність проходження даних від користувацького інтерфейсу до рівня збереження інформації та дозволяє краще зрозуміти внутрішню організацію системи.

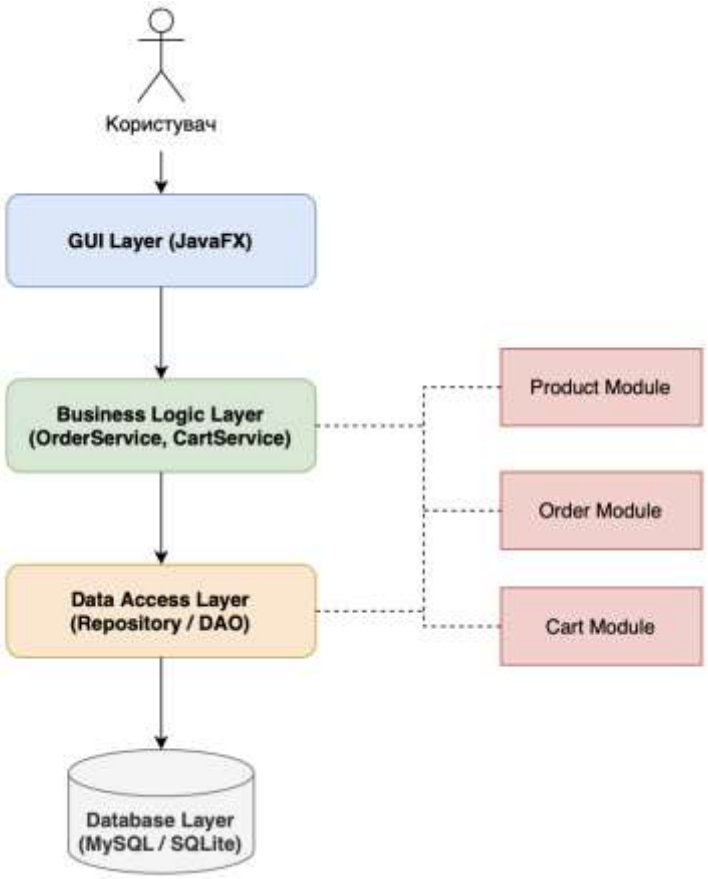


Рисунок 2.2 – Загальна архітектура програмного засобу

Як показано на рисунку 2.2, система побудована за багаторівневим принципом. Кожен рівень відповідає за окрему функціональну частину та взаємодіє лише із суміжними компонентами. Подібний підхід дозволяє зменшити кількість жорстких залежностей між модулями та спрощує подальшу підтримку програмного засобу [11, 21].

Модуль бізнес-логіки є центральним компонентом системи, що забезпечує координацію обчислювальних процесів. Тут виконуються розрахунок вартості замовлення, перевірка коректності введених даних, зміна статусів і формування структури замовлення. По суті, цей компонент координує роботу всіх інших частин програми і відповідає за те, щоб кожна операція виконувалась правильно.

Модуль роботи з даними займається збереженням і завантаженням інформації – про товари, користувачів і замовлення. Його виділення в окремий шар дає важливу перевагу: якщо в майбутньому знадобиться змінити спосіб

зберігання або перейти на іншу систему керування базами даних, це можна зробити без суттєвого втручання в бізнес-логіку.

Модуль керування замовленнями відповідає за створення об'єктів замовлення та зміну їх поточного стану. Наприклад, після підтвердження оформлення система автоматично присвоює статус «CREATED», а після виконання відповідних дій – змінює його на «CONFIRMED».

Для більш детального відображення структури системи та взаємодії між окремими модулями було використано UML-діаграму компонентів. Вона дозволяє показати, які частини програмного засобу відповідають за окремі функції та яким чином між ними організована взаємодія.

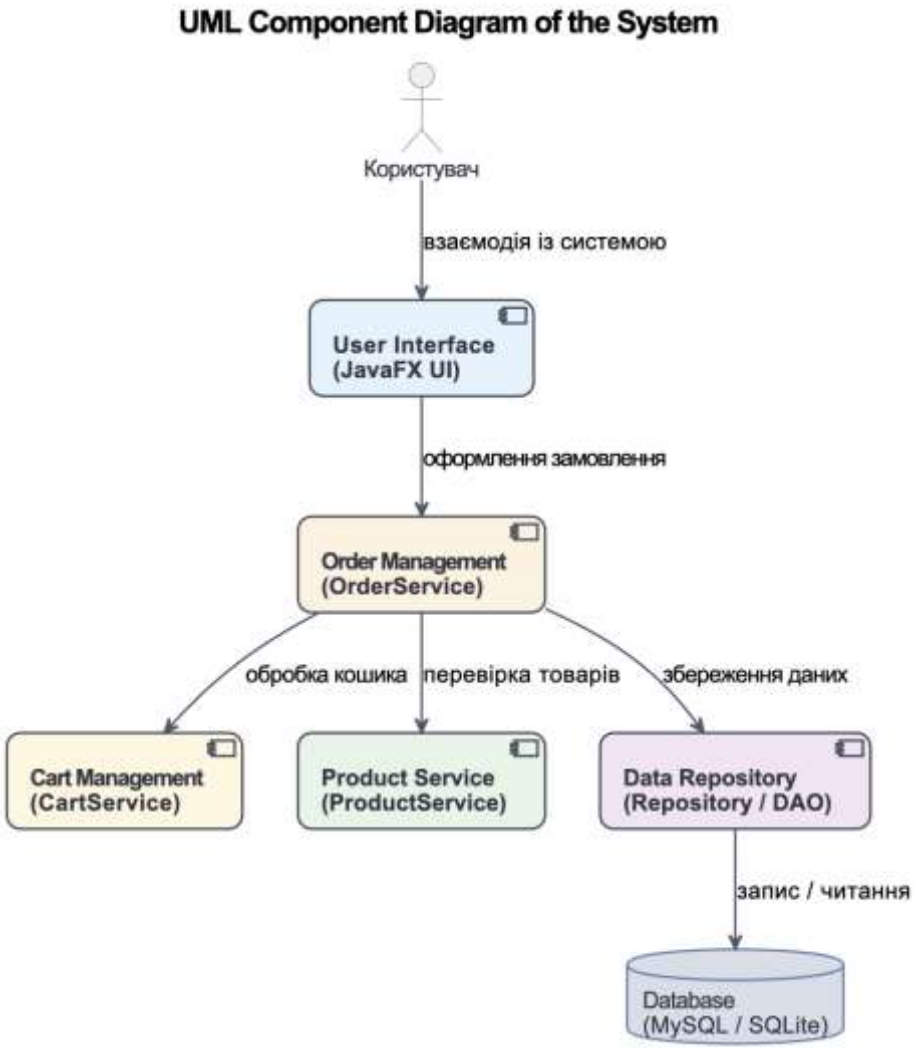


Рисунок 2.3 – UML-діаграма компонентів системи

Представлена UML-діаграма відображає архітектурні зв'язки між основними підсистемами та підтверджує чіткий поділ функціональної відповідальності в межах розроблюваного програмного засобу.

Окремо під час проєктування враховувалась масштабованість. Навіть якщо поточна версія системи містить лише базовий функціонал, структура проєкту повинна не заважати, а навпаки – спрощувати подальше розширення. У перспективі до системи можна додати механізми авторизації, інтеграцію з платіжними сервісами або аналітику замовлень, і це не вимагатиме переписування вже готових частин.

При цьому весь код побудований з дотриманням принципів ООП і мінімальними жорсткими залежностями між компонентами. Це робить код структурованим, спрощує його тестування і полегшує роботу з ним у довгостроковій перспективі.

Для узагальнення функціонального призначення основних компонентів системи доцільно окремо представити їх у вигляді таблиці. Це дозволяє швидко оцінити роль кожного модуля в архітектурі програмного засобу та краще зрозуміти структуру системи.

Таблиця 2.3 – Основні модулі системи та їх функції

Модуль	**Основні функції**
Модуль користувацького інтерфейсу	Взаємодія користувача із системою
Модуль бізнес-логіки	Обробка замовлень та бізнес-операцій
Модуль роботи з даними	Збереження та завантаження інформації
Модуль керування замовленнями	Створення та зміна статусів замовлення
Модуль кошика	Робота з кошиком товарів
Модуль роботи з товарами	Обробка інформації про товари

Як видно з таблиці 2.3, кожен модуль системи виконує окрему групу задач і відповідає лише за власну функціональність. Подібний поділ дозволяє

мінімізувати залежності між компонентами та спрощує подальший розвиток програмного засобу.

Таким чином, правильно спроєктована архітектура програмного засобу є основою стабільної роботи всієї системи. Багаторівнева структура і чіткий поділ функціональності між компонентами дозволяють створити гнучкий продукт, готовий до подальшого розвитку.

2.3. Об'єктно-орієнтоване моделювання системи засобами UML

Перш ніж писати перший рядок коду, варто зрозуміти, як система влаштована зсередини. Для цього в сучасній програмній інженерії використовується UML (Unified Modeling Language) – набір нотацій, які дозволяють описати архітектуру, взаємодію компонентів і логіку роботи системи ще до того, як вона реально існує [7].

Для проєкту, що будується на об'єктно-орієнтованому підході, UML є особливо природним інструментом: він думає в тих самих категоріях – класи, об'єкти, повідомлення, сценарії. Саме тому моделювання засобами UML стало невід'ємною частиною етапу проєктування цієї системи [22].

Першим кроком стала побудова діаграми варіантів використання. Вона дає відповідь на просте запитання: хто користується системою і що саме робить? В нашому випадку основний актор – користувач, який взаємодіє з функціями перегляду товарів, формування кошика та оформлення замовлення.

Для більш наочного відображення взаємодії користувача з функціональними можливостями системи було побудовано UML-діаграму варіантів використання. Вона дозволяє визначити основні сценарії роботи користувача та окреслити межі функціональності програмного засобу.

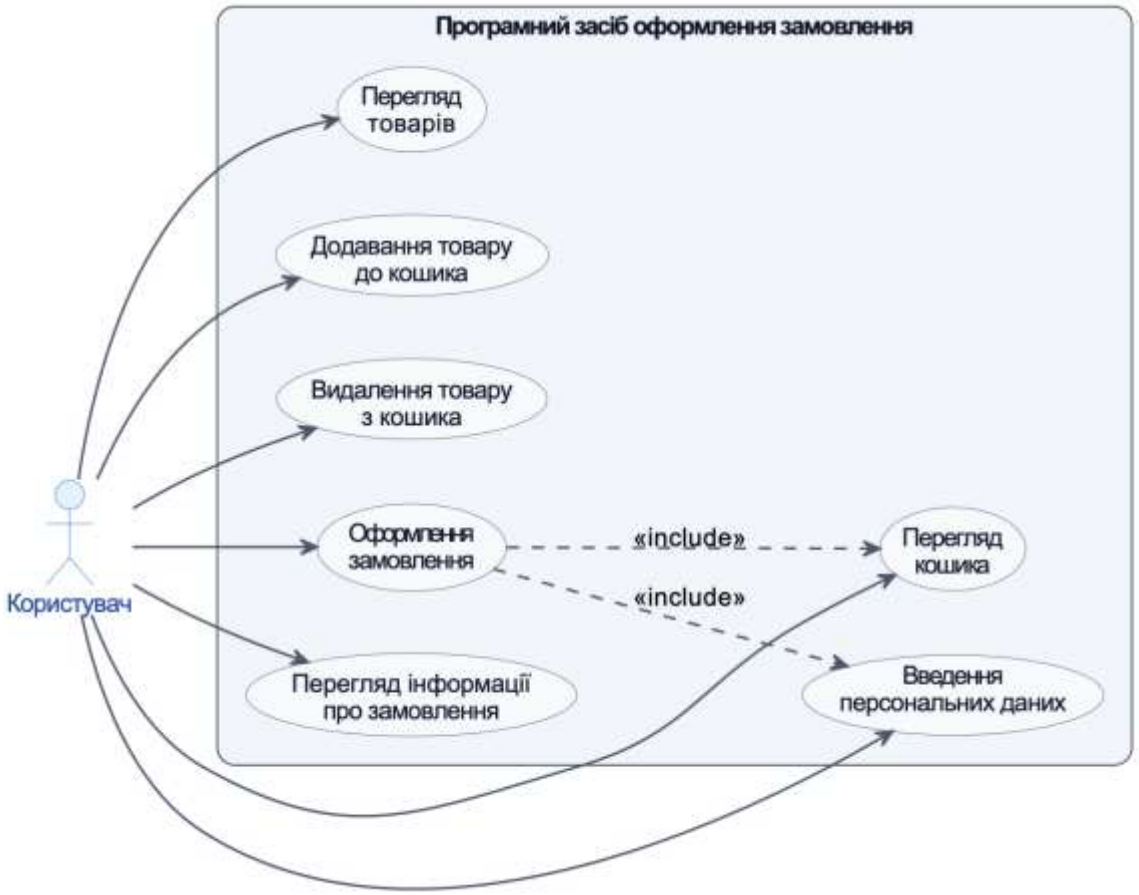


Рисунок 2.4 – UML-діаграма варіантів використання системи

Як показано на рисунку 2.4, центральним учасником системи є користувач, який взаємодіє з основними функціями програмного засобу. Діаграма демонструє повний цикл роботи із системою: від перегляду товарів і формування кошика до створення та перегляду замовлення. Представлені варіанти використання охоплюють усі ключові етапи бізнес-процесу оформлення замовлення.

Конкретно, у межах системи користувач може виконувати такі дії:

– переглядати список доступних товарів;

– додавати товари до кошика;

– видаляти товари з кошика;

– оформлювати замовлення;

– вводити персональні дані;

– переглядати інформацію про створене замовлення.

Після того як сценарії роботи визначені, наступний крок – діаграма класів. Саме вона відображає структуру об'єктів предметної області та зв'язки між ними, фактично формуючи кістяк майбутньої архітектури.

У системі виділено сім основних класів:

– Product;

– Customer;

– Cart;

– CartItem;

– Order;

– OrderItem;

– DeliveryDetails;

– OrderService.

Виділення саме цих класів обумовлене структурою предметної області. Кожен із них відповідає окремій сутності або бізнес-функції, що дозволяє забезпечити високий рівень зв'язності всередині класів та мінімізувати залежності між окремими компонентами системи.

Клас Product зберігає характеристики товару: назву, ціну, опис і кількість. Клас Customer – персональні дані покупця. Клас Cart відповідає за роботу з кошиком: додавання і видалення товарів та підрахунок загальної суми. Центральна роль належить класу Order, який об'єднує інформацію про покупця, товари і поточний статус замовлення. Для інкапсуляції логіки створення і обробки замовлень виділено окремий сервісний клас OrderService.

Після визначення основних сценаріїв використання системи було виконано моделювання її внутрішньої структури. Для цього побудовано UML-діаграму класів, яка відображає ключові об'єкти предметної області, їх атрибути та взаємозв'язки між ними. Саме ця діаграма стала основою для подальшої реалізації системи мовою Java.

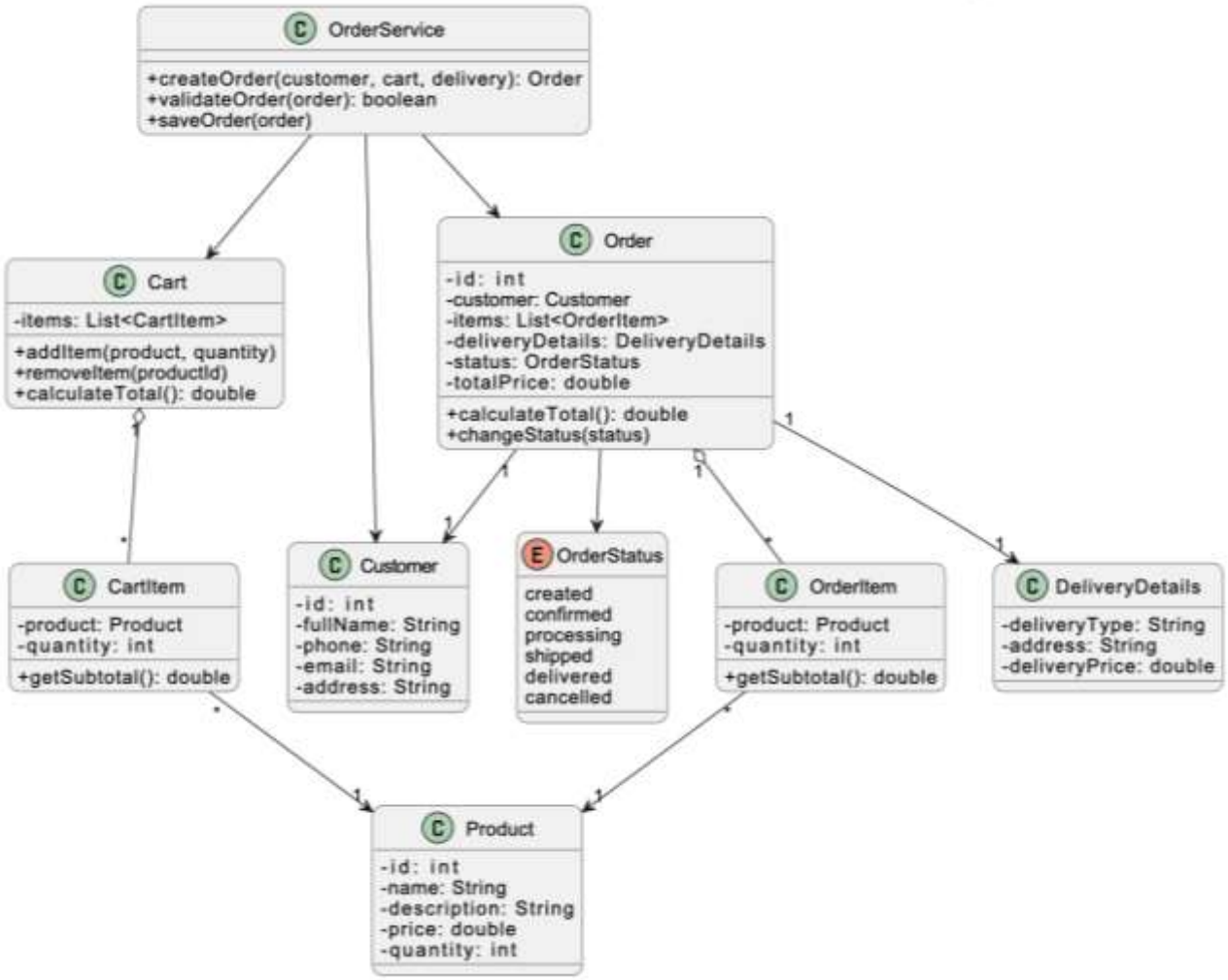


Рисунок 2.5 – UML-діаграма класів системи

Представлена діаграма класів відображає об'єктну модель бізнес-процесу оформлення замовлення. Центральне місце в ній займає клас Order, який об'єднує інформацію про покупця, перелік товарів та параметри доставки. Допоміжні класи Product, Customer, Cart, CartItem OrderItem і DeliveryDetails забезпечують зберігання та обробку даних окремих елементів системи, тоді як клас OrderService інкапсулює бізнес-логіку створення та супроводу замовлення.

Статична картина класів – це лише частина розуміння системи. Не менш важливо побачити, як об'єкти взаємодіють між собою в динаміці. Для цього використовується діаграма послідовності, яка показує, в якому порядку

компоненти обмінюються повідомленнями під час виконання конкретного сценарію.

Наприклад, у бізнес-процесі оформлення замовлення така діаграма відображає весь ланцюжок подій: від моменту, коли користувач додає товар до кошика, до фінального збереження замовлення в системі. Це дозволяє ще до написання коду побачити, де можуть виникнути проблеми з логікою або зайві залежності між компонентами.

Для аналізу поведінки системи в процесі виконання основного сценарію було побудовано UML-діаграму послідовності. Вона дозволяє простежити порядок взаємодії між об'єктами та визначити послідовність виконання операцій під час оформлення замовлення.

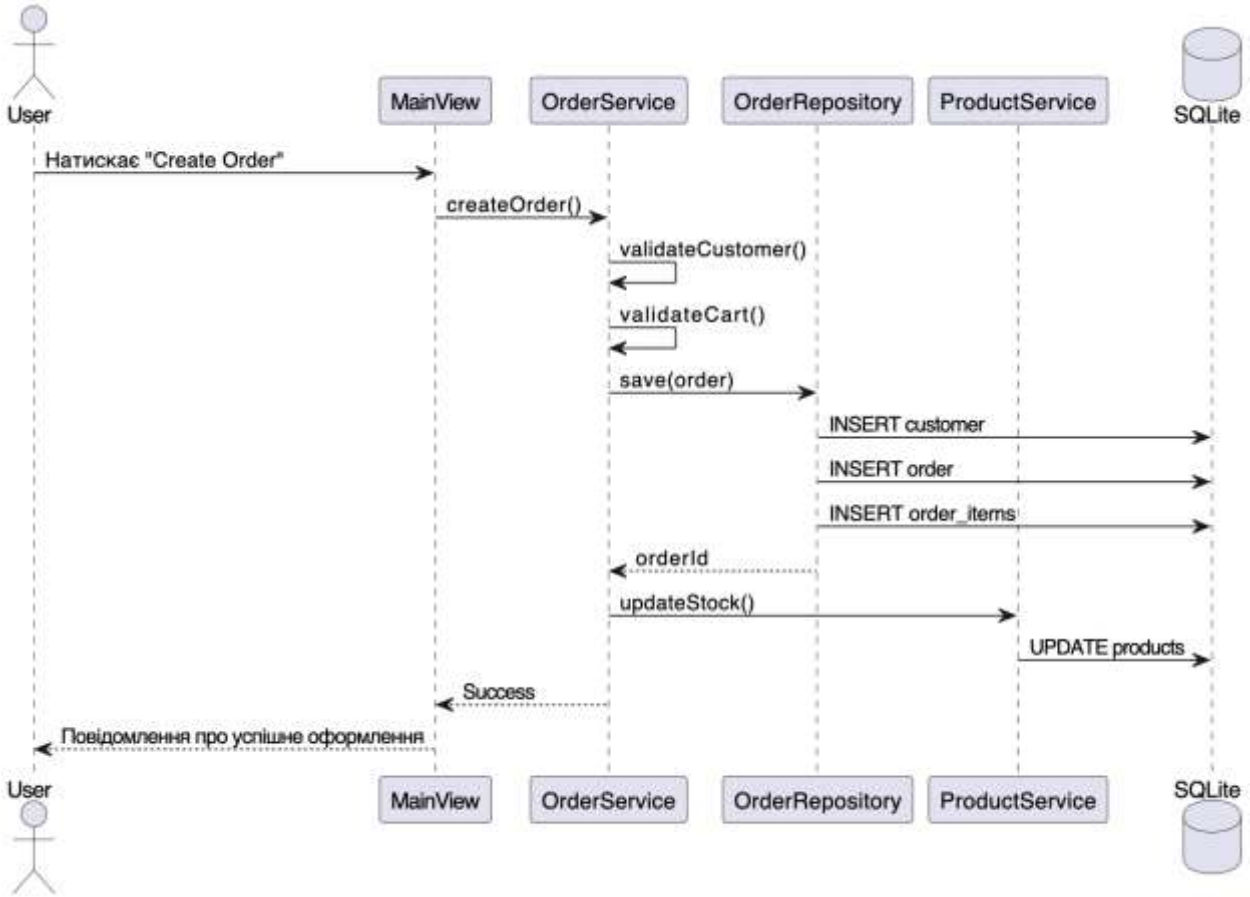


Рисунок 2.6 – UML-діаграма послідовності оформлення замовлення

Як видно з рисунка 2.6, процес оформлення замовлення складається з послідовності взаємопов'язаних дій. Користувач додає товари до кошика,

після чого система виконує перевірку введених даних, формує об'єкт замовлення та передає його до підсистеми збереження даних. Подібне моделювання дозволяє ще на етапі проєктування перевірити коректність логіки роботи програмного засобу та уникнути потенційних помилок під час програмної реалізації.

Загалом, UML-моделювання є важливим етапом проєктування, який дозволяє виявити потенційні архітектурні недоліки ще до початку програмної реалізації. Помилки в архітектурі, виявлені на рівні діаграм, виправляються за хвилини. Ті самі помилки, виявлені в готовому коді, можуть коштувати днів роботи.

2.4. Проєктування структури даних та бази даних системи

Будь-яка система, яка зберігає свій стан між сеансами роботи, вирішує одне принципове питання: як і де функціонує збереження даних. Для системи оформлення замовлення це питання стоїть особливо гостро, адже під час роботи вона повинна гарантувати надійну фіксацію інформації про товари, користувачів, замовлення та доставку. Якщо структуру зберігання спроєктовано незадовільно – з'являється дублювання, виникають помилки при обробці і зростає складність супроводу.

Щоб цього уникнути, ще на етапі проєктування були виділені основні сутності предметної області:

– користувач;

– товар;

– кошик;

– елемент кошика;

– замовлення;

– елемент замовлення;

– дані доставки.

Між цими сутностями існують зв'язки, які потрібно врахувати при проєктуванні. Одне замовлення може містити кілька товарів. Кожен товар має

власні характеристики. Замовлення пов'язане з конкретним користувачем і містить інформацію про спосіб доставки. Ці зв'язки є визначальними для точного відображення бізнес-процесу.

Для наочного представлення основних сутностей та зв’язків між ними було побудовано ER-діаграму структури даних системи. Вона показує, як між собою пов’язані користувач, товар, замовлення, кошик, позиції замовлення та дані доставки.

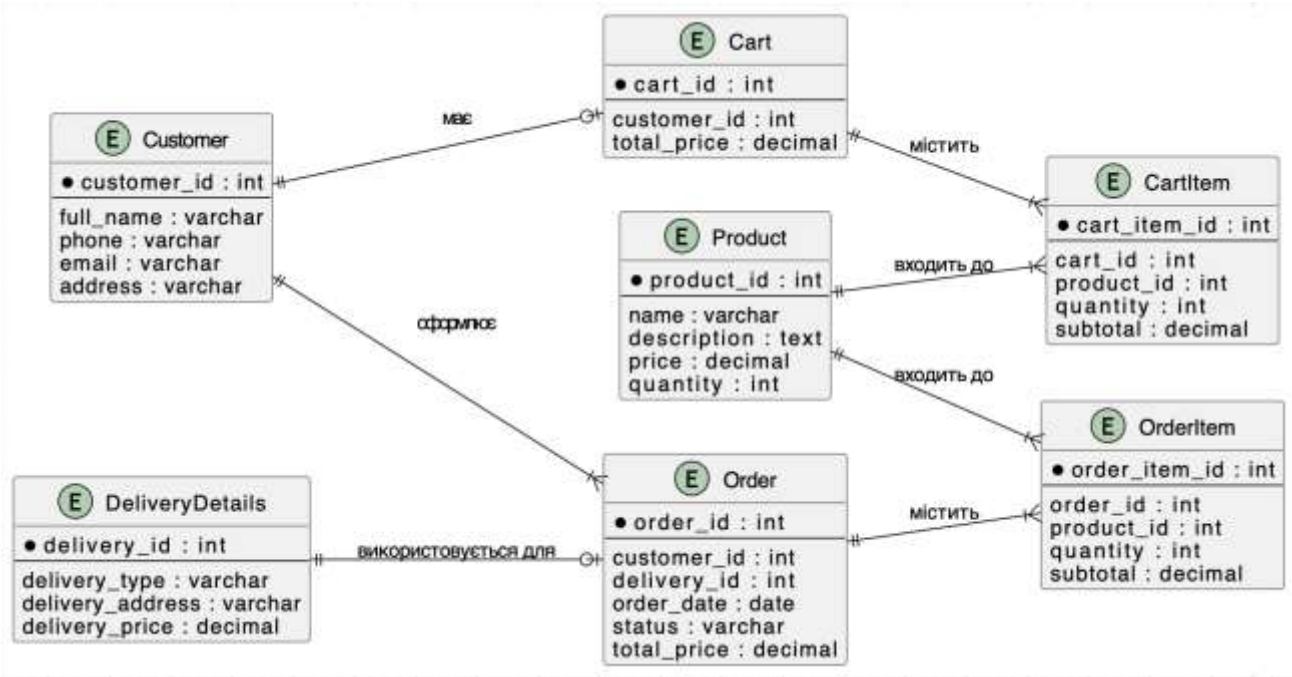


Рисунок 2.7 – ER-діаграма структури даних системи

Як показано на рисунку 2.7, центральною сутністю є Order, оскільки саме вона об’єднує дані про покупця, перелік товарів та параметри доставки. Зв’язок між замовленням і товарами реалізується через сутність OrderItem, що дозволяє коректно зберігати кілька товарних позицій у межах одного замовлення.

Сутність Product описує товар і включає такі поля:

– унікальний ідентифікатор;

– назву;

– опис;

– ціну;

– кількість на складі.

Сутність Customer зберігає дані покупця: ім'я, номер телефону, адресу доставки та електронну пошту. Виділення її в окрему структуру дозволяє уникнути ситуації, коли ті самі персональні дані дублюються в кожному замовленні.

Сутності Cart та CartItem відповідають за фіксацію проміжного стану вибору користувача. Cart містить ідентифікатор користувача та загальну вартість обраних позицій, тоді як CartItem деталізує кожну додану одиницю товару, її кількість та проміжну суму (subtotal).

Центральна сутність – Order. Саме вона об'єднує всю інформацію про оформлене замовлення:

– номер замовлення;

– дату створення;

– поточний статус;

– загальну вартість;

– список товарів;

– дані покупця.

Оскільки одне замовлення може містити кілька позицій, для реалізації цього зв'язку використовується окрема сутність OrderItem. Вона фіксує, який конкретно товар входить до замовлення і в якій кількості.

Для узагальнення структури даних основні сутності системи та їх поля наведено у таблиці 2.4.

Таблиця 2.4 – Основні сутності системи та їх поля

Сутність	Основні поля	Призначення
Customer	customer_id, full_name, phone, email, address	Зберігає персональні та контактні дані покупця, необхідні для оформлення та обробки замовлення.
Cart	cart_id, customer_id, total_price	Призначений для тимчасового зберігання обраних користувачем товарів до моменту оформлення замовлення.

Продовження таблиці 2.4

CartItem	cart_item_id, cart_id, product_id, quantity, subtotal	Містить інформацію про окрему товарну позицію в кошику, її кількість та проміжну вартість.
Product	product_id, name, description, price, quantity	Містить відомості про товари, їх характеристики, вартість та доступну кількість на складі.
Order	order_id, customer_id, delivery_id, order_date, status, total_price	Зберігає загальну інформацію про оформлене замовлення, його статус та підсумкову вартість.
OrderItem	order_item_id, order_id, product_id, quantity, subtotal	Фіксує склад замовлення та містить дані про кожний товар, що входить до нього.
DeliveryDetails	delivery_id, delivery_type, delivery_address, delivery_price	Зберігає інформацію про спосіб доставки, адресу отримання та вартість логістичних послуг.

Як видно з таблиці 2.4, кожна сутність відповідає окремій частині бізнес-процесу оформлення замовлення. Такий поділ дозволяє уникнути дублювання даних і зробити структуру системи більш зрозумілою для подальшої реалізації.

Під час вибору механізму збереження даних розглядалися різні підходи. Використання текстових файлів або формату JSON є неефективним через відсутність підтримки складних зв'язаних запитів та типізації при збільшенні обсягів інформації. Тому реляційна база даних є найбільш оптимальним варіантом для даної роботи, оскільки вона дозволяє наочно показати, як програмний засіб взаємодіє зі структурованим сховищем даних у реальних умовах [29, 30].

Водночас під час зберігання даних важлива не лише структура, але й надійність. Система повинна гарантувати, що після оформлення замовлення вся інформація – дані користувача, перелік товарів, параметри доставки – буде збережена повністю і без спотворень. Саме тому структура даних спроєктована послідовно і без зайвої надмірності.

Окремо було сформовано схему взаємозв’язків між таблицями бази даних. Вона деталізує, які поля використовуються як первинні та зовнішні ключі, а також показує логіку збереження пов’язаних записів.

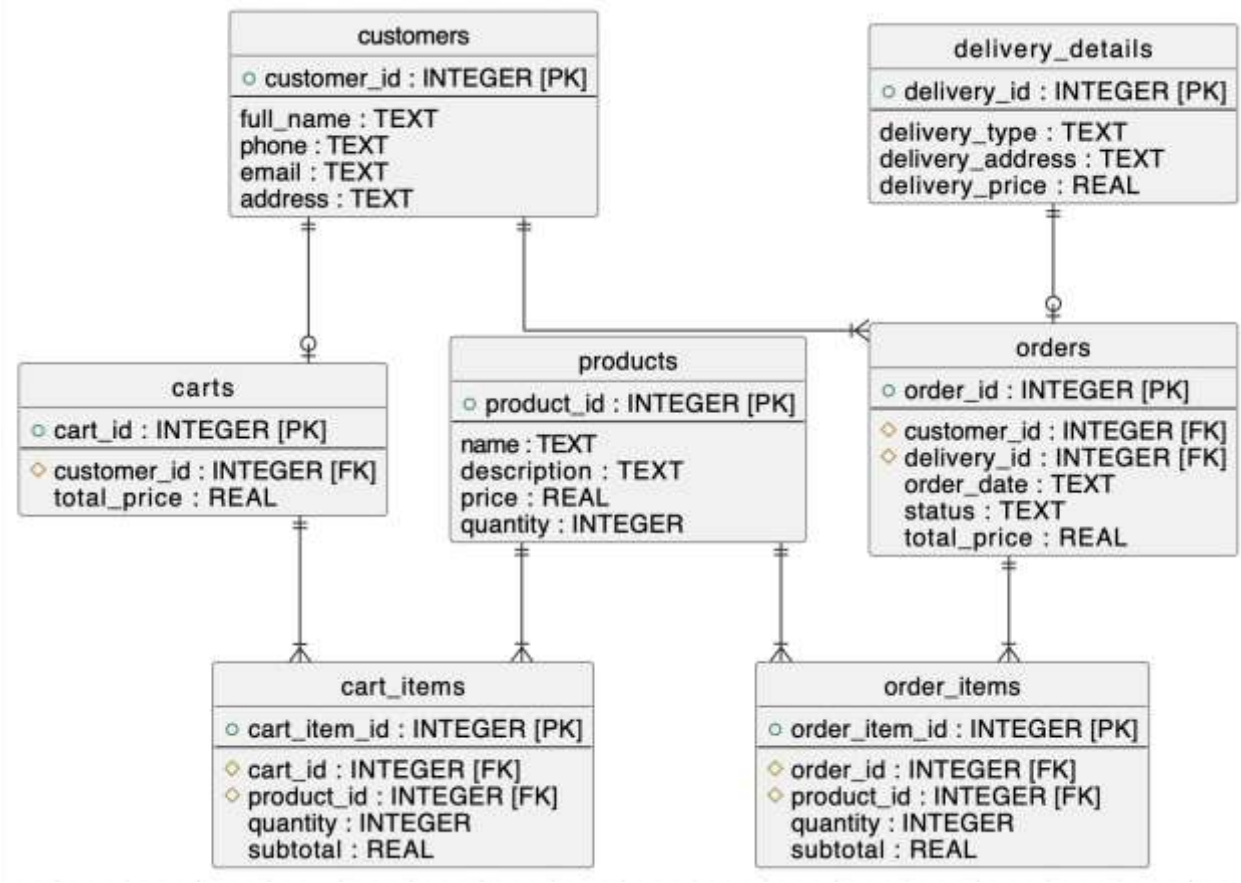


Рисунок 2.8 – Схема взаємозв’язків між таблицями бази даних

Схема фізичної структури бази даних на рисунку 2.8 демонструє взаємозв’язки між усіма таблицями системи. Зокрема, таблиця orders пов’язана із таблицями customers, delivery_details та order_items, де остання забезпечує зв’язок із номенклатурою товарів products. Аналогічно, поточний стан користувацьких кошиків фіксується через таблиці carts та cart_items, що дозволяє підтримувати транзакційність процесу оформлення замовлення без дублювання інформації.

Таким чином, грамотно спроєктована структура даних – це не просто технічна деталь, а важлива умова коректної роботи всієї системи. Чітке визначення сутностей і зв'язків між ними забезпечує коректну обробку

замовлень, спрощує реалізацію бізнес-логіки і залишає простір для подальшого розвитку.

Висновки до розділу 2

У другому розділі було виконано проєктування архітектури програмного засобу та обґрунтовано вибір технологій розробки. Java обрана як основна мова завдяки підтримці об'єктно-орієнтованого підходу, платформонезалежності та зрілості інструментального середовища. Для побудови інтерфейсу обрано JavaFX, для організації проєкту – Maven, для контролю версій – Git.

Розглянуто архітектурну організацію системи. В основу покладено багаторівневу структуру з виділенням модулів користувацького інтерфейсу, бізнес-логіки, роботи з даними та керування замовленнями. Такий підхід забезпечує чіткий розподіл відповідальності між компонентами і спрощує подальшу підтримку коду.

Виконано об'єктно-орієнтоване моделювання системи засобами UML. Побудовано діаграми варіантів використання, класів та послідовності, які відображають структуру і динаміку взаємодії компонентів системи. Це дозволило сформувати цілісне уявлення про внутрішню організацію програмного засобу ще до початку його реалізації.

Окремо спроєктовано структуру даних і визначено основні сутності предметної області: користувач, товар, кошик, замовлення, елемент замовлення та дані доставки. Встановлено зв'язки між сутностями і обрано реляційну базу даних як механізм зберігання інформації.

Результати цього розділу формують архітектурну основу системи і створюють необхідну базу для практичної реалізації програмного засобу мовою Java у наступному розділі роботи.

На основі сформованих архітектурних рішень, UML-моделей та структури даних можна переходити до безпосередньої програмної реалізації системи та перевірки її працездатності.

РОЗДІЛ 3.ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ

3.1. Реалізація структури програмного засобу мовою Java

Після завершення етапу проєктування архітектури виконано безпосередню програмну реалізацію засобу мовою Java. Основне завдання на цьому етапі полягало в трансляції розробленої об'єктної моделі в діючий програмний проєкт із повним збереженням принципів об'єктно-орієнтованого програмування та чітким розмежуванням зон відповідальності компонентів.

Першим кроком стало формування загальної структури Java-проєкту. Щоб класи не розміщувались хаотично, систему одразу було розділено на окремі пакети за їх функціональним призначенням. Такий підхід помітно спрощує орієнтування в коді та полегшує подальшу підтримку програми.

У структурі проєкту було виділено п'ять основних пакетів:

– model;

– service;

– repository;

– ui;

– util.

Пакет model – це основа предметної області. Саме тут знаходяться класи Product, Customer, Cart, CartItem, Order, OrderItem та DeliveryDetails, які відповідають за зберігання інформації про товари, користувачів та замовлення.

У пакеті service зосереджена бізнес-логіка системи: сервіси, що відповідають за створення замовлення, роботу з кошиком, зміну статусів та виконання ключових операцій над даними. Завдяки такому поділу бізнес-логіка не переплітається ні з графічним інтерфейсом, ні з механізмами збереження даних – кожен шар існує окремо.

Така організація ізолює предметну модель від бізнес-логіки та механізмів збереження, що спрощує супровід коду при подальшому розширенні системи [1, 11].

Пакет repository відповідає за роботу з даними: читання та запис інформації про замовлення у базу даних SQLite. Виділення окремого шару доступу до даних дає змогу ізолювати механізми збереження від усієї іншої частини системи – якщо в майбутньому потрібно буде змінити спосіб зберігання, це не торкнеться бізнес-логіки.

Для реалізації графічної частини було створено пакет ui, де розміщуються JavaFX-класи: вікна програми, форми оформлення замовлення та все, що стосується взаємодії користувача з системою.

Нарешті, пакет util містить допоміжні класи – для валідації даних, форматування інформації та різних службових операцій. Це дозволяє уникнути ситуації, коли одна й та сама логіка дублюється в кількох місцях програми.

Фактичну організацію файлів у середовищі розробки показано на рисунку 3.1.

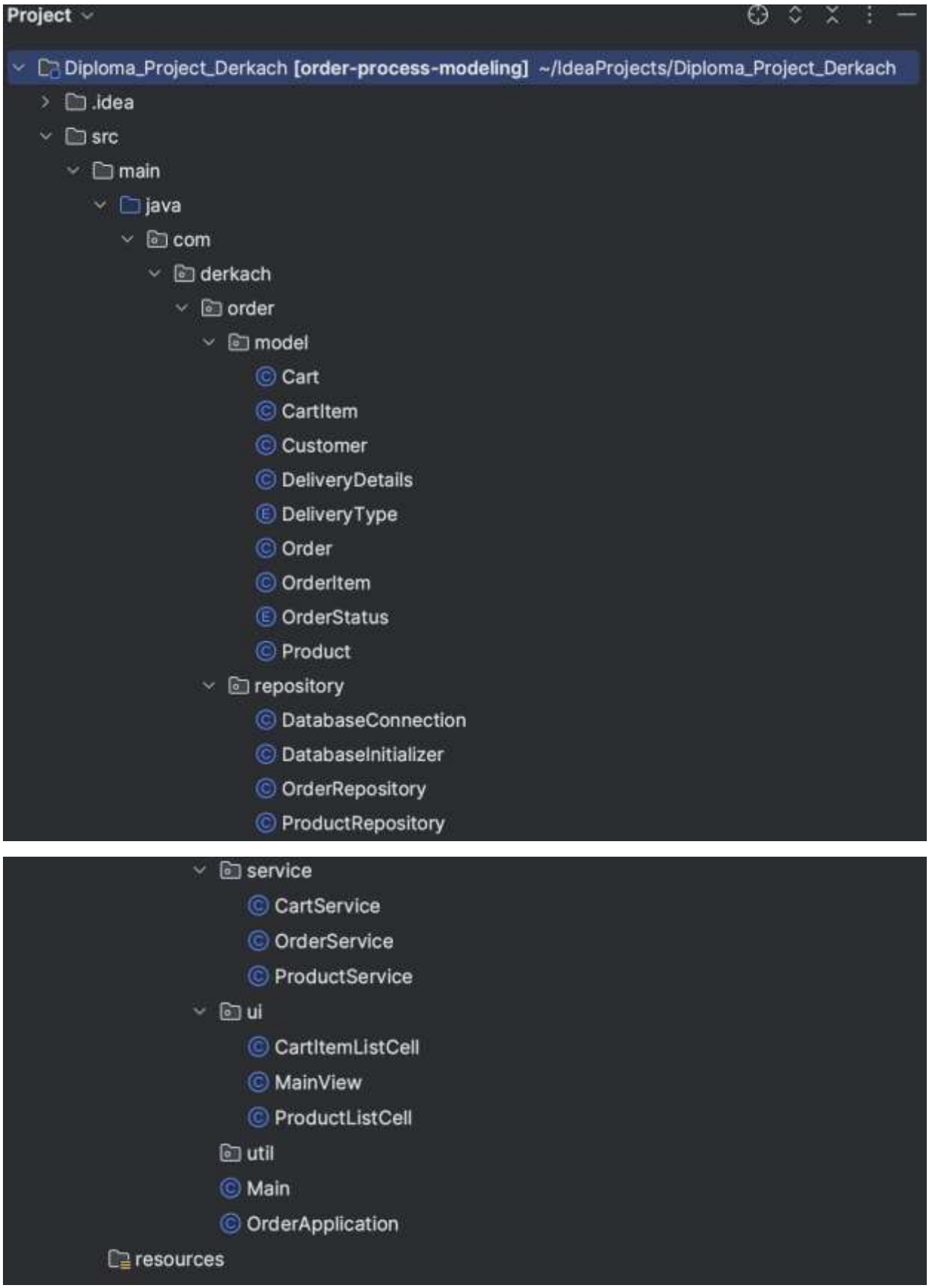


Рисунок 3.1 – Структура проєкту в середовищі IntelliJ IDEA

Розподіл класів відповідає архітектурним рішенням, прийнятим на етапі проєктування.

Під час реалізації системи особливу увагу приділяли принципу єдиної відповідальності. Кожен клас виконує рівно одне завдання: клас Order займається лише самим замовленням, а клас OrderService – бізнес-операціями над ним. Завдяки цьому код залишається зрозумілим, а будь-які зміни в одному місці не тягнуть за собою непередбачені наслідки в іншому.

Взаємодія між компонентами побудована через методи та об'єкти класів. Наприклад, коли користувач створює нове замовлення, система формує об'єкт Order, наповнює його переліком товарів та даними про покупця, а потім передає цей об'єкт у сервісний шар для подальшої обробки.

Одним із ключових класів системи є клас Order, який представляє оформлене замовлення та об'єднує інформацію про покупця, перелік товарів, параметри доставки і поточний стан замовлення.

Фрагмент реалізації цього класу наведено нижче.

```
public class Order {

    private int id;
    private Customer customer;
    private List<OrderItem> orderItems;
    private DeliveryDetails deliveryDetails;
    private OrderStatus status;

    public Order(Customer customer,
                 List<OrderItem> orderItems,
                 DeliveryDetails deliveryDetails) {

        this.customer = customer;
        this.orderItems = orderItems;
        this.deliveryDetails = deliveryDetails;
        this.status = OrderStatus.CREATED;
    }
}
```

Лістинг 3.1 – Фрагмент реалізації класу Order

Представлений клас реалізує принцип інкапсуляції: усі дані замовлення зосереджені в одному місці, що спрощує їх обробку.

Окремо варто згадати про обробку помилок та перевірку введених даних. Система не дозволяє створити замовлення, якщо номер телефону введено

некоректно, кошик порожній або кількість товару виходить за допустимі межі. Для цього використовуються механізми валідації та обробки виключень.

Важливо й те, що структура програми від початку проєктувалась з думкою про майбутнє. За потреби систему можна розширити: додати нові способи оплати, підключити авторизацію користувачів або інтеграцію із зовнішніми сервісами – і все це без суттєвої перебудови вже існуючої архітектури.

Після реалізації основних класів було сформовано схему їх взаємодії в межах програмного засобу. Вона демонструє, як об'єкти предметної області та сервісні компоненти співпрацюють під час виконання бізнес-процесу оформлення замовлення.

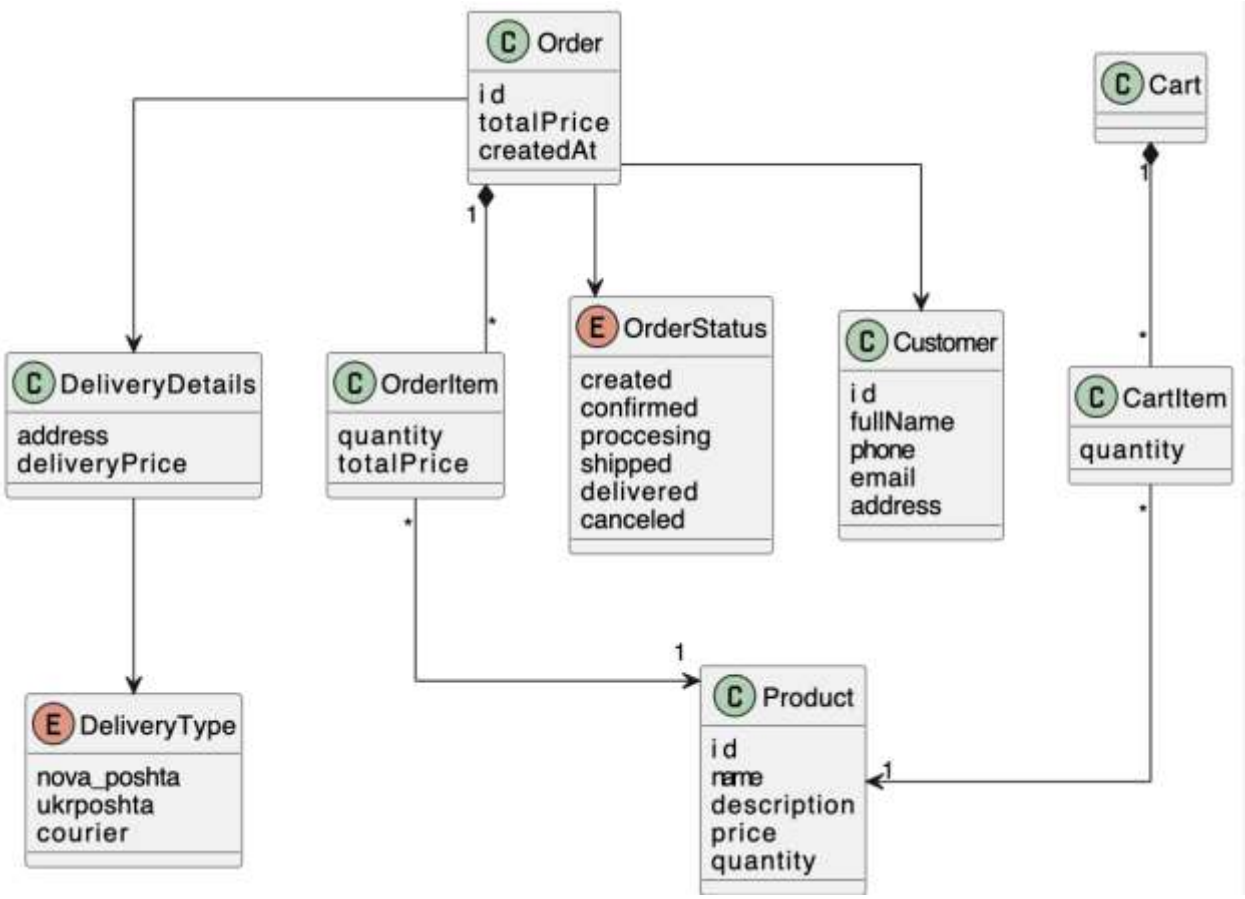


Рисунок 3.2 – Взаємодія основних класів програмного засобу

Як видно зі схеми, центральне місце займає клас Order, навколо якого організована взаємодія інших компонентів системи. Саме через нього

відбувається зв'язок між покупцем, товарами та параметрами доставки, тоді як сервісний рівень забезпечує виконання бізнес-операцій і збереження результатів роботи.

Таким чином, реалізація структури програмного засобу дозволила отримати логічно організований проєкт, де кожен компонент чітко відповідає за свою роль. Використання принципів ООП та багаторівневої архітектури забезпечило зрозумілу структуру коду та заклало міцну основу для реалізації бізнес-логіки оформлення замовлення.

3.2. Реалізація бізнес-логіки оформлення замовлення

Після того як загальна структура програмного засобу була сформована, настав час реалізувати бізнес-логіку системи оформлення замовлення. Саме цей компонент забезпечує виконання внутрішніх обчислювальних процесів системи під час взаємодії користувача з інтерфейсом: обробку кошика, верифікацію вхідних даних, безпосереднє створення замовлення та відстеження його статусів.

На відміну від графічного інтерфейсу, бізнес-логіка не видима користувачеві – але саме вона визначає, як система поводитиметься у кожній конкретній ситуації. Що має статися, якщо кошик порожній? Як відреагувати на некоректний номер телефону? Чи можна завершити оформлення замовлення, якщо не всі поля заповнені? Відповіді на всі ці питання закладені саме тут.

Ключовим елементом реалізації бізнес-логіки є метод створення замовлення, розміщений у класі OrderService. Саме цей метод об’єднує основні дії системи: перевірку стану кошика, формування об’єкта замовлення, передачу даних до репозиторію та оновлення залишків товарів після успішного оформлення покупки.

Фрагмент реалізації цього механізму наведено у лістингу 3.2.

```
public Order createOrder(Customer customer, Cart cart, DeliveryDetails
deliveryDetails) {
    if (customer == null) {
        throw new IllegalArgumentException("Customer cannot be null.");
    }
    if (cart == null || cart.isEmpty()) {
        throw new IllegalArgumentException("Cart cannot be empty.");
    }
    if (deliveryDetails == null) {
        throw new IllegalArgumentException("Delivery details cannot be
null.");
    }

    List<OrderItem> orderItems = new ArrayList<>();

    for (CartItem cartItem : cart.getItems()) {
        OrderItem orderItem = new OrderItem(cartItem.getProduct(),
cartItem.getQuantity());
        orderItems.add(orderItem);
    }

    Order order = new Order(customer, orderItems, deliveryDetails);
    orderRepository.save(order);

    for (CartItem cartItem : cart.getItems()) {
        productService.decreaseProductQuantity(
                cartItem.getProduct().getId(),
                cartItem.getQuantity()
        );
    }

    cart.clear();

    return order;
}
```

Лістинг 3.2 – Фрагмент реалізації методу створення замовлення в класі OrderService

Як видно з лістингу 3.2, метод створення замовлення виконує роль координатора між окремими компонентами системи. На початку методу здійснюється базова перевірка вхідних даних: наявності покупця, стану кошика та даних доставки. Якщо один із необхідних об’єктів відсутній або кошик є порожнім, система припиняє створення замовлення та генерує відповідне виключення. Після успішного проходження перевірок дані з кошика перетворюються на список позицій замовлення, створюється об’єкт Order, який передається до репозиторію для збереження. Після цього система оновлює залишки товарів і очищує кошик.

Такий підхід дозволяє зосередити основний сценарій оформлення замовлення в одному сервісному методі, не змішуючи його з кодом графічного інтерфейсу або прямими SQL-операціями. Завдяки цьому бізнес-логіка залишається відокремленою від представлення даних і механізмів їх збереження, що спрощує подальшу підтримку програмного засобу [11, 21].

Окрім перевірок, які виконуються безпосередньо під час створення замовлення, у програмному засобі реалізовано окремі механізми валідації даних покупця. Їх призначення полягає у запобіганні створенню замовлень із некоректною або неповною інформацією. Особлива увага приділяється перевірці контактних даних користувача, оскільки саме вони використовуються для подальшої обробки та доставки замовлення.

```
public void setPhone(String phone) {
    if (phone == null || phone.isBlank()) {
        throw new IllegalArgumentException("Phone number cannot be empty.");
    }
    if (!phone.matches("\\+?[0-9]{10,15}")) {
        throw new IllegalArgumentException("Phone number has invalid
format.");
    }
    this.phone = phone;
}
```

```
public void setEmail(String email) {
    if (email == null || email.isBlank()) {
        throw new IllegalArgumentException("Email cannot be empty.");
    }
    if (!email.matches("^[\\w.-]+@[\\w.-]+\\.[A-Za-z]{2,}$")) {
        throw new IllegalArgumentException("Email has invalid format.");
    }
    this.email = email;
}
```

Лістинг 3.3 – Фрагмент реалізації валідації контактних даних покупця

Як видно з лістингу 3.3, перевірка даних здійснюється безпосередньо в методах встановлення значень полів класу Customer. У разі відсутності введених даних або невідповідності встановленому формату система генерує виключення типу IllegalArgumentException. Для перевірки номера телефону використовується регулярний вираз, який допускає введення від 10 до 15 цифр із можливим префіксом «+». Перевірка електронної пошти забезпечує

відповідність стандартній структурі email-адреси. Такий підхід дозволяє запобігти створенню некоректних об’єктів та підвищує надійність роботи програмного засобу.

Після успішного проходження перевірок дані про вибрані товари перетворюються на складові майбутнього замовлення. Для цього в системі використовується клас OrderItem, який представляє окрему позицію замовлення та зберігає інформацію про товар, його кількість і вартість. Під час обробки кошика для кожного вибраного товару створюється відповідний об’єкт OrderItem, що дозволяє сформувати повну структуру замовлення та виконувати подальші розрахунки. Такий підхід робить модель замовлення гнучкою, зрозумілою та зручною для подальшого розширення функціональності системи.

Окремо було реалізовано механізм розрахунку загальної вартості замовлення. Система послідовно проходить по всіх позиціях кошика, обчислює суму для кожного товару та формує підсумкову вартість покупки. Цей механізм спроєктовано так, що в майбутньому його можна легко розширити – наприклад, додати підтримку знижок, промокодів або бонусних програм.

Фрагмент реалізації механізму обчислення вартості товарних позицій у кошику та загальної суми замовлення наведено у лістингу 3.4.

```
public double getItemsTotalPrice() {
    double total = 0;

    for (CartItem item : items) {
        total += item.getSubPrice();
    }

    return total;
}

public double getTotalPrice() {
    return getItemsTotalPrice() + deliveryDetails.getDeliveryPrice();
}
```

Лістинг 3.4 – Фрагмент реалізації розрахунку вартості замовлення

Як видно з лістингу 3.4, підсумкова вартість замовлення формується у два етапи. Спочатку система обчислює загальну вартість усіх товарних позицій шляхом послідовного підсумовування вартості кожного об’єкта OrderItem. Після цього до отриманої суми додається вартість доставки, що дозволяє отримати кінцеву ціну замовлення. Такий підхід забезпечує прозорість розрахунків і дає можливість у майбутньому легко розширити алгоритм додатковими механізмами, наприклад застосуванням знижок або бонусних програм.

Після формування структури замовлення та обчислення його вартості система створює об’єкт Order і присвоює йому початковий стан. Для відображення поточного етапу обробки замовлення в програмному засобі використовується механізм статусів, реалізований за допомогою переліку OrderStatus. У програмному засобі передбачено такі стани замовлення:

1. CREATED;
2. CONFIRMED;
3. PROCESSING;
4. SHIPPED;
5. DELIVERED;
6. CANCELLED.

Завдяки статусам система може моделювати повний життєвий цикл замовлення та відображати його поточний стан у будь-який момент часу.

Фрагмент реалізації переліку статусів замовлення наведено у лістингу 3.5.

```
public enum OrderStatus {
    CREATED,
    CONFIRMED,
    PROCESSING,
    SHIPPED,
    DELIVERED,
    CANCELLED
}
```

Лістинг 3.5 – Реалізація переліку статусів замовлення

Як видно з лістингу 3.5, кожному етапу обробки замовлення відповідає окреме значення переліку OrderStatus. Після створення замовлення йому автоматично присвоюється статус CREATED, після підтвердження – CONFIRMED, далі замовлення переходить до стадій обробки, відправлення та доставки. У разі необхідності замовлення може бути скасоване шляхом присвоєння статусу CANCELLED. Використання переліку дозволяє уникнути помилок, пов’язаних із текстовими значеннями статусів, та забезпечує централізоване керування життєвим циклом замовлення.

Для наочного відображення переходів між окремими станами замовлення було побудовано діаграму життєвого циклу замовлення, наведену на рисунку 3.3.

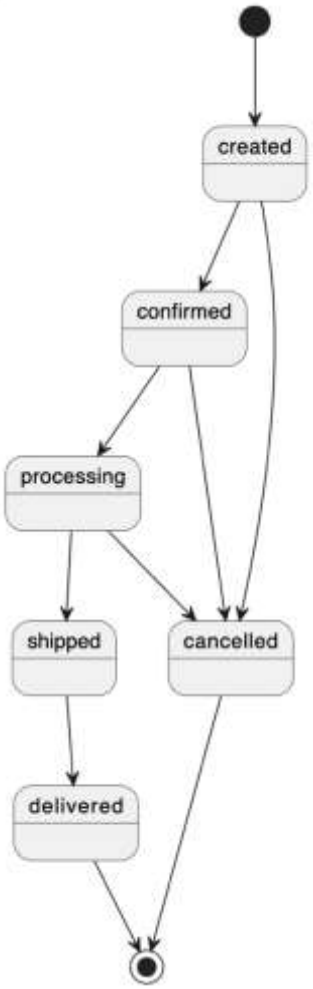


Рисунок 3.3 – Діаграма життєвого циклу замовлення

Як видно з рисунка 3.3, замовлення проходить послідовний шлях від моменту створення до завершення доставки. На будь-якому з початкових етапів обробки воно може бути скасоване, після чого переходить до фінального стану CANCELLED. Такий підхід дозволяє коректно відображати поточний стан замовлення та контролювати допустимі переходи між окремими етапами його обробки.

Паралельно реалізовано механізми централізованої обробки виключних ситуацій. У разі виникнення помилок під час виконання операцій система забезпечує стабільність своєї роботи, коректно інформуючи користувача про проблему через графічний інтерфейс без аварійного завершення потоків виконання.

Під час реалізації бізнес-логіки також враховувалась перспектива подальшого розвитку системи. Структура сервісного шару побудована таким чином, що до нього можна без значних зусиль додати інтеграцію з платіжними сервісами, механізм авторизації або автоматичне формування повідомлень після створення замовлення.

Таким чином, реалізована бізнес-логіка забезпечує коректну роботу всіх ключових механізмів оформлення замовлення та організовує злагоджену взаємодію між компонентами системи. Сервісний шар, об'єктно-орієнтований підхід і механізми валідації разом утворюють гнучку та добре структуровану основу програмного засобу.

3.3. Розробка користувацького інтерфейсу системи

У розробці програмного засобу важлива не лише внутрішня логіка – не менше значення має те, наскільки зручно і зрозуміло користувач взаємодіє з програмою. Ефективність функціонування системи суттєво знижується, якщо її графічний інтерфейс є надмірним або складним для сприйняття. Саме тому побудова зрозумілого і логічного GUI стала окремим і важливим завданням у цій роботі.

Графічну частину системи реалізовано за допомогою технології JavaFX. Вона дозволила створити повноцінний десктопний застосунок із підтримкою візуальних компонентів, обробки подій та динамічного оновлення даних. Порівняно зі звичайними консольними програмами JavaFX суттєво наближає застосунок до реального вигляду систем електронної комерції та робить взаємодію з ним природнішою [14, 20, 37].

В основі користувацького інтерфейсу знаходиться головне вікно програми, реалізоване у класі MainView. Інтерфейс побудовано на основі контейнера BorderPane, що дозволяє логічно розділити робочу область на дві основні частини. У лівій частині вікна відображається список доступних товарів, а в центральній частині розміщено кошик, поля для введення даних покупця, вибір типу доставки та кнопку створення замовлення. Для побудови інтерфейсу використовуються стандартні компоненти JavaFX: Label, ListView, TextField, ComboBox та Button.

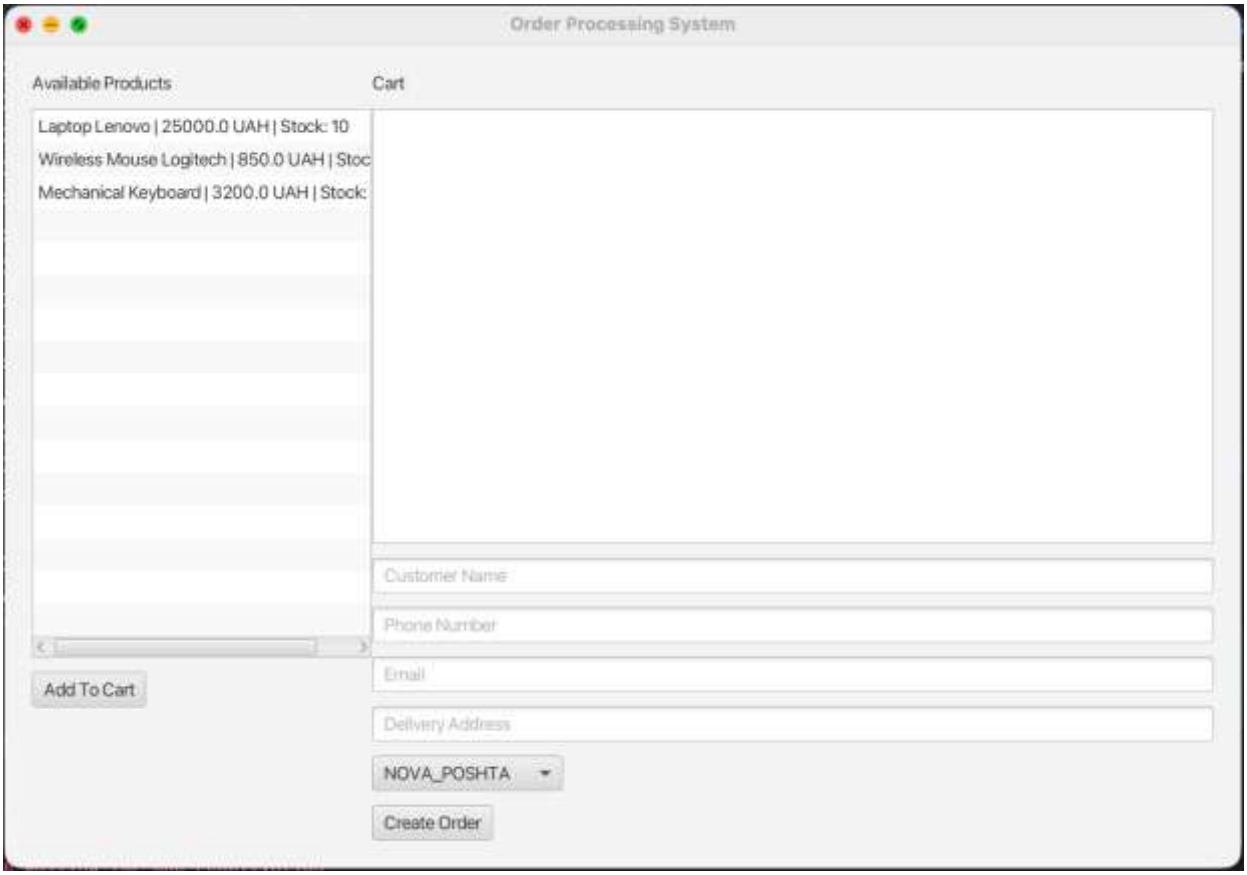


Рисунок 3.4 – Головне вікно програмного засобу

Як видно з рисунка 3.4, інтерфейс програмного засобу має просту двопанельну структуру. Ліва частина призначена для перегляду доступних товарів і додавання їх до кошика, а права частина використовується для відображення кошика та введення даних, необхідних для оформлення замовлення. Така організація інтерфейсу дозволяє користувачу виконувати всі основні дії в межах одного вікна без переходу між додатковими екранами.

Під час проєктування інтерфейсу одним із ключових завдань було уникнути перевантаженості. Для підвищення ергономічності та зниження когнітивного навантаження інтерфейс побудовано у вигляді чіткого і послідовного маршруту користувача.

Товари відображаються у вигляді списку з основною інформацією про кожну позицію: назвою, ціною та доступною кількістю. Після натискання кнопки додавання товар автоматично переноситься до кошика, що дозволяє користувачу формувати майбутнє замовлення.

Окрему увагу було приділено формі оформлення замовлення. Тут користувач вводить своє ім'я, номер телефону, адресу доставки та обирає спосіб отримання. Система перевіряє коректність цих даних ще до того, як замовлення буде створено.

Для оформлення замовлення користувач повинен заповнити контактні дані та обрати спосіб доставки.

Приклад заповненої форми наведено на рисунку 3.5.

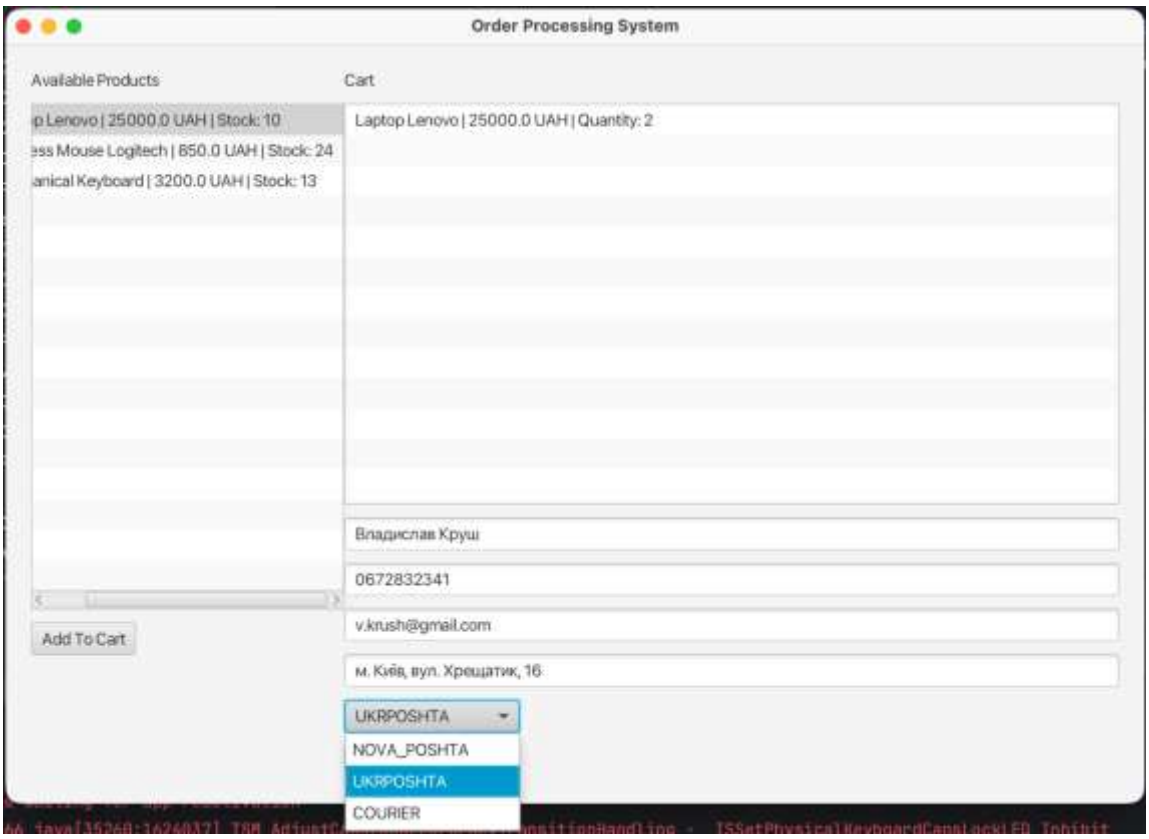


Рисунок 3.5 – Форма оформлення замовлення із заповненими даними покупця

Як видно з рисунка 3.5, форма оформлення замовлення містить поля для введення імені покупця, номера телефону, електронної пошти та адреси доставки. Крім того, користувач може обрати один із доступних способів доставки за допомогою компонента ComboBox. Заповнення цих полів є необхідною умовою для успішного створення замовлення.

Дизайном інтерфейсу передбачено, що у разі незаповнення обов'язкових реквізитів або введення даних у невідповідному форматі, операція блокується на рівні графічної оболонки. Технічні аспекти цієї валідації та візуалізація відповідних діалогових вікон детально розглянуті нижче у процесі тестування системи.

Приклад повідомлення, яке відображається при введенні некоректних даних, наведено на рисунку 3.6.

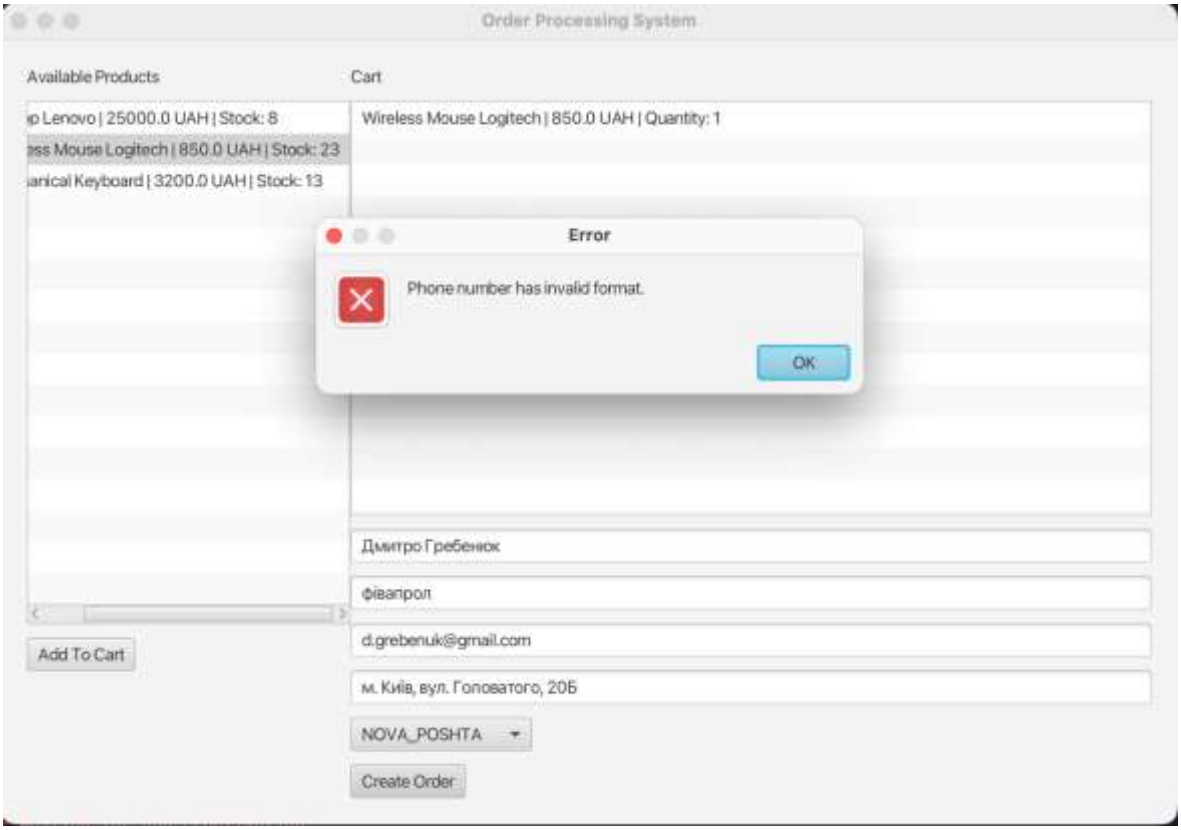


Рисунок 3.6 – Повідомлення про помилку під час введення некоректного номера телефону

Як видно з рисунка 3.6, у разі виявлення помилки система не дозволяє продовжити оформлення замовлення та повідомляє користувача про причину відмови. Такий механізм забезпечує додатковий контроль коректності введених даних і сприяє підвищенню надійності роботи програмного засобу.

Для обробки дій користувача застосовувались механізми Event Handling у JavaFX. Кожна кнопка та кожен інтерактивний елемент мають власний обробник подій, що викликає відповідні методи сервісного шару. Наприклад, після натискання кнопки «Оформити замовлення» запускається ланцюжок дій: перевірка даних, створення об'єкта замовлення та збереження інформації в системі.

Фрагмент реалізації обробника події натискання кнопки створення замовлення наведено у лістингу 3.6.

```
createOrderButton.setOnAction(event -> {

        if (cart.isEmpty()) {
        showError("Cart is empty.");
        return;
        }

        if (nameField.getText().isBlank()
        || phoneField.getText().isBlank()
        || emailField.getText().isBlank()
        || addressField.getText().isBlank()) {

        showError("Please fill in all customer fields.");
        return;
        }

        Order order = orderService.createOrder(
        customer,
        cart,
        deliveryDetails
        );
        });
```

Лістинг 3.6 – Фрагмент реалізації обробника події створення замовлення

Як видно з лістингу 3.6, після натискання кнопки створення замовлення система виконує перевірку стану кошика та коректності заповнення обов’язкових полів. Лише після завершення контролю керування передається сервісному шару, який відповідає за створення та збереження замовлення. Такий підхід дозволяє відокремити логіку інтерфейсу від бізнес-логіки та забезпечує коректну взаємодію між компонентами програмного засобу.

Після вдалого проходження всіх перевірок система створює замовлення та відображає користувачеві інформаційне повідомлення з результатом виконання операції.

Приклад такого повідомлення наведено на рисунку 3.7.

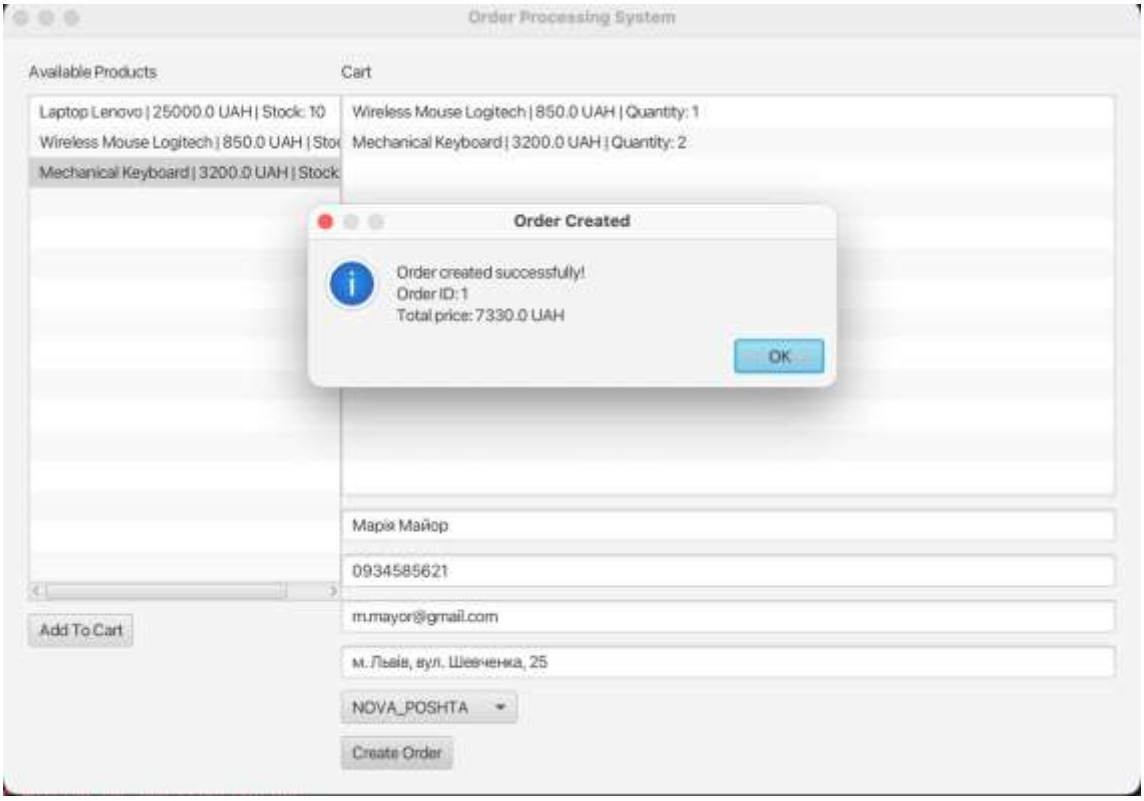


Рисунок 3.7 – Повідомлення про успішне створення замовлення

Як видно з рисунка 3.7, після успішного завершення оформлення замовлення система відображає інформацію про результат виконаної операції, зокрема номер створеного замовлення та його загальну вартість. Таке повідомлення забезпечує користувача зворотним зв’язком і підтверджує успішне завершення процесу оформлення покупки.

Структура інтерфейсу була спроєктована таким чином, щоб забезпечити простий і послідовний сценарій роботи користувача. Усі основні дії – перегляд товарів, формування кошика, введення контактних даних та створення замовлення – виконуються в межах одного вікна без необхідності переходу між додатковими екранами.

Для наочного відображення послідовності взаємодії користувача з програмним засобом було побудовано схему, наведену на рисунку 3.8.

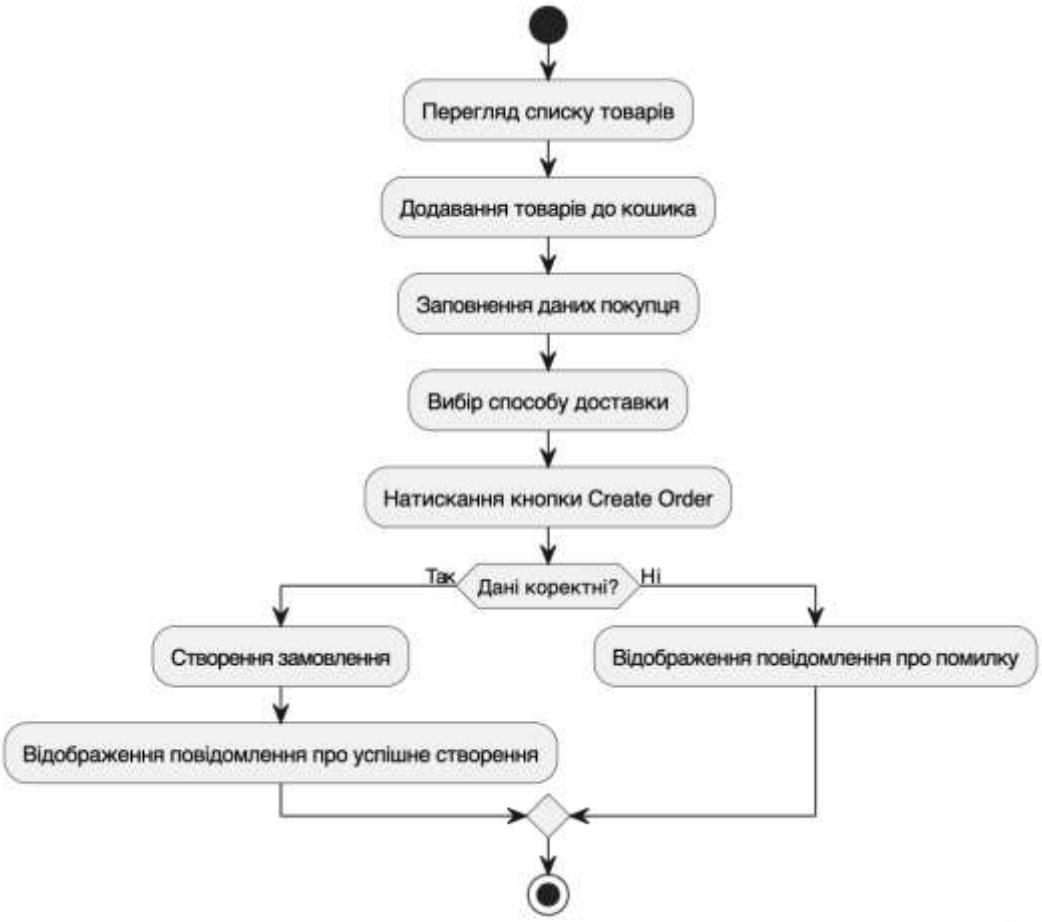


Рисунок 3.8 – Схема взаємодії користувача з інтерфейсом системи

Як видно з рисунка 3.8, робота користувача із системою складається з послідовності взаємопов’язаних дій: вибору товарів, формування кошика, введення контактних даних та створення замовлення. У разі виявлення помилки система повідомляє користувача про необхідність виправлення даних, а при успішному проходженні перевірок створює нове замовлення та відображає відповідне інформаційне повідомлення.

Таким чином, розроблений користувацький інтерфейс забезпечує зручну та послідовну взаємодію користувача з системою оформлення замовлення. Використання технології JavaFX дозволило реалізувати єдине робоче середовище для перегляду товарів, керування кошиком та оформлення замовлення. Поєднання візуальних компонентів, механізмів обробки подій та перевірки введених даних забезпечує комфортну роботу користувача та підвищує загальну якість програмного засобу.

3.4. Тестування програмного засобу та аналіз результатів роботи

Після завершення реалізації основних компонентів системи було проведено функціональне тестування програмного засобу. Оскільки розроблений застосунок є навчально-демонстраційною моделлю з графічним інтерфейсом, перевірка здійснювалась шляхом запуску програми, послідовного проходження основних користувацьких сценаріїв та контролю результатів безпосередньо у базі даних SQLite [8, 9].

Метою тестування було підтвердити коректність роботи ключових функцій: відображення товарів, додавання позицій до кошика, валідації введених даних, створення замовлення, збереження інформації у базі даних та автоматичного оновлення залишків товарів після покупки. Центральним об'єктом перевірки став процес оформлення замовлення – як основний бізнес-процес розробленої системи.

У межах тестування перевірялись такі сценарії:

– відображення списку товарів, завантажених із бази даних;

– додавання товару до кошика;

– спроба створення замовлення з порожнім кошиком;

– перевірка заповнення обов'язкових полів покупця;

– перевірка формату номера телефону та електронної пошти;

– успішне створення замовлення;

– збереження покупця, замовлення та позицій у таблицях SQLite;

– зменшення кількості товарів на складі після оформлення покупки.

Узагальнені результати перевірки наведено в таблиці 3.1.

Таблиця 3.1 – Результати функціонального тестування програмного засобу

№	Тестовий сценарій	Очікуваний результат	Фактичний результат	Статус
1	Запуск програми	Відкривається головне вікно з переліком товарів	Головне вікно відкрито, товари завантажено	Успішно

Продовження таблиці 3.1

2	Додавання товару до кошика	Обраний товар з’являється у кошику	Товар додано до кошика	Успішно
3	Створення замовлення з порожнім кошиком	Система показує повідомлення про помилку	Відображено повідомлення `Cart is empty.`	Успішно
4	Створення замовлення з порожніми полями покупця	Система блокує створення замовлення	Відображено повідомлення про необхідність заповнити поля	Успішно
5	Введення некоректного номера телефону	Система повідомляє про помилковий формат	Відображено повідомлення про некоректний формат номера	Успішно
6	Успішне створення замовлення	Система створює замовлення і показує його номер та суму	Замовлення створено, відображено повідомлення з ID і сумою	Успішно
7	Перевірка запису в таблицях SQLite	Дані покупця, замовлення та позицій збережені в БД	Записи наявні в таблицях `customers`, `orders`, `order_items`	Успішно
8	Перевірка залишків товарів	Кількість придбаних товарів зменшується	Значення в таблиці `products` оновлено	Успішно

Як видно з таблиці 3.1, усі перевірені сценарії виконані успішно. Система коректно реагує як на правильні дії користувача, так і на помилкові ситуації – зокрема, спробу оформити замовлення з порожнім кошиком або з некоректно введеними контактними даними.

Окремо було перевірено коректність збереження замовлень у базі даних SQLite. Після оформлення замовлення через графічний інтерфейс виконувався перегляд таблиці orders. Результат наведено на рисунку 3.9.

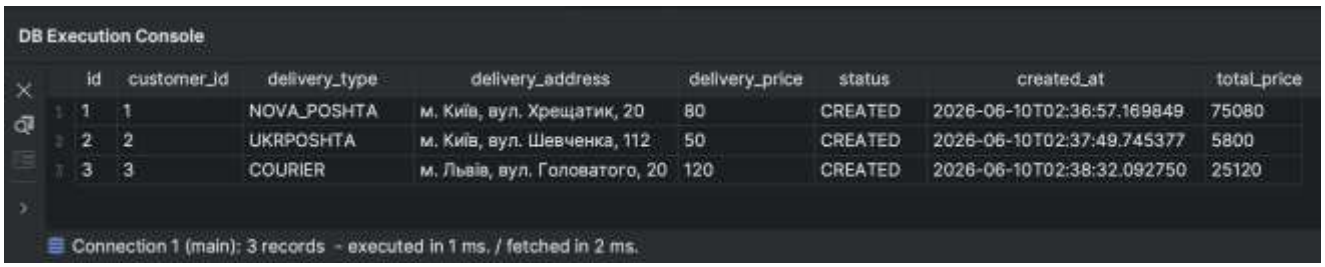


Рисунок 3.9 – Результат збереження замовлень у таблиці orders бази даних SQLite

Як видно з рисунка 3.9, кожне замовлення отримує унікальний ідентифікатор і пов'язується з відповідним покупцем через поле customer_id. Разом із цим зберігаються спосіб доставки, адреса, вартість доставки, статус, дата створення та загальна сума. Наявність цих записів підтверджує, що дані не лише відображаються в інтерфейсі, а й надійно зберігаються у базі даних.

Також було перевірено механізм оновлення залишків товарів. Після успішного оформлення замовлення система автоматично зменшує кількість товарів на складі відповідно до придбаних позицій. Результат наведено на рисунку 3.10.

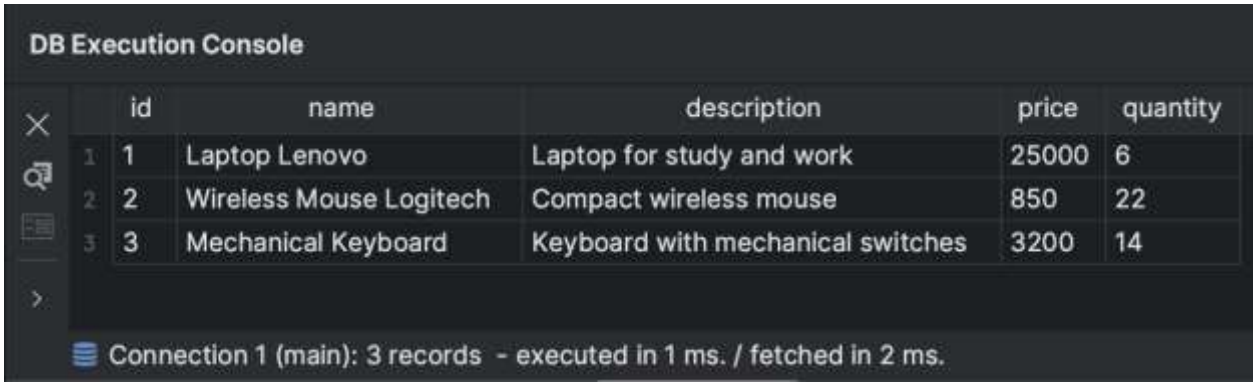


Рисунок 3.10 – Стан таблиці products після оформлення замовлень

З рисунка 3.10 видно, що після створення замовлень значення поля quantity у таблиці products автоматично змінюється відповідно до кількості придбаних товарів. Це підтверджує коректну взаємодію між бізнес-логікою системи та базою даних.

Додатково перевірялась узгодженість роботи всіх трьох рівнів системи: графічного інтерфейсу, сервісного шару та бази даних. Результати підтвердили, що після будь-якої дії користувача зміни коректно відображаються як у вікнах програми, так і у відповідних таблицях SQLite. Зокрема, замовлення успішно зберігаються в таблицях customers, orders та order_items, а залишки товарів оновлюються автоматично.

Під час тестування бізнес-логіки окрему увагу було приділено розрахунку загальної вартості замовлення. Перевірялись різні комбінації товарів і кількостей – у всіх випадках система правильно обчислила підсумкову суму та сформувала коректну структуру замовлення.

За результатами проведеного тестування підтверджено, що програмний засіб коректно реалізує основні функції моделювання бізнес-процесу оформлення замовлення. Перевірка також засвідчила стабільну взаємодію між усіма компонентами системи – інтерфейсом, бізнес-логікою і базою даних. Отримані результати свідчать про готовність програмного засобу до використання як демонстраційної моделі системи електронної комерції.

3.5. Інструкція користувача та практичне застосування системи

Після завершення розробки та тестування програмного засобу важливим етапом є оцінка його зручності для кінцевого користувача. Навіть функціонально завершена система не може вважатися успішною, якщо її використання є складним або неінтуїтивним. Саме тому під час розробки особлива увага приділялась створенню простого та зрозумілого сценарію взаємодії користувача із системою.

Розроблений програмний засіб призначений для моделювання бізнес-процесу оформлення замовлення в системі електронної комерції. Програма дозволяє переглядати перелік доступних товарів, формувати кошик, вводити контактні дані покупця, обирати спосіб доставки та створювати замовлення з подальшим збереженням інформації у базі даних SQLite [13, 14, 16].

Для запуску системи необхідно відкрити проєкт у середовищі розробки IntelliJ IDEA та виконати запуск головного класу застосунку. Після запуску на екрані відображається головне вікно програми, яке містить список доступних товарів, кошик користувача та форму оформлення замовлення.

Крок 1. Запуск та ознайомлення з інтерфейсом. Після ініціалізації проєкту на екрані з'являється головне вікно програми.

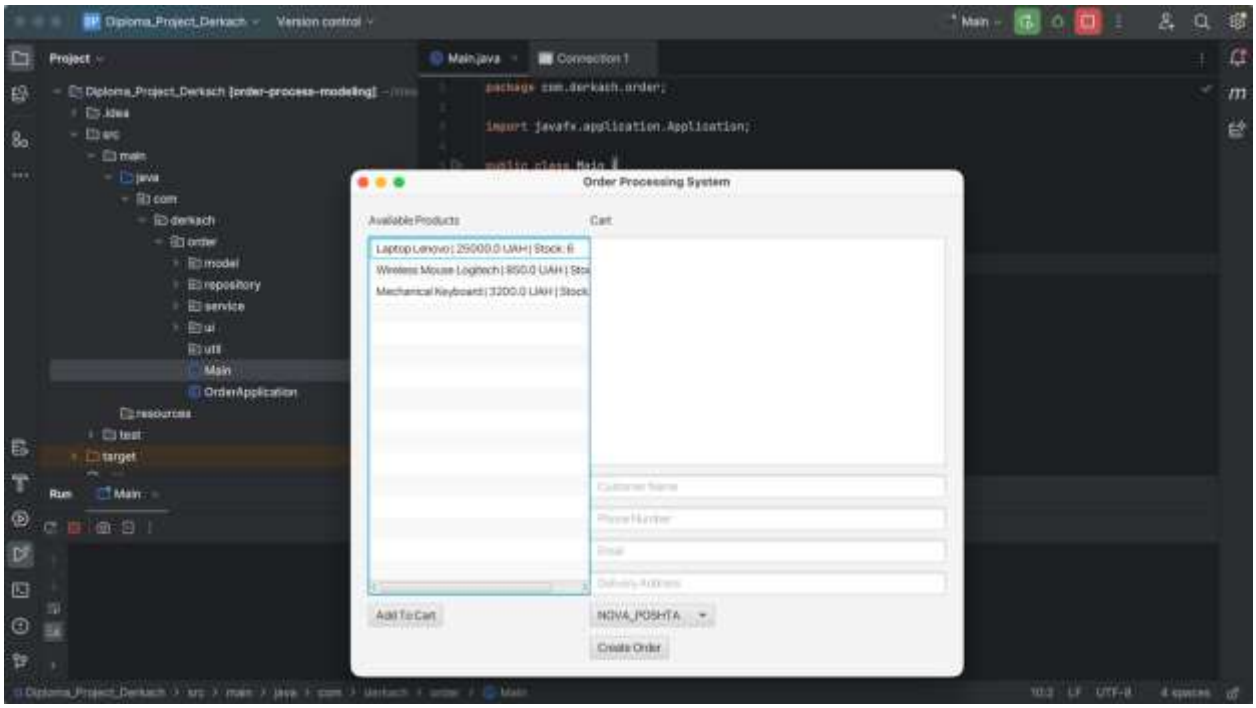


Рисунок 3.11 – Головне вікно системи після запуску

Як видно з рисунка 3.11, усі основні функції системи зосереджені в одному вікні. У лівій частині інтерфейсу відображається список доступних товарів із зазначенням їх вартості та кількості на складі. У правій частині розташовано кошик користувача та форму для введення даних, необхідних для оформлення замовлення.

Робота із системою починається з вибору товарів. Для додавання товару до кошика користувач обирає потрібну позицію зі списку та натискає кнопку «Add To Cart». Після цього інформація про вибраний товар автоматично відображається у кошику. Користувач може додати декілька товарів та сформувати замовлення відповідно до власних потреб.

Після формування кошика необхідно заповнити форму оформлення замовлення. Для цього користувач вводить своє ім’я, номер телефону, електронну пошту, адресу доставки та обирає один із доступних способів доставки.

Крок 2. Наповнення кошика та заповнення реквізитів. Користувач обирає товари натисканням кнопки «Add To Cart» та переходить до форми введення персональних даних.

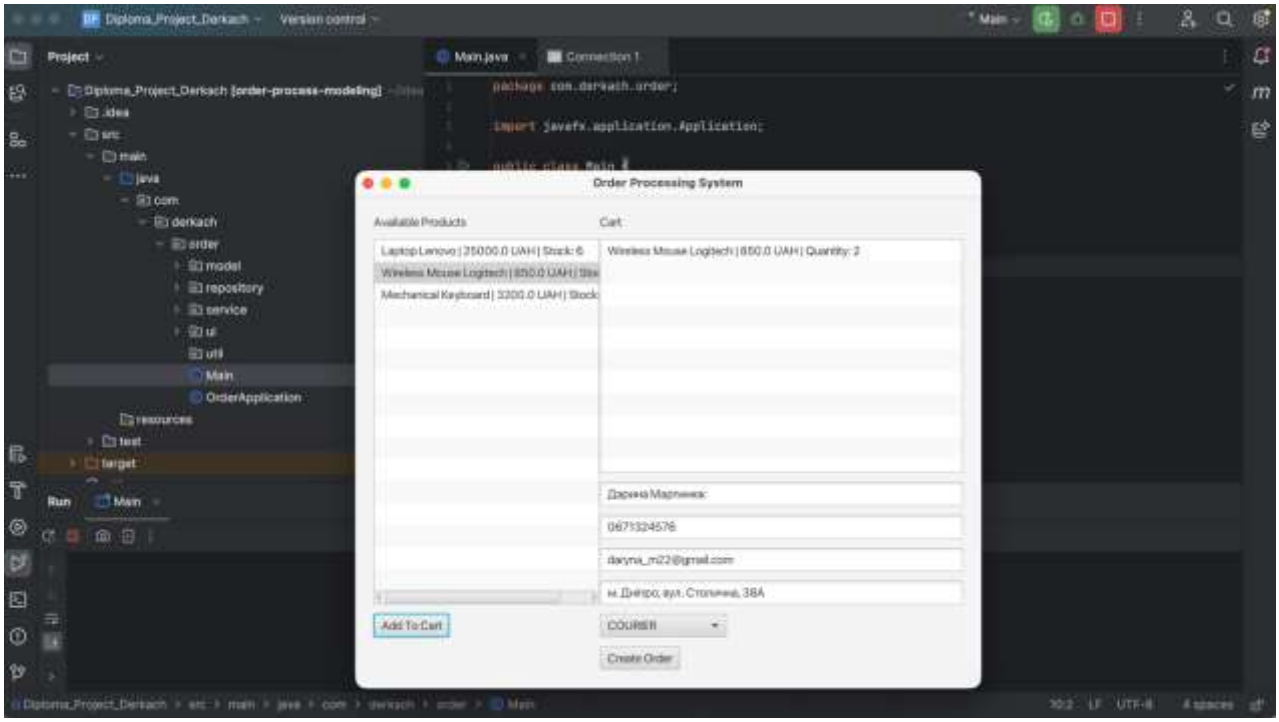


Рисунок 3.12 – Процес оформлення замовлення

Як видно з рисунка 3.12, система надає можливість ввести всі необхідні дані для створення замовлення в межах одного вікна. Перед обробкою введеної інформації виконується перевірка коректності заповнення всіх обов’язкових полів. Це дозволяє уникнути створення замовлень із неповними або некоректними даними.

У випадку виявлення помилки система повідомляє користувача про причину відмови та не дозволяє продовжити виконання операції до моменту виправлення введених даних. Такий механізм підвищує надійність роботи

програмного засобу та забезпечує коректність інформації, що зберігається у базі даних.

Після успішного проходження всіх перевірок система створює нове замовлення, зберігає інформацію про покупця та замовлені товари у базі даних, оновлює залишки товарів на складі та відображає повідомлення про успішне завершення операції.

Крок 3. Підтвердження транзакції. Після натискання кнопки «Оформити замовлення» система генерує фінальне вікно-повідомлення.

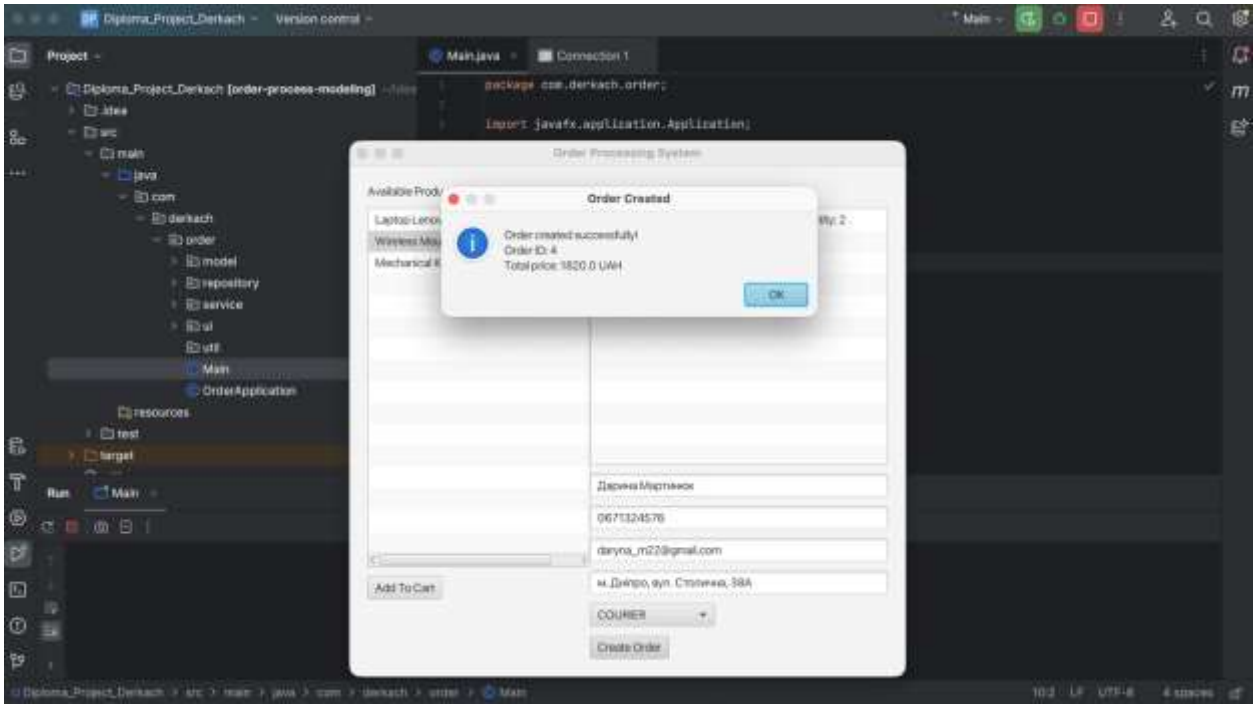


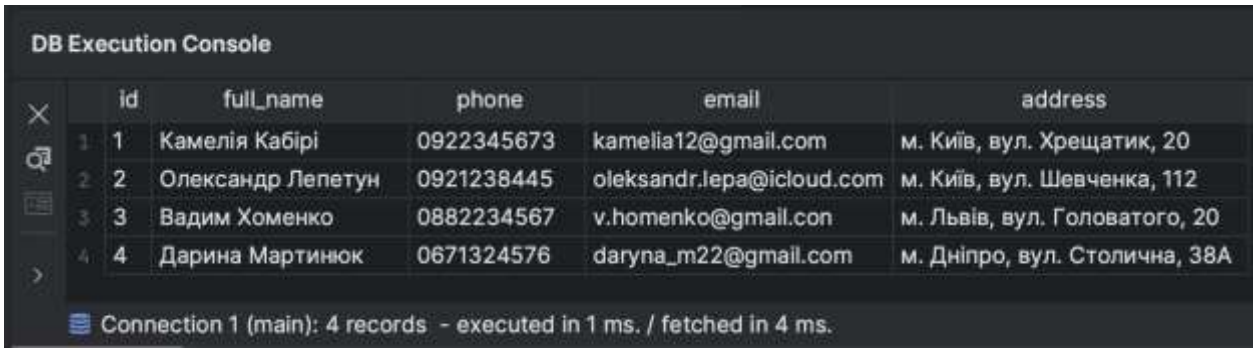
Рисунок 3.13 – Повідомлення про успішне створення замовлення

Як видно з рисунка 3.13, після завершення операції система відображає інформацію про створене замовлення, зокрема його ідентифікатор та загальну вартість. Це підтверджує успішне виконання операції та надає користувачеві зворотний зв’язок щодо результату роботи системи.

Після завершення операції результати роботи системи також відображаються у базі даних SQLite. Під час створення замовлення автоматично формуються записи про покупця та саме замовлення, які зберігаються у відповідних таблицях бази даних.

Крок 4. Перевірка синхронізації з базою даних. Для адміністратора системи результатом успішної роботи користувача є поява нових записів у реляційних таблицях.

Приклад збереження інформації про нового покупця наведено на рисунку 3.14.



DB Execution Console

	id	full_name	phone	email	address
1	1	Камелія Кабірі	0922345673	kamelia12@gmail.com	м. Київ, вул. Хрещатик, 20
2	2	Олександр Лепетун	0921238445	oleksandr.lepa@icloud.com	м. Київ, вул. Шевченка, 112
3	3	Вадим Хоменко	0882234567	v.homenko@gmail.con	м. Львів, вул. Головатого, 20
4	4	Дарина Мартинюк	0671324576	daryna_m22@gmail.com	м. Дніпро, вул. Столична, 38А

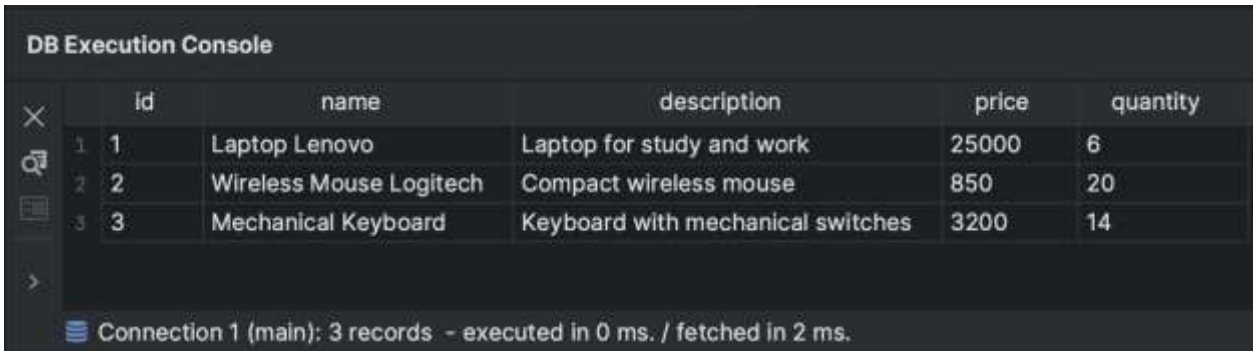
Connection 1 (main): 4 records - executed in 1 ms. / fetched in 4 ms.

Рисунок 3.14 – Збереження інформації про покупця у таблиці customers

Як видно з рисунка 3.14, після успішного оформлення замовлення система автоматично створює новий запис у таблиці customers та зберігає контактні дані покупця. Це підтверджує коректну роботу механізму взаємодії між графічним інтерфейсом, бізнес-логікою та базою даних.

Крім збереження інформації про покупця, система також автоматично оновлює залишки товарів на складі відповідно до кількості придбаних позицій.

Результат оновлення даних наведено на рисунку 3.15.



DB Execution Console

	id	name	description	price	quantity
1	1	Laptop Lenovo	Laptop for study and work	25000	6
2	2	Wireless Mouse Logitech	Compact wireless mouse	850	20
3	3	Mechanical Keyboard	Keyboard with mechanical switches	3200	14

Connection 1 (main): 3 records - executed in 0 ms. / fetched in 2 ms.

Рисунок 3.15 – Оновлення залишків товарів після створення замовлення

Як видно з рисунка 3.15, після оформлення замовлення значення поля quantity автоматично змінюється відповідно до кількості придбаних товарів. Це свідчить про коректну реалізацію механізму обліку товарних залишків та підтверджує правильність роботи сервісного шару програми.

Після створення замовлення користувач отримує підтвердження успішного виконання операції, а результати роботи системи зберігаються у базі даних. Це дозволяє використовувати програмний засіб не лише як приклад графічного інтерфейсу, а як повноцінну демонстраційну модель, у якій взаємодія користувача призводить до реальних змін у структурованих даних.

З практичної точки зору розроблений програмний засіб може використовуватись як навчальна модель інформаційної системи електронної комерції. Система демонструє застосування об’єктно-орієнтованого підходу, використання багаторівневої архітектури, реалізацію бізнес-логіки оформлення замовлення, побудову графічного інтерфейсу засобами JavaFX та організацію роботи з базою даних SQLite.

Крім того, архітектура програмного засобу залишає можливість для подальшого розвитку. У майбутньому до системи можуть бути додані механізми авторизації користувачів, перегляд історії замовлень, інтеграція з платіжними сервісами, формування звітів та інші функції, характерні для сучасних систем електронної комерції.

Таким чином, розроблений програмний засіб реалізує базовий цикл оформлення замовлення та демонструє практичне застосування об’єктно-орієнтованого підходу під час створення інформаційних систем мовою Java.

Висновки до розділу 3

У третьому розділі було виконано повну практичну реалізацію програмного засобу для моделювання бізнес-процесу оформлення замовлення мовою Java. У процесі розробки було реалізовано всі основні компоненти програмного засобу – від структури пакетів до готового застосунку з

дотриманням принципів об'єктно-орієнтованого підходу та чіткого розподілу відповідальності між компонентами.

Програмний засіб охоплює чотири основні шари: моделі предметної області, сервісний шар із бізнес-логікою, графічний інтерфейс на основі JavaFX та механізм збереження даних у базі даних SQLite. Завдяки такому поділу бізнес-логіка залишається незалежною від інтерфейсу, а кожен компонент можна змінювати або розширювати, не зачіпаючи решту системи.

У частині бізнес-логіки реалізовано роботу з кошиком, створення та збереження замовлень, валідацію введених даних, розрахунок загальної вартості та керування статусами. Разом ці механізми охоплюють повний життєвий цикл замовлення – від додавання першого товару до фінального підтвердження покупки.

Графічний інтерфейс спроєктовано так, щоб користувач міг пройти весь процес оформлення послідовно й без зайвих кроків: перегляд товарів, формування кошика, введення контактних даних і підтвердження замовлення – усе в межах одного зрозумілого робочого середовища.

Проведене функціональне тестування підтвердило коректність роботи всіх ключових функцій системи. Перевірка охопила як штатні сценарії, так і граничні випадки – порожній кошик, некоректні контактні дані, відсутність обов'язкових полів. У всіх випадках система повелась очікувано: замовлення зберігаються у базі даних, залишки товарів оновлюються автоматично, а помилки користувача обробляються коректно.

Таким чином, результатом третього розділу є повністю працездатний програмний засіб, який не лише реалізує основні механізми оформлення замовлення, а й наочно демонструє, як об'єктно-орієнтований підхід і багаторівнева архітектура застосовуються при побудові реальних інформаційних систем мовою Java.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було розроблено програмний засіб для моделювання бізнес-процесу оформлення замовлення мовою Java. Дослідження охопило повний життєвий цикл проєктування – від аналізу предметної області до створення функціонуючого застосунку з графічним інтерфейсом та реляційним сховищем даних.

На першому етапі було досліджено предметну область і встановлено, що процес оформлення замовлення є центральною ланкою будь-якої системи електронної комерції. Саме він завершує взаємодію користувача з платформою і запускає подальший ланцюжок обробки покупки. Аналіз існуючих рішень показав, що готові платформи, попри свою функціональність, рідко дозволяють гнучко моделювати власну бізнес-логіку – що і обґрунтувало доцільність розробки окремого засобу.

На етапі проєктування було сформовано архітектуру системи, виконано UML-моделювання та спроєктовано структуру даних. Для реалізації обрано мову Java, яка повноцінно підтримує об'єктно-орієнтований підхід і відповідає специфіці предметної області, де кожна сутність – покупець, товар, замовлення, доставка – має чітко визначену поведінку і стан.

Програмна реалізація базується на використанні багаторівневої архітектури, що включає чотири взаємопов'язані шари системи: моделей предметної області, сервісного шару з бізнес-логікою, графічного інтерфейсу на основі JavaFX та механізму збереження даних у SQLite. Такий поділ дозволив забезпечити незалежність компонентів і зробити систему зручною для підтримки та розширення.

Проведене функціональне тестування підтвердило коректність роботи всіх ключових сценаріїв: від додавання товарів до кошика до збереження готового замовлення в базі даних із автоматичним оновленням товарних залишків. Програмний засіб демонструє стабільність при виконанні

стандартних користувацьких сценаріїв та коректно обробляє виключні ситуації.

Практичне значення роботи полягає в тому, що розроблений засіб може використовуватись як навчальна модель для демонстрації принципів ООП, багаторівневої архітектури та роботи з базами даних у середовищі Java. Поставлену мету досягнуто, усі визначені завдання виконано.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Bloch J. Effective Java. 3rd ed. Boston : Addison-Wesley Professional, 2018. 416 p.
2. Schildt H. Java: The Complete Reference. 12th ed. New York : McGraw-Hill Education, 2021. 1248 p.
3. Horstmann C. Core Java. Volume I – Fundamentals. 12th ed. Hoboken : Pearson, 2021. 928 p.
4. Horstmann C. Core Java. Volume II – Advanced Features. 12th ed. Hoboken : Pearson, 2022. 1120 p.
5. Freeman E., Robson E. Head First Design Patterns. 2nd ed. Sebastopol : O'Reilly Media, 2020. 694 p.
6. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. Boston : Addison-Wesley Professional, 1994. 416 p.
7. Fowler M. UML Distilled: A Brief Guide to the Standard Object Modeling Language. 3rd ed. Boston : Addison-Wesley Professional, 2003. 208 p.
8. Pressman R. S., Maxim B. R. Software Engineering: A Practitioner’s Approach. 9th ed. New York : McGraw-Hill Education, 2019. 880 p.
9. Sommerville I. Software Engineering. 10th ed. Boston : Pearson, 2015. 816 p.
10. Martin R. C. Clean Code: A Handbook of Agile Software Craftsmanship. Boston : Prentice Hall, 2008. 464 p.
11. Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Boston : Prentice Hall, 2017. 432 p.
12. Evans E. Domain-Driven Design: Tackling Complexity in the Heart of Software. Boston : Addison-Wesley Professional, 2003. 560 p.
13. Oracle Corporation. Java Platform, Standard Edition Documentation. URL: https://docs.oracle.com/en/java/ (дата звернення: 10.06.2026).

14. Oracle Corporation. JavaFX Documentation. URL: https://openjfx.io (дата звернення: 10.06.2026).
15. Oracle Corporation. JDBC API Documentation. URL: https://docs.oracle.com/javase/tutorial/jdbc/ (дата звернення: 10.06.2026).
16. SQLite Documentation. URL: https://www.sqlite.org/docs.html (дата звернення: 10.06.2026).
17. Apache Maven Project Documentation. URL: https://maven.apache.org/guides/ (дата звернення: 10.06.2026).
18. Git Documentation. URL: https://git-scm.com/doc (дата звернення: 10.06.2026).
19. IntelliJ IDEA Documentation. URL: https://www.jetbrains.com/help/idea/ (дата звернення: 10.06.2026).
20. OpenJFX Documentation. URL: https://openjfx.io/javadoc/ (дата звернення: 10.06.2026).
21. Fowler M. Patterns of Enterprise Application Architecture. Boston : Addison-Wesley Professional, 2002. 533 p.
22. Larman C. Applying UML and Patterns. 3rd ed. Upper Saddle River : Prentice Hall, 2004. 736 p.
23. BPMN Method and Style. 3rd ed. Cody-Cassidy Press, 2021. 540 p.
24. Laudon K. C., Traver C. G. E-Commerce 2023–2024: Business, Technology, Society. 18th ed. Pearson, 2023. 912 p.
25. Єрмошенко М. М., Нестеренко О. В., Штулер І. Ю. Інформаційні технології аналізу даних у маркетингу : навчальний посібник. Київ : Національна академія управління, 2021. 141 с.
26. Нестеренко О. В. Метод пріоритезації вимог в програмній інженерії. Проблеми програмування. 2024. № 2–3. С. 132–139.
27. Dumas M., La Rosa M., Mendling J., Reijers H. Fundamentals of Business Process Management. 3rd ed. Berlin : Springer, 2023. 527 p.

28. Нестеренко О. В. Інформаційні системи управління підприємствами : навчальний посібник. Київ : УкрНЦ РІТ, 2019. 135 с.
29. Coronel C., Morris S. Database Systems: Design, Implementation and Management. 14th ed. Boston : Cengage Learning, 2022. 832 p.
30. Elmasri R., Navathe S. Fundamentals of Database Systems. 7th ed. Boston : Pearson, 2017. 1272 p.
31. Silberschatz A., Korth H., Sudarshan S. Database System Concepts. 7th ed. New York : McGraw-Hill Education, 2019. 1376 p.
32. Falovskyi O. O., Nesterenko O. V. Basics of Database Design and Using : Tutorial. Kyiv : Тропеа, 2023. 83 p.
33. Beaulieu A. Learning SQL. 3rd ed. Sebastopol : O'Reilly Media, 2020. 394 p.
34. SQLite Home Page. URL: https://www.sqlite.org (дата звернення: 10.06.2026).
35. SQLite Tutorial. URL: https://www.sqlitetutorial.net (дата звернення: 10.06.2026).
36. JetBrains. IntelliJ IDEA Documentation. URL: https://www.jetbrains.com/help/idea (дата звернення: 10.06.2026).
37. Oracle Corporation. JavaFX API Documentation. URL: https://openjfx.io/javadoc (дата звернення: 10.06.2026).
38. OpenJFX Project Documentation. URL: https://openjfx.io (дата звернення: 10.06.2026).
39. Apache Maven Documentation. URL: https://maven.apache.org/guides (дата звернення: 10.06.2026).
40. Git SCM Documentation. URL: https://git-scm.com/doc (дата звернення: 10.06.2026).

ДОДАТКИ

Додаток А – Фрагменти програмного коду сервісного шару

```
package com.derkach.order.service;

import com.derkach.order.model.Cart;
import com.derkach.order.model.CartItem;
import com.derkach.order.model.Customer;
import com.derkach.order.model.DeliveryDetails;
import com.derkach.order.model.Order;
import com.derkach.order.model.OrderItem;
import com.derkach.order.model.OrderStatus;
import com.derkach.order.repository.OrderRepository;

import java.util.ArrayList;
import java.util.List;

public class OrderService {
    private final OrderRepository orderRepository;
    private final ProductService productService;

    public OrderService(OrderRepository orderRepository, ProductService
productService) {
        if (orderRepository == null) {
            throw new IllegalArgumentException("Order repository cannot be
null.");
        }
        if (productService == null) {
            throw new IllegalArgumentException("Product service cannot be
null.");
        }

        this.orderRepository = orderRepository;
        this.productService = productService;
    }

    public Order createOrder(Customer customer, Cart cart, DeliveryDetails
deliveryDetails) {
        if (customer == null) {
            throw new IllegalArgumentException("Customer cannot be null.");
        }
        if (cart == null || cart.isEmpty()) {
            throw new IllegalArgumentException("Cart cannot be empty.");
        }
        if (deliveryDetails == null) {
            throw new IllegalArgumentException("Delivery details cannot be
null.");
        }

        List<OrderItem> orderItems = new ArrayList<>();

        for (CartItem cartItem : cart.getItems()) {
            OrderItem orderItem = new OrderItem(cartItem.getProduct(),
cartItem.getQuantity());
            orderItems.add(orderItem);
        }

        Order order = new Order(customer, orderItems, deliveryDetails);
        orderRepository.save(order);

        for (CartItem cartItem : cart.getItems()) {
```

```
            productService.decreaseProductQuantity(
                    cartItem.getProduct().getId(),
                    cartItem.getQuantity()
            );
        }

        cart.clear();

        return order;
    }

    public void confirmOrder(Order order) {
        changeOrderStatus(order, OrderStatus.CONFIRMED);
    }

    public void processOrder(Order order) {
        changeOrderStatus(order, OrderStatus.PROCESSING);
    }

    public void shipOrder(Order order) {
        changeOrderStatus(order, OrderStatus.SHIPPED);
    }

    public void deliverOrder(Order order) {
        changeOrderStatus(order, OrderStatus.DELIVERED);
    }

    public void cancelOrder(Order order) {
        if (order == null) {
            throw new IllegalArgumentException("Order cannot be null.");
        }

        order.cancel();
    }

    private void changeOrderStatus(Order order, OrderStatus status) {
        if (order == null) {
            throw new IllegalArgumentException("Order cannot be null.");
        }

        order.changeStatus(status);
    }
}
```

```
package com.derkach.order.repository;

import com.derkach.order.model.Customer;
import com.derkach.order.model.Order;
import com.derkach.order.model.OrderItem;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class OrderRepository {

    public void save(Order order) {
        if (order == null) {
            throw new IllegalArgumentException("Order cannot be null.");
        }

        try (Connection connection = DatabaseConnection.getConnection()) {
```

```
            connection.setAutoCommit(false);

            int generatedCustomerId = saveCustomer(connection,
order.getCustomer());
            order.getCustomer().setId(generatedCustomerId);

            int generatedOrderId = saveOrder(connection, order);
            order.setId(generatedOrderId);

            saveOrderItems(connection, order);

            connection.commit();
        } catch (SQLException e) {
            throw new RuntimeException("Order saving failed.", e);
        }
    }

    private int saveCustomer(Connection connection, Customer customer) throws
SQLException {
        String sql = """
                INSERT INTO customers (full_name, phone, email, address)
                VALUES (?, ?, ?, ?);
                """;

        try (PreparedStatement statement = connection.prepareStatement(sql,
Statement.RETURN_GENERATED_KEYS)) {
            statement.setString(1, customer.getFullName());
            statement.setString(2, customer.getPhone());
            statement.setString(3, customer.getEmail());
            statement.setString(4, customer.getAddress());
            statement.executeUpdate();

            try (ResultSet generatedKeys = statement.getGeneratedKeys()) {
                if (generatedKeys.next()) {
                    return generatedKeys.getInt(1);
                }
            }
        }

        throw new SQLException("Creating customer failed, no ID obtained.");
    }

    private int saveOrder(Connection connection, Order order) throws
SQLException {
        String sql = """
                INSERT INTO orders (
                    customer_id, delivery_type, delivery_address,
                    delivery_price, status, created_at, total_price
                )
                VALUES (?, ?, ?, ?, ?, ?, ?);
                """;

        try (PreparedStatement statement = connection.prepareStatement(sql,
Statement.RETURN_GENERATED_KEYS)) {
            statement.setInt(1, order.getCustomer().getId());
            statement.setString(2,
order.getDeliveryDetails().getDeliveryType().name());
            statement.setString(3, order.getDeliveryDetails().getAddress());
            statement.setDouble(4,
order.getDeliveryDetails().getDeliveryPrice());
            statement.setString(5, order.getStatus().name());
            statement.setString(6, order.getCreatedAt().toString());
            statement.setDouble(7, order.getTotalPrice());
            statement.executeUpdate();
```

```
            try (ResultSet generatedKeys = statement.getGeneratedKeys()) {
                if (generatedKeys.next()) {
                    return generatedKeys.getInt(1);
                }
            }
        }

        throw new SQLException("Creating order failed, no ID obtained.");
    }

    private void saveOrderItems(Connection connection, Order order) throws
SQLException {
        String sql = """
                INSERT INTO order_items (
                    order_id, product_id, product_name,
                    quantity, price_at_purchase, total_price
                )
                VALUES (?, ?, ?, ?, ?, ?);
                """;

        try (PreparedStatement statement = connection.prepareStatement(sql))
{
            for (OrderItem item : order.getItems()) {
                statement.setInt(1, order.getId());
                statement.setInt(2, item.getProduct().getId());
                statement.setString(3, item.getProduct().getName());
                statement.setInt(4, item.getQuantity());
                statement.setDouble(5, item.getPriceAtPurchase());
                statement.setDouble(6, item.getTotalPrice());
                statement.addBatch();
            }

            statement.executeBatch();
        }
    }
}
```

```
SELECT * FROM orders;
SELECT * FROM order_items;
SELECT * FROM customers;
SELECT * FROM products;
```

Додаток В - Результати роботи бази даних

Таблиця Orders

id	customer_id	delivery_type	delivery_address	delivery_price	status	created_at	total_price
1	1	NOVA_POSHTA	м. Київ, вул. Хрещатик, 20	80	CREATED	2026-06-10T02:36:57.169849	75080
2	2	UKRPOSHTA	м. Київ, вул. Шевченка, 112	50	CREATED	2026-06-10T02:37:49.745377	5800
3	3	COURIER	м. Львів, вул. Головатого, 20	120	CREATED	2026-06-10T02:38:32.092750	25120
4	4	COURIER	м. Дніпро, вул. Столична, 38А	120	CREATED	2026-06-10T13:39:09.054644	1820

Таблиця Orders Item

id	order_id	product_id	product_name	quantity	price_at_purchase	total_price
1	1	1	Laptop Lenovo	3	25000	75000
2	2	2	Wireless Mouse Logitech	3	850	2550
3	2	3	Mechanical Keyboard	1	3200	3200
4	3	1	Laptop Lenovo	1	25000	25000
5	4	2	Wireless Mouse Logitech	2	850	1700

Таблиця Customers

	id	full_name	phone	email	address
1	1	Камелія Кабірі	0922345673	kamelia12@gmail.com	м. Київ, вул. Хрещатик, 20
2	2	Олександр Лепетун	0921238445	oleksandr.lepa@icloud.com	м. Київ, вул. Шевченка, 112
3	3	Вадим Хоменко	0882234567	v.homenko@gmail.con	м. Львів, вул. Головатого, 20
4	4	Дарина Мартинюк	0671324576	daryna_m22@gmail.com	м. Дніпро, вул. Столична, 38А

Таблиця Products

id	name	description	price	quantity
1	Laptop Lenovo	Laptop for study and work	25000	6
2	Wireless Mouse Logitech	Compact wireless mouse	850	20
3	Mechanical Keyboard	Keyboard with mechanical switches	3200	14

```
package com.derkach.order.ui;

import com.derkach.order.model.Cart;
import com.derkach.order.model.CartItem;
import com.derkach.order.model.Customer;
import com.derkach.order.model.DeliveryDetails;
import com.derkach.order.model.DeliveryType;
import com.derkach.order.model.Order;
import com.derkach.order.model.Product;
import com.derkach.order.repository.OrderRepository;
import com.derkach.order.repository.ProductRepository;
import com.derkach.order.service.OrderService;
import com.derkach.order.service.ProductService;
import javafx.geometry.Insets;
import javafx.scene.control.Alert;
import javafx.scene.control.Button;
import javafx.scene.control.ComboBox;
import javafx.scene.control.Label;
import javafx.scene.control.ListView;
import javafx.scene.control.TextField;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.VBox;

public class MainView extends BorderPane {

    public MainView() {

        setPadding(new Insets(20));

        ProductRepository productRepository = new ProductRepository();
        ProductService productService = new
ProductService(productRepository);

        OrderRepository orderRepository = new OrderRepository();
        OrderService orderService = new OrderService(orderRepository,
productService);

        Cart cart = new Cart();

        Label productsLabel = new Label("Available Products");

        ListView<Product> productsList = new ListView<>();
        productsList.setCellFactory(list -> new ProductListCell());
        productsList.getItems().addAll(productService.getAllProducts());

        Button addToCartButton = new Button("Add To Cart");

        Label cartLabel = new Label("Cart");

        ListView<CartItem> cartList = new ListView<>();
        cartList.setCellFactory(list -> new CartItemListCell());

        TextField nameField = new TextField();
        nameField.setPromptText("Customer Name");

        TextField phoneField = new TextField();
        phoneField.setPromptText("Phone Number");

        TextField emailField = new TextField();
        emailField.setPromptText("Email");

        TextField addressField = new TextField();
```

```
        addressField.setPromptText("Delivery Address");

        ComboBox<DeliveryType> deliveryTypeComboBox = new ComboBox<>();
        deliveryTypeComboBox.getItems().addAll(DeliveryType.values());
        deliveryTypeComboBox.setValue(DeliveryType.NOVA_POSHTA);

        Button createOrderButton = new Button("Create Order");

        addToCartButton.setOnAction(event -> {
            Product selectedProduct =
productsList.getSelectionModel().getSelectedItem();

            if (selectedProduct != null) {
                try {
                    cart.addProduct(selectedProduct, 1);
                    refreshCartList(cartList, cart);
                } catch (IllegalArgumentException exception) {
                    showError(exception.getMessage());
                }
            }
        });

        createOrderButton.setOnAction(event -> {

            if (cart.isEmpty()) {
                showError("Cart is empty.");
                return;
            }

            if (nameField.getText().isBlank()
                    || phoneField.getText().isBlank()
                    || emailField.getText().isBlank()
                    || addressField.getText().isBlank()) {

                showError("Please fill in all customer fields.");
                return;
            }

            try {
                Customer customer = new Customer(
                        nameField.getText(),
                        phoneField.getText(),
                        emailField.getText(),
                        addressField.getText()
                );

                DeliveryDetails deliveryDetails = new DeliveryDetails(
                        deliveryTypeComboBox.getValue(),
                        addressField.getText()
                );

                Order order = orderService.createOrder(
                        customer,
                        cart,
                        deliveryDetails
                );

                Alert alert = new Alert(Alert.AlertType.INFORMATION);
                alert.setTitle("Order Created");
                alert.setHeaderText(null);
                alert.setContentText(
                        "Order created successfully!\n"
                                + "Order ID: " + order.getId() + "\n"
                                + "Total price: " + order.getTotalPrice() + "
```

```
UAH"
                );
                alert.showAndWait();

                cartList.getItems().clear();
                refreshProductsList(productsList, productService);

                nameField.clear();
                phoneField.clear();
                emailField.clear();
                addressField.clear();
                deliveryTypeComboBox.setValue(DeliveryType.NOVA_POSHTA);

            } catch (IllegalArgumentException | IllegalStateException
exception) {
                showError(exception.getMessage());
            }
        });

        VBox leftPanel = new VBox(10);
        leftPanel.getChildren().addAll(
                productsLabel,
                productsList,
                addToCartButton
        );

        VBox rightPanel = new VBox(10);
        rightPanel.getChildren().addAll(
                cartLabel,
                cartList,
                nameField,
                phoneField,
                emailField,
                addressField,
                deliveryTypeComboBox,
                createOrderButton
        );

        setLeft(leftPanel);
        setCenter(rightPanel);
    }

    private void refreshCartList(ListView<CartItem> cartList, Cart cart) {
        cartList.getItems().clear();
        cartList.getItems().addAll(cart.getItems());
    }

    private void refreshProductsList(ListView<Product> productsList,
ProductService productService) {
        productsList.getItems().clear();
        productsList.getItems().addAll(productService.getAllProducts());
    }

    private void showError(String message) {
        Alert alert = new Alert(Alert.AlertType.ERROR);
        alert.setTitle("Error");
        alert.setHeaderText(null);
        alert.setContentText(message);
        alert.showAndWait();
    }
}
```